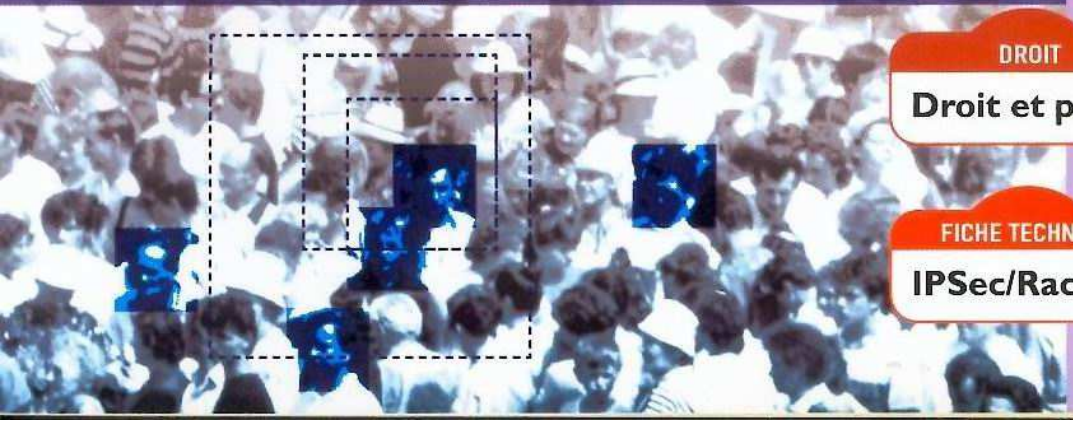
misc
MULTI-SYSTEM & INTERNET SECURITY COOKBOOK

15 septembre octobre 2004

100 % SÉCURITÉ INFORMATIQUE

Authentification

La serrure du système d'information : qui accède à vos données ?



SYSTÈME
Heap overflows fiables sous Windows

DROIT
Droit et pot de miel

FICHE TECHNIQUE
IPSec/Racoon sous Linux

Sommaire

[1] http://support.microsoft.com/default.aspx?kbid=884130
http://support.microsoft.com/default.aspx?kbid=842242
http://support.microsoft.com/default.aspx?kbid=875357
http://support.microsoft.com/default.aspx?kbid=875351

[2] http://archives.neohapsis.com/archives/dailydave/2004-q3/0070.html

Édito

Qui suis-je, où vais-je, dans quelle étagère ?

Je ne sais pas vous, mais moi, je reste effaré devant l'attachement aux lettres recommandées. Certes, le destinataire doit prouver au facteur qui il est en produisant une pièce d'identité. En revanche, n'importe qui peut envoyer un recommandé au nom d'une autre personne. De plus, rien ne lie l'accusé de réception et le contenu de la lettre, même s'il est directement collé à ladite lettre. Bref, du point de vue de la sécurité, le recommandé, comme le dit si bien mon ami Brice de Nice, c'est comme le H de Hawaï, ça sert à rien.

C'est partant de ce constat que Brice (Emmanuel Bouillon de son vrai nom) m'a proposé de réaliser un dossier sur l'authentification. Je tiens donc à le remercier pour son sérieux et son implication dans les pages que vous tenez entre les mains.

En même temps, ça tombait bien puisque j'avais reçu des menaces de mort de la part de compagnes de certains auteurs si, par hasard, je venais à discuter d'une idée d'article, ou à accepter une de leurs propositions ;-)

À part ça, l'évènement des vacances, outre le retrait de Zinedine Zidane de l'équipe de France de football, est la sortie du *Service Pack 2* [1] pour Windows XP par Microsoft. D'un côté, on peut se réjouir de la volonté affichée par la firme de Redmond de combattre ses failles de sécurité. Mais d'un autre, quand on voit la liste de logiciels qui ne fonctionnent plus correctement une fois ce SP2 installé, ça fait peur ! Sans oublier les patches pour tous les programmes concernés ... Appliquer, retarder, ne pas appliquer, c'est un choix qui dépend de l'importance que vous attachez à votre sécurité.

En revanche, la sortie de ce SP2 me conforte dans l'idée que prendre en compte la sécurité le plus tôt possible (c'est-à-dire dès le début) est un bien meilleur pari sur l'avenir. Le preuve en est d'ailleurs qu'une des protections contre l'exploitation des débordements de buffer présentes dans la SP2 est déjà obsolète [2]. De l'efficacité d'une rustine ...

La question qui se pose et à laquelle je n'ai toujours pas de réponse est : comment encourager la prise en compte de la sécurité le plus tôt possible ? Quand il s'agit de développer un logiciel dans son coin ou de contribuer à l'effort d'une communauté, une volonté personnelle (ou de quelques individus) suffit pour réussir. En revanche, dès que des enjeux économiques interviennent, la "pression de la marge" a plus que tendance à faire zapper la sécurité, ou à la rejeter sur un strapontin. A priori, seul le "client" (nous les consommateurs, industriels, entreprises, états, etc.) est en mesure d'exiger des garanties au constructeur. Mais en a-t-il les moyens et les compétences ?

Si on prend le cas du SP2, je doute que Microsoft se soit lancé là-dedans pour le bien de l'Humanité. En revanche, leur image de marque commençait à pâtir trop sévèrement des Blaster et autres aberrations du même acabit. Mais se défait-on aussi facilement de ses vieux démons ?

Frédéric Raynal

Introduction juridique aux *honeypots* : du droit à mettre du miel dans le pot !

Résumé : Les systèmes pot de miel, qui continuent de se développer dans le domaine de la recherche, connaissent un regain d'intérêt auprès des responsables de systèmes d'information qui souhaitent optimiser les outils en place (IDS par exemple) ou justifier une augmentation des budgets alloués à la sécurité auprès des directions générales. Certains freins demeurent néanmoins, et en particulier des incertitudes liées aux risques juridiques posés par ces dispositifs.

A cet égard, une question anime en particulier les forums de discussion consacrés à la « légalité des pots de miel », qui concerne l'idée suivant laquelle les honeypots constitueraient une forme condamnable de « provocation au crime » ; de plus, par leur conception même, on veut également prétendre que la poursuite des attaquants serait empêchée soit au titre du consentement de la victime, soit sur le fondement d'une négligence coupable dans la sécurité du système d'information sur lequel ils sont installés.

Ainsi, une clarification du droit applicable apparaît nécessaire pour permettre aux responsables de honeypots de mieux cerner l'environnement juridique dans lequel s'inscrivent ces dispositifs.

Janvier 2004. Les prévisions des analystes américains indiquent un pis-aller en matière de délinquance informatique et de cyber-terrorisme et les experts s'alarment de la sophistication croissante des crimes informatiques.

Pour tenter de cerner cette menace de l'intérieur et pouvoir étudier la faune de ces cyber-pirates, la communauté des experts en sécurité informatique continue de déployer un nouveau[1] concept : celui des honeypots ou, dans sa traduction française, de (systèmes) « pots de miel »…

Un concept « pot-pourri », emprunt des différentes techniques anti-intrusion [HB95] qui forment l'état de l'art actuel, et qui consiste, d'une manière générique, à mettre en place des systèmes volontairement vulnérables, c'est-à-dire conçus pour être scannés, attaqués et compromis, dans le but soit d'observer les comportements et de connaître les outils et les méthodes d'attaque des pirates (honeypot de recherche), soit de contribuer directement à la politique de sécurité d'une organisation (honeypot de production). Un concept haut en couleurs, à la pointe de la lutte contre la criminalité informatique, mais qui se révèle rapidement lui-même en proie à des doutes sur le régime juridique applicable à ces dispositifs.

En effet, tant les responsables de *honeypots* en France qu'aux Etats-Unis [LANCE, chapitre 15] semblent continuer de s'interroger sur la légalité de ces systèmes, notamment au regard de la « provocation au crime » que pourrait représenter, selon certains, la conception volontairement vulnérable des pots de miel (1), et dont le second impact serait la perte du droit à poursuivre les attaquants (2).

L'ambition simple de cet article consistera donc à donner l'éclairage nécessaire à la correcte appréhension des aspects juridiques afférents à ces deux principaux questionnements, ainsi qu'à celui de la maîtrise des risques de « dommages collatéraux » (3).

1 *Honeypots*, le « piège à pirates » ?

Le terme de *honeypot* ou, en français, de système « pot de miel » recèle une suggestivité très forte, véhiculant l'idée que ces systèmes consistent essentiellement à attirer et piéger les pirates informatiques par la ruse. Or, une telle définition suggère très vite que l'on se situe sur un terrain très proche de la provocation aux crimes et aux délits.

Cette association d'idée explique peut-être en partie le faible taux d'utilisation des *honeypots* dans les environnements de production, l'incertitude quant aux risques juridiques relatifs à cet outil de sécurité tendant ainsi à freiner les organisations dans leur adoption. Pourtant, cette image de « piège à pirates » assimilée à de la provocation s'avère rapidement une idée trompeuse.

1.1 Principe de fonctionnement

Pour s'en convaincre, il suffit de rappeler le principe de fonctionnement des *honeypots*, seule variable commune à l'ensemble des outils classés dans cette catégorie.

Les *honeypots* sont des systèmes de sécurité qui n'ont aucune valeur de production. Dès lors, aucun utilisateur ni aucune autre ressource ne devrait en principe avoir à communiquer avec lui. L'activité ou le trafic attendu sur le *honeypot* étant nul à l'origine, on en déduit *a contrario* que toute activité enregistrée par cette ressource est suspecte par nature.

Ainsi, tout trafic, tout flux de données envoyé à un *honeypot* est probablement un test, un scan ou une attaque. Tout trafic initié par un *honeypot* doit être interprété comme une probable compromission du système et signifie que l'attaquant est en train d'effectuer des connexions par rebond.

Généralement, un *honeypot* se comporte telle une boîte noire, enregistrant passivement toute l'activité et tout le trafic qui passe

[1] En réalité, le concept de honeypot n'est pas si nouveau que cela… Petit rappel historique : Cliff Stoll rapporte dans son livre [CUCKOO88] les premiers balbutiements du concept de honeypot, lorsque dans le cadre d'une investigation qu'il menait à l'Université de Berkeley (en 1986), celui-ci a été amené à alimenter un pirate en fausses informations, de façon à garder l'intrus en ligne suffisamment longtemps pour réussir à le localiser et finalement le faire appréhender par les forces de police. Ainsi l'idée de pot de miel ou honeypot naît, dans sa première appréhension, dès le milieu des années 1980 ; puis le concept devient plus sophistiqué, faisant l'objet de véritables études et donnant lieu au développement d'outils spécifiques et également de projets complexes parmi lesquels le Honeynet Project, dirigé notamment par Lance Spitzner, le maître à penser du domaine. [MISC 8]

Marie Barel
marie.barel@legalis.net
Juriste, spécialiste en droit des technologies de l'information et de la communication et sécurité de l'information.

Le présent article a fait l'objet d'une communication à la conférence SSTIC'04.

© 2004 – Tous droits réservés. La reproduction et la représentation à des fins d'enseignement et de recherche sont autorisées sous réserve que soit clairement indiqués le nom de l'auteur et la source. Pour toute autre utilisation, contactez l'auteur à l'adresse de courrier électronique ci-contre.

par lui, sur la base du principe de fonctionnement précédent. L'ensemble de ces critères va se révéler déterminant dans la définition de l'impact juridique en matière de *honeypots*.

1.2. De la provocation aux crimes et aux délits ?

Partant de ce principe de fonctionnement, on peut en effet s'interroger sur la réalité de la provocation suggérée et l'applicabilité au responsable d'un système *honeypot* du chef de complicité prévu sous l'incrimination de « provocation au crime ou au délit », tel que défini à l'article 23 de la loi du 29 juillet 1881 sur la liberté de la presse[2] :

« Seront punis comme complices d'une action qualifiée crime ou délit ceux qui, soit par des discours, cris ou menaces proférés dans des lieux ou réunions publics, soit par des écrits, imprimés, dessins, gravures, peintures, emblèmes, images ou tout autre support de l'écrit, de la parole ou de l'image vendus ou distribués, mis en vente ou exposés dans des lieux ou réunions publics, soit par des placards ou des affiches exposées au regard du public, soit par tout moyen de communication audiovisuelle, auront directement provoqué l'auteur ou les auteurs à commettre ladite action, si la provocation a été suivie d'effet.

Cette disposition sera également applicable lorsque la provocation n'aura été suivie que d'une tentative de crime prévue par (l'article 121-5 du code pénal). »

Au-delà de la question du support utilisé, c'est la constatation de l'élément constitutif principal de l'infraction visée, à savoir la publicité, qui suscite le plus d'interrogations.

En effet, à la lecture de sa définition dans le Petit Larousse, la publicité implique bien une pro-activité et tend à faire connaître au public un produit ou service, en l'incitant à acheter ou utiliser ce produit ou service par le moyen d'annonces, encarts, films publicitaires, etc. En l'occurrence, en quoi la construction des *honeypots*, « *systèmes conçus pour être scannés, attaqués et compromis* » [LANCE], constitue-t-elle une provocation *directe* aux pirates ? Ceux-ci font-ils l'objet d'une publicité à destination de la communauté *underground* ? Les responsables de systèmes *honeypots* mettent-ils cette communauté au défi à travers les forums de discussion ou les canaux IRC ? Ou encore, ont-ils fait des déclarations sur le Web susceptibles de déclencher les hostilités ?...

Ainsi, en l'absence de démarche particulière à destination de la communauté cible, la seule visibilité/détectabilité du honeypot sur le réseau en tant que système vulnérable ne doit pas permettre de qualifier l'acte de publicité nécessaire à la présente incrimination de provocation au crime ou au délit.

Bien au contraire, on peut même affirmer que la publicité serait en réalité tout à fait antinomique avec les *honeypots*, le succès de ces systèmes étant conditionné en premier lieu par leur furtivité. Ainsi, pour pouvoir remplir leur principal objectif (capter et enregistrer un maximum de traces d'activités malveillantes), les systèmes pots de miel doivent impérativement se développer dans l'ombre.

De plus, force est de souligner que, dans la majorité des scénarios d'attaque, la résolution criminelle de l'attaquant elle-même est indépendante de la vulnérabilité apparente (détectable) du système. Aussi, la performance d'un *honeypot* n'est tout simplement nullement conditionnée par une quelconque publicité concernant sa vulnérabilité.

En effet, la majorité des pirates sont aujourd'hui animés par le « easy kill » et une « shotgun approach » ; dès lors, les attaquants perdent rarement de temps à analyser les systèmes qu'ils visent, leur but étant de toucher le maximum de machines ou de voir simplement à quoi ils accèdent avant de recommencer. Pire encore, la plupart des attaques ne sont pas exécutées en direct par les pirates, mais bien de façon programmée par des outils automatisant les attaques (comme les vers par exemple). Seule exception notable, les attaquants aux « cibles choisies » par opposition aux « attaques par cible d'opportunité » susvisées, et qui concernent les rares pirates de haut vol, plus impliqués dans des actions proches de l'espionnage et du contre-espionnage industriel ou du sabotage[3]... Cas extrêmes s'il en est, qui ne font pas partie de notre champ d'étude dans le cadre du présent article.

En définitive, il est très difficile de voir comment les *honeypots* répondent à l'exigence de publicité requise dans l'incrimination de « provocation aux crimes et aux délits » ni comment la conception volontairement vulnérable des ressources *honeypot* peut constituer en elle-même une incitation déterminante dans l'intention criminelle des attaquants.

Pour d'autres, c'est la nécessité d'une provocation « *suivie d'effet* » (ou à tout le moins d'une tentative) qui fait « structurellement défaut » et conduit à rejeter la provocation aux crimes et délits :

[2] Nous n'évoquerons pas ici la question de la provocation policière, en principe interdite en droit pénal français sauf certains domaines spécifiques (stupéfiants notamment), mais très usitée dans d'autres espaces juridictionnels (notamment aux USA).

[3] Dans ce cadre exceptionnel, les *honeypots* pourraient servir efficacement des objectifs de déception et de dissuasion. Ainsi, s'agissant d'organisations titulaires d'informations à forte valeur ajoutée, dans des domaines de recherche sensible (tel que le nucléaire) et/ou les ressources sont classifiées ou soumises à contrôle, l'attaquant qui construit son approche à partir d'un objectif pré-déterminé pourra être impacté par ces techniques. Par exemple, on pourra construire un *honeypot* de façon à tromper et divertir l'attaquant, tout en prévenant des attaques contre les données de production réelles ; dans ce cas de figure, on pourrait créer un serveur de fichiers jouant le rôle de registre central pour des documents classifiés « secret défense », mais au lieu d'y placer de la documentation valide, ce sont des fausses informations (*fake data*) qui seraient créées et déposées sur le registre du *honeypot*. Ainsi, l'attaquant pensera par exemple avoir obtenu les plans d'un cœur de réacteur dernière génération, alors qu'il met en œuvre les procédés qu'il a volés, il n'obtiendra aucun résultat utile.

> *« il ne peut y avoir d'atteinte à un système de traitement automatisé de données au sens pénal, si le maître du système est d'accord pour qu'une telle atteinte soit réalisée. Ce qui semble bien le cas lorsqu'il met à la disposition du public un système destiné à faire l'objet d'une intrusion. On ne serait ici donc que dans un cas d'application classique de la théorie des faits justificatifs chère au droit pénal : le consentement de la victime a, en l'espèce, la libre disposition de l'intérêt protégé par la loi pénale et s'érige comme une condition de réalisation de l'infraction*[4]. »

Il est important, en effet, en matière de *honeypot*, d'envisager cette conception suivant laquelle la mise en place de systèmes volontairement vulnérables constituerait à la fois une forme de consentement implicite de la victime (en l'occurrence le responsable du système) et une négligence coupable, faisant l'un et l'autre obstacle à la possibilité même de poursuivre les attaquants pour atteinte au système.

2 *Honeypots* et perte du droit à poursuivre les attaquants

Suivant nos développements précédents, nous avons déjà souligné comment, conformément à leur principe commun de fonctionnement, les *honeypots* permettent simplement, grâce à un « effet microscope »[5] et une réduction drastique du niveau de bruit, de mieux capter, sans provocation, une activité non autorisée et illicite. Ce faisant, la vocation de ces ressources à être scannées, attaquées ou compromises ne permet pas pour autant de présumer systématiquement d'une forme de consentement implicite du responsable de *honeypot* et il nous faut relativiser cette première affirmation (2.1).

Ensuite, certains voudraient prétendre que la conception volontairement vulnérable des systèmes pot de miel pourrait emporter perte du droit à poursuivre les attaquants car elle constituerait une négligence coupable au regard de l'exigence de protection des systèmes. Pourtant, nous verrons qu'il convient là encore de relativiser certaines décisions récentes pour réaffirmer le principe suivant lequel la mise en place (voire même l'existence) de dispositifs de sécurité n'est pas une condition des incriminations de fraude informatique (2.2).

2.1 Observer ou poursuivre, faut-il choisir ? : à propos du consentement de la victime

L'admission du fait justificatif de consentement de la victime peut facilement apparaître comme une évidence. Cependant, la simplicité du raisonnement conduit selon nous à ignorer les réalités techniques propres à la distinction entre *honeypots* de recherche et *honeypots* de production. (Paul Valéry n'écrivait-il pas : « ce qui est simple est faux » !).

Cette distinction est celle proposée par Marty Roesch[6] face à la diversité de formes et d'outils et l'absence de concept unitaire pour définir les *honeypots*. Selon cette « classification », les *honeypots* de recherche visent uniquement à la connaissance[7], tandis que les *honeypots* de production sont conçus moins pour apprendre que pour protéger une organisation spécifique et apporter une valeur ajoutée à sa politique de sécurité :

- soit dans une optique de prévention, par exemple en faisant croire à un pirate qu'il a réussi à accéder aux ressources système alors qu'il a été dirigé dans un environnement préparé et contrôlé[8] - Cf [Hb95] : techniques de déflection, de déception et de dissuasion ;
- soit dans une optique de détection d'intrusion, par exemple pour augmenter les performances des NIDS (*Network Intrusion Detection System*) notamment dans la recherche de nouvelles attaques ou vulnérabilités[9] ;
- soit dans une optique de réponse aux incidents en organisant par exemple la pré-constitution des preuves de l'intrusion[10].

Dès lors, parce que les *honeypots* de production ne remplacent pas les autres mécanismes de sécurité (*firewall* et IDS par exemple) et s'inscrivent à part entière dans la politique globale de sécurité, conclure d'emblée au consentement de la victime serait selon nous antinomique avec les objectifs possiblement assignés à ces ressources (notamment en fonction de leur place dans l'architecture du réseau d'entreprise), objectifs où la volonté est moins ici d'être attaqué (sondé ou compromis) pour observer le mieux et le plus, que de seulement *faciliter la captation pour mieux protéger*[11].

[4] Voir notamment Thiébaut Devergranne : *Du droit dans le pot, quelques réflexions juridiques autour des « honeypots »* - MISC 8, pp.34-35 (consultable sur : **http://hstd.net/honeypots.pdf**)

[5] Seule l'activité dirigée contre les *honeypots* est enregistrée, à l'exception de toute intrusion sur d'autres ressources du système de production, ce qui constitue dans le même temps l'un des principaux avantages (meilleure gestion des faux positifs et faux négatifs) et inconvénients (champ de vision réduit) de ces ressources.

[6] Le développeur du logiciel SNORT.

[7] … considérant, suivant le conseil du chinois Sun Zu dans son ouvrage « *L'art de la guerre* », daté du IVème siècle avant J.-C , qu'il faut s'efforcer « *de vaincre par la ruse, sans livrer combat. Les grands stratèges remportent le succès en découvrant le jeu caché de leurs adversaires, en déjouant leurs plans […]* ». Extrait de l'article de Fabrice Deblock, « *on n'attrape pas les pirates avec du vinaigre, mais avec du miel !* » - JDNet, 26 novembre 2003 : **http://solutions.journaldunet.com/0311/031126_honeypots.shtml**

[8] Par exemple le système *Bait and Switch* qui joue le rôle de passerelle entre l'extérieur et le réseau interne à protéger, permet, après « isolement » du trafic suspicieux, de demander la redirection de ce trafic vers un miroir « pot de miel », qui simule l'environnement de production.

[9] Par exemple, les *honeypots* sont peu sensibles aux faux négatifs car ils ne fonctionnent pas, à la différence des NIDS, à partir d'une base de données de signatures, mais sur la simple observation du trafic et de l'activité enregistrée. Dès lors, un *honeypot* pourra détecter un exploit lancé avec ADMmutate, alors que le NIDS en place sera leurré…

[10] Cf [MISC 11] : au sujet des *markers* (en français, « fichiers trace » ; on parle aussi de « fichiers témoins ») dans la dernière version du logiciel *Specter*.

[11] Il semble bien que cette opinion soit partagée par quelques-uns. En effet, si l'on admettait qu'il a fait justificatif de la victime et qu'en tout état de cause, les intrusions sur les *honeypots* ne peuvent être poursuivies judiciairement, quelle serait l'utilité de développer des fonctionnalités comme les *markers* offerts dans le logiciel SPECTER (version 7) – Cf *infra* section 3 – et dont l'objectif est d'aménager des preuves de l'intrusion ?

A l'inverse, il nous semble effectivement que les *honeypots de recherche*, qui ne participent qu'indirectement à la sécurité[12], impliquent bien le consentement du responsable de projet : en effet, la valeur ajoutée de ce type de *honeypot* augmentant à l'aune du nombre de scans, d'attaques ou de compromissions dont ils sont l'objet, on peut raisonnablement penser qu'ils les appellera de ses propres vœux.

2.2 Des systèmes volontairement vulnérables, partant non éligibles à la protection des articles 323-1 et suivants du code pénal ?

Une autre idée répandue est de considérer comme une négligence coupable le fait de, volontairement, laisser des vulnérabilités et « autoriser » ainsi les attaquants à s'introduire sur le système. Dès lors, la poursuite de l'attaquant sous le chef d'accès ou de maintien frauduleux (article 323-1 du Code pénal) deviendrait impossible.

En ce sens, force est de rappeler que le Sénat lui-même, au cours des travaux préparatoires relatifs à la loi dite Godfrain, avait souligné qu'une exigence de protection du système pour que l'infraction d'accès frauduleux à un système informatique soit constituée lui paraissait raisonnable, « *le droit pénal ne devant pas compenser l'insuffisance ou la défaillance des mesures de sécurité* »... Cependant, l'Assemblée nationale[13] a jugé excessive cette position du Sénat, et la jurisprudence ou les textes ultérieurs ont pris la même position : la protection du système n'est pas une condition de l'incrimination.

Par exemple, rappelons que la Cour d'appel de Paris dans un arrêt du 5 avril 1994 a posé le principe que : « pour être punissable, cet accès ou ce maintien doit être fait sans droit et en pleine connaissance de cause, étant précisé à cet égard qu'il n'est pas nécessaire pour que l'infraction existe, que l'accès soit limité par un dispositif de protection [...] ».Ainsi l'accès tombe sous le coup de l'article 323-1 du Code pénal dès lors qu'il est le fait d'une personne qui n'a pas le droit d'accéder au système et « la barrière technique n'est pas indispensable au jeu de l'interdiction » [VS03].

Par analogie, Christian Le Stanc considère de la même façon que : « il n'est pas licite de pénétrer chez autrui sans autorisation et que, notamment, l'infraction de violation de domicile peut être constituée sans qu'il faille avoir égard à la hauteur du mur d'enceinte ou à la résistance de la serrure ». Quid en l'absence même de toute protection !? Le fait de ne pas fermer sa porte à clé, voire même de laisser la porte ouverte (idée plus proche encore du concept de honeypot), constitue-t-il une négligence coupable de la victime l'empêchant d'engager des poursuites contre un intrus ?

La Commission européenne, dans une proposition de décision-cadre relative aux attaques visant des systèmes d'information[14], nous donne une réponse à cet égard :

> *« la Commission ne souhaite nullement mettre en cause l'importance qu'elle attache à l'utilisation de mesures techniques efficaces pour protéger les systèmes d'information. Le fait est néanmoins qu'une grande partie des utilisateurs s'exposent malheureusement à des attaques faute d'une protection technique adéquate (voire même de toute protection). En vue de prévenir les attaques contre ces utilisateurs, le droit pénal doit couvrir l'accès non autorisé à leurs systèmes, même si ces systèmes ne bénéficient pas d'une protection technique appropriée. C'est pour cela (...) qu'il n'est pas nécessaire que des mesures de sécurité aient dû être déjouées ».*

Cependant, certains voient dans la retentissante affaire Kitetoa/Tati[15] un revirement récent de la jurisprudence française à cet égard. Qu'en est-il exactement ? Le tribunal correctionnel de Paris dans sa décision du 13 décembre 2002[16] avait condamné l'animateur du site Kitetoa.com pour accès frauduleux à des données qui n'étaient pas du tout sécurisées (absence de mot de passe ou de restriction d'accès).

Finalement, c'est seulement suite à un appel introduit par le Parquet général que le webmestre a été relaxé par décision du 30 octobre 2002 de la Cour d'appel de Paris[17], décision fondée sur l'accès « par des moyens informatiques réguliers » (accès par un simple navigateur).

Cette dernière décision, qui fait suite à une démarche peu courante du Parquet, demeure selon nous un cas d'espèce aux circonstances particulières ; en l'occurrence, les données accédées étant des données personnelles, elles étaient soumises à une obligation de sécurité[18], ce qui justifie que, dès le jugement de première instance, la constitution de partie civile de la société Tati ait été rejetée par le tribunal au motif que « *celle-ci ne saurait se prévaloir de ses propres carences et négligences pour arguer d'un prétendu préjudice* ».

En définitive, il convient de souligner que, si le niveau de protection du système n'est pas une condition de l'incrimination d'accès ou de maintien frauduleux, l'insuffisance des moyens de sécurité mis en œuvre fait néanmoins courir au responsable du système des risques connexes non négligeables.

[12] Par exemple le projet Honeynet avait permis en janvier 2002 de découvrir l'exploit *dtscp*. Pour plus d'informations : **http://www.honeynet.org**

[13] Jérôme Dupré, *Pour un droit de la sécurité économique de l'entreprise* - Thèse 2000, Université de Nice-Sophia Antipolis, n°346

[14] Com/2002/0173 (Final) - JOCE du 27 août 2002

[15] Rappelons en quelques mots les faits de l'espèce : en 1999, l'animateur du site Kitetoa.com signale à l'hébergeur du site des magasins Tati une faille de sécurité permettant d'accéder au contenu des bases de données clients du serveur grâce à un simple navigateur. Constatant près d'un an après que les failles détectées et signalées existaient toujours, il décide de publier sur son site un article relatant cette faille de sécurité. L'information était ensuite reprise dans un magazine spécialisé, ce qui détermina la société Tati à poursuivre l'animateur du site pour accès frauduleux dans un système informatisé.

[16] *Revue Communication Commerce électronique*, mai 2002, p.31, note Grynbaum

[17] *Revue Communication Commerce électronique*, janvier 2003, p.30, note Grynbaum

[18] Obligation issue de l'article 29 de la loi « Informatique et Libertés » (loi du 6 janvier 1978) et sanctionnée par l'article 226-17 du Code pénal : « *le fait de procéder ou de faire procéder à un traitement automatisé de données nominatives sans prendre toutes les précautions utiles pour préserver la sécurité de ces informations et notamment empêcher qu'elles ne soient déformées, endommagées ou communiquées à des tiers non autorisés est puni de cinq ans d'emprisonnement et 300.000 euros d'amende.* »

Il s'agit en particulier des :

- recours de la tierce victime (victime collatérale par rebond –voir paragraphe 3 ci-après-, « personne concernée » au sens de la Directive 95/46, sur le fondement de l'obligation de sécurité précitée) ;
- risques en matière d'assurance : perte du droit à indemnisation (Cf paragraphe 3.1).

Ces enjeux et ces risques devront être anticipés et pris en considération dès la phase de conception par le responsable d'un projet *honeypot*, en particulier lorsqu'il définira les spécifications techniques en matière d'étanchéité des systèmes et de contrôle des données (Cf paragraphe 3.2). De la même façon, la conception d'un *honeypot* pourra être modulée en fonction des contraintes juridiques applicables en matière de protection des données personnelles (ou, dans un sens plus large encore, de « *privacy* » - Cf *infra*)…

3 Responsabilité du fait de la compromission d'un système *honeypot* : maîtriser le risque de « dommage collatéral »

Un point souvent discuté[19] en matière de *honeypot* est celui de la responsabilité encourue en cas de dommages causés à un tiers suite à la compromission du système pot de miel.

3.1 Origine et nature du risque

Le cas ici envisagé est donc celui où l'attaquant réussit à « s'échapper » du pot de miel et à l'utiliser comme rebond pour conduire de nouvelles attaques contre des systèmes étrangers ou, pourquoi pas, d'autres machines du réseau. En réalité, les niveaux de risques sont différents d'un *honeypot* à l'autre ; on oppose classiquement [LANCE] les *honeypots* dits « à forte interaction » aux *honeypots* « à faible interaction ».

Pour ces derniers, le risque potentiel est considéré comme faible car ils ne font pour la plupart que simuler des services sur des machines virtuelles, sans offrir beaucoup de privilèges à l'attaquant. C'est d'ailleurs généralement de cette limitation dans l'interaction que l'attaquant perdra son intérêt pour le *honeypot*, se mettant en quête de nouvelles cibles.

Par opposition, les *honeypots* de recherche (type *honeynets*), qui sont conçus pour récolter un maximum d'informations, sont classés dans les systèmes dits « à forte interaction », car ils offrent de vrais services et un environnement avec lequel l'attaquant va pouvoir interagir après compromission du système, que ce soit en lançant des attaques par rebond, en se servant des machines comme zombie pour réaliser un déni de service, etc.

Le risque est alors important de voir sa responsabilité engagée par un tiers ayant subi des dommages suite à cette compromission, moins sur le plan pénal que par le biais d'une action civile en dommages-intérêts sur le fondement des articles 1382[20] et suivants du Code civil. Pour autant, il faudra au demandeur prouver à la fois la faute du responsable de *honeypot*, son préjudice et le lien de causalité entre les deux.

Sur ce point, des doutes peuvent tout à fait être émis sur l'existence d'une faute, les systèmes pot de miel n'étant pas nécessairement des machines *non sécurisées* ; bien au contraire, les *honeypots* doivent être astreints à un contrôle très scrupuleux du trafic vers l'extérieur (Cf paragraphe 3.2 ci-après)…

Enfin, il ne faudra pas négliger, à côté du risque de l'action civile en réparation par le tiers, le risque assurantiel. En effet, la mise en place d'un *honeypot*, si elle introduit des risques pour le système d'exploitation assuré, pourrait conduire à exclure les garanties de la police d'assurance en matière de risques informatiques. Il faudra donc valider préalablement les conditions dans lesquelles le *honeypot* pourra être déployé en particulier en environnement de production.

A l'occasion du rapprochement avec l'assureur, on fera notamment état des mesures de sécurité mises en œuvre pour prévenir les utilisations malveillantes après compromission.

3.2 Un risque techniquement maîtrisable : le contrôle des données

En effet, la limitation du risque d'action en responsabilité passe au premier chef par la mise en place de mesures de sécurité de nature à permettre la surveillance et la maîtrise des connexions sortantes de l'attaquant.

Ainsi, pour éviter la réalisation du risque d'attaque par rebond, plusieurs solutions techniques ont pu être adoptées :

- l'interdiction de tout trafic vers l'extérieur : à l'évidence, cette solution, si elle a le mérite de l'efficacité, pêche par manque de furtivité puisque ce comportement sera très vite détecté et considéré comme suspect, poussant ainsi l'attaquant à abandonner son attaque (d'où d'ailleurs l'échec des premiers essais du *Honeynet Project* [HONEYNET]…) ;
- la limitation de la bande passante ou du nombre de connexions sortantes (Cf *Netfilter* e.g.). Ainsi, dans l'architecture dite de première génération (GenI) du *Honeynet Project*, les communications sortantes sont limitées dans le temps (typiquement entre 5 et 10 par heure) ;
- l'analyse des paquets sortants et leur modification à la volée pour les rendre inoffensifs (Cf architecture de seconde génération - GenII, Snort_inline).

Dans tous les cas, ne rien contrôler serait une politique éminemment dangereuse et aussi la pire des défenses en cas d'action judiciaire ; le responsable de *honeypot*, en bon gestionnaire de risques, appréciera donc les mesures de sécurité appropriées en fonction du niveau d'interactivité du système.

[19] Cf les messages postés dans certaines listes de discussion : « Use a honeypot, Go to Prison ? » > **http://www.securityfocus.com/news/4004** ; « les pots de miel sont-ils légaux ?" > **http://linuxfr.org/2003/06/16/12887.html**

[20] « Tout fait de l'homme, qui cause à autrui un dommage, oblige celui par la faute duquel il est arrivé, à le réparer ».

Conclusion

Il ressort de nos développements qu'il n'existe pas véritablement d'obstacle juridique, sinon de simples *a priori*, susceptibles d'expliquer le faible taux d'installation de systèmes *honeypots* sur les réseaux en production. Le présent article en a fait la démonstration et invite dès lors à un intérêt renouvelé pour ces techniques à la pointe de la lutte contre la cybercriminalité. Pour autant, le déploiement de ces systèmes ne devra pas prospérer sans une conduite de projet très détaillée et une approche de gestion de risques destinés à appréhender les points sensibles de la construction des systèmes pot de miel.

Par exemple, une question fondamentale sera de savoir quels sont les objectifs du *honeypot* : observer, défendre... et poursuivre ? Ces objectifs devront être déterminés dès l'origine du projet et documentés dans un document de conception générale et détaillée, incluant une analyse de risques, un plan de sécurité et tous documents permettant de retracer l'historique du projet et les étapes de sécurisation (techniques, organisationnelles et juridiques) qui ont été menées. Bien sûr, la frontière entre *honeypot* de recherche (dont on a dit qu'ils étaient plus fortement exposés au risque en matière de responsabilité) et les *honeypots* de production (qui nécessitent des précautions sur le plan juridique, en particulier s'agissant de la problématique des « attaquants internes »[21]) est à géométrie variable, ce qui implique, encore une fois, une casuistique et une gestion de projet adaptée à ses spécifications techniques... *In fine*, le responsable de projet devra définir son niveau de risque acceptable, en fonction notamment de la valeur ajoutée attendue[22], de la nature et de l'étendue des dommages potentiels et de l'« entropie » des scénarios de risque juridique.

Références

[CUCK'88] Cliff Stoll : *The Cuckoo's egg : tracking a spy through the Maze of Computer Espionnage* (1988) – ISBN 074341 1463

[HONEYNET] The Honeynet Project – **http://www.honeynet.org**

[LANCE] Lance Spitzner – *Honeypots : tracking hackers* (2002) – ISBN 0321108957; **http://www.tracking-hackers.com/book/**

[HB95] L. Halme and R. Bauer, *AINT misbehaving : a taxonomy of anti-intrusion techniques* (1995) – **http://www.sans.org/resources/idfaq/aint.php**

[MISC 8] *Honeypots, le piège à pirates !* – Dossier spécial, MISC Juillet-Août 2003, pp.24-61

[MISC 11] *Specter, un honeypot qui compromet les pirates : techniques et légalité*, Elisabeth Stella et Thierry Martineau – article paru dans MISC, Janvier-Février 2004, pp.6-9

[VS03] Valérie Sédallian, *Légiférer sur la sécurité informatique : la quadrature du cercle* - 2003, Juriscom.net

[21] Sur les limites de la cybersurveillance des salariés sur leur lieu de travail, consulter le site de la CNIL (Commission Nationale Informatique et Libertés) : **http://www.cnil.fr**

[22] « En matière de *honeypot* de production, on se rendra vite compte par exemple qu'il est fastidieux de dépenser du temps pour regarder des traces de sécurité sur un système non critique, à moins d'avoir déjà terminé d'analyser les traces des vrais systèmes de production à protéger (le *honeypot* n'étant pas, en général, la priorité sur un réseau d'entreprise. » Cf [MISC 8], au sujet du démon *Honeyd* développé par Niels Provos – pp.43 à 53.

Le facteur humain : vecteur ou maillon... à la force surprenante

Organiser la sécurité n'a de sens qu'en tenant compte de la composante humaine : les personnes qui mettent en œuvre, configurent et maintiennent les dispositifs techniques, ou celles qui les utilisent, cette dernière catégorie, déviante, pouvant aussi se révéler malveillante...

Psychologie, comportements, règles. Ce sont, quasi caricaturalement, les trois pivots très classiques qui déterminent la force (ou la faiblesse) d'un système d'information, indépendamment de toutes les précautions techniques qui auront été prises à plus ou moins grands frais. Quoi de plus normal : la nature humaine est aussi généreuse qu'elle peut se montrer déviante et redoutable d'inventivité perverse. Voilà une caractéristique exploitée sans difficulté lorsque l'humain est l'oublié de la politique de sécurité et qu'il n'a donc pas été renforcé par un plan de sécurisation approprié.

Plutôt que de céder à la facilité de toujours incriminer la faiblesse de notre propre nature, essayons d'y croire un peu : roseaux nous sommes certes, frêles et délicats, mais capables de plier sans rompre, et en pensant qui plus est ! Plutôt que de montrer que l'homme (ou la femme, pas de sexisme) n'est qu'un « maillon faible », nous allons au contraire nous intéresser aux dispositifs de protection *humains* correspondant à des attaques *humaines* sur des composants des systèmes d'information (SI). Bien avant d'être *technique*, la cible ou le vecteur est informationnel.

Comment exploiter le facteur humain pour agir sur le système d'information et sur sa partie logique ? Quatre exemples illustrent quelques aspects. Pour chacun, nous cherchons à comprendre la défaillance exploitée et les contre-mesures appliquées. Vous pourrez trouver ces exemples drôles, étranges ou aberrants. Ils sont cependant tous tirés de cas réels, vécus ou rencontrés, et peuvent très bien se produire dans votre propre environnement.

1. L'ingénierie sociale : entre la discipline des DRH et celle de la Sûreté

L'*ingénierie sociale* est la discipline qui travaille sur l'humain, son fonctionnement et ses règles. Celles-ci peuvent être « internes » à la personne (capacité et niveau de raisonnement, texture émotionnelle, mode de réflexion, etc.), comportementales et relationnelles pour rendre possible et faciliter la vie en société.

Cette *ingénierie* est, en temps normal, le domaine d'action des « responsables des ressources humaines ». Pour notre part, professionnels et responsables de la sécurité, nous regardons l'exploitation et le détournement de ces règles afin de disposer de contre-mesures adaptées. Cependant, nous ne rentrerons pas, dans le cadre de cet article, dans des détails de psychologie, de dynamique de groupe, et encore moins dans les dimensions sociétales, nous limitant aux interactions avec le SI. Aussi, le facteur humain est vu dans une perspective systémique comme un composant interactif au même titre que n'importe quel autre constituant du SI.

La sûreté est une discipline historiquement pratiquée dans les environnements principalement industriels ou de Défense. Que ce soit sur les installations ou les produits, cette science de l'ingénieur vise à s'assurer du maintien d'un système dans une plage donnée de risques identifiés et mesurés (qualifiés et quantifiés). Certains réagiront en soulignant qu'en sécurité c'est pareil ! Ce n'est pas tout à fait vrai : la différence entre sûreté et sécurité se situe sur l'origine du risque généré. Sans trop entrer dans les détails, pour un système donné, un risque d'origine *interne* (ex : une défaillance) est un problème de *sûreté*, un risque d'origine *externe* (ex : une agression) est un problème de *sécurité*. Une vulnérabilité du système est un problème de sûreté (origine interne au système) qui peut devenir un problème de sécurité si une menace la vise. Tout ceci est applicable en particulier au périmètre des systèmes d'information.

La composante humaine est considérée assez systématiquement dans les différentes approches méthodologiques, tant de sûreté (thématiques « erreurs » par exemple) que de sécurité (en particulier, les « malveillances »). La façon d'appréhender le facteur humain varie selon qu'il soit intégré comme composant du système ou non.

D'où toute l'importance de correctement définir, d'une part la notion de « *système d'information* », d'autre part le périmètre qui y correspond. En effet, l'humain peut être considéré comme cible, vecteur (principe de « *stepping stones* ») ou menace selon les cas. Un point important est l'interaction entre humains et machines pour agir sur un processus informationnel ou en exfiltrer des renseignements.

L'ingénierie sociale, placée non plus dans les champs de compétences des DRH, mais dans ceux de la sûreté et de la sécurité, est une discipline à double usage : défensive comme offensive.

Diverses possibilités sont à exploiter pour maintenir, voire renforcer la sûreté du SI en exploitant les techniques de l'ingénierie sociale : psychologie des intervenants, sensibilité à la détection des menaces, conscience des actions malveillantes et des dangers, efficience des dispositifs organisationnels et de leurs protections, etc. Un travail rapproché entre RSSI (ou Sûreté) et DRH, apparaît donc important. Nous n'approfondirons pas ces sujets dans cet article.

Une « anomalie de fonctionnement » humaine, tout comme une défaillance organisationnelle, sont plus que possibles : elles sont, pour ainsi dire, monnaie courante. N'étant pas systématiquement exploitées, ces anomalies ne constituent pas pour autant des incidents de sécurité.

Yannick Fourastier
yfourastier@miscmag.com

Cependant, l'observation dans le temps des pratiques dans une entreprise permet de déterminer un potentiel de risques dans une démarche rationnelle de management de la sécurité.

Cette même observation permet à une entité organisée et motivée d'élaborer un scénario d'intelligence, de manipulation ou d'attaque quasiment indétectable ou qui le sera peut-être bien tard. Le nombre de sociétés spécialisées en « intelligence économique » a crû de façon très importante ces dernières années. Un marché existe donc pour ces « observateurs de sources d'informations ouvertes ». Si la menace est réelle, l'importance des risques est, quant à elle, souvent plus difficile à démontrer : elle dépend et varie beaucoup selon les contextes et les spécificités de chacun.

Exemple 1

L'intérimaire qui en savait trop : comment la précarisation rapporte à la concurrence

▷ Les enjeux

Le produit *Phare* d'un grand industriel agroalimentaire français *A*, constituait une part importante de son chiffre d'affaires. *Phare*, issu d'un savoir-faire particulier, jouissait d'une image de marque bien entretenue. Un industriel *B*, d'origine française également, n'avait pas dans son catalogue de produit équivalent à *Phare*. Aussi, si *B* ne concurrençait pas directement *A* sur ce produit, il avait néanmoins valorisé la part de marché qu'il pouvait capter en proposant un produit alternatif. Plutôt que d'investir en analyse marketing (étude de marché et attentes consommateurs), recherche et développement, il l'a contrefait en « s'inspirant » des procédés qu'il a récupérés. Ses produits *Lampion* ont ainsi été commercialisés, notamment sur le marché français, en exploitant un packaging, un visuel et une appellation dans l'esprit très proches de *Phare*. Bref, différentes caractéristiques permettant un détournement en règle de l'image de marque de *Phare* ont été utilisées. Ce n'est jamais en définitive qu'une banale histoire de contrefaçon.

Vous avez ainsi pu trouver *Phare* et *Lampion*, tous deux produits d'origine purement européenne, dans les rayons de vos grandes surfaces préférées...

▷▷ Le mode opératoire

Les caractéristiques marketing de *Phare* étaient par nature, publiques. D'accès aisé, il suffit d'avoir le produit entre les mains pour disposer des informations. La réelle contrefaçon n'est pas là, bien que ce détournement (à défaut de protection adéquate sur la propriété intellectuelle) puisse également être répréhensible.

La reconstitution d'un produit aux caractéristiques très proches nécessite en revanche la mise en œuvre de procédures (séquence d'opérations industrielles) quasi identiques. En effet, en matière d'industrie, sauf utilisation de procédés fondamentalement révolutionnaires, le degré de fidélité de la reproduction n'est pas vraiment le fruit du hasard. Pour que *Lampion* ressemble autant à *Phare*, une procédure suffisamment voisine, aux variables renseignées avec les bonnes valeurs (directives de production : origine et qualité des matières premières, quantités, niveaux d'affinages, temps de préparation, etc.) doit être amorcée.

Des travaux de rétro-ingénierie sur le produit fini auraient permis de parvenir à un résultat proche. D'un autre côté, une bonne exploitation de renseignements précis permet d'atteindre aussi cet objectif de manière certainement moins ardue. Après tout, la fabrication d'une unité *Phare* suit strictement toujours la même séquence d'opération (reproductibilité industrielle) : modéliser la procédure finement n'est pas si compliqué pour une personne qui en a le temps, et qui est présente sur la chaîne de production. De la même façon, connaître les « variables » de cette procédure, leurs « valeurs » et les conséquences de leurs variations sur le produit fini, est possible en s'intéressant aux contenus des contrôles qualité.

▷▷▷ L'exploitation

Pour réguler la production et certaines caractéristiques de saisonnalité, la politique de gestion des ressources humaines de *A* consiste à lisser les besoins de main-d'œuvre avec des intérimaires. Il est ainsi facile aussi de remplacer le personnel défaillant ou de rapidement « reconfigurer son parc de compétences » (en français dans le texte de la communication interne de cet industriel).

Classique ... D'un autre côté, ce mode de gestion, en soi rationnel vu du petit côté de la lorgnette, a une quantité d'effets pervers non négligeables sur le plan du management et financier (comme c'est ici le cas au final : perte d'exploitation, de parts de marché, etc.), et réel cauchemar en particulier pour un RSSI.

Dans le cas présent, les possibilités d'introduction de personnel étranger via le recours intensif à des contrats précaires (donc de la façon la plus légale qui soit !) sont grandement démultipliées, avec en prime une certaine facilité pour l'adversaire. Le personnel introduit se fond facilement dans le paysage, renseigne depuis l'intérieur, et agit en complicité interne. Peu importe le poste auquel ce « trojan » est affecté : tant qu'il est libre de ses mouvements, de discuter (ce qui constitue déjà un accès à une partie du SI) ou d'accéder aux systèmes informatiques.

Voilà comment s'est introduit le voleur « d'immatériel » chez *A*.

Les informations recherchées par ce curieux intérimaire sont concentrées au service qualité : manuel des procédures, description des procédés, fiches de contrôles qualité, etc. Tout cela sous format papier et bien sûr en version électronique, stocké sur un serveur bureautique (partages Windows). Cet intérimaire avait un peu l'embarras du choix pour satisfaire son commanditaire : la version papier, l'accès « légitime » au serveur, la récupération de bandes de sauvegardes...

Le plus simple a encore été de récupérer les bandes de sauvegardes : l'intérimaire s'est rapproché des exploitants informatiques qui avaient l'habitude de déjeuner à la cantine locale, et de conversations en plaisanteries, il s'est retrouvé à faire partie de ce groupe.

Entré d'abord dans « le cercle de confiance », il a au bout de quelques semaines fini par raccompagner ses nouveaux copains jusqu'en salle informatique, le plus naturellement du monde qui soit : une conversation qui dure et chemin faisant, les voilà déjà arrivés.

L'habitude s'installe, jusqu'au jour où ledit intérimaire se retrouve à bien connaître l'organisation de la salle des serveurs, les procédures, et le lieu de rangement des bandes. D'un naturel avenant, sympathique, et tout ce qu'il y a de plus commun, cet intérimaire est de la même façon capable de raconter tout ce qui se passe à la Qualité. Cependant, ressortir avec quelques bandes DAT dans la poche après un bête échange est beaucoup plus facile que de sortir plusieurs classeurs volumineux. Les bandes n'étant pas contrôlées avant d'être stockées ou ressorties, la manœuvre est vraiment passée inaperçue.

La phase de préparation de cette opération d'extrusion d'informations aura probablement duré un peu plus d'un an, pour une phase active au final de trois mois : le temps d'un pic (prévisible) de production de *Phare*, durée également programmée d'un contrat précaire.

▷▷▷▷ Les contre-mesures

Les caractéristiques de cette opération d'extrusion d'informations (et réel incident de sécurité ... informatique), reposent toutes sur des défaillances organisationnelles et humaines.

Trois grandes catégories se distinguent :

- **> le cadre sécuritaire :** l'absence de politique de sécurité est flagrante. Ou plutôt, la politique de sécurité est réduite à un minimum très dangereux ;
- **> l'organisation :** si des responsabilités sont définies, l'absence de contrôle rend l'organisation bien vulnérable ;
- **> l'humain :** le personnel peu ou pas sensibilisé n'a acquis ni discipline, ni réflexes de précaution.

Au-delà d'une réaction quasi immédiate du CEO et de la Direction Générale de *A* avec les recours en justice adéquats, cet évènement a provoqué un électrochoc.

Cependant, s'il aura fallu cela pour éveiller l'attention du *top management* aux risques d'activités encourus, les moyens nécessaires pour éviter qu'un incident majeur de ce type ne se reproduise n'ont pas pour autant été mis en œuvre. En effet, une culture d'entreprise ne se change pas aussi facilement !

Une conduite du changement a été amorcée avec de la communication interne, l'animation d'une dynamique de réflexion et des séances de sensibilisation pour travailler en profondeur l'esprit du personnel, et compléter ainsi les dispositions techniques de sécurité informatique.

Exemple 2 Les secrets du photocopieur

▷ Les enjeux

Rachats, fusions, absorptions... la vie de nos entreprises est bien mouvementée. Dans une opération de ce type, l'acquéreur est bien friand d'informations pour disposer d'arguments et négocier au mieux ses intérêts. Les méthodes d'acquisition d'informations peuvent être d'une esthétique douteuse. Le copieur illustre ici ce fâcheux incident qui s'est ainsi produit, à quelques détails près, sur le territoire français il n'y a pas si longtemps. Tout ne serait-il affaire que d'habillage ?

▷▷ Le mode opératoire

La façon de préparer une opération offensive est parfois bien particulière, au point d'exploiter des moyens illégaux pour collecter des renseignements. Ce sont les dures limites des activités d'intelligence économique.

Dans le cas présent, la victime est un copieur numérique : une brave machine œuvrant paisiblement dans son antre, à l'abri des passages courants, en compagnie d'un broyeur toujours affamé. Le couloir à cet étage est fréquenté par des gens importants et toujours pressés. Ce copieur n'a pas à traîner au milieu, d'autant que pour des raisons de sécurité, tous les documents hautement confidentiels, copiés ou imprimés via cette machine en réseau (évidemment bien cloisonné), doivent être systématiquement détruits au moyen du broyeur. Discipline que toute personne respecte particulièrement à cet étage.

Pourtant, cet équipement si banal soit-il, a permis une exfiltration d'informations ultra sensibles. Ces dernières ont été utilisées suffisamment maladroitement pour impliquer une réaction rapide des victimes.

Ce copieur présente un double intérêt pour un violeur de bastion aguerri : il *voit* passer des documents toujours intéressants (sinon ils ne seraient pas copiés !), et il est raccordé au réseau comme imprimante à gros tirage. La difficulté est d'atteindre cette machine en évitant de se faire repérer.

Pour un individu malveillant, voilà un composant important du système d'information, construit sur des briques relativement standards (Windows, Solaris, Linux, etc.) et objet d'une attention très relâchée.

Cependant, ce type d'appareil est rarement reconnu comme constituant du système d'information : ce n'est qu'un vulgaire copieur (à côté de son broyeur). Dans les grandes structures, il dépend plus souvent des services généraux que des services informatiques ou de la DSI. Aussi, il est souvent oublié des politiques de sécurité, et rarement sujet aux mises à jour logicielles. Déjà, patcher les postes utilisateurs à cause des virus, les serveurs parce qu'on a peur des vilains pirates, c'est laborieux. Si en plus, il faut s'occuper des copieurs, pourquoi pas aussi de la sécurité logique du système de gestion des ascenseurs, de celui de l'énergie, ou des PABX... Soyons sérieux : rien ne risque d'arriver !

Dans le cas présent, ce copieur n'a tout de même pas toutes les tares : il aurait aussi pu être raccordé à une ligne de télémaintenance. L'agent de maintenance n'aurait pas besoin de se déplacer.

▷▷▷ L'exploitation

L'existence de ce copieur est au départ une simple supputation. Partant du principe que n'importe quelle instance dirigeante utilise ce genre d'appareil, il est important de cibler correctement le type de matériel (marque, modèle, installation). Aussi, le plus simple est encore de venir sur place pour vérifier. Initier le repérage n'est pas particulièrement compliqué en soi : il faut par exemple identifier le mainteneur « légitime », une assistante au niveau désiré, signaler un rendez-vous de contrôle de maintenance.

L'identification du mainteneur est aisée : pour des questions de rationalisation des coûts, les contrats de maintenance sont rassemblés et confiés à un interlocuteur unique. Ce dernier prend un soin tout particulier d'étiqueter sur ou dans ces machines, sa publicité et la date d'intervention. Une visite dans les étages (entrer et se balader dans des étages secondaires d'une société peut être vraiment trivial) suffit à collecter ces informations.

L'identification des personnes est aussi une tâche peu complexe (c'est le job de tout bon commercial qui se respecte !). Le plus dur reste encore la prise de rendez-vous qui doit *paraître* anodine et vraiment de routine.

Sa préparation inclut bien sûr une vérification d'existence de procédure de contrôle de la société cible auprès de la société de maintenance.

Le jour convenu, voilà donc un faux agent de maintenance qui s'authentifie naturellement à l'accueil, qui passe les différents filtres de contrôles, pour finir avec sa caisse à outils devant le copieur numérique assoupi. Il connaît maintenant précisément la marque, le modèle et après un rapide contrôle, le niveau de version de ses constituants logiciels.

Reste à repasser « parce qu'une fatigue sur le fil corona a été trouvée et que n'ayant pas la bonne pièce pour ce modèle, il faut revenir demain ». L'assistante, professionnelle et froide, mais compréhensive, approuve vaguement (elle a des choses importantes à faire).

Voilà donc comment le copieur se retrouve avec un système « mis à jour » de nouvelles fonctionnalités : la sauvegarde compressée systématique des documents copiés ou imprimés, un module d'exploration furtive de réseau, un autre de communication subtile, etc. Un disque dur de bonne capacité préparé pour l'occasion achève de compléter le dispositif. La « réparation » du fil corona effectuée, le faux agent de maintenance laisse l'ancienne pièce à l'assistante, avec un double de son bon d'intervention signé (n'oublions pas : c'est la procédure normale).

Après avoir très professionnellement précisé qu'avec l'usage important de cet appareil, une visite de contrôle rapprochée serait peut-être nécessaire (un vrai agent cette fois-ci), il ressort tout tranquillement avec son disque dur d'origine rangé parmi les pièces de rechange de sa caisse à outils.

Une autre fausse intervention échangera de nouveau les disques si besoin. Et voilà quelques dizaines de gigaoctets d'informations hautement confidentielles dehors. En effet, le recouvrement des données effacées d'un disque dur reste encore aujourd'hui un exercice que quelques sociétés spécialisées maîtrisent. N'en déduisez pas qu'elles soient incriminées dans ce délit : d'autres sociétés ou organisations disposent aussi des moyens comme du savoir-faire approprié.

▷▷▷▷ Les contre-mesures

À la différence du premier cas, le niveau de maturité des pratiques sécuritaires de la société victime est déjà important.

L'exploitation joue principalement sur une défaillance organisationnelle dans la sécurité des systèmes d'information : une vision très réductrice de ce que recouvrent réellement les notions de *système d'information*, de sécurité des informations et de sécurité informatique. Les approches, le plus souvent techniciennes (principalement par confort ou par limitation « politique »), oblitèrent de fait une quantité importante d'axes de protection comme de composants.

Une autre caractéristique de cette opération consiste à exploiter, d'une part les relations entre gros donneurs d'ordres et petits fournisseurs, et d'autre part, des « pré-programmations » psychologiques de quelques intervenants gentiment manipulés pour l'occasion.

Les contre-mesures portent sur quatre domaines :

> **une sérieuse révision du rôle du RSSI**, qui se voit repositionné par la même occasion dans l'organisation. Ses pouvoirs sont élargis en même temps que ses moyens, ses responsabilités et ses attributions en sûreté des informations ;

> **la redéfinition du périmètre des systèmes d'information et du champ de ses moyens** : copieurs, fax, pabx, interactions entre personnes, etc., en plus des systèmes informatiques et de communications plus traditionnels ;

> **un sérieux durcissement de la discipline** avec l'application stricte des procédures de prévention, de contrôle et de réaction, en particulier dans les environnements classifiés. Le règlement intérieur prévoit des sanctions qui vont jusqu'à la faute professionnelle, en plus de rappeler à chacun ses responsabilités, autant civiles que pénales. Les audits sont menés pour vérifier la conformité des pratiques sécuritaires.

> **la sélection des personnes clefs** (et les assistantes en sont aussi !), l'entretien de leur sensibilité, et le contrôle de leur capacité de réaction aux anomalies sécuritaires.

De fait, cette société victime a subi des pertes suffisamment importantes pour comprendre, dans la douleur, la nécessité d'appliquer une réelle politique de sécurité, rigoureusement définie pour la globalité du SI.

Un moyen d'y parvenir est une interaction très forte du RSSI avec les différents processus de la société, en particulier le top management, la gestion des ressources humaines et de l'organisation, les services généraux autant que le management des systèmes d'information... et bien sûr l'informatique.

Exemple 3 Si ce n'est pas par devant, alors par derrière... sinon à travers

▷ Les enjeux

Un grand opérateur dispose d'une activité de centre d'appels. Proposant ses prestations à de grandes sociétés, il capte ainsi une partie de la gestion de la relation client de celles-ci : télé-prospection, enquêtes de satisfaction, support après-vente (*hot line*), etc. Ces sociétés utilisatrices ont ainsi une dépendance fonctionnelle importante à cet opérateur.

Une manœuvre de déstabilisation sur l'un des clients de cet opérateur, la société *A*, doit permettre à une société *B* de prendre un avantage concurrentiel important lors de la phase de lancement d'un nouveau produit.

Cet avantage peut s'avérer durablement déterminant pour *B*, non seulement par rapport à *A*, mais aussi pour le positionnement stratégique de *B* sur son marché général. Bloquer la communication de *A* avec ses clients pendant quelques jours suffirait à créer des conditions optimales.

▷▷ Le mode opératoire

Pour satisfaire des objectifs de coûts très réduits, l'opérateur héberge ses activités de *call center* dans des bâtiments anciens peu entretenus (le minimum légal). Les conditions de travail peu engageantes sont à l'origine d'un *turn-over* très important.

Qu'il s'agisse des télé-opérateurs comme de l'encadrement, le personnel se connaît peu, et dans l'ensemble, est peu motivé ou démobilisé. Les contrôles sont par là même soit inexistants, soit bâclés quand ils sont réalisés.

Le bâtiment concerné est facile d'accès. La porte principale est toujours grande ouverte, sans gardien, dans une rue assez passante. Les allées et venues du personnel (qui ne se connaît pas) animent le passage. À l'arrière du bâtiment, une ruelle dessert un accès aux sous-sols, qui hébergent comme très souvent les moyens énergétiques du bâtiment et leurs secours. Cet accès est très faiblement protégé et non surveillé.

La salle informatique est bien protégée : les locaux techniques sont isolés et fermés à clef. Malheureusement, ils donnent sur l'extérieur : les faisceaux de câbles passent au travers d'une ouverture béante qui donne sur la rue. La distribution se fait naturellement en façade... de bâtiment !

Ce qui est encore plus intéressant, c'est que ladite ouverture se trouve à une hauteur accessible depuis la rue.

Une fois de plus, un être vil et malveillant a vraiment l'embarras du choix pour opérer en toute quiétude. Aussi, il décide d'intervenir de nuit, avec une bonne échelle et des cisailles à partir de la rue. Il aurait pu choisir de jouer sur les moyens énergétiques en programmant d'abord une surtension trop importante qui aurait pu griller tous les appareils du bâtiment, puis une destruction méthodique des systèmes d'alimentation principaux et des générateurs de secours. Trop fatigant.

▷▷▷ L'exploitation

Un faisceau cisaillé rend déjà inexploitable un site. Lorsque les distributions sont en plus soigneusement enlevées, le bâtiment est condamné au recâblage. Le call center est ainsi paralysé, bloquant par la même occasion les relations clients des sociétés utilisatrices des services de cet opérateur. Le personnel du call center travaille dans de trop mauvaises conditions pour se mobiliser sérieusement. Personne n'ayant particulièrement envisagé ce cas, tant chez l'opérateur que dans ses sociétés utilisatrices, la gestion de crise se révèle assez catastrophique.

La société *A*, comme les autres utilisatrices, improvisent des solutions de secours pour tenter d'assurer la continuité des relations informationnelles avec ses clients. Le temps de réagir, de procéder à une chasse aux responsables, et surtout remettre en service l'informatique du call center et ses réseaux (un réseau complet est à retirer, depuis la distribution des plateaux jusqu'au locaux techniques de la salle informatique), trois bonnes semaines se passent avant de revenir à un fonctionnement de secours.

La société *B* a, entre-temps, eu le champ libre pour parvenir à ses fins.

▷▷▷▷ Les contre-mesures

Dans le cas présent, deux niveaux sont à distinguer : celui de la société utilisatrice et celui de son fournisseur.

En optant pour l'externalisation de processus vitaux pour son activité économique, la société utilisatrice crée une dépendance dont elle doit pouvoir s'affranchir en cas de crise. En proposant des services d'externalisation, le fournisseur, quant à lui, doit accepter de mettre en œuvre les moyens et les pratiques garantissant ses engagements, en particulier sécuritaires. Ce doit être contractuel et auditable par la société utilisatrice. Une première série de contre-mesures concerne donc l'intégration des dispositions de sécurité des systèmes d'information dans les contrats qui formalisent les relations client/fournisseur.

Pour ce qui concerne l'opérateur, la remise à niveau de la SSI pour cette activité de centre d'appels impose pratiquement de revoir le bâtiment, les méthodes de management, les conditions de travail, les pratiques de pilotage de l'informatique, etc. En effet, le plus stupéfiant dans ce cas n'est pas forcément de voir un bâtiment câblé de la sorte, mais plutôt de constater que ça n'ait pas choqué les responsables.

Enfin, la préparation des dispositifs de gestion de crise est un impératif pour permettre la continuité des activités et, au moins, assurer le bon déroulement des plans de secours. Cela nécessite préparation et entraînement, tant de la société utilisatrice, que de son fournisseur et des deux ensemble. Bref, ça s'organise.

Exemple 4 Sans les protections ça produit davantage... sauf que

▷ Les enjeux

Un grand industriel veut augmenter son rendement pour gagner en productivité et donc en rentabilité. Sa production est réalisée sur un site classé SEVESO II. Autrement dit, c'est un site *classifié* du fait de sa dangerosité par les risques industriels qu'il présente. Ce dernier cas est donc très différent des précédents. Ce n'est plus le « centre de gestion » qui est concerné par la sécurité des informations ou des systèmes d'informations, mais un site qui peut tuer ou être utilisé pour ce faire.

Les enjeux peuvent donc être de deux natures bien distinctes, économiques pour l'industriel, ou à caractère indirect de système d'*arme* pour d'autres...

▷▷ Le mode opératoire

Les systèmes de production (systèmes automatisés et automates) intègrent des mécanismes de régulation, de contrôle et de réaction prévus pour se déclencher en cas d'anomalie. L'utilisation de ces systèmes, en dehors de leurs plages normales d'exploitation, peut permettre de produire davantage. Cependant, les mécanismes de sécurité programmés sont prévus pour

fonctionner dans un cadre « normal » défini. Par conséquent, pour produire davantage, il est nécessaire de faire sauter ces protections ou de les configurer de manière arbitraire pour les rendre inopérantes.

Outre le contrôle de la sécurité par le responsable local, le site est assujetti à une surveillance particulière des DRIRE et des sociétés de contrôles (APAVE, etc.). Cependant, si ces dernières vérifient surtout les dispositifs généraux de contrôles (incendie, etc.), l'intégrité d'exploitation des automates reste quant à elle plus délicate (si tant est seulement qu'elle soit l'objet de vérifications) : les mécanismes de logging et de gestion des traces sont souvent assez vulnérables, et les fonctions de journalisation n'étant pas implémentées ou non configurées, le constat du détournement des appareils est assez difficile à réaliser.

Ces processus industriels sont historiquement bien éloignés du monde calfeutré et globalement aseptisé des processus de gestion et de pilotage des sociétés : c'est le monde de la Production, celui du cambouis, des fumées et des poussières. On n'a jamais vu descendre un RSSI en bleu de travail et casque de chantier dans des bureaux d'études ou des salles de production (quelle idée saugrenue).

La mesure du risque liée à l'informatique est le plus souvent réduite à néant : les DRIRE comme les sociétés de contrôles ont une culture peu adaptée, voire faussée, de la sécurité informatique et limitée à la trilogie Internet / virus / vilains pirates. Ce n'est jamais que la vision réductrice courante de la sécurité des technologies de l'information et des communications !

Par ailleurs, la réalité de la situation de la sûreté de l'informatique industrielle peut laisser pantois. Les systèmes des plates-formes de supervison et de contrôle des automates en particulier sont rarement mis à jour (et du Windows 98 ou NT4 jamais patché est assez vulnérable). Ils ne font pas l'objet non plus de contrôles d'intégrité logicielle...

Pour achever, si les automates se trouvent généralement sur un réseau industriel dédié, leurs systèmes de supervision sont le plus souvent aussi raccordés au LAN standard pour interagir avec la gestion de production (GPAO), celle des stocks, etc.

De plus, cette informatique « industrielle » étant rarement sous la responsabilité de la DSI, mais plus généralement sous celle des services techniques de l'usine, sa sécurité informatique échappe au RSSI (l'éternel problème du champ d'action de ce dernier). En local sur le site, culturellement très éloignés de ces préoccupations, le responsable de la sécurité « industrielle » et les CHSCT ne considèrent pas cette part non identifiée de sécurité informatique dans leurs attributions.

Voici donc l'informatique industrielle d'un site SEVESO II en réseau, utilisant des technologies standards, sans surveillance ni protection particulières, avec quelque part au détour d'une interconnexion, Internet ou des bornes wifi...

La désactivation des « grilles de sécurité » implique le plus souvent une opération manuelle : c'est en général un mode de maintenance pour les automates qui peuvent alors tourner « à vide », sans risque de faire de mélanges ou de provoquer des réactions en chaîne, non contrôlables.

La modification logicielle pour une programmation détournée des automates est réalisable par simple modification de sa grille de valeurs. L'augmentation du rendement par suspension des sécurités n'est pas le fait de l'opérateur. Cependant, il charge en toute bonne foi dans sa chaîne automatisée le fichier qui lui est transmis pour un ordre de fabrication donné.

Ce fichier est généré en bureau d'étude (peu protégé et peu contrôlé), stocké sur un serveur bureautique ou transmis par le mail interne. Une variante sur certains sites est de transmettre, via le système de gestion de production, une « gamme » associée à un « ordre de fabrication » : un ensemble de valeurs parmi lesquels se retrouvent les enregistrements pour la grille de sécurité.

La circulation de ces informations n'est pas associée à des protocoles d'authentification ou de contrôle d'intégrité particuliers. À la limite, un *checksum* des valeurs, uniquement stocké avec elles, existe parfois. Il ne sert donc à rien.

▷▷▷ L'exploitation

L'élévation des rendements pour satisfaire des objectifs économiques est une décision qui provient de la hiérarchie. La responsabilité généralement très diluée permet de générer toutes les conditions favorables pour que le personnel ne remarque pas d'anomalie (c'est une méthode de management très courante, notamment dans les environnements sociaux à forte présence syndicale).

Le résultat est une déviance du fonctionnement systémique de l'entité industrielle. Son système d'information au sens large n'incluant pas de dispositifs de sécurité et de contrôle des informations, les risques ne sont pas perçus.

En bureau d'étude, la manipulation conceptuelle des valeurs et des programmes des automates éloignent de la réalité de leur danger. L'appareil industriel peut ainsi être « configuré » pour un meilleur rendement.

Les sécurités programmées « au minimum » sur certaines machines et suspendues sur d'autres permettent d'avoir une chaîne de production optimisée pour la performance, mais dangereuse : il n'y a plus de contrôles.

Cette configuration a provoqué l'emballement d'une machine (une conduite sous pression a lâché). Par réaction en chaîne, l'ensemble du processus industriel est rapidement débordé.

Cet accident aurait pu être très grave (destruction violente de l'usine) : une erreur de programmation a bloqué un automate...

Au final, ce ne sera que la production d'un mois qui sera à jeter. En matière de rendement, ce n'est pas très payant.

▷▷▷▷ Les contre-mesures

Étant donné l'importance de l'usage des technologies de l'information dans les équipements qui composent l'environnement de travail, industriel notamment, il est devenu impératif de considérer la sécurité informatique sur ces périmètres.

Comme le spécifie l'article L231-3-1 du Code du Travail, tout chef d'établissement est tenu d'organiser une formation pratique et appropriée en matière de sécurité, au bénéfice des travailleurs qu'il embauche, de ceux qui changent de poste de travail ou de technique, des travailleurs liés par un contrat de travail temporaire, etc.

La législation récente concernant les risques technologiques (au sens très large) renforce encore les obligations, même si des décrets d'application manquent encore. Malgré les efforts récents de communication de certains préfets envers des directions de sites, il reste encore beaucoup à faire pour éviter de se tromper de débat lorsqu'on parle de sécurité informatique. Un renforcement de la sensibilisation des DRIRE sur ces aspects serait en outre nécessaire, mais ça relève d'une volonté politique...

Les contre-mesures à des situations ubuesques (mais réelles !) du cas présenté sont aussi courantes que peu nombreuses. Bien qu'elles ne soient pas évidentes à mettre en œuvre (pour cause d'inertie culturelle), la volonté stratégique a permis de corriger un certain nombre de pratiques dans la société évoquée, avec :

→ une responsabilité identifiée pour surveiller, contrôler et alerter ;

→ un positionnement adapté du RSSI dans la structure et une organisation cohérente de la filière SSI/Sûreté ;

→ une politique de sécurité des systèmes d'information qui considère aussi les périmètres de production ;

→ une première campagne de sensibilisation sur les personnels concernés ;

→ un contrôle technique (« audit ») pour mesurer, par automates, périmètres et systèmes automatisés, les besoins en sécurité informatique ;

Un renforcement de la sécurité informatique sur le périmètre technique devrait ainsi suivre. Sur le plan organisationnel, la sécurité des procédures industrielles (séquences d'opérations) est revue, depuis la R&D qui spécifie les systèmes programmables adaptés aux procédés, en passant par les bureaux d'études, la Fabrication et jusqu'à la Qualité. La SSI de production est aussi un aspect de la sûreté industrielle.

Conclusion

Les sociétés humaines, à l'instar des individus qui les font fonctionner, développent des habitudes. Certaines permettent de renforcer la sécurité des systèmes d'information et d'autres d'affaiblir leur sûreté. Viser un niveau de sécurité absolu dans la formidable complexité d'une grande société est une utopie tout juste bonne pour un jeune théoricien. En revanche, atteindre un certain niveau de maturité par rapport à quelques pratiques clefs permet de sérieusement limiter les dégâts. Ces « meilleures pratiques » relèvent le plus souvent du bon sens :

1. Connaître ses risques d'activité en les réactualisant régulièrement par du contrôle interne ou externe (audits) ;
2. Avoir une politique de sécurité rationnelle, objective et adaptée (au minima une approximation de l'ISO17799 !) ;
3. Disposer de plans de secours pour la continuité d'activité des processus clefs (à moins d'aimer les aventures du type du call center...) ;
4. Sensibiliser et entraîner les employés à authentifier les personnes qu'ils reçoivent ;
5. Maintenir à jour ses équipements et contrôler régulièrement leur intégrité, en particulier celle des configurations ;
6. Chiffrer les informations sensibles et assurer la cohésion des chaînes de confiance (les équipements comme les copieurs peuvent vous trahir : des moyens existent) ;
7. Reporter au CEO, Président ou PDG : il est seul responsable devant la loi, socialement, civilement et pénalement. Il n'est censé ignorer ni la loi, ni la situation sécuritaire dont la connaissance est une obligation pour lui.

Dans la pratique, pour contrer la menace informationnelle, CEO, DRH, Sûreté ou RSSI, à chacun de prendre et assumer ses responsabilités en matière d'ingénierie sociale. Intégrer ces pratiques à la culture d'entreprise ne coûte pas tant que ça. C'est même une action de management qui peut rapporter gros : les agresseurs l'ont bien compris eux. En effet, leur facteur humain n'est pas un « maillon faible », mais un vecteur à la force surprenante... Dans la chaîne alimentaire du monde économique, ces prédateurs n'hésitent pas.

L'observation de ces pratiques permet un meilleur emploi des professionnels de la sécurité des systèmes d'information : ils peuvent alors apporter une vraie valeur ajoutée quant aux spécificités de l'activité, de l'organisation, de la société, par rapport à des menaces réelles.

Authentification : clé de voûte de la confiance

Etant donné l'interaction, voire la dépendance grandissante entre les systèmes d'information et le monde réel, la confiance que nous mettons dans les SI est essentielle. La capacité à connaître et reconnaître ses interlocuteurs est un point indispensable à cette confiance, d'où la nécessité de fournir un ou plusieurs mécanismes d'authentification au SI. Un tel mécanisme permet à une entité de prouver son identité à tout élément du SI. Des qualités de ce mécanisme dépend le bien-fondé de la confiance que nous avons dans cette authentification.

Pourquoi a-t-on besoin d'authentification ?

Les objectifs classiques de la sécurité d'un SI sont souvent regroupés en trois points :

- la confidentialité garantit que l'accès aux informations n'est permis qu'aux entités (personnes, processus) autorisées ;
- l'intégrité est la propriété d'un SI interdisant l'altération d'un élément du SI par une personne illégitime ;
- la disponibilité assure la continuité des services offerts par le SI, que ce soit face à des problématiques d'administration ou à des problématiques de sécurité.

On ajoute également les notions d'auditabilité et de preuve, qui permettent d'identifier les actions effectuées sur le SI et de prouver les responsabilités correspondantes. Cet objectif est atteint en déployant des mesures de traçabilité fiables et de non-répudiation des échanges.

L'authentification sert ces objectifs ; directement pour la confidentialité, l'auditabilité et la génération de preuves, indirectement pour l'intégrité et la disponibilité en limitant les personnes malveillantes ou maladroites accédant au système. C'est pourquoi ce service a suscité autant d'intérêt depuis les débuts de l'informatique. Aujourd'hui plus que jamais, le besoin de connaître de manière fiable l'identité de l'utilisateur d'un système est essentiel.

Le temps où une adresse IP était une preuve suffisante pour s'authentifier auprès d'une machine est heureusement révolu, même si cela a permis certaines attaques célèbres à l'origine de bien des vocations [1].

Gardons à l'esprit cependant que ce service n'est pas suffisant. Notamment, l'autorisation, mise en œuvre par le contrôle d'accès, est un autre service distinct de l'authentification (qui lui est nécessaire). La mise en œuvre de ces services passe parfois par les mêmes mécanismes (ex : les Pluggable Authentication Modules - PAM - interviennent à la fois dans l'authentification et l'autorisation).

Qu'est-ce que l'authentification ?

L'authentification cherche à garantir à une entité (personne, machine, application) l'identité d'une autre entité, aux défauts, limites, failles du mécanisme près.

Un tel mécanisme repose sur les éléments suivants :

- une entité cliente ou prouveur : tout élément du SI (individus compris) ayant besoin de s'authentifier ;
- le vérifieur : entité à laquelle l'entité cliente doit prouver son identité ;
- une caractéristique discriminante : propriété d'une entité cliente, représentative de son identité ;
- un mécanisme d'authentification : destiné à vérifier la présence de la caractéristique discriminante chez l'entité cliente.

Dans le cas d'un service d'autorisation, il faut ajouter les éléments suivants :

- la ressource accédée : service (au sens large) dont l'accès est demandé par l'entité client ;
- les droits : opérations qu'une entité cliente peut effectuer sur la ressource ;
- un mécanisme d'autorisation : destiné à vérifier les droits d'accès demandés par l'entité cliente sur la ressources accédée.

Le Tableau 1 ci-dessous illustre les caractéristiques précédentes à trois niveaux différents : sécurité physique, sécurité réseau et sécurité des applications.

Tableau 1

	Physique	Réseau	Application
Objectif	Accès à la salle informatique	Accès à un flux	Accès à un segment de la mémoire partagée
Entité cliente	Individu	Nœud réseau	Processus
Vérifieur	Lecteur de badge	Firewall	Noyau
Caractéristique	Badge magnétique	Adresse IP	PID
Mécanisme d'authentification	Reconnaissance du badge	Vérification IP source	Test du PID
Ressource	Salle informatique	Flux	Segment de mémoire
Droits	Entrer	Traverser l'infrastructure	Lire
Mécanisme	Contrôle d'accès	Filtre sur adresse IP	ACL Unix

ManuX - <ManuX@rstack.org>
hb - <hb@rstack.org> - http://hb.rstack.org/

Mécanismes d'authentification

Du point de vue du prouveur, l'accès à une ressource s'effectue selon deux étapes :

→ **1.** le prouveur annonce son identité ;

→ **2.** en mettant en œuvre le mécanisme d'authentification, le prouveur convainc (ou non) le vérifieur de sa possession de la caractéristique discriminante.

Les mécanismes d'authentification sont regroupés en fonction de la preuve apportée par le prouveur pendant la deuxième phase [2]. On distingue à cet effet quatre méthodes primaires d'authentification : je connais, je possède, je sais faire et je suis.

Je connais

Cette approche est la plus répandue parmi les mécanismes d'authentification. Sous sa forme la plus simple, elle consiste à vérifier que l'utilisateur connaît un mot de passe. Il s'agit alors d'un système fondé sur le partage d'un secret (mot de passe en clair) ou d'un dérivé du secret (hash du mot de passe).

Cette méthode est également utilisée au niveau d'applications, ainsi une application accédant à une base de données « connaît » le mot de passe stocké dans un script, ou au niveau d'un réseau sans fil avec le partage d'une clé WEP.

D'autres mises en œuvre font évoluer le système telle cette implémentation, plus artistique [3].

Dans cette présentation, figure 1, au « Security Symposium » 2000 de Usenix, R. Dhamija propose une solution qui s'appuie sur la capacité d'un utilisateur à reconnaître un ensemble d'images abstraites parmi d'autres. Ce schéma secret est censé être facile à retenir et difficile à communiquer, accroissant ainsi le niveau de sécurité du mécanisme d'authentification.

Je possède

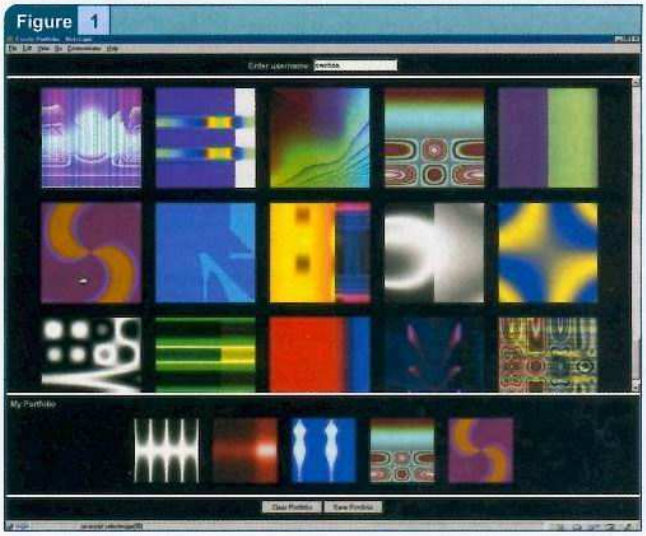
Cette méthode d'authentification s'appuie sur un composant tiers caractéristique de son identité. Une carte magnétique donnant accès à une salle informatique ou une clé USB permettant de démarrer un ordinateur sont deux exemples de cette catégorie.

Dans le cas d'une authentification par clé publique / clé privée, il est impossible de se souvenir de la clé privée. Elle est stockée sur un médium tel qu'une carte à puce. De cette manière, l'authentification par « je connais » devient une authentification par « je possède ». Enfin, le cas le plus connu de cette approche est le serveur SSL qui possède un certificat l'authentifiant.

Je suis

L'authentification par « je suis » définit un élément que l'entité cliente personnifie. Dans cette approche, l'entité cliente fournit au mécanisme d'authentification une caractéristique physique intrinsèque telle que l'empreinte digitale, la reconnaissance de la

Figure 1



rétine, de l'iris, de la voix, ou encore la géométrie du visage ou de la main. Cette authentification biométrique est un sujet très à la mode, notamment en raison de son côté spectaculaire.

Cependant, cette approche est loin d'être une panacée. Dans [5], P. Wolf décrit les raisons de cette « fascination » et surtout les limites de l'authentification biométrique. La falsification de données biométriques est tout à fait envisageable (notamment la faisabilité « simple » de la falsification d'empreintes digitales a été démontrée [6]). Pourtant, leur diffusion ne correspond pas aux critères de sécurité d'une clé d'authentification. Que peut-on dire en effet du caractère « confidentiel » d'une empreinte digitale ? La difficulté de contrôler cette diffusion fait les beaux jours de la police scientifique, mais pas forcément ceux d'un système d'authentification sûr. De plus, certains pays pratiquent un recueil systématique de données biométriques à leur frontière. Dans ces conditions, la compromission d'une donnée biométrique est tout à fait envisageable.

Si l'exemple des données biométriques est un cas particulièrement à la mode, il ne faut pas oublier que l'authentification par « je suis » existe depuis longtemps pour les composants du SI, sous la forme de l'adresse MAC ou IP. Bien entendu, la faculté de modifier son adresse MAC rend une telle technique particulièrement faible, néanmoins elle reste un cas d'école.

Je sais faire

L'authentification par « je sais faire » consiste à évaluer la capacité de l'entité cliente à effectuer une opération spécifique. Encore peu appliquée dans le monde logique, elle est très largement utilisée dans le monde réel sous la forme de la signature manuscrite, intégrant des paramètres cinétiques que l'on ne peut plus associer au savoir, mais au savoir-faire.

Un autre exemple est la rythmique d'écriture ou de saisie d'un mot. Répéter plusieurs centaines de fois une telle opération va faire

naître des automatismes de saisie chez l'utilisateur. Ces automatismes peuvent s'avérer caractéristiques de l'identité d'une entité cliente.

Authentification forte

Une définition usuelle de l'authentification forte, même si ce n'est pas celle des cryptographes, est la suivante : l'authentification forte ou « *Two factors authentication* » consiste à allier au moins deux méthodes parmi celles décrites ci-dessus. Ainsi, une clé USB contenant une clé privée (je possède) et protégée par un code PIN (je connais) est l'exemple typique d'authentification forte. De la même manière, la reconnaissance d'une empreinte palmaire associée à un ID unique fournit une authentification forte de type (je suis + je connais). Enfin, un mécanisme de saisie des mots de passe intégrant une notion de rythmique, telle que décrite précédemment, est un mécanisme de type je connais + je sais faire.

L'authentification forte a cependant ses limites. En effet, reprenons le cas d'une clé USB protégée par un code PIN. L'accès à une telle clé peut être assez simple, que ce soit par le vol ou en exploitant un oubli de l'utilisateur (dans la salle à café, sur son bureau, etc.). Une fois l'accès acquis, le secret n'est plus protégé que par un mot de passe de 4 caractères numériques, ce qui n'est pas particulièrement résistant.

Protocoles et outils

Vecteurs de ces différentes méthodes d'authentification, de nombreux protocoles et outils mettent en œuvre, plus ou moins adroitement, ces différentes approches. Il est alors important de distinguer les protocoles d'authentification à proprement parler, des protocoles « annexes », utilisés soit pour le transport des informations (RADIUS), soit pour fournir des briques à l'architecture globale d'authentification.

Protocoles d'authentification

Parmi les protocoles d'authentification existants, trois seront détaillés dans ce dossier : OTP, Kerberos et SRP.

OTP (One Time Password)

Ce système d'authentification simple et très utilisé [9] cherche à résoudre les problèmes de sécurité consécutifs à la capture et la réutilisation de mots de passe. L'objectif est de fournir le moyen à l'entité cliente de fournir un mot de passe différent lors de chaque session d'authentification interdisant *de facto* toute tentative de rejeu.

Bien que de nombreuses implémentations existent, il est important de garder en tête qu'une norme existe, ratifiée par l'IETF (RFC 1938 - OTP Systems), sous le nom de S/Key.

Kerberos

Ce protocole d'authentification réseau a été mis au point au MIT. Plusieurs implémentations, propriétaires ou Open Source, sont disponibles, rendant ce protocole supporté par un grand nombre de systèmes. Il fait l'objet d'un article détaillé dans ce dossier.

SRP (Secure Remote Password)

Plusieurs versions de ce protocole, qui cherche lui aussi à résoudre les problèmes classiques de l'authentification client/serveur sur un réseau, ont vu le jour. Il présente l'avantage de ne pas posséder certaines faiblesses de Kerberos par exemple.

Protocoles et outils annexes

Les annuaires et leurs protocoles

L'utilisation du service d'annuaire pour centraliser les informations liées à l'authentification est ancienne. On obtient ainsi un système d'authentification distribuée. Le système NIS (Network Information System), conçu par Sun Microsystems dans les années 1980, est le plus ancien d'entre eux, supporté par un grand nombre de systèmes, y compris Windows (avec les outils des « Services For Unix »).

Allié à NFS, une autre invention de Sun, l'utilisation de NIS simplifiait énormément l'administration d'un grand parc. Malheureusement, c'était au prix de problèmes de sécurité sévères. La tentative d'amélioration avec le système NIS+, conçu pour corriger les faiblesses des NIS, est restée peu utilisée. Même Sun semble s'intéresser davantage à Kerberos avec son implémentation propre (Sun Enhanced Authentication Mecanism).

Les annuaires X500 et leur protocole d'accès LDAP (Lightweight Directory Access Protocol) fournissent un exemple plus réaliste d'utilisation d'un annuaire pour l'authentification dans des conditions acceptables de sécurité.

RADIUS (Remote Authentication Dial-In User Service)

Le protocole RADIUS est un standard ([7][8]) permettant d'authentifier des utilisateurs accédant à un réseau. Plusieurs méthodes peuvent être utilisées : login/mot de passe, challenge/réponse, Token, etc. Ce protocole fait l'objet d'un article détaillé dans ce dossier.

Généricité

Face à ce grand nombre de protocoles, en apparaissent d'autres fournissant un service d'authentification permettant de s'abstraire du mécanisme utilisé.

C'est l'objectif de la GSSAPI (Generic Security Service Application Programming Interface [10]) par exemple, qui fournit aux développeurs un moyen de réaliser de manière générique une application client/serveur sécurisée indépendante du mécanisme de sécurité sous-jacent.

C'est aussi le cas de EAP (Extensible Authentication Protocol), utilisé par la norme 802.1X. EAP est compatible avec plusieurs protocoles d'authentification décrits précédemment (SRP, Kerberos, RADIUS, LDAP, etc.) et a déjà fait l'objet d'un article dans MISC [11].

Les attaques sur l'authentification

Pierre angulaire de la sécurité d'un SI, les systèmes d'authentification ont toujours fait l'objet d'attaques afin d'en déterminer les failles. Ces attaques peuvent être actives ou passives, suivre différents schémas [12] :

- **l'impersonnification :** l'attaquant parvient à faire croire au vérifieur qu'il est le prouveur ;
- **le rejeu :** l'attaquant rejoue un défi qu'il a intercepté auprès d'un vérifieur, ce qui conduit aussi à une impersonnification ;
- **attaque à clairs choisis :** l'attaquant choisit méticuleusement les défis qu'il lance au prouveur afin d'extraire de l'information sur son secret. Il peut ensuite essayer de se faire passer pour lui ;

■ **attaque par dictionnaire ou exhaustive :** l'attaquant essaie successivement les mots d'un dictionnaire ou l'intégralité de l'espace des clés.

Ces attaques visent plusieurs éléments constituant du mécanisme d'authentification. L'objectif de l'authentification n'est pas de déterminer si une entité connaît un secret partagé, mais de vérifier de manière fiable l'identité de cette entité. La validation d'un mot de passe, par exemple, n'est qu'une modélisation de cette vérification. Son objectif est alors différent de celui de l'authentification.

La confiance dans le résultat repose sur le caractère acceptable de cette approximation, car elle peut faire naître des problèmes soit au niveau de la caractéristique discriminante, soit au niveau de la mise en œuvre de sa vérification.

Caractéristique discriminante imparfaite

Nous avons vu que la modélisation de l'authentification aboutit au choix d'une caractéristique propre à l'identité du client. La qualité de cette caractéristique, c'est-à-dire sa capacité à représenter de manière exclusive l'identité du client, a des conséquences directes sur la confiance accordée à l'authentification.

Ainsi, considérer que la connaissance d'un mot de passe est une caractéristique capable de représenter de manière exclusive l'identité d'un utilisateur, revient notamment à exclure que ce mot de passe puisse être trivial (facile à deviner ou à casser) ou communiqué à une autre personne.

Dans ce cas, l'objectif de l'authentification implique deux autres mesures :

→ assurer la robustesse des mots de passe ;

→ assurer leur caractère personnel (mot de passe caché sous le clavier, etc.).

Problèmes de mise en œuvre

Les mécanismes d'authentification ont pour but de vérifier la présence de la caractéristique discriminante. Même si on suppose que les algorithmes cryptographiques utilisés sont sûrs (ce qui n'est pas toujours vrai), la mise en œuvre de ces mécanismes ou leur intégration dans un protocole d'authentification réseau peut aboutir à la fragilisation de l'authentification.

Citons les procédures de diffusion, de changement de mots de passe, la distribution de clés publiques [13], ou les faiblesses des protocoles eux-mêmes (ex : [14]).

Facteur humain

Ce point, déjà évoqué, reste primordial dans la confiance envers le mécanisme d'authentification (divulgation, prêt, vol d'authentifiant) ainsi que dans son utilité même. Un mécanisme d'authentification, aussi sophistiqué soit-il, est inutile s'il suffit d'une ligne téléphonique et d'un « ingénieur social » pour obtenir l'information.

Il y aurait même un lien entre l'amélioration des moyens d'authentification et l'exploitation de cette « faille humaine ». « A mesure que les développeurs continueront d'améliorer les mesures de sécurité, rendant l'exploitation des failles techniques de plus en plus difficile, les agresseurs se tourneront davantage vers l'exploitation de l'élément humain [15] »

Conclusion

L'objet de l'authentification est la confiance. En ces temps où la « confiance dans l'économie numérique » [16] est un sujet brûlant, on comprend l'importance de disposer d'un service d'authentification fiable.

Les techniques et protocoles permettant d'atteindre cet objectif sont nombreux et parfois complexes à mettre en œuvre. La compréhension de ces techniques et de leurs limites est nécessaire pour savoir si cette confiance est bien fondée.

Par ailleurs, le service fourni par chacune de ces solutions n'est pas identique. C'est pourquoi certaines seront privilégiées à d'autres selon l'environnement et le besoin (gestion centralisée des utilisateurs, authentification unifiée, portabilité, extensibilité, API disponible, etc.).

Ce dossier propose quelques éclaircissements sur certaines de ces techniques couramment utilisées, sachant que leur apport doit être évalué dans son ensemble en considérant également le facteur humain et la sécurité physique.

Références

[1] J. Littman, *L'intrus - en ligne avec Kevin Mitnick, le Fugitif du Cyberspace*, Grasset, 1996, ISBN 2-246-54051-8

[2] E. Amoroso, *Fundamentals of computer security technology*, Prentice Hall, 1994, ISBN 0-13-108929-3

[3] R. Dhamija et A. Perrig, Déjà Vu : *A user study using image for authentication*, 9th USENIX Security Symposium, August 2000

[4] W. Diffie et M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, November 1976

[5] P. Wolf, *De l'authentification biométrique*, Sécurité Informatique n°46, octobre 2003

[6] **http://cryptome.org/gummy.html**

[7] RFC 2865

[8] RFC 2866

[9] Voir article sur OTP dans ce dossier

[10] RFC 2743 : « GSSAPIv2 »
http://www.ietf.org/rfc/rfc2743.txt

[11] MISC 11, janvier/février 2004

[12] Bruce Schneier, *Cryptographie appliquée*, Thomson Publishing

[13] Dossier PKI, MISC 13, mai/juin 2004

[14] E. Bouillon, *Kerberos et la sécurité*, SSTIC, juin 2004

[15] Kevin D. Mitnick, *L'art de la supercherie*, CampusPress, 2003, ISBN 2-7440-1570-9

[16] Loi n°2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique

De l'usage des mots de passe jetables

Comme son nom l'indique, le principe d'un système d'authentification One Time Password (OTP) est d'utiliser des mots de passe à usage unique. L'utilisateur doit, à chaque fois qu'il lui est demandé de s'authentifier auprès du système, fournir un nouveau mot de passe.

I. État des lieux

Le masque à usage unique, aussi appelé *One Time Pad* ou chiffrement de Vernam, peut être vu comme le lointain précurseur du mot de passe à usage unique. Un One Time Pad est une suite aléatoire de caractères : le chiffrement des données se fait alors en appliquant un OU exclusif avec le texte en clair. La solidité de ce système repose sur trois points :

→ le caractère aléatoire du masque ;

→ l'exclusivité de la connaissance du masque par les deux parties ;

→ son usage unique [1].

Le lecteur peut se référer aux travaux de Ding et Rabin [2] pour un aperçu des dernières recherches dans ce domaine.

Neil M. Haller a écrit un papier en 1994 [3] dans lequel il présente une implémentation de système d'authentification OTP pour Unix qui s'intitule S/KEY. Dans des temps où l'accès aux machines se fait via Telnet et l'usage des *r-commands* est affaire de tous les jours, la compromission des machines est aisée car les flux d'authentification transitent en clair. Ce système a donc pour principal but de contrer les attaques de type écoute du réseau. Cette implémentation évite le transit du mot de passe Unix de l'utilisateur sur le réseau et ne stocke d'information secrète nulle part. Ainsi, un attaquant qui intercepte un mot de passe de ce type ne peut pas le réutiliser pour s'introduire sur le système (*replay attack*). Une RFC [4] décrivant un système d'authentification OTP voit le jour en 1998.

Malgré sa simplicité de mise en œuvre, le déploiement de cette méthode d'authentification est faible. Les chiffres du Global Information Security Survey 2004 [5] montrent que le pourcentage d'entreprises qui l'emploient varie de 5% à 20% selon leur localisation géographique.

Néanmoins, Verisign a récemment choisi cette technologie à deux reprises et ces pourcentages pourraient augmenter dans le futur. Dans le premier cas [6], on retrouve le système OTP comme l'une des trois méthodes d'authentification de l'Open Authentication reference architecture (OATH). Dans le second cas [7], on le retrouve dans une couche d'authentification forte pour Windows Server 2003.

Avec la banalisation de l'utilisation des canaux chiffrés, le problème de la circulation du mot de passe en clair n'est plus d'actualité. La menace vient de l'environnement depuis lequel un utilisateur accède à un système. Le mot de passe à usage unique reste donc valable pour un utilisateur itinérant par exemple, qui se trouve contraint d'effectuer un accès à un système en utilisant un poste client qui n'est pas digne de confiance. Ainsi, il est intéressant d'utiliser un système d'authentification de type OTP pour se prémunir du *shoulder surfing* et des *keyloggers*.

II. Fonctionnement général d'un système OTP

La sécurité d'un système OTP réside dans deux points essentiels : le premier consiste en l'irréversibilité du protocole de hachage utilisé et le second provient du fait qu'il n'est pas utile de stocker le mot de passe sur le système (même sur la machine à protéger).

Les fonctions de hachage

La plus couramment utilisée dans le cadre des systèmes OTP est MD5. On peut toutefois trouver des utilisations de MD4, SHA (Secure Hash Algorithm) ou DES-MAC. OPIE (« One-Time Passwords in Everything ») utilise le hachage MD5 par défaut. Quoi qu'il en soit, le principe reste toujours le même et repose sur l'irréversibilité de ces fonctions : à partir du résultat, on ne peut pas remonter à la valeur initiale.

Génération des mots de passe non réutilisables

Nous parlons ici de la création d'une séquence de mots de passe « jetables ». La valeur de référence permettant de générer cette suite est trouvée comme suit :

- L'utilisateur fournit un mot de passe secret (ici nommé Secret). Celui-ci peut en théorie être de n'importe quelle longueur. Néanmoins dans la pratique, il est compris entre 10 et 63 caractères (donc plus long que la plupart des mots de passe Unix).
- Secret est alors concaténé avec une « graine » (Seed). Cette « graine » est longue de 1 à 16 caractères alphanumériques (de la norme ISO-646) convertis en minuscules avant d'être ajoutés à Secret.
- Le résultat de la concaténation Seed + Secret est passé à travers une fonction de hachage (sur 64 bits). On obtient ainsi la valeur initiale (Init-Value) de la future séquence de One-Time-Passwords.

Cette première étape achevée, les OTP sont générés en passant N fois (N choisi par l'utilisateur) Init-Value à travers la fonction de hachage. Ainsi, si on donne à N la valeur 500, le premier mot de passe utilisé sera Init-Value qu'on aura haché 500 fois. Le deuxième sera Init-Value passé 500-1 fois dans la fonction de hachage et ainsi de suite.

Pourquoi décrémenter le nombre de hachages ? Si une personne capture un mot de passe, comme la fonction de hachage n'est pas inversible, elle ne peut pas trouver le mot de passe suivant (ce qui serait par contre possible si on incrémentait N).

Gianpaolo Fasoli, **gfasoli@mail.com**
Jean Scholzen, **jscholzen@citali.com**

Figure 1

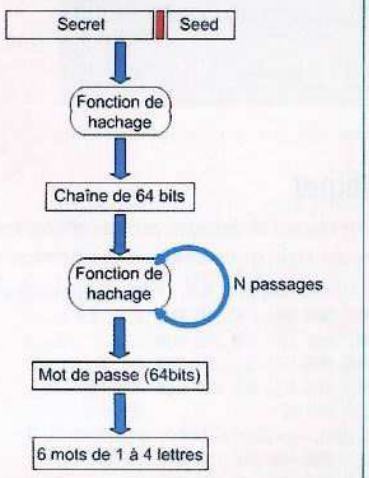
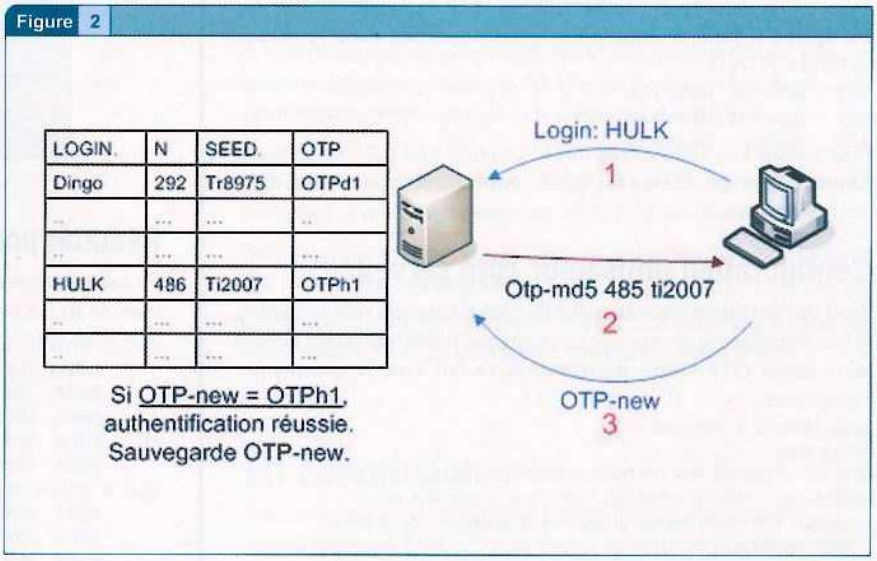


Figure 2



Représentation du mot de passe

Le mot de passe fait donc 64 bits. Pour faciliter la tâche de l'utilisateur (qui devra souvent transférer à la main le mot de passe), celui-ci peut être traduit en 6 mots de une à quatre lettres. Chacun de ses mots est codé sur 11 bits et choisi à partir d'un dictionnaire de 2048 mots. Tous les One Time Passwords peuvent être codés. La chaîne fait au total 66 bits : 64 bits du mot de passe + 2 bits de checksum (figure 1).

Un mot de passe représenté en mot serait par exemple de la forme : VENT BURG JIBE OR JOHN FOR.

Le challenge

Le otp-challenge qui sera utilisé lors d'un échange entre deux machines (voir l'exemple plus loin) est une chaîne comprenant tous les renseignements nécessaires pour le calcul du mot de passe :

- → le nom de la fonction de hachage utilisée : md5, md4, sha1 ;
- → un entier N (le numéro de séquence) ;
- → une graine.

La forme de ce challenge doit être comprise par tous les systèmes OTP : précédé de «otp-» avec un espace entre chacun des trois champs.

Exemple d'otp-challenge : `otp-md5 494 ob0071`

Fonctionnement d'une connexion

Nous étudions ici le cas d'un client désirant s'identifier auprès d'un serveur grâce au mécanisme OTP. La machine cliente possède un générateur d'OTP. Quant au serveur, il possède, outre un générateur, une base de données où sont enregistrés, pour chaque identité, le dernier OTP ainsi que la graine, le numéro de séquence et l'algorithme de hachage associés.

Les étapes de l'authentification sont les suivantes :

- Le client envoie son login au serveur.
- Le serveur génère un otp-challenge à partir des informations contenues dans sa base de données : otp-algo + (Dernier numéro de séquence – 1) + Seed. Il envoie ce challenge au client.
- Le client génère un otp à partir du challenge reçu : otp-algo + (numéro de séquence reçu) + (`Secret+Seed`).

Le serveur reçoit l'otp, le hache, puis le compare au dernier mot de passe de sa base de données : s'ils sont égaux, alors le client est authentifié (figure 2).

III. Mise en place de OPIE

La distribution choisie pour cette illustration est Debian. Il est bien évidemment possible de déployer ce logiciel sous d'autres Unix compatibles POSIX au format binaire, ou en recompilant le tout à partir des sources [8].

Installation côté serveur

Les paquetages nécessaires sont `opie-server` et `libpam-opie`. Le premier fournit les programmes nécessaires à la gestion et à la manipulation des mots de passe. Lors de son installation, il crée le conteneur `/etc/opiekeys` qui est construit de la manière suivante : une ligne par utilisateur constituée du username, du numéro de séquence courant + 1, de la graine, du hash du dernier mot de passe et de la date du dernier login.

On voit bien ici qu'à aucun moment, le serveur ne stocke de mot de passe en clair sur le disque. Le second paquetage fournit la librairie PAM nécessaire (`/lib/security/pam_opie.so`) pour ajouter l'authentification OTP aux services désirés.

Configuration PAM côté serveur

Pour qu'un utilisateur s'authentifie via OTP depuis la console ou via Telnet, il suffit d'ajouter les lignes suivantes dans le fichier `/etc/pam.d/login` :

```
auth    sufficient  pam_opie.so
auth    sufficient  pam_unix.so nullok
auth    required    pam_deny.so
```

Dans le cas de SSH, ces lignes doivent être ajoutées dans `/etc/pam.d/ssh`.

Configuration utilisateur côté serveur

Lors de l'installation, le conteneur de clefs a été créé, vide au départ. Il faut initialiser le conteneur pour chaque utilisateur ayant besoin d'un accès OTP. Cette initialisation se fait avec la commande `opiepasswd` :

```
gfasoli@obelix:~$ opiepasswd -c
Adding gfasoli:
Only use this method from the console; NEVER from remote. If you are
using telnet, xterm, or a dial-in, type ^C now or exit with no
password.  Then run opiepasswd without the -c parameter.  Using MD5 to
compute responses.

Enter new secret pass phrase: MOTDEPASSEPOUROTP
Again new secret pass phrase: MOTDEPASSEPOUROTP
```

Ce mot de passe servira ultérieurement à l'utilisateur pour calculer ses mots de passe OTP. De ce fait, il est **TRÈS FORTEMENT** conseillé de choisir un mot de passe différent de celui utilisé pour le login Unix.

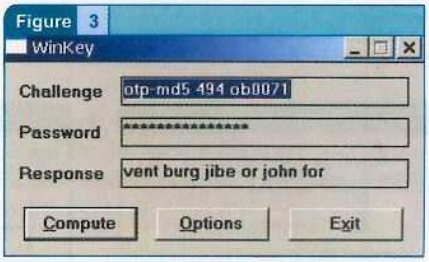
Utilisation du système mis en place

Une fois le système d'authentification en place, l'utilisateur peut, au choix, pré-générer un certain nombre de clefs qu'il emportera sur lui (avec les risques que cela comporte) ou calculer son mot de passe lors de chaque login. Dans le second cas, l'utilisateur ne court pas le risque de se faire dérober une liste de mots de passe, mais doit avoir un calculateur à disposition à chaque login. Bien évidemment, l'utilisateur peut télécharger un logiciel pour cette tâche, mais la question de l'intégrité du poste client se repose, alors que dans le premier cas, il n'en est nullement question.

Pour le calcul des clefs, l'utilisateur doit connaître la séquence, la graine et son mot de passe. Sous Debian, le calcul se fait avec `otp-md5` (dans le paquetage `opie-client`). Pour en générer 5 en partant de la 495ème, l'utilisateur procédera comme ceci :

```
gfasoli@obelix:~$ otp-md5 -n 5 495 ob0071
Using the MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase: MOTDEPASSEPOUROTP
491: ASH BALD AFRO DANK COT BALI
492: HUT BONN SWUM LOIS LEST GRIN
493: DANK DISH EEL DISC WADE LIEU
494: VENT BURG JIBE OR JOHN FOR
495: PUP BREW FUSE GIG TREE FILL
gfasoli@obelix:~$
```

Il existe aussi des générateurs de clefs pour HP48 [10] et Palm [11]. Des calculateurs pour téléphones portables commencent même à apparaître [12]. Sous Windows, l'utilisateur peut se tourner vers un outil graphique du type WinKey [9] (figure3 ci-dessous).



Résultat pour Telnet

En examinant la capture réseau ci-dessous, on voit effectivement le mot de passe circuler en clair pour une authentification telnet.

```
19:49:38.062199 IP localhost.telnet > localhost.36561: P 125:158(33) ack 86 win 32767
        0x0000:  4510 0049 f595 4000 4006 4707 7f00 0001  E..I..@.@.G.....
        0x0010:  7f00 0001 0017 8ed1 aa19 5d4b a9c6 aa6d  ..........]K...m
        0x0020:  5018 7fff de40 0000 6f74 702d 6d64 3520  P....@..otp-md5.
        0x0030:  3439 3420 6f62 3030 3731 2065 7874 2c20  494.ob0071.ext,.
        0x0040:  5265 7370 6f6e 7365 3a                   Response:
19:49:38.062210 IP localhost.36561 > localhost.telnet: . ack 158 win 32767
        0x0000:  4510 0028 0377 4000 4006 3947 7f00 0001  E..(.w@.@.9G....
        0x0010:  7f00 0001 8ed1 0017 a9c6 aa6d aa19 5d6c  ...........m..]l
        0x0020:  5010 7fff 4730 0000                      P...G0..
19:50:01.613457 IP localhost.36561 > localhost.telnet: P 86:112(26) ack 158 win 32767
        0x0000:  4510 0042 0378 4000 4006 392c 7f00 0001  E..B.x@.@.9,....
        0x0010:  7f00 0001 8ed1 0017 a9c6 aa6d aa19 5d6c  ...........m..]l
        0x0020:  5018 7fff a32a 0000 7665 6e74 2062 7572  P....*..vent.bur
        0x0030:  6720 6a69 6265 206f 7220 6a6f 686e 2066  g.jibe.or.john.f
        0x0040:  6f72                                     or
```

Faiblesses du système

Quelle est la robustesse de ce système face aux différentes attaques ? OPIE présente potentiellement deux défauts. Premièrement, il est possible de savoir si un compte existe ou pas [13]. Si le login est inexistant dans la base OPIE, alors la graine et le numéro de séquence du challenge changent à chaque tentative. En revanche, pour un compte existant, ces paramètres restent inchangés.

Deuxièmement, du fait de l'invariance du challenge, le comportement précédent permettrait de procéder à une attaque de type *brut force* sur des comptes existants. Ce type d'attaque peut être contré en imposant, par exemple, un nombre maximal de tentatives infructueuses au terme desquelles le client ne peut plus se connecter pendant un certain temps.

IV. Intégrer OPIE dans une application

Le paquetage `libopie-dev` aide à l'ajout d'une couche d'authentification OTP en quelques lignes de code (comparativement, une authentification maison avec du MD5 demande trois fois plus de lignes de code avec `libgcrypt-dev`).

Pour des questions de place, le mot de passe ainsi que son hash ont été câblés directement dans le code, ce qu'il ne faut **jamais** faire !

NOTE : Ces codes sources sont donnés à titre éducatif et l'auteur décline toutes responsabilités découlant de son usage.

Compiler avec : `gcc -lopie auth-opie.c -o auth-opie`

```
1 #include <stdio.h>
2 #include <opie.h>
3
4 main(void)
5 {
6         char *user_name;

7         struct opie opiedata;
8         char opieprompt[OPIE_CHALLENGE_MAX+1];
9         char typed_password[OPIE_RESPONSE_MAX+1];

10
11        user_name = "gfasoli";
12
13        opiechallenge(&opiedata, user_name, opieprompt);

14
15        if (opieverify(&opiedata, typed_password)) {
16                fprintf (stderr, "Login incorrect.\n");

17                return 1;
18        } else {
19                fprintf (stdout, "User Authenticated.\n");

20                return 0;
21        }
22
23        return 0;

24 }
```

V. Les systèmes à boîtiers

Le tour d'horizon des systèmes OTP ne serait pas complet sans mentionner les fameuses solutions à « calculette ». Celles-ci reposent sur l'utilisation d'une petite calculatrice servant à générer le mot de passe à usage unique.

Il en existe de deux sortes :

- **Asynchrone :** le serveur transmet un challenge à l'utilisateur. Il le rentre dans son token qui lui fournit alors la réponse à retourner au serveur.
- **Synchrone :** comme son nom l'indique, cette méthode repose sur la synchronisation du token et du serveur (en fonction de l'heure ou d'un compteur par exemple). La calculatrice fournit automatiquement à l'utilisateur un mot de passe valide (mais toujours à usage unique) pour s'authentifier. La plupart du temps, le mot de passe est systématiquement recalculé toutes les 60 secondes.

À titre d'exemple, on peut notamment citer ici SecurID, une solution synchrone commercialisée par RSASecurity ou la gamme de produits synchrones/asynchrones de la société ActivCard SA.

Pour SecurID, il faut compter 3500 euros pour des tokens sans clavier utilisables trois ans,et 6500 euros pour des tokens avec pinpad (où l'utilisateur rentre un code PIN sur le clavier).

Pour la solution Activcard, le coût est d'environ 3500 euros pour des tokens, avec ou sans clavier.

Ces prix, donnés à titre purement indicatif, comprennent les licences et les tokens pour 25 utilisateurs, mais pas la maintenance.

En conclusion

Nous avons vu que ce système à mots de passe jetables est facile à déployer. OPIE offre un niveau de sécurité plus élevé que les classiques mots de passe dans des environnements peu sûrs. De ce fait, il est pratique de l'utiliser en complément des systèmes d'authentification en place pour réduire le risque de compromission du mot de passe (par un keylogger par exemple).

De plus, l'API pour interagir avec OPIE s'intègre avec un très faible coût de développement.

Références

[1] G. S. Vernam. *Cipher printing telegraph systems for secret wire and radio telegraphic communications*, 1926.

[2] Y. Z. Ding et M. O. Rabin. *Hyper-Encryption and Everlasting Security*, STACS 2002. LNCS 2285 1-26.

[3] N. M. Haller. *The S/KEY One Time Password System*, Bellcore 1994.

[4] RFC 2289, *A One-Time Password System*.

[5] Cost dictates security plans. **http://www.vnunet.com/features/1156593**

[6] VeriSign Introduces Collaborative Vision to Drive Ubiquitous Adoption of Strong Authentication Solutions. **http://www.verisign.com/corporate/news/2004/pr_20040223b.html**

[7] VeriSign Delivers Roadmap For Strong Authentication Services Built On The Microsoft Windows Environment. **http://www.verisign.com/corporate/news/2004/pr_20040225.html**

[8] One Time Passwords in Everything (OPIE). **http://inner.net/opie**

[9] WinKey. **http://inner.net/pub/opie/contrib/WinKey.exe**

[10] S/KEY for HP48. **http://web.baz.org/~adam/otp/skey-hp48.tar.gz**

[11] pilOTP. **http://www.mirrors.wiretapped.net/security/authentication/otp/palm/**

[12] OTPC. **http://www.basun.net/software/otpc/**

[13] OpenSSH & S/Key information leakage. **http://archives.neohapsis.com/archives/bugtraq/2001-11/0113.html**

PKCS #11 et authentification

Un dossier dans MISC sur l'authentification ne peut pas passer sous silence le cas des cartes à puce. Et il est difficile de parler de l'utilisation de cartes à puce comme fournisseur de services cryptographiques sans parler de l'API qui a été définie pour ça : PKCS #11.

Historique

L'algorithme RSA est publié en février 1978 dans la revue scientifique Communications of the ACM [1]. En 1982, la société RSA Security est créée par les inventeurs de RSA pour valoriser le brevet US4405829 Cryptographic communications system and method déposé en décembre 1977 par le MIT (Massachusetts Institute of Technology). L'idée étant déjà publique, car décrite dans l'article [1], le brevet n'a pas pu être déposé ailleurs qu'aux États-Unis.

Les RSA Labs sont le centre de recherche de RSA Security. Une partie du travail de recherche est de proposer des standards tels que les PKCS : Public-Key Cryptography Standards ou standards de cryptographie à clé publique, afin de développer l'utilisation des algorithmes à clé publique, donc de RSA, et donc, que les fournisseurs de solutions utilisant RSA payent des droits de licence à RSA Security. Les documents PKCS ont été publiés en 1991 pour la première fois. Des contributions issues des PKCS ont été intégrées dans de nombreux standards de fait ou officiels comme les documents ANSI X9, PKIX, SET, S/MIME et SSL.

Le brevet sur RSA a expiré le 21 septembre 2000.

Famille PKCS

Les standards PKCS sont librement accessibles sur le site Web [2] de RSA Security.

Tableau 1

PKCS #1	RSA Cryptography Standard
PKCS #3	Diffie-Hellman Key Agreement Standard
PKCS #5	Password-Based Cryptography Standard
PKCS #6	Extended-Certificate Syntax Standard
PKCS #7	Cryptographic Message Syntax Standard
PKCS #8	Private-Key Information Syntax Standard
PKCS #9	Selected Attribute Types
PKCS #10	Certification Request Syntax Standard
PKCS #11	Cryptographic Token Interface Standard
PKCS #12	Personal Information Exchange Syntax Standard
PKCS #13	Elliptic Curve Cryptography Standard
PKCS #15	Cryptographic Token Information Format Standard

Note : PKCS #2 et PKCS #4 sont maintenant incorporés dans PKCS #1.

Les documents PKCS les plus connus sont :

- **PKCS #1 :** décrit l'utilisation de l'algorithme RSA et en particulier les algorithmes de bourrage (padding) à utiliser.
- **PKCS #7 :** décrit le format de messages cryptographiques utilisés par exemple dans PEM : Privacy-Enhanced Mail (RFC 1421).
- **PKCS #10 :** décrit le format des demandes de certificats à faire auprès d'une autorité de certification.
- **PKCS #12 :** décrit le format de transport et d'échange de certificats, clés privées. C'est le format utilisé par Mozilla ou Internet Explorer pour exporter/importer des certificats dans des fichiers `*.p12`.

PKCS #11

PKCS #11, que nous allons étudier dans le présent article, décrit les fonctions d'accès à un ou plusieurs Cryptographic Token ou jeton cryptographique.

L'idée de PKCS #11 est de fournir une API (Application Programming Interface) qui puisse fournir des services cryptographiques de base, mais sans dépendre d'une solution particulière. Ces services cryptographiques de base peuvent être fournis par du logiciel ou par un composant matériel comme une carte PCMCIA ou une carte à puce. Cette API est aussi appelée Cryptoki.

PKCS #11 ne réalise donc pas directement de service d'authentification, mais peut être utilisé pour en fournir un, par exemple à travers un module d'authentification PAM (Pluggable Autentication Module).

Microsoft ne voulant, comme d'habitude, pas suivre un standard défini par un autre, a inventé sa propre API cryptographique. L'équivalent Microsoft de Cryptoki (PKCS #11) est CAPI (Crypto API).

Architecture de PKCS #11

Modèle général

Le modèle général de Cryptoki est décrit sur la figure 1. Les parties Cryptoki sont en jaune.

Les objets

Cryptoki (PKCS #11) définit 3 classes d'objets (voir figure 2) :

- **donnée :** objet défini par l'application
- **certificat :** contient un certificat
- **clé :** contient une clé qui peut être de 3 types (publique, privée ou secrète)

Cryptoki permet à une application d'accéder à des objets. Les objets sont de deux types :

Georges Bart <georges.bart@free.fr>

Figure 1

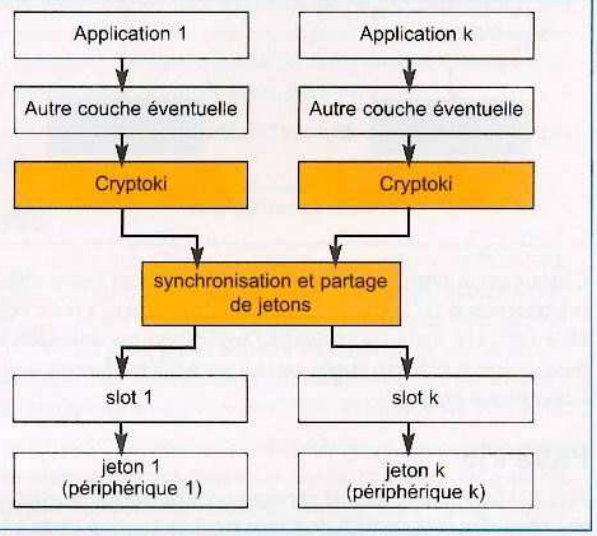
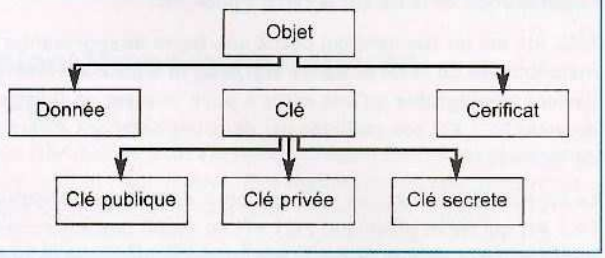


Figure 2



- **token :** ces objets sont visibles depuis toutes les applications connectées. Ils sont persistants dans le jeton ;
- **session :** ces objets sont temporaires et visibles uniquement depuis l'application qui les a créés.

Une application qui utilise la bibliothèque PKCS #11 sait de quels objets elle a besoin. PKCS #11 lui donne la possibilité de créer, copier, rechercher et détruire des objets sur le jeton. PKCS #11 ne crée pas d'objets pour ses besoins internes. Les applications manipulent leurs objets à travers PKCS #11.

Les utilisateurs

Les objets sont accessibles par trois utilisateurs différents :

- **objets publics :** ils sont accessibles sans authentification préalable ;
- **objets utilisateurs :** ils sont accessibles après authentification de l'utilisateur du jeton ;
- **objets de l'administrateur :** ils sont accessibles après authentification de l'administrateur du jeton.

On peut remarquer qu'un jeton n'a qu'un seul utilisateur. En effet, un jeton est personnel et ne peut donc être associé qu'à un seul utilisateur.

Les fonctions

Cryptoki fournit des fonctions pour différents usages, fonctions présentées dans le tableau ci-dessous.

Tableau 2

générales	C_Initialize, C_Finalize, C_GetInfo, C_GetFunctionList
gestion de slot	C_GetSlotList, C_GetSlotInfo, C_WaitForSlotEvent
gestion de jeton (token)	C_GetTokenInfo, C_InitToken, C_InitPIN, etc.
gestion de session	C_OpenSession, C_CloseSession, C_Login, C_Logout, C_GetSessionInfo, etc
gestion d'objets	C_CreateObject, C_CopyObject, C_DestroyObject, C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal, etc.
chiffrement/déchiffrement	C_EncryptInit, C_Encrypt, C_EncryptUpdate, C_EncryptFinal, C_DecryptInit, etc.
hachage	C_DigestInit, C_Digest, C_DigestUpdate, C_DigestKey, C_DigestFinal
signature et MAC	C_SignInit, C_Sign, C_SignUpdate, C_SignFinal, C_SignRecoverInit, etc.
vérification de signature	C_VerifyInit, C_Verify, etc.
fonctions à deux étapes	C_DigestEncryptUpdate, C_SignEncryptUpdate, C_DecryptVerifyUpdate, etc.
gestion des clés	C_GenerateKey, C_GenerateKeyPair, C_WrapKey, C_DeriveKey, etc.
génération d'aléas	C_SeedRandom, C_GenerateRandom

Une implémentation de Cryptoki n'a pas à fournir toutes les fonctions, ni à supporter tous les algorithmes possibles. Une fonction peut retourner l'erreur CKR_FUNCTION_NOT_SUPPORTED pour indiquer que la fonction n'est pas supportée.

PKCS #11 et la carte à puce

Les fonctions fournies par Cryptoki sont des fonctions cryptographiques très générales. Une carte à puce ayant été conçue pour stocker des clés cryptographiques et les utiliser de façon sûre (sans les sortir de la carte), c'est un jeton tout trouvé pour fournir les fonctions PKCS #11.

La carte à puce ne fournit pas une interface PKCS #11. Il faut donc écrire une couche logicielle qui convertit un appel PKCS #11 en une ou plusieurs commandes pour la carte à puce. L'architecture générale d'une bibliothèque PKCS #11 pour carte à puce est décrite sur la figure 3.

Au lecteur de carte est associé un pilote spécifique. PC/SC Lite est une infrastructure qui fournit une interface commune PC/SC (Personnal Computer / Smart Card) quel que soit le lecteur utilisé. PC/SC Lite [3] est une implantation en logiciel libre de cette interface appelée WinSCard.

Les cartes à puce sont définies par les normes ISO 7816-1 à 7816-15. En réalité, seules les trois premières spécifications sont vraiment utilisées. Les autres parties sont un peu ou pas du tout utilisées.

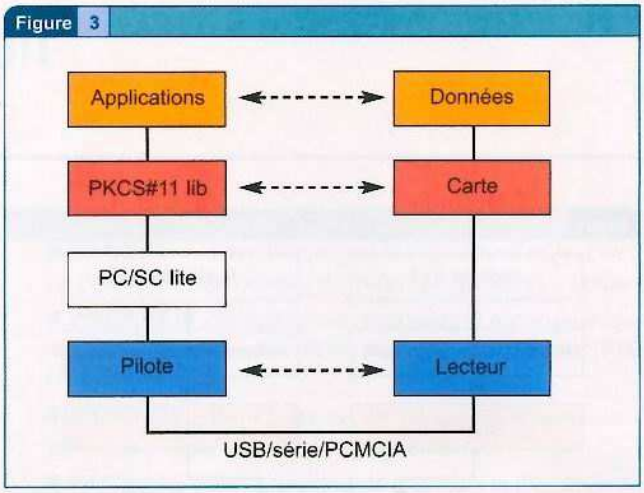
Tableau 3

Partie 1	Caractéristiques physiques
Partie 2	Cartes à contact - Dimensions et emplacements des contacts
Partie 3	Signaux électroniques et protocoles de transmission
Partie 4	Commandes intersectorielles pour les échanges
Partie 5	Système de numérotation et procédure d'enregistrement d'identificateurs d'applications
Partie 6	Éléments de données intersectoriels pour les échanges
Partie 7	Commandes intersectorielles pour langage d'interrogation de carte structurée (SCQL)
Partie 8	Commandes pour des opérations de sécurité
Partie 9	Commandes pour la gestion des cartes
Partie 10	Signaux électroniques et réponse à la mise à zéro des cartes synchrones
Partie 11	Vérification personnelle par méthodes biométriques
Partie 15	Application des informations cryptographiques

Les titres des documents sont les titres officiels de la version française de la norme ISO. Je n'ai rien fait pour les rendre encore plus abscons.

L'utilisation d'un sous-ensemble seulement des spécifications est due au fait que toutes les cartes n'ont pas les mêmes fonctions, ni le même but. Les parties 1 à 3 définissent le minimum pour qu'une carte à puce communique avec un lecteur de carte à puce. Ensuite, chaque carte peut suivre ou non les autres spécifications. Par exemple, les cartes à puce Java Card [4] suivent les parties 1 à 3, mais sont complètement différentes au niveau du jeu de commandes et des possibilités.

Figure 3



Chaque carte ayant un jeu de commandes plus ou moins différent, la bibliothèque PKCS #11 est dépendante de la carte à puce utilisée. Mais PKCS #11 étant un standard, l'application qui utilise PKCS #11 peut utiliser n'importe quelle carte à puce qui est fournie avec une bibliothèque PKCS #11.

PKCS #15

PKCS #11 fournit une liste de services cryptographiques, mais ne dit pas comment une clé RSA doit être stockée sur une carte à puce. Chaque fournisseur d'une bibliothèque PKCS #11 peut donc utiliser un format différent pour indiquer la taille du module de la clé RSA, l'identification de la clé sur la carte à puce, etc.

PKCS #15 est un standard qui décrit une façon de représenter les métadonnées de la carte à puce sur la carte à puce elle-même. Il devient envisageable qu'une carte à puce utilisant un format de données PKCS #15 soit utilisable par deux implantations différentes de PKCS #11.

Le support de PKCS #15 ne suffit pas pour définir une bibliothèque PKCS #11 qui serait générique. PKCS #15 ne définit que le format des données. Il faut ensuite définir le format des commandes (la norme ISO 7816 partie 4, mais pas uniquement).

Il est envisageable d'avoir une librairie PKCS #11 qui ne soit pas liée à un modèle de carte à puce, mais qui puisse fonctionner avec plusieurs cartes à puce de fabricants différents, mais qui partagent une base commune.

Implantations libres de PKCS #11

Mozilla et dérivés

Mozilla contient une implantation d'un jeton cryptographique logiciel. C'est ce module qui est utilisé lorsque vous utilisez les fonctions de chiffrement et signature de messages électroniques ou d'authentification du client Web. À ma connaissance, cette implantation n'est pas disponible en tant que bibliothèque indépendante.

L'avantage d'avoir choisi l'API PKCS #11 est qu'il est facile de remplacer la gestion uniquement logicielle de Mozilla par une bibliothèque utilisant une carte à puce. Vous pouvez alors signer vos messages électroniques avec une clé privée qui ne sort pas de votre carte à puce.

MuscleCard

M.U.S.C.L.E.
MOVEMENT FOR THE USE OF SMART CARDS IN A LINUX ENVIRONMENT

Le projet MUSCLE [5] (Movement for the Use of Smart Cards in a Linux Environment - mouvement pour l'utilisation des cartes à puce en environnement Linux), qui fournit déjà PC/SC Lite, offre aussi une bibliothèque PKCS #11. Afin de ne pas dépendre d'une carte à puce en particulier, une applet Java Card a été développée et ainsi (presque) n'importe quelle carte à puce Java Card qui peut utiliser l'applet peut servir de jeton PKCS #11.

Le projet PKCS #11 de MuscleCard est intéressant, mais n'est pas très actif.

OpenSC

Un autre groupe a pris une voie différente. Plutôt que d'utiliser des cartes Java Card encore assez chères, ils ont écrit une bibliothèque qui permet de supporter un assez grand nombre de modèles de cartes à puce non Java Card, mais fournissant le jeu de commandes nécessaire à l'utilisation des cartes à puce avec PKCS #15 présenté plus haut.

Le projet OpenSC [6] fournit un ensemble de programmes en ligne de commande permettant d'initialiser une carte à puce, de l'explorer, de manipuler les données stockées, calculer des cryptogrammes, les vérifier, etc. Le projet fournit aussi une bibliothèque PKCS #11.

C'est avec les outils de ce projet que nous allons travailler.

Implantations propriétaires de PKCS #11

Chaque fabricant de jetons cryptographiques fournit, en général, le module PKCS #11 qui va avec. Ces implantations sont quasiment toujours fournies uniquement pour Microsoft Windows.

Nous ne nous y attarderons pas, mais sachez que ces librairies fonctionnent de la même façon et fournissent les mêmes services.

Utilisations

À quoi peut bien servir un jeton cryptographique utilisable à travers une bibliothèque PKCS #11 ?

Mozilla

Comme nous l'avons déjà vu précédemment, Mozilla utilise PKCS #11 comme interface de base pour ses fonctions cryptographiques. L'avantage est que si la version interne et logicielle fournie par Mozilla ne vous convient pas, parce que vous voulez une authentification des courriers électroniques plus forte par exemple, vous pouvez très facilement utiliser un autre fournisseur (bibliothèque) de services cryptographiques (PKCS #11).

PAM

Une autre utilisation est l'authentification locale sur une machine, en remplaçant le classique login/mot de passe par une authentification beaucoup plus forte.

Exemple d'utilisation de PKCS #11 : authentification PAM à carte à puce

Nous allons utiliser les logiciels fournis par le projet OpenSC [6]. La version stable actuelle de OpenSC est la 0.8.1, mais une nouvelle version majeure 0.9.1 est prévue pour bientôt.

Je vous conseille d'attendre la version 0.9.1 finale ou d'utiliser la version 0.9.1 bêta déjà disponible. Cette nouvelle version est/sera une grande amélioration et devrait vous éviter beaucoup de désagréments.

Configuration de la carte à puce

La première étape est de récupérer une carte à puce et de la configurer pour l'usage que l'on veut en faire.

Structure de fichiers

On commence par créer une structure de fichiers PKCS #15 sur la carte.

```
$ pkcs15-init -verbose -create-pkcs15 -so-pin 567890 -so-puk
098765 -use-default-transport-keys
Connecting to card in reader AseIIIeUSB 00 00...
Using card driver GPK driver.
About to create PKCS #15 meta structure.
```

→ `--verbose` donne un affichage des traitements en cours. Vous pouvez dupliquer l'argument pour avoir encore plus d'informations sur les traitements en cours.

→ `--create-pkcs15` crée la structure PKCS #15 sur la carte et permet d'initialiser les PIN (Personnal Identification Numbers ou code secrets).

→ `--so-pin 567890` indique que le code secret (PIN) du responsable sécurité (SO pour Security Officer) est 567890. Il est bien sûr possible de ne pas donner le code secret en ligne de commande. La commande va alors demander de l'entrer au clavier.

→ `--so-puk 098765` indique que le code de déblocage (PUK pour PIN Unlock Code) du code secret du responsable sécurité est 098765. Ce code secret est utilisé si le code secret du responsable sécurité est bloqué, par exemple après trois essais infructueux. Ce code de déblocage doit être gardé encore plus secret que le code secret du responsable sécurité.

→ `--use-default-transport-keys` indique que le programme doit utiliser les clés de transport par défaut.

Lorsqu'une carte à puce sort de l'usine, elle est soit déjà personnalisée avec les clés et certificats qui vont bien et ceux-ci sont protégés par le code secret de l'utilisateur et du responsable sécurité, soit la carte à puce est vierge (c'est le cas ici) et l'accès à la carte est protégé par une clé de transport.

Avant de pouvoir créer des fichiers sur la carte, il faut d'abord que le monde extérieur s'authentifie auprès de la carte par une commande External Authenticate. La carte à puce génère un nombre aléatoire et l'extérieur (le PC) doit chiffrer ce nombre avec la clé de transport et le renvoyer à la carte à puce.

Si le cryptogramme est correct, la carte à puce autorise la création de fichiers.

Code secret utilisateur

Il faut ensuite créer un code secret pour l'utilisateur de la carte à puce. Ce code est, en général, différent du code du responsable sécurité et connu uniquement de l'utilisateur.

```
$ pkcs15-init -verbose -store-pin -auth-id 01 -label 'User PIN' -pin 1234 -puk
4321 -so-pin 567890
Connecting to card in reader AseIIIeUSB 00 00...
Using card driver GPK driver.
Found OpenSC Card
About to store PIN.
```

→ `--store-pin` indique l'action désirée.

→ `--auth-id 01` indique l'identité (au sens de la carte) qui est protégée par ce code secret.

→ `--label 'User PIN'` est une description du code secret.

→ `--pin 1234` est la valeur (très originale) du code secret.

→ `--puk 4321` est la valeur du code de déblocage du code secret. Comme pour le code secret du responsable sécurité, l'utilisateur peut arriver à bloquer son code. Plutôt que de jeter la carte bloquée et d'en générer une nouvelle, il est possible de débloquer le code de l'utilisateur.

→ `--so-pin 567890` est nécessaire pour que la carte autorise la création de nouveaux objets.
L'autorisation n'est maintenant plus conditionnée à la connaissance de la clé de transport, mais à la connaissance du code secret du responsable sécurité.

Génération d'une paire de clés RSA

Un des avantages d'une carte à puce est la génération de clés à bord de la carte (toutes les cartes à puce ne le permettent pas). La paire de clés RSA est générée par la carte et la clé privée ne quitte jamais la carte à puce. Les droits d'accès de la clé privée sont tels que l'utilisation de la clé (pour signer ou chiffrer) est possible après authentification avec le code secret de l'utilisateur, mais l'exportation de la clé hors de la carte à puce est, normalement, impossible (à ce sujet voir [7, 8]).

```
$ pkcs15-init -verbose -generate-key rsa/1024 -auth-id 01 -label 'PAM key' -pin
1234 -so-pin 567890
Connecting to card in reader AseIIIeUSB 00 00...
Using card driver GPK driver.
Found OpenSC Card
About to generate key.
```

→ `--generate-key rsa/1024` : nous voulons une paire de clés de type RSA et de taille de module 1024 bits.

→ `--auth-id 01` : la clé sera protégée par l'identité 01.

→ `--pin 1234` : code secret de l'utilisateur.
Ce code secret est nécessaire puisque la clé privée sera stockée sur la carte à puce dans un fichier qui est protégé par le code secret de l'utilisateur.

→ `--so-pin 567890` : code secret du responsable sécurité.
Ce code secret est nécessaire puisque la commande modifie le système de fichiers pour indiquer qu'une paire de clés est maintenant présente sur la carte à puce.

Identification de la carte à puce

Il faut pouvoir associer la carte à puce à un utilisateur. Pour notre utilisation (authentification PAM) nous avons besoin de stocker l'identifiant de connexion, ici `gbart`.

```
$ echo -n gbart > fichier_temporaire
$ pkcs15-init -verbose -store-data fichier_temporaire -label 'PAM username' -so-
pin 567890
Connecting to card in reader AseIIIeUSB 00 00...
Using card driver GPK driver.
Found OpenSC Card
About to store data object.
```

Liste des objets présents sur la carte

On va maintenant vérifier l'état de la carte à puce.

```
$ pkcs15-tool -verbose -list-public-keys -list-keys -list-certificates -list-
data-objects
Connecting to card in reader AseIIIeUSB 00 00...
Using card driver GPK driver.
Trying to find a PKCS#15 compatible card...
Found OpenSC Card!
Card has 0 certificate(s).

Reading data object <0>, label = 'PAM username'
Data Object (5 bytes): < 67 62 61 72 74 >
Card has 1 private key(s).

Private RSA Key [PAM key]
        Com. Flags  : 3
        Usage       : [0x4], sign
        Access Flags: [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength   : 1024
        Key ref     : 0
        Native      : yes
        Path        : 3F005015045
        Auth ID     : 01
        ID          : 45

Card has 1 public key(s).

Public RSA Key [Public Key]
        Com. Flags  : 2
        Usage       : [0x4], sign
        Access Flags: [0x0]
        ModLength   : 1024
        Key ref     : 0
        Native      : no
        Path        : 3F005015048
        Auth ID     :
        ID          : 45
```

Le chemin (Path) `3F005015045` correspond au nom du fichier dans le système de fichier de la carte à puce.

→ `3F00` est le nom de la racine (Meta File). C'est l'équivalent de / sur un système de fichier Unix.

→ `5015` est le nom d'un répertoire (Dedicated File). Ce nom est spécifié par `PKCS #15`.

→ `3045` est le nom du fichier (Elementary File).

On peut remarquer aussi que les deux clés n'ont pas les mêmes conditions d'accès. La clé publique peut être lue et utilisée en accès libre. Par contre, la clé privée ne peut être utilisée (pour signer) qu'après authentification de l'identité 01 et ne peut pas être extraite de la carte à puce.

Récupération de la clé publique

OpenSC permet de récupérer la clé publique au format PEM.

```
$ pkcs15-tool -read-public-key 45 > ~/.pam/pk.pem
$ openssl rsa -inform PEM -in ~/.pam/pk.pem -pubin -text
Modulus (1024 bit):
```

```
    00:be:5e:be:c9:a5:a9:77:22:ef:25:67:14:a0:94:
    fb:1c:ab:96:54:90:b2:6c:8b:13:18:ed:b5:e3:67:
    c7:d0:f2:56:9c:07:1c:9c:15:29:e8:43:53:62:9f:
    0e:d9:cd:f2:81:89:df:d1:f1:02:37:e0:8a:b8:ca:
    9b:c9:39:81:00:2d:2c:eb:d7:dc:00:38:ce:b1:bc:
    0a:96:d9:7c:b1:05:94:05:ff:cb:24:7e:f1:dd:ee:
    f8:97:a6:c2:c7:89:7e:6f:df:00:5b:88:51:b1:49:
    de:3b:8d:01:5b:d2:f6:2a:64:43:52:a3:c7:4a:5d:
    71:cd:8c:a7:2e:fa:d4:9b:79
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+Xr7Jpal3Iu8lZxSglPscq5ZU
kLJsixMY7bXjZ8fQ8lacBxycFSnoQ1Ninw7ZzfKBid/R8QI34Iq4ypvJOYGALSzr
19wAOM6xvAqW2XyxBZQF/BskfvHd7viXpsLHiX5v3wBbiFGxSd47jQFb0vYqZENS
o8dKXXHNjKcu+tSbeQIDAQAB
-----END PUBLIC KEY-----
```

Le module a bien une taille de 1024 bits. L'exposant public est $2^{16}+1$. C'est une valeur classique d'exposant public, car c'est un nombre premier petit et creux, c'est-à-dire qu'il n'a pas de bit à 1 hormis le bit de poids fort et le bit de poids faible. Cet exposant permet d'avoir des calculs de vérification de signature très rapides.

Utilisation de PKCS #11

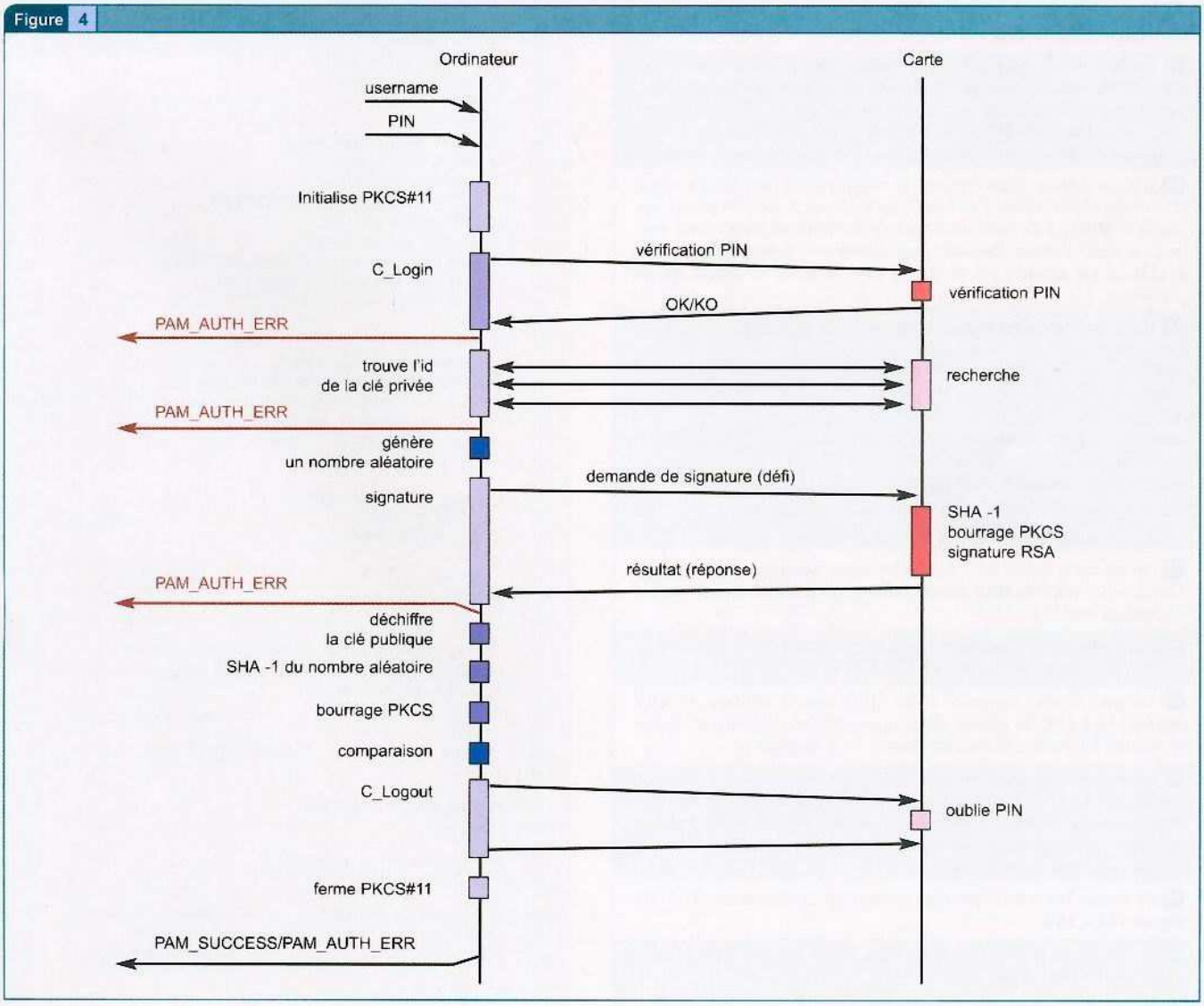
Il nous faut maintenant écrire un bout de code qui va utiliser PKCS #11 pour réaliser une authentification. Le projet OpenSC fournit déjà un module PAM utilisant la carte à puce. Ce module permet d'utiliser des certificats stockés localement dans le fichier `~/.eid/authorized_certificates` de l'utilisateur ou bien de récupérer les certificats dans une base LDAP. Ce module PAM n'utilise pas PKCS #11, mais directement des fonctions fournies par la bibliothèque `libopensc.so`.

Cet article étant sur PKCS #11, nous allons développer nous-même un module PAM minimal utilisant PKCS #11.

Algorithme

Nous allons utiliser un algorithme d'authentification extrêmement simple. L'ordinateur choisit un nombre aléatoire, le fait chiffrer par la clé privée de la carte à puce et vérifie que le chiffrement est correct en utilisant la clé publique correspondante. Une description plus détaillée est donnée dans la figure 4 ci-dessous.

Figure 4



1 Le module PAM reçoit le nom de login (username) en paramètre (**lignes 175 à 177**).

2 À l'aide des fonctions de conversation de PAM, le module demande à afficher un message spécifique « Entrez votre PIN: » à la place du classique « Password: » (**lignes 178 à 182**).

3 Le code gestion de PKCS #11 est isolé dans la fonction pkcs11_authenticate() appelée **ligne 183**. En fonction du résultat de cette fonction, le module PAM renvoie PAM_SUCCESS ou PAM_AUTH_ERR.

4 Le module initialise la bibliothèque PKCS #11 et récupère un accès aux fonctions fournies (**lignes 47 à 74**).

5 Le module utilise C_Login() pour authentifier l'utilisateur auprès de la carte à puce (**lignes 75 à 77**). Si le code secret (PIN) de l'utilisateur n'est pas correct, la carte rend un message d'erreur et le module PAM retourne un code d'échec d'authentification PAM_AUTH_ERR.

6 Le module va ensuite chercher dans les objets PKCS #11 de la carte celui qui contient la clé privée d'authentification (**lignes 78 à 89**). Dans le programme d'exemple, la recherche est faite sur le nom de cette clé PAM key. Ici aussi, si le module PAM ne trouve pas la clé recherchée, il retourne un code d'échec.

7 Le module génère un nombre aléatoire de 128 bits en utilisant le générateur du noyau Linux : fichier spécial /dev/urandom (**lignes 90 à 98**).

8 Le module demande à la carte de hacher et signer ce message (le nombre aléatoire) (lignes **99 à 103**).

9 L'algorithme (mécanisme au sens PKCS #11) utilisé est CKM_SHA1_RSA_PKCS. Le message est d'abord haché avec l'algorithme SHA-1, puis des octets de bourrage sont ajoutés pour passer de 20 octets (la sortie du SHA-1, 160 bits) à 128 octets (l'entrée du RSA, 1024 bits). Ce bourrage est spécifié dans PKCS #1. La carte à puce renvoie la signature obtenue.

10 Si le calcul de signature s'est mal déroulé, la carte a renvoyé une erreur et le module PAM retourne un code d'échec. Sinon, il déchiffre le message avec la clé publique de l'utilisateur. Pour ne pas écrire un code source trop long, j'ai choisi d'utiliser OpenSSL pour déchiffrer le message (**lignes 104 à 127**). La clé publique est récupérée dans le fichier ~/.pam/pk.pem de l'utilisateur.

11 Nous avons maintenant un message déchiffré de la forme :

```
0000000: 0001 FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
0000010: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
0000020: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
0000030: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
0000040: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
0000050: FFFF FFFF FFFF FFFF FFFF FFFF 0030 2130  .............0!0
0000060: 0906 052B 0E03 021A 0500 0414 A941 EC12  ...+.........A..
0000070: 5424 BEDA 159C AD6A 3EB2 1958 7832 0716  T$.....j>..Xx2..
```

Il faut vérifier que ce message est bien celui attendu.

12 On calcule le haché SHA-1 de nos 16 octets aléatoires (**lignes 128 à 129**). Les 20 octets résultant doivent correspondre aux 20 derniers octets du message déchiffré.

13 Il faut ensuite appliquer l'algorithme de bourrage PKCS #1, ce qui remplit les 108 premiers octets (**lignes 130 à 152**).

14 On peut ensuite comparer notre calcul avec le message déchiffré (**lignes 153 à 154**). Le résultat de cette comparaison donnera le résultat de réussite ou d'échec de l'authentification de l'utilisateur.

15 Le module se déconnecte de la bibliothèque PKCS #11 avec C_Logout() (**lignes 156 à 157**). Cette fonction permet principalement d'invalider l'authentification de l'utilisateur auprès de la carte à puce. Il ne faudrait pas qu'un autre programme puisse utiliser la clé privée sans s'authentifier à nouveau auprès de la carte à puce.

16 Le module libère les ressources allouées par la bibliothèque PKCS #11 (**lignes 158 à 160**).

17 Le résultat de la comparaison à l'étape 14 est renvoyé comme résultat du module PAM (**ligne 163**).

Code source

```
 1 #include <stdio.h>
 2 #include <dlfcn.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <fcntl.h>
 6 #include <unistd.h>
 7 #include <openssl/sha.h>

 8 /* définitions nécessaires à l'utilisation de pkcs11.h */
 9 #define CK_PTR *
10 #define CK_DEFINE_FUNCTION(returnType, name) \
11    returnType name
12 #define CK_DECLARE_FUNCTION(returnType, name) \
13    returnType name
14 #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
15    returnType (* name)
16 #define CK_CALLBACK_FUNCTION(returnType, name) \
17    returnType (* name)
18 #define NULL_PTR 0
19 #include «pkcs11.h»

20 /* macro personnelle pour tester le code de retour des fonctions PKCS #11 */
21 #define CHECK_RC(rc, f) \
22    if (rc != CKR_OK) { printf(f « failed: 0x%.8lX\n », rc); return -1; }

23 int pkcs11_authenticate(const char *username, char *pincode)
24 {
25    void *lib_handle;
26    CK_RV (*get_function_list)();
27    CK_FUNCTION_LIST *funcs;
28    CK_RV rc;
29    CK_INFO info;
30    CK_SESSION_HANDLE session_handle;
31    CK_SLOT_ID_PTR slot_ids;
32    CK_ULONG slot_count, ulCount;
33    CK_OBJECT_CLASS key_class = CKO_PRIVATE_KEY;
34    CK_BBOOL key_sign = CK_TRUE;
35    CK_ATTRIBUTE key_template[] = {
36       { CKA_CLASS, &key_class, sizeof(CK_OBJECT_CLASS) },
37       { CKA_SIGN, &key_sign, sizeof(key_sign) },
38       { CKA_LABEL, «PAM key», 7 }
39    };
40    CK_OBJECT_HANDLE key;
41    /* bourrage PKCS, SHA1 puis RSA */
42    CK_MECHANISM mechanism = { CKM_SHA1_RSA_PKCS, NULL, 0 };
43    unsigned char signature[128], clear[128], expected[128]; /* 1024 bits */
44    unsigned char random[16], sha1[20];
45    char openssl_cmd[200];
46    int fd, i, ret;

47    /* charge la bibliothèque PKCS #11 */
48    lib_handle = dlopen(«/usr/lib/pkcs11/opensc-pkcs11.so», RTLD_LAZY);
49    if (NULL == lib_handle)
50    {
51       perror(«dlopen»);
52       return -1;
53    }

54    /* récupère la fonction de récupération des fonctions */
55    get_function_list = (CK_RV (*)())dlsym(lib_handle , «C_GetFunctionList»);
56    if (NULL == get_function_list)
57    {
58       perror(«dlsym»);
59       return -1;
60    }

61    rc = get_function_list(&funcs);
62    CHECK_RC(rc, «C_GetFunctionList»)

63    rc = funcs->C_Initialize(NULL_PTR);
64    CHECK_RC(rc, «C_Initialize»)

65    rc = funcs->C_GetInfo(&info);
66    CHECK_RC(rc, «C_GetInfo»)
67    printf(«PKCS #11 manufacturer: %s\n», info.manufacturerID);
```

```
68    rc = funcs->C_GetSlotList(TRUE, NULL, &slot_count);
69    CHECK_RC(rc, «C_GetSlotList»)

70    slot_ids = calloc(slot_count, sizeof(CK_SLOT_ID));

71    rc = funcs->C_GetSlotList(TRUE, slot_ids, &slot_count);
72    CHECK_RC(rc, «C_GetSlotList»)

73    rc = funcs->C_OpenSession(0 /* slot 0 */, CKF_SERIAL_SESSION, NULL_PTR,
NULL, &session_handle);
74    CHECK_RC(rc, «C_OpenSession»)

75    /* login pour accéder aux clés privées */
76    rc = funcs->C_Login(session_handle, CKU_USER, pincode, strlen(pincode));
77    CHECK_RC(rc, «C_Login»)

78    /* on cherche une clé privée de signature */
79    rc = funcs->C_FindObjectsInit(session_handle, key_template, 3);
80    CHECK_RC(rc, «C_FindObjectsInit»)

81    rc = funcs->C_FindObjects(session_handle, &key, 1, &ulCount);
82    CHECK_RC(rc, «C_FindObjects»)

83    rc = funcs->C_FindObjectsFinal(session_handle);
84    CHECK_RC(rc, «C_FindObjectsFinal»)

85    if (0 == ulCount)
86    {
87       printf(«Pas de clé privée 'PAM key'\n»);
88       goto end;
89    }

90    /* génère 128 bits aléatoires */
91    fd = open(«/dev/urandom», O_RDONLY);
92    if (fd < 0)
93    {
94       perror(«open»);
95       goto end;
96    }
97    read(fd, random, 16);
98    close(fd);

99    rc = funcs->C_SignInit(session_handle, &mechanism, key);
100   CHECK_RC(rc, «C_SignInit»)

101   ulCount = sizeof(signature);
102   rc = funcs->C_Sign(session_handle, random, 16, signature, &ulCount);
103   CHECK_RC(rc, «C_Sign»)

104 #define CIPHER_FILENAME «/tmp/pam_result»
105   fd = open(CIPHER_FILENAME, O_WRONLY | O_CREAT, 0600);
106   if (fd < 0)
107   {
108      perror(«open»);
109      goto end;
110   }
111   write(fd, signature, 128);
112   close(fd);

113 #define CLEAR_FILENAME «/tmp/pam_clear»
114   snprintf(openssl_cmd, sizeof(openssl_cmd), «openssl rsautl -pubin -inkey»
115      «/home/%s/.pam/pk.pem  -in» CIPHER_FILENAME « -encrypt -raw -out »
116      CLEAR_FILENAME, username);
117   system(openssl_cmd);
118   unlink(CIPHER_FILENAME);

119   fd = open(CLEAR_FILENAME, O_RDONLY);
120   if (fd < 0)
121   {
122      perror(«open»);
123      goto end;
124   }
125   read(fd, clear, 128);
126   close(fd);
127   unlink(CLEAR_FILENAME);

128   /* hache le nombre aléatoire */
129   SHA1(random, 16, sha1);

130   /* calcule le résultat attendu : bourrage PKCS de SHA1(random) */
131   expected[0] = 0x00;
132   expected[1] = 0x01;
133   for (i=2; i<92; i++)
134      expected[i] = 0xFF;
135   expected[i++] = 0x00;   /* ces constantes sont définies dans PKCS #1 */
136   expected[i++] = 0x30;
137   expected[i++] = 0x21;
138   expected[i++] = 0x30;
139   expected[i++] = 0x09;
140   expected[i++] = 0x06;
141   expected[i++] = 0x05;
142   expected[i++] = 0x2B;
143   expected[i++] = 0x0E;
144   expected[i++] = 0x03;
145   expected[i++] = 0x02;
146   expected[i++] = 0x1A;
147   expected[i++] = 0x05;
148   expected[i++] = 0x00;
149   expected[i++] = 0x04;
150   expected[i++] = 0x14;
151   for (i=0; i<20; i++)
152      expected[i+108] = sha1[i];

153   /* authentification OK si clear et expected sont identiques */
154   ret = memcmp(clear, expected, 128);

155 end:
156   rc = funcs->C_Logout(session_handle);
157   CHECK_RC(rc, «C_Logout»)

158   rc = funcs->C_CloseSession(session_handle);
159   CHECK_RC(rc, «C_CloseSession»)

160   rc = funcs->C_Finalize(NULL_PTR);
161   CHECK_RC(rc, «C_Finalize»)

162   dlclose(lib_handle);

163   return ret;
164 }

165 #define PAM_SM_AUTH
166 #include <security/pam_modules.h>

167 PAM_EXTERN int pam_sm_authenticate(pam_handle_t * pamh, int flags, int argc,
168    const char **argv)
169 {
170    struct pam_conv *conv;
171    struct pam_message mesg = { PAM_PROMPT_ECHO_OFF, «Entrez votre PIN : »
};
172    const struct pam_message *mesg_p = &mesg;
173    struct pam_response *resp;
174    const char *username;

175    /* récupère le nom de login */
176    if (PAM_SUCCESS != pam_get_user(pamh, &username, NULL))
177       return PAM_CONV_ERR;

178    if (PAM_SUCCESS != pam_get_item(pamh, PAM_CONV, (const void **) &conv))
179       return PAM_CONV_ERR;

180    /* récupère le code secret */
181    if (PAM_SUCCESS != conv->conv(1, &mesg_p, &resp, conv->appdata_ptr))
182       return PAM_CONV_ERR;

183    if (0 == pkcs11_authenticate(username, resp->resp))
184       return PAM_SUCCESS;
185    else
186       return PAM_AUTH_ERR;
187 }
```

Améliorations possibles et souhaitables de l'exemple

La clé publique de l'utilisateur est récupérée dans un fichier sur son compte. Il est sûrement plus simple pour l'administrateur du système de stocker les clés publiques dans un annuaire LDAP centralisé.

Les messages d'erreur sont affichés en utilisant simplement `printf()`. Lorsque le module PAM est exécuté, il est plus utile d'envoyer les messages d'erreur à `syslog` ou équivalent pour que l'administrateur de la machine puisse récupérer ces messages d'erreur, diagnostiquer et résoudre le problème.

Le module PAM ne vérifie rien sur la clé publique. La clé étant publique, il n'est pas nécessaire d'en protéger l'accès en lecture. En revanche, il est (très) important d'en protéger l'accès en écriture. Si un attaquant réussit à remplacer la clé publique d'un utilisateur par une clé publique choisie par l'attaquant (et dont il connaît la clé privée correspondante), il peut créer une fausse carte à puce et réussir l'authentification.

L'utilisateur peut aussi vouloir changer sa propre clé publique par une autre dont il connaît la clé secrète correspondante. Ainsi, il peut lui-même se fabriquer plusieurs fausses cartes à puce et les donner à des complices.

L'utilisation de cartes à puce (non copiables) est alors mise en défaut. La solution est d'utiliser un certificat de la clé publique. Ce certificat permet au module PAM de vérifier que la clé publique est bien valide, qu'elle est bien associée à cet utilisateur et qu'elle n'a pas été modifiée par un attaquant. Je vous renvoie au dossier PKI d'un précédent numéro de MISC [9].

L'utilisation de fichiers temporaires (`/tmp/pam_result` et `/tmp/pam_clear`) pour communiquer avec le programme `openssl` n'est pas sécurisée. Les fichiers ont des noms fixes et vont écraser les fichiers précédemment existants et de même nom. Le module PAM étant exécuté avec les privilèges du super-utilisateur c'est très dangereux et donc à éviter.

Ce code source n'est qu'un exemple d'utilisation de `PKCS #11` pour illustrer son application à l'authentification système. Plutôt que d'essayer d'améliorer ce code source, je recommande aux lecteurs intéressés par une authentification PAM à carte à puce de regarder le projet `PKCS #11` *PAM Login Module* [10]. Ce projet résout les problèmes discutés ci-dessus.

Conclusion

Le standard `PKCS #11` définit une interface de programmation donnant accès à des fonctions cryptographiques de bas niveau. Son principal avantage est que l'API n'est pas spécifique à un jeton cryptographique particulier (carte à puce, clé USB, carte PCI ou PCMCIA, implantation logicielle). Une application utilisant cette API peut donc supporter un large éventail de jetons cryptographiques sans modification.

Nous avons utilisé quelques-unes des fonctions de cette API pour réaliser un (tout petit) module d'authentification PAM à titre d'exemple. L'API est beaucoup plus riche que ce que nous avons utilisé dans l'exemple.

L'API `PKCS #11` est aussi, en général, utilisée par les cartes matérielles « d'accélération cryptographique » qui permettent de décharger le processeur principal du serveur Web lors des calculs cryptographiques pour sécuriser les connexions Web avec SSL/TLS.

Il y a beaucoup d'autres aspects à traiter à propos des cartes à puce et de la sécurité. Certains ont déjà été traités, comme l'utilisation d'une carte SIM dans le réseau GSM [11]. D'autres le seront peut-être dans de prochains numéros de MISC. Toute suggestion d'article est la bienvenue.

Références

[1] «*A Method for Obtaining Digital Signatures and Public-Key Crypto-systems*», R. Rivest, A. Shamir, L. Adleman, Communications of the ACM, Vol. 21 (2), 1978, pages 120—126. Previously released as an MIT «Technical Memo» in April 1977.

[2] PKCS, **http://www.rsasecurity.com/rsalabs/node.asp?id=2124**

[3] PC/SC Lite, **http://pcsclite.alioth.debian.org/**

[4] Java Card Technology, **http://java.sun.com/products/javacard/**

[5] MUSCLE, **http://www.musclecard.com/**

[6] OpenSC, **http://www.opensc.org/**

[7] «*Sortir une clé RSA par la prise de courant*», Multi System & Internet Security Cookbook, n°7, mai-juin 2003

[8] «*Comment récupérer une clé privée RSA avec un marteau*», Multi System & Internet Security Cookbook, n°11, janvier-février 2004

[9] «*PKI, Public Key Infrastructure, ou comment organiser et gérer votre cryptographie*», Multi System & Internet Security Cookbook, n°13, mai-juin 2004

[10] «*PKCS #11 PAM Login Module*», Mario Strasser, 2003, **http://n.ethz.ch/student/mariost/pkcs11_login/**

[11] «*La sécurisation d'UMTS ou GSM 3e génération*», Multi System & Internet Security Cookbook, n°5, janvier-février 2003

Le protocole RADIUS

hb - <hb@rstack.org>
http://hb.rstack.org/
S. Dralet - <zg@kernsh.org)>

RADIUS est un monstre mythique sur lequel maintes histoires extraordinaires circulent depuis des années. Des mots de passe chiffrés par RADIUS aux mots de passe en clair, en passant par l'authentification RADIUS, tout ce qui pouvait être imaginé sur la bête a été dit. Et pourtant, à l'inverse du monstre du loch Ness, RADIUS existe, est normalisé, et mis en production. Ce qui confirme que d'une part, il est possible de savoir exactement ce que fait (ou devrait faire) votre serveur RADIUS, et d'autre part, que la plupart des gens ne savent pas ce qu'ils font.

Le protocole et son histoire

Qu'est-ce que RADIUS ?

RADIUS est l'acronyme de *Remote Authentication Dial-In User Service*. Une définition plus explicite serait « un protocole de transport de données d'authentification ». Normalisé en janvier 1997 dans la RFC 2058 [1], RADIUS avait essentiellement pour objectif de fournir aux FAI un moyen pour ne gérer qu'une seule base d'utilisateurs quel que soit le POP auquel ces derniers se connectaient. La principale fonction était par conséquent de transférer les informations d'authentification depuis les RAS (*Remote Access Servers*) à un mécanisme central, lui-même capable de s'interfacer avec un système d'authentification.

Si le protocole supportait dès l'origine les fonctions de transfert des informations réseau vers le poste client (adresse IP, masque de sous-réseau, etc.), le support du *roaming* via la mise en place de proxy et les capacités de *call-back*, il s'est également enrichi de fonctions d'*accounting* [2] concernant l'accès à certains services, du support d'IPv6 [3], et plus récemment encore, du support de EAP (*Extensible Authentication Protocol*) [4]. Actuellement dans sa seconde version, la dernière définition du protocole a été établie en Juin 2000 dans la RFC 2865 [5]. L'utilisation de RADIUS n'est donc plus limitée aux RAS, mais est désormais rendue possible sur l'ensemble des composants du système d'information servant de « porte d'entrée » tels que des firewalls, des routeurs, ou encore des Access Points, voire des commutateurs.

Enfin, RADIUS utilise UDP comme couche transport sur le port 1812. Le choix d'UDP est justifié dans les RFC [1] et [5] comme répondant à des critères techniques tels que la possibilité de passer de manière transparente d'un serveur à l'autre en cas d'indisponibilité ou l'inutilité de supporter l'*overhead* induit par TCP pour un protocole sans état.

Mode opératoire

Afin de comprendre le mode opératoire de RADIUS, il est nécessaire de distinguer trois acteurs : le poste utilisateur, le client et le serveur Radius :

- **le poste utilisateur** : il s'agit tout simplement de la station de travail, du PDA ou de tout composant informatique à partir duquel sera émis la requête d'authentification ;
- **le client RADIUS** : le client est le point d'accès au réseau (RAS, firewall, AP, etc.) ;
- **le serveur RADIUS** : c'est le point central où les clients transmettent les données d'authentification.

En complément de cette infrastructure, un serveur d'authentification (serveur autonome, PDC ou autre) est nécessaire pour effectuer l'opération d'authentification à proprement parler.

Dans un premier temps, l'utilisateur effectue une requête auprès du client RADIUS. La nature de cette requête est variable en fonction du service cible. Il peut s'agir d'un accès telnet, d'un protocole propriétaire ou d'un protocole tel que PPP. A l'exception de ce dernier cas, une invite spécifique à l'implémentation du client RADIUS sur le service en question sera transmise à l'utilisateur lui demandant ses informations d'authentification, login et mot de passe par exemple. Dans le cas de PPP, ces informations sont déjà contenues dans le paquet.

A partir de ces informations, le client RADIUS émet une requête de type `Access-Request` à destination du serveur et contenant les informations d'authentification. À la réception, le serveur valide l'ensemble des critères utiles de la requête, dont l'authentification effectuée auprès du serveur idoine. En cas d'échec, un paquet `Access-Reject` est transmis au client qui gère en fonction de son implémentation. Le succès, en revanche, est validé par un paquet `Access-Accept`, et également traité par le client comme bon lui semble.

Dans le cas d'une authentification par challenge, le serveur transmet au client un paquet de type `Access-Challenge` contenant le challenge, et optionnellement, un texte pouvant être affiché sur l'interface de l'utilisateur. La réponse est renvoyée par le client sous forme d'une nouvelle requête `Access-Request` et dont l'ID est différent de celui de la requête initiale. La suite du processus est identique au mécanisme décrit ci-dessus.

Les attributs

Un des principaux aspects de RADIUS est la richesse des informations potentiellement transmises entre le client et le serveur, voire jusqu'au poste utilisateur. Ces informations sont appelées attributs et sont positionnées à la fin du paquet selon le format suivant :

- **Type :** 1 octet, correspondant au type de données de l'attribut. Les valeurs sont définies dans la RFC 1700 [6].
- **Longueur :** 1 octet, taille (en octets) de l'attribut.
- **Attribut :** valeur de l'attribut à proprement parler.

La nature de ces informations est variable et dépend de la fonction du paquet RADIUS. Ainsi, si lors des phases d'authentification les

attributs `User-Name` et `User-Password` sont évidemment utilisés, d'autres peuvent être ajoutés, tels que `NAS-IP-Address` précisant l'adresse IP du point d'accès au réseau, `Callback-Number` qui précise le numéro de téléphone à rappeler ou encore `Framed-IP-Address` et `Framed-IP-Netmask` qui fournissent des paramètres IP à appliquer au poste utilisateur.

La norme définit plus de 40 attributs, enrichis depuis par des attributs spécifiques à Microsoft [7] ou d'autres améliorations génériques [8]. Cette structure ouverte fait la richesse de RADIUS et sa facilité d'adaptation à de nouveaux besoins tels que EAP.

La sécurité dans RADIUS

RADIUS est un des éléments de la chaîne d'authentification. Par conséquent, il est naturel que les problématiques de sécurité soient nativement intégrées à la norme. Cependant, il convient de garder à l'esprit que la première spécification du protocole date de 1997 et que par conséquent, ses mécanismes de sécurité ne peuvent plus être considérés aujourd'hui comme entièrement fiables.

Le secret partagé

Le principal élément de la sécurité RADIUS est l'existence d'un secret partagé entre le client et le serveur. Ce secret est utilisé pour les mécanismes de chiffrement/déchiffrement des informations ainsi que pour l'authentification et le contrôle d'intégrité de certains paquets. Selon la norme, un secret partagé distinct peut être défini pour chaque client.

Cependant, certaines implémentations ne permettent la définition que d'un seul secret pour l'ensemble des clients... La seule restriction imposée par la norme est l'interdiction du secret nul. Elle recommande néanmoins (le fameux SHOULD des RFC) d'utiliser un secret ayant le même niveau de complexité qu'un mot de passe et faisant plus de 16 caractères.

Authentification et intégrité

Comme nous l'avons vu, une opération de transfert d'informations d'authentification s'effectue en 2 étapes (4 en cas de challenge). Afin de garantir l'intégrité et l'identité de l'émetteur des paquets, RADIUS propose un mécanisme fondé sur des `Request Authenticator` et `Response Authenticator`. Lors de l'émission du paquet `Access-Request`, le client RADIUS génère un nombre aléatoire de 16 octets : le `Request Authenticator` (RA). Selon cette même norme, ce nombre devrait être aléatoire et unique pendant toute la durée d'utilisation du secret. La réponse du serveur (quelle qu'elle soit) contient un `Response Authenticator`. Ce champ est le hash MD5 de la concaténation des champs suivants : Code, ID, Longueur, RA, Attributs et secret (les trois premiers sont des champs « administratifs » du paquet RADIUS dont la signification est évidente).

Le secret étant connu uniquement du serveur et du client, ce dernier peut recalculer le `Response Authenticator` et vérifier ainsi que le message a bien été envoyé par le serveur et qu'il n'a pas été modifié en cours de transmission.

Notons au passage qu'un mécanisme identique est utilisé pour l'authentification et le contrôle d'intégrité des paquets d'accounting (cf. [2]).

Sécurité des mots de passe

Il apparaît comme une évidence que le mot de passe (transmis dans les paquets `Access-Request`) ne peut être transmis en clair. Cependant, et compte tenu du fait que le serveur doit être à même de transmettre les éléments d'authentification à n'importe quel type de serveur, il est également nécessaire que le mécanisme de chiffrement soit réversible.

Le mécanisme mis en œuvre est par conséquent le suivant :

1. Une fonction de padding est appliquée au mot de passe afin que la longueur de celui-ci soit un multiple de 16 octets ;
2. Le mot de passe est ensuite découpé en blocs de 16 octets ;
3. Le premier bloc (`p1`) de 16 octets est chiffré par la fonction : `c1 = MD5(secret+RA) xor p1` ;
4. Si la longueur du mot de passe est supérieure à 16 octets, les blocs suivants (`pi`) sont chiffrés par la fonction : `ci = MD5(secret+c(i-1)) xor pi` ;

Le RA étant transmis en clair dans le paquet `Access-Request`, le secret apparaît comme l'élément essentiel de la confidentialité de cet échange.

Un mécanisme identique est utilisé pour le chiffrement des clés MS-CHAPv1 (cf [7]). En revanche, pour la mise en œuvre de tunnels [9], un vecteur d'initialisation de 16 bits est ajouté : `c1 = MD5(secret+RA+VI) xor p1`. Ce vecteur d'initialisation doit être regénéré pour chaque requête d'accès.

Les failles de RADIUS

Si la sécurité a été nativement intégrée à RADIUS, les mécanismes mis en œuvre comportent néanmoins des failles permettant de trouver le secret partagé et / ou d'accéder aux informations d'authentification.

Attaque du secret

Une attaque du secret est rendue possible par l'interception du paquet `Access-Request` et de la réponse du serveur. En effet, le `Request-Authenticator` est transmis en clair dans la requête, ainsi que les champs Code, ID, Longueur et les attributs dans la réponse. À partir de ces éléments, l'état MD5 est pré-calculé pour (Code+ID+Longueur+RA+Attributs), le signe « + » étant utilisé pour la concaténation. Il ne reste alors qu'à passer à une attaque au dictionnaire ou à la force brute pour trouver MD5(Code+ID+Longueur+RA+Attributs**+Secret**). La possibilité de pré-calculer un état intermédiaire augmente considérablement la faisabilité et l'efficacité d'une telle attaque.

Le secret peut également être attaqué en effectuant une tentative d'authentification avec un mot de passe court (valide ou non). Dans ce cas, la valeur de l'attribut `User-Password` du paquet `Access-Request` sera `User-Password = MD5(Secret+RA) xor Password`. On en déduit `MD5(Secret+RA) = User-Password xor Password`. Le mot de passe est connu (nous venons de le fournir) et il ne reste plus qu'à attaquer `MD5(Secret+RA)`.

Le `Request-Authenticator` étant hashé en second, il n'est pas possible de pré-calculer l'état MD5 et l'opération s'avère par conséquent plus longue que l'attaque précédente.

Une des principales conséquences de cette attaque est l'exposition des réseaux sans-fil sécurisés via EAP/802.1x. En effet, bien que la RFC 3579 [4] précise que le mécanisme de transport de la clef WEP devrait (SHOULD) supporter IPSEC ESP, la majeure partie des implémentations se contente du mécanisme de base. Ainsi, la découverte du secret RADIUS permet la récupération de la clef WEP et par conséquent, le déchiffrement à la volée des informations transitant sur le réseau. La gravité de cette attaque est telle qu'elle a donné lieu à un draft Internet [11].

Attaque du mot de passe

Nous avons vu précédemment qu'il est possible d'obtenir la valeur de `MD5(Secret+RA)`. En créant des paquets utilisant le même RA, il devient trivial d'envoyer des requêtes d'authentification `Access-Request` multiples en ne modifiant que la valeur de l'attribut `User-Password` à `MD5(Secret+RA) xor Password` où `Password` est obtenu à partir d'un dictionnaire, voire à la force brute. Si un paquet `Access-Accept` est reçu en réponse à une de ses requêtes, c'est que le mot de passe a été deviné. Cette attaque est rendue possible si aucune limite n'est imposée au rythme des requêtes / réponses.

Les deux attaques suivantes s'appuient sur le fait que le RA ne suit pas les recommandations de la norme, et que par conséquent, le même RA peut être utilisé lors de deux sessions différentes à partir de clients RADIUS ayant la même clé partagée (ce qui est un cas très fréquent).

La première attaque consiste à écouter le réseau et à intercepter les paquets `Access-Request`. Un dictionnaire est alors créé avec les champs `RA` et `User-Password`. En cas de répétition d'un RA, un XOR est effectué entre les deux champs `User-Password`. Si les mots de passe font moins de 16 caractères (cas très fréquent également), le résultat de cette opération est le XOR des deux mots de passe. Ces derniers ayant été « paddés » avec des caractères nuls jusqu'à 16 octets, les derniers caractères du mot de passe le plus long sont ainsi trouvés automatiquement.

Plus efficace, la seconde attaque consiste à créer un dictionnaire des champs RA et de la valeur de `MD5(RA+Secret)` associée en générant des demandes d'authentification à partir d'un mot de passe connu (valide ou non). Dès lors, si une nouvelle requête est interceptée avec un RA contenu dans le dictionnaire, le calcul des 16 premiers octets du mot de passe s'effectue trivialement via `Password[16] = User-Password XOR MD5(RA+Secret)`.

Rejeu

Le rejeu n'a d'intérêt que pour les réponses transmises du serveur au client RADIUS, ce dernier étant responsable de l'ouverture des services à destination de l'utilisateur final une fois qu'il s'est authentifié. Cette opération est rendue possible, encore une fois, si les RA se répètent. Dans ce cas, il est nécessaire d'établir un dictionnaire contenant les champs RA et le paquet de réponse `Access-Accept` associé. Des demandes d'authentification sont alors générées depuis le poste utilisateur jusqu'à ce qu'une requête soit transmise par le client avec un RA contenu dans le dictionnaire. UDP facilitant le travail, il ne reste plus qu'à envoyer au client le paquet d'autorisation en prenant soin de se faire passer pour le serveur.

Mise en œuvre

Plusieurs implémentations du protocole RADIUS existent. Pour ne citer que les plus connues, nous avons Microsoft Radius parmi les versions commerciales et Cistron Radius, Livingston Radius et FreeRadius [10] (dont l'un des développeurs est celui du projet Cistron Radius) pour les versions libres. C'est cette dernière implémentation qui sera installée et configurée.

Cas d'école

Pour démontrer la simplicité de configuration et d'utilisation de FreeRadius, nous utilisons le schéma suivant : FreeRadius avec comme serveur d'authentification un serveur MySQL et comme client RADIUS le client SSH avec un module PAM. La base de données contient les comptes utilisateurs et le service RADIUS fait le lien entre le client RADIUS installé sur le poste utilisateur et le serveur d'authentification. Cette configuration est idéale pour ceux qui souhaitent faire des tests chez eux, c'est-à-dire aucune dépendance par rapport à un logiciel propriétaire ou à un matériel quelconque comme un routeur Cisco.

La dernière version stable de FreeRadius est la version 0.9.3. Nous vous épargnons l'installation des packages, mais attardons-nous plutôt sur la configuration de chacun des acteurs.

Le serveur d'authentification MySQL

Sa configuration est simple :

→ création d'une base de données `radius` ;

→ création d'un utilisateur avec le *password test* qui aura les droits sur cette base (ce n'est pas avec cet utilisateur que l'on s'authentifiera) ;

→ création de la structure de la base à l'aide du fichier `db_mysql.sql` fourni avec les sources de FreeRadius (`freeradius-0.9.3/src/modules/rlm_sql/drivers/rlm_sql_mysql/db_mysql.sql`). Remarquez qu'il existe la même chose pour les serveurs de base de données Oracle et PostgresSQL. C'est dans ce fichier qu'est référencé le nom de la base de données. Si vous voulez utiliser un autre nom de base que `radius`, il faudra le modifier aussi dans ce fichier (et dans d'autres fichiers) ;

→ attribution des droits à l'utilisateur nouvellement créé sur la base de données.

Rien de sorcier. Dans la pratique cela nous donne :

```
$ mysql -u root -p -s
[...]
mysql> create database radius;
mysql> use radius;
mysql> source /tmp/freeradius-
0.9.3/src/modules/rlm_sql/drivers/rlm_sql_mysql/db_mysql.sql;
mysql> use mysql;
mysql> insert into user (Host, User, Password) values ('localhost', 'radius', '
');
mysql> update user set Password = password('test') where user = 'radius';
mysql> exit
$ mysqladmin reload -u root -p
Enter password:
$ mysql -u root -p -s
mysql> use mysql;
mysql> grant all on radius.* to radius;
mysql> exit
```

La structure `db_mysql.sql` a permis de créer 6 tables dans la base de données :

- → `radacct` : table pour les informations d'accounting ;
- → `radcheck` : déclaration des comptes utilisateurs ;
- → `radgroupcheck` : déclaration des groupes ;
- → `radgroupreply` : déclaration d'attributs pour être retournés à tous les membres d'un groupe ;
- → `radreply` : même chose que `radgroupreply` mais pour un utilisateur ;
- → `usergroup` : attribution d'un groupe à un utilisateur.

Parmi toutes ces tables, seule la table `radcheck` est importante pour une architecture RADIUS simple comme la nôtre. Tous nos utilisateurs y seront enregistrés. Les autres tables n'ont pas besoin de contenir d'enregistrements.

FreeRadius

Pour l'architecture que nous souhaitons monter, trois fichiers de configuration sont à modifier une fois FreeRadius installé : `/etc/freeradius/sql.conf`, `/etc/freeradius/radiusd.conf` et `/etc/freeradius/clients.conf`. Nous ne détaillerons pas chaque paramètre de configuration de ces fichiers, ils sont assez bien documentés.

Concernant `sql.conf`, ce sont les paramètres de connexion à la base de données qui doivent être modifiés :

```
[...]
# Connect info
server = «localhost»
login = «radius»
password = «test»

# Database table configuration
radius_db = «radius»
[...]
```

Concernant `radiusd.conf`, il est juste nécessaire de modifier la section `authorize`. Cette section sert à déclarer les différentes méthodes d'autorisation (CHAP, MSCHAP, etc.), processus qui sert à vérifier que l'information fournie dans la requête `Access-Request` à propos de l'utilisateur est suffisante pour passer à la phase d'authentification. Dans notre cas, nous ajoutons la méthode `sql` :

```
[...]
authorize {
[...]
        #
        #  Look for IPASS style 'realm/', and if not found, look for
        #  '@realm', and decide whether or not to proxy, based on
        #  that.
        #       realmslash
        suffix

        #
        #  Read the 'users' file
        # files
        sql
[...]
}
```

FreeRadius n'accepte que des requêtes venant de machines strictement identifiées et authentifiées. Le fichier `clients.conf` sert à déclarer ces machines.

Chaque enregistrement dans ce fichier se présente sous la forme suivante :

```
clients <IP> {
        secret = <secret à déclarer>
        shortname = <nomination du client>
}
```

Dans notre cas, nous avons :

```
clients <127.0.0.1> {
        secret = <radiustest>
        shortname = <localhost>
}
```

Notre serveur RADIUS est fin prêt. L'outil `radtest` teste si la communication entre FreeRadius et MySQL se fait bien. Le test consiste à envoyer une requête `Access-Request` au serveur FreeRadius. Sa réponse `Access-Accept` ou `Access-Reject` dépendra de la validité de la requête, comme cela a été expliqué dans la première partie de l'article.

Il reste une dernière petite chose à faire pour effectuer nos tests : déclarer un utilisateur dans la base de données MySQL à l'aide de la requête `SQL` suivante et qui servira à nous authentifier (prenez soin lors de vos tests d'utiliser un login déjà déclaré dans `/etc/passwd`) :

```
insert into radcheck (Username, Attribute, op, Value) VALUES
('compaq', 'Password', '==', 'passtest');
```

On lance le serveur Radius de manière à déboguer notre configuration :

```
# /usr/sbin/freeradius -A -xx -y
[...]
Listening on IP address *, ports 1812/udp and 1813/udp, with proxy on 1814/udp.
Ready to process requests.
```

Sur une autre console, nous fournissons les arguments nécessaires à `radtest`, c'est-à-dire le compte de l'utilisateur enregistré dans la base de données MySQL, l'adresse du serveur RADIUS, la valeur de l'attribut NAS-Port (le port source de la connexion) qui n'est pas important selon la page man de `radtest` et le secret :

```
# radtest compaq passtest 127.0.0.1 1234 radiustest
Sending Access-Request of id 140 to 127.0.0.1:1812
        User-Name = «compaq»
        User-Password = «passtest»
        NAS-IP-Address = pluton
        NAS-Port = 1234
rad_recv: Access-Accept packet from host 127.0.0.1:1812, id=140, length=20
```

Le client reçoit bien le paquet `Access-Accept` signifiant que l'authentification s'est bien déroulée.

Le client

Nous avons décidé d'utiliser SSH comme client RADIUS du fait qu'il est omniprésent sur les machines. Évidemment, fourni comme il est par les distributions, il ne supporte pas le protocole RADIUS et ne permet donc pas de s'authentifier auprès du serveur MySQL.

La solution est d'installer un module PAM. Vous trouverez le module à l'adresse **http://www.freeradius.org/pam_radius_auth/** ou bien utilisez le package fourni avec votre distribution s'il existe. Dans notre cas, nous avons installé le package `libpam-radius-auth` de la Debian.

Il faut ensuite paramétrer le module PAM du serveur `sshd` pour qu'il utilise l'authentification RADIUS avant tout autre mode d'authentification :

```
# cat /etc/pam.d/ssh
#%PAM-1.0
auth       sufficient   pam_radius_auth.so
```

```
auth       required     pam_nologin.so
auth       required     pam_unix.so
auth       required     pam_env.so # [1]
[...]
```

`sufficient` signifie qu'un tel module suffit à l'authentification en cas de réussite.

Reste maintenant à dire à SSH quel **secret** utiliser. Le fichier `/etc/pam_radius_auth.conf` est là pour ça. Ses commentaires suffisent à comprendre comment le configurer :

```
# server[:port] shared_secret      timeout (s)
127.0.0.1       radiustest                  1
```

L'architecture RADIUS est définitivement installée et configurée. Il est nécessaire maintenant de la tester. Lancez de manière classique SSH toujours avec le même compte utilisateur et déboguez le serveur FreeRadius. Les logs sont commentés au fur et à mesure de manière à bien comprendre ce qui se passe entre le client et le serveur :

```
# /usr/sbin/freeradius -A -xx -y
[...]
Listening on IP address *, ports 1812/udp and 1813/udp, with proxy on 1814/udp.
Ready to process requests.
rad_recv: Access-Request packet from host 127.0.0.1:5658, id=56, length=83
Thread 1 assigned request 0
Requête Access-Request de la part du client 127.0.0.1

-- Walking the entire request list --
Threads: total/active/spare threads = 5/1/4
Waking up in 5 seconds...
Thread 1 handling request 0, (1 handled so far)
        User-Name = «compaq»
        User-Password = «passtest»
        NAS-IP-Address = 127.0.0.1
        NAS-Identifier = «ssh»
        NAS-Port = 4633
        NAS-Port-Type = Virtual
        Service-Type = Authenticate-Only
        Calling-Station-Id = «pluton»
```

Ce sont les valeurs des attributs. Nous reconnaissons notre login/passwd et le client qui dialogue avec le serveur FreeRadius

```
modcall: entering group authorize for request 0
  modcall[authorize]: module «preprocess» returns ok for request 0
  modcall[authorize]: module «chap» returns noop for request 0
  modcall[authorize]: module «eap» returns noop for request 0
    rlm_realm: No '@' in User-Name = «compaq», looking up realm NULL
    rlm_realm: No such realm «NULL»
  modcall[authorize]: module «suffix» returns noop for request 0
radius_xlat:  'compaq'
rlm_sql (sql): sql_set_user escaped user -> 'compaq'
radius_xlat:  'SELECT id,UserName,Attribute,Value,op FROM radcheck WHERE Username
= 'compaq' ORDER BY id'
rlm_sql (sql): Reserving sql socket id: 4
radius_xlat:  'SELECT radgroupcheck.id,radgroupcheck.GroupName,radgroupcheck.
Attribute,radgroupcheck.Value,radgroupcheck.op  FROM radgroupcheck,usergroup
WHERE usergroup.Username = 'compaq' AND usergroup.GroupName = radgroupcheck.
GroupName ORDER BY radgroupcheck.id'
radius_xlat:  'SELECT id,UserName,Attribute,Value,op FROM radreply WHERE Username
= 'compaq' ORDER BY id'
radius_xlat:  'SELECT radgroupreply.id,radgroupreply.GroupName,radgroupreply.
Attribute,radgroupreply.Value,radgroupreply.op  FROM radgroupreply,usergroup
WHERE usergroup.Username = 'compaq' AND usergroup.GroupName = radgroupreply.
GroupName ORDER BY radgroupreply.id'
rlm_sql (sql): Released sql socket id: 4
  modcall[authorize]: module «sql» returns ok for request 0
  modcall[authorize]: module «mschap» returns noop for request 0
modcall: group authorize returns ok for request 0
```

Si vous jetez un œil, vous allez vous apercevoir que le serveur exécute les différents modules déclarés dans le groupe `authorize`. Si c'est effectué avec succès, le module répond par `ok`, `noop` dans le cas où il ne fait rien et `fail` si ça échoue (cf `freeradius-0.9.3/doc/rlm_always`).

Dans notre cas, `preprocess` (qui reconfigure certains attributs non standards) et évidemment `sql` s'effectuent avec succès.

```
auth: type Local
auth: user supplied User-Password matches local User-Password
Sending Access-Accept of id 56 to 127.0.0.1:5658
Finished request 0
Going to the next request
Thread 1 waiting to be assigned a request
-- Walking the entire request list --
Threads: total/active/spare threads = 5/0/5
Waking up in 1 seconds...
-- Walking the entire request list --
Cleaning up request 0 ID 56 with timestamp 41013dd7
Nothing to do.  Sleeping until we see a request.
```

Les mots de passe fournis par le client RADIUS et celui enregistré dans la base MySQL coïncident, le serveur RADIUS renvoie donc une réponse `Access-Accept`

Un mot sur le serveur RADIUS en mode proxy

Dans l'exemple précédent, le serveur RADIUS gérait lui-même la demande d'authentification en se connectant au serveur MYSQL. Il était autonome, mais il ne s'agissait pas réellement d'authentification forte (utilisation de token, génération de mots de passe uniques...).

En configurant FreeRadius en mode proxy, le serveur peut alors relayer les requêtes RADIUS vers un serveur d'authentification forte, quel qu'il soit à condition d'être compatible RADIUS. D'un point de vue technique, il suffira de configurer les fichiers `/etc/freeradius/radiusd.conf` et `/etc/freeradius/proxy.conf`.

Dans le premier fichier, le seul champ à modifier est `proxy_requests` qu'il suffit de mettre à `yes` pour activer le mode proxy du serveur RADIUS. Décommentez aussi la ligne `$INCLUDE ${confdir}/proxy.conf` pour pouvoir utiliser le fichier de configuration `proxy.conf`.

Concernant le fichier `/etc/freeradius/proxy.conf`, deux champs sont à modifier pour une configuration basique du mode proxy :

→ `synchronous = yes` ; il permet de renvoyer la réponse à la requête d'authentification au client RADIUS initial.

→ les objets `realm` : ces objets permettent d'identifier la provenance des utilisateurs. Ils donnent au serveur RADIUS l'information nécessaire (les suffixes adossés au nom des utilisateurs) pour faire suivre la requête d'authentification auprès du ou des serveurs d'authentification correspondants. Il existe dans le fichier `proxy.conf` toute une série d'exemples d'objets de ce type. Nous vous laissons les regarder.

Conclusion

RADIUS, parfaitement intégré à 802.1x et EAP, est toujours incontournable. Simple (sinon trivial) et ouvert, il offre une solution fiable de centralisation des informations d'authentification. Et s'il n'est pas parfaitement sûr, son niveau de sécurité reste tout à fait acceptable tant que les implémentations suivent les recommandations de la norme.

Ainsi, une implémentation s'appuyant sur un générateur vraiment aléatoire pour les RA et supportant des secrets différents pour chaque client couplée avec des mots de passe et des secrets partagés « solides », garantit un niveau de sécurité on ne peut plus satisfaisant.

Références

[1] *Remote Authentication Dial-In User Service (RADIUS)* - RFC 2058 - C. Rigney, A. Rubens, W. Simpson, S. Willens - Janvier 1997 - **http://www.ietf.org/rfc/rfc2058.txt**

[2] *RADIUS Accounting* - RFC 2866 - C. Rigney, A. Rubens, W. Simpson, S. Willens - Juin 2000 - **http://www.ietf.org/rfc/rfc2866.txt**

[3] *RADIUS and IPv6* - RFC 3162 - B. Aboba, G. Zorn, D. Mitton - Août 2001 - **http://www.ietf.org/rfc/rfc3162.txt**

[4] *RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)* - RFC 3579 - B. Aboba, P. Calhoun - Septembre 2003 - **http://www.ietf.org/rfc/rfc3579.txt**

[5] *Remote Authentication Dial-In User Service (RADIUS)* - RFC 2865 - C. Rigney, A. Rubens, W. Simpson, S. Willens - Juin 2000 - **http://www.ietf.org/rfc/rfc2865.txt**

[6] *Assigned Numbers* - STD 2, RFC 1700 - Reynolds, J. and J. Postel - Octobre 1994 - **http://www.ietf.org/rfc/rfc2865.txt**

[7] *Microsoft Vendor-specific RADIUS Attributes* - RFC 2548 - G. Zorn - Mars 1999 - **http://www.ietf.org/rfc/rfc2548.txt**

[8] *RADIUS Extensions* - RFC 2869 - C. Rigney, W. Willats, P. Calhoun - Juin 2000 - **http://www.ietf.org/rfc/rfc2869.txt**

[9] *RADIUS Attributes for Tunnel Protocol Support* - RFC 2868 - G. Zorn, D. Leifer, A. Rubens, J. Shriver, M. Holdrege, I. Goyret - Juin 2000 - **http://www.ietf.org/rfc/rfc2868.txt**

[10] FreeRadius - **http://www.freeradius.org**

[11] *RADIUS Vulnerabilities in Wireless and Wired Environments* - R. Chou, M. Andrade, J. Wright - Juillet 2004 - **http://home.jwu.edu/jwright/radius_vuln_00.txt**

Kerberos

emmanuel.bouillon@cea.fr
gilles.wiber@cea.fr

1 Introduction

Selon la mythologie grecque, Kerberos est le chien à trois têtes gardant les portes de l'enfer. Plus « récemment », Kerberos fut le nom choisi pour le mécanisme d'authentification d'un projet mené au MIT appelé Athena. Aujourd'hui, Kerberos est un protocole d'authentification réseau. Les versions 1 à 3 de ce protocole ne dépassèrent pas les portes des laboratoires du MIT. La version 4 ([1] et [2]) fut la première réellement utilisée. La version actuelle du protocole, la version 5, est un standard défini dans la RFC 1510 [3].

Kerberos est un protocole ancien, cependant, son adoption par plusieurs systèmes d'exploitation propriétaires comme Solaris ou Windows a suscité un regain d'intérêt pour cette méthode d'authentification. En effet, peu de mécanismes résolvent le problème de l'authentification réseau unifiée (SSO pour *Single Sign On*) en milieu hétérogène. Kerberos présente l'avantage d'être un standard disponible sur les systèmes « libres » (Linux, *BSD) et sur les systèmes propriétaires, Solaris (Sun-Microsystem), Irix (SGI), Windows 200[0|3] et XP. La portabilité des implémentations Open Source permet de compléter cette liste. Par ailleurs, Kerberos est aujourd'hui le principal mécanisme de sécurité sous-jacent supporté dans les implémentations de la GSSAPI [4], cette API étant la base des `RPCSEC_GSS` [5], mécanisme désigné pour sécuriser la prochaine version de NFS (version 4, [6]).

Après avoir donné une description (superficielle) du protocole, cet article présentera sa mise en œuvre ainsi que les difficultés liées à son déploiement et à son administration, pour finir par les perspectives qu'offrent les évolutions à venir.

2 Le protocole Kerberos

L'objectif de cette partie n'est pas de décrire les détails du protocole, plusieurs documentations existent en la matière ([3],[7]). On se contentera de donner les grands principes nécessaires à la compréhension de son fonctionnement et de ses contraintes.

2.1 Caractéristiques du protocole

Trois caractéristiques essentielles sont à retenir au sujet de Kerberos :

- Kerberos permet l'authentification des utilisateurs et des services sur un **réseau supposé non sûr** (« open, unprotected network » selon la RFC 1510). Par exemple, les données sur ce réseau peuvent être lues ou modifiées, les adresses des machines peuvent être faussées, elles n'interviendront pas pour assurer l'authentification.
- Kerberos utilise une **tierce partie de confiance** : toutes les entités du réseau, utilisateurs et services, appelés principaux et regroupés en « royaume », font confiance à cette tierce partie (le serveur Kerberos ou KDC pour *Key Distribution Center*).
- Enfin, Kerberos utilise des mécanismes de **chiffrement symétrique** (à clé secrète) : tous les principaux partagent une clé secrète avec le serveur Kerberos.

2.2 Fonctionnement du protocole

Plusieurs approches pour présenter Kerberos existent, l'une des plus célèbres a la forme d'une pièce de théâtre en 4 actes [8]. Celle qui sera utilisée ici consiste à montrer comment on part d'une authentification par secret partagé pour arriver à la version 5 de Kerberos [9].

Dans toute la suite, Alice (A) sera l'entité qui initie la communication (analogie avec le client, l'utilisateur), Bob (B) l'entité qui répond (analogie avec le service, serveur applicatif). K_{ab} sera la clé partagée entre Alice et Bob (on a $K_{ab} = K_{ba}$). $K\{Texte\}$ désigne le résultat du chiffrement de « Texte » avec la clé K.

2.2.1 Authentification par secret partagé

Kerberos utilise des algorithmes de chiffrement à clé secrète pour l'authentification. Un moyen de faire de l'authentification à l'aide de tels algorithmes est le suivant (N_a et N_b sont deux nombres aléatoires générés par Alice et Bob respectivement) :

1. Alice appelle Bob
2. Bob envoie $K_{ab}\{N_b\}$
3. Alice déchiffre le message, obtient N_b et envoie $K_{ab}\{N_b\text{-}1\}$
4. Alice envoie $K_{ab}\{N_a\}$
5. Bob déchiffre le message, obtient N_a et envoie $K_{ab}\{N_a\text{-}1\}$

Ce qui, en regroupant les messages indépendants, donne :

1. Alice envoie $K_{ab}\{N_a\}$
2. Bob déchiffre le message, obtient N_a, envoie $K_{ab}\{N_a\text{-}1\}$ et $K_{ab}\{N_b\}$
3. Alice déchiffre le message, obtient N_b et envoie $K_{ab}\{N_b\text{-}1\}$

Ce schéma (appelons le **schéma 1**) apparaît dans le protocole de Needham et Schroeder [10], sur lequel se fonde Kerberos, avec une amélioration toutefois : l'utilisation d'une tierce partie de confiance.

2.2.2 Utilisation d'une tierce partie de confiance

L'un des inconvénients majeurs du schéma 1 est son manque d'extensibilité. La généralisation à m utilisateurs et n services, implique la distribution préalable de mxn clés partagées. Une solution à cette problématique est l'introduction d'une tierce partie de confiance.

Cette tierce partie (TP) de confiance connaît les secrets partagés de chaque entité à authentifier (K_a pour Alice, K_b pour Bob). L'authentification mutuelle se déroule alors comme suit :

1. Alice demande à TP d'accéder à Bob
2. TP génère une clé de session entre Alice et Bob, K_{ab}
3. TP envoie $K_a\{K_{ab}, Bob\}$ à Alice et $K_b\{K_{ab}, Alice\}$ à Bob

À ce niveau, on peut appliquer le schéma I pour l'authentification mutuelle par algorithme à clé secrète. En effet, Alice connaissant K_a, peut en déduire K_{ab}. De même, Bob connaissant K_b peut en déduire K_{ab}.

On remarque cependant que pour la distribution de K_{ab}, Alice, le client envoie un message alors que la tierce partie commune à tous les utilisateurs en envoie deux. On préfère en général faire porter la charge sur le client et non sur la tierce partie qui doit gérer tous les clients. On modifie donc légèrement le schéma précédent qui devient :

1. Alice demande à TP d'accéder à Bob
2. TP génère une clé de session entre Alice et Bob, K_{ab}
3. TP envoie $K_a\{K_{ab}, Bob\}$ et $K_b\{K_{ab}, Alice\}$ à Alice
4. Alice appelle Bob et lui envoie $K_b\{K_{ab}, Alice\}$

Alice, qui ne connaît pas K_b, ne peut rien faire de $K_b\{K_{ab}, Alice\}$ si ce n'est l'envoyer à Bob. Cette information est le ticket pour dialoguer avec Bob. On aboutit alors au schéma de Needham et Schroeder, le nombre aléatoire N_1, envoyé avec la première requête n'est là que pour rendre unique cette requête et sa réponse pour éviter les attaques de type « rejeu».

1. Alice demande à TP d'accéder à Bob et joint N_1
2. TP génère une clé de session entre Alice et Bob, K_{ab}
3. TP envoie $K_a\{N_1, K_{ab}, Bob, Ticket_b\}$ à Alice, où $Ticket_b = K_b\{K_{ab}, Alice\}$
4. Alice déchiffre la réponse, en tire $Ticket_b$ et K_{ab}
5. Alice appelle Bob avec $Ticket_b$ et $K_{ab}\{N_a\}$ (à partir d'ici, on retrouve le schéma I défini précédemment)
6. Bob déchiffre le Ticket, en tire K_{ab}, répond avec $K_{ab}\{N_a\text{-}1, N_b\}$
7. Alice répond avec $K_{ab}\{N_b\text{-}1\}$

2.2.3 Kerberos

Le protocole Kerberos améliore le schéma de Needham et Schroeder d'une part en remplaçant les nombres aléatoires par des horodateurs (*timestamps*), d'autre part en séparant le rôle de la tierce partie de confiance en deux services :

→ le service d'authentification (AS pour *Authentication Service*) ;

→ le générateur de ticket de service (TGS pour *Ticket Granting Service*).

L'utilisation des horodateurs permet d'introduire la péremption de ticket et de réduire la fenêtre de temps pendant laquelle le rejeu de ticket est possible. Ce type d'attaque est empêché définitivement par l'utilisation d'un cache stockant les authentificateurs reçus et dont la date de validité n'a pas expiré. On note que cette propriété introduit une dépendance entre Kerberos et le service de temps. La séparation du rôle du KDC en deux services induit deux types de ticket :

→ **Le TGT** (*Ticket Granting Ticket*) est envoyé par l'AS. Une demande de TGT est en fait une demande d'accès au service TGS. Le TGT est donc le ticket du service TGS. La réponse de l'AS contient aussi la clé de session entre le client (A) et le service TGS ($K_{a,tgs}$) chiffrée par la clé secrète du client (K_a).

→ **Le TS** (*Ticket Service*) pour un service donné (S) est envoyé par le TGS sur présentation d'un TGT et d'un horodateur chiffré avec la clé de session $K_{a,tgs}$. Le TGS sait donc que le client a pu déchiffrer la réponse de l'AS et connaît la clé secrète K_a. Le TGS génère une clé de session entre A et S, $K_{a,s}$. La réponse du TGS contient notamment cette clé de session chiffrée par $K_{a,tgs}$ (A pourra ainsi la déduire) et le TS proprement dit, constitué, entre autres, de $K_{a,s}$ chiffré par K_s, la clé secrète du service S (S, sur présentation de ce ticket pourra en déduire $K_{a,s}$).

La même clé $K_{a,tgs}$ permet d'obtenir les clés de session des services accédés pendant toute la validité du TGT. Le principal avantage de cette séparation des rôles du KDC est de permettre le SSO (*Single Sign On*). En effet, seule l'exploitation du TGT nécessite l'utilisation de la clé secrète du client. Ainsi, tant que son TGT est valide, le client pourra acquérir des TS sans réutiliser sa clé, qui est directement déduite du mot de passe dans le cas d'un utilisateur. Cette propriété, couplée avec la possibilité de rendre le TGT « forwardable », c'est-à-dire réutilisable une fois connecté au service, permet à Kerberos de fournir un SSO.

2.2.4 Résumé simplifié

Trois échanges entrent en jeu au cours d'une authentification Kerberos :

→ le client fait une demande de TGT auprès du service d'authentification de Kerberos (AS) ;

→ le client utilise ce TGT pour demander au service délivreur de tickets (TGS) un ticket de service (TS) pour le service auquel il souhaite accéder ;

→ le client utilise ce ticket de service pour s'authentifier auprès du service accédé.

Voir figure 1 ci-contre.

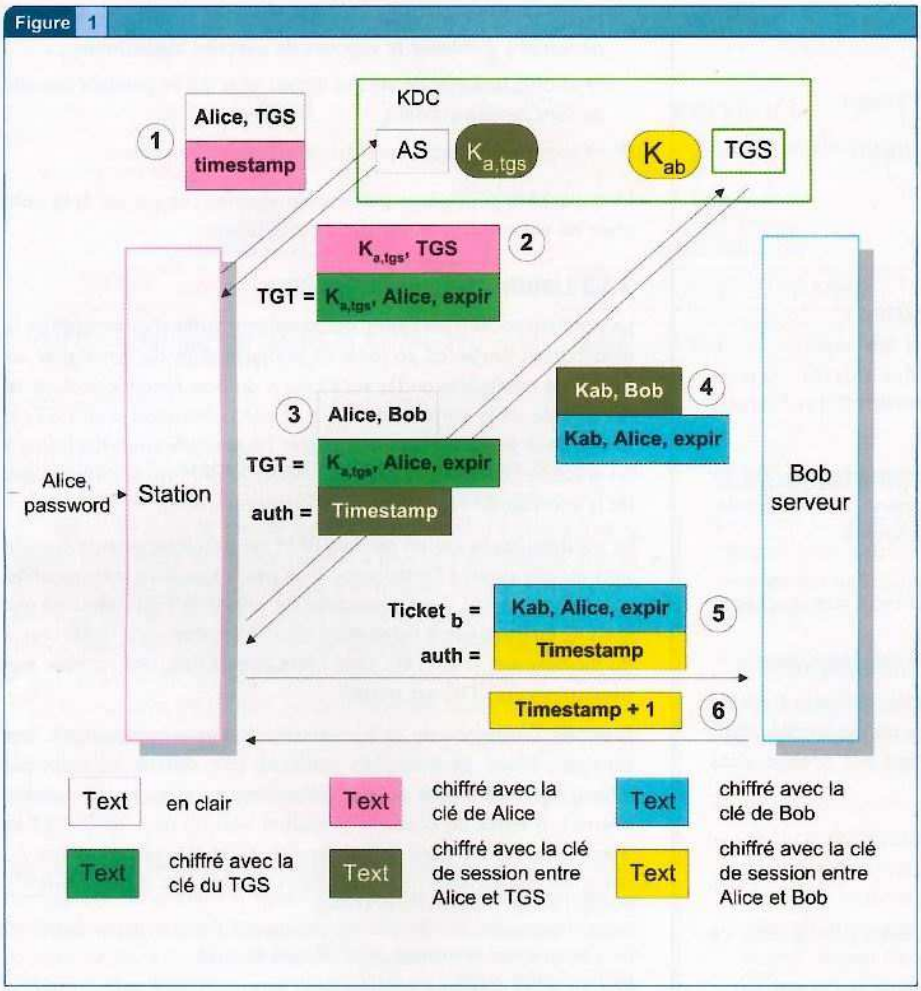
Une analogie avec la vie courante illustre la différence entre TGT et TS. Le mot de passe est le passeport qui permet d'obtenir des TGT (des visas). Grâce à ce visa, on obtient les billets d'avion pour les destinations souhaitées, les tickets de service, pendant toute la durée de validité du visa (TGT).

2.2.4 La pré-authentification

Au cours des échanges qui prennent place lors d'une authentification Kerberos, on constate qu'il n'est pas nécessaire de connaître K_a pour obtenir de la tierce partie de confiance un TGT et un message chiffré par K_a. Il suffit de le demander à l'AS pour l'obtenir.

La pré-authentification résout ce problème. Elle exige que la requête à l'AS, émise par le client, soit accompagnée d'un horodateur chiffré par K_a. L'AS assure ainsi la requête qui émane bien d'une entité connaissant sa clé secrète.

Figure 1



Les gros avantages de la solution constructeur sont en théorie :

→ le support fourni ;

→ l'intégration au système d'exploitation : le comportement des commandes reste conforme au comportement décidé par le constructeur (par exemple, le positionnement des variables d'environnement lors du login).

Or dans les faits, le support de Kerberos par les constructeurs est faible. De plus, cette solution comporte des inconvénients majeurs :

→ l'incompatibilité entre les différentes implémentations, notamment au niveau de l'administration de la base Kerberos à distance (commandes `kadmin` et `kpasswd`) ;

→ certains constructeurs ne supportent pas Kerberos ;

→ certaines implémentations propriétaires sont construites sur d'anciennes versions Open Source.

D'où la nécessité d'introduire une version Open Source sur le réseau pour en tirer les avantages suivants :

→ uniformisation de la solution sur tout le réseau et simplification de son administration ;

→ maîtrise des sources, un atout majeur dans le cadre de la mise en place d'un élément de sécurité.

2.2.5 Les relations de confiance inter-royaume

Kerberos prévoit la possibilité qu'un principal puisse s'authentifier auprès de principaux n'appartenant pas à son royaume : il s'agit de l'authentification inter-royaume. Cela implique d'établir une relation de confiance entre les deux royaumes, relation pouvant être unilatérale ou bilatérale.

3 Administration

Kerberos a la réputation d'être difficile à administrer, à tel point qu'on déconseille parfois son déploiement [11] : *« D'après nous, la plupart des sites feraient mieux de ne pas l'utiliser [sic]. »*

Dans cette partie, certains de ces problèmes sont abordés.

3.1 Mise en œuvre

3.1.1 Choix de l'implémentation

Il existe plusieurs implémentations de Kerberos V : certaines sont Open Source, comme la version du MIT [12] ou Heimdal [13], d'autres sont propriétaires comme la version SEAM de SUN (construite sur une ancienne version de Kerberos du MIT) ou Active Directory de Microsoft. Deux possibilités existent donc : la solution constructeur ou une solution Open Source.

3.1.2 Installation d'un KDC

L'installation (basique) d'un KDC maître se fait en quelques commandes. Deux fichiers de configuration sont à éditer :

- `krb5.conf`, nécessaire à toutes les machines du royaume Kerberos. Il indique notamment le royaume par défaut, les noms et ports des KDC (maître et esclaves) des différents royaumes, les noms et ports des services d'administration à distance (`kadmind`), les options par défaut des tickets et des services kerbérisés ;
- `kdc.conf`, spécifique au KDC. Il indique notamment les informations relatives à la base des utilisateurs, les algorithmes de chiffrement supportés, la durée de vie des tickets et de leur renouvellement, ainsi que les attributs par défaut des principaux (par exemple, la pré-authentification).

Avec l'implémentation du MIT, la création d'un KDC maître se fait comme suit :

Création de la base Kerberos :

```
kdc-master:# kdb5_util create -r TEST.FR -s
```

Cette commande crée la base pour le royaume `TEST.FR`. Elle sera chiffrée avec le mot de passe demandé. Celui-ci sera stocké dans un fichier (`option -s`) sur le KDC. Ainsi, le service pourra démarrer seul. A ce stade, l'accès à la base peut se faire

grâce à la commande kadmin.local exécutée en tant que root sur le KDC :

```
kdc-master:~# kadmin.local
Authenticating as principal paul/admin@TEST.FR with password.
kadmin.local: listprincs
K/M@TEST.FR (principaux créés automatiquement à l'initialisation du royaume)
kadmin/admin@TEST.FR
kadmin/changepw@TEST.FR
kadmin/history@TEST.FR
krbtgt/TEST.FR@TEST.FR
kadmin.local:
```

2 Création d'un principal d'administration :

```
kadmin.local: ank admin/admin (ank = add new key)
Enter password for principal «admin/admin@TEST.FR»: xxxxxxx
Re-enter password for principal «admin/admin@TEST.FR»: xxxxxxx
Principal «admin/admin@TEST.FR» created.
```

On attribue les privilèges aux principaux en éditant le fichier kadm5.acl. Par exemple, la ligne suivante donne tous les privilèges aux principaux ayant une instance « admin ».

```
*/admin@TEST.FR *
```

Enfin, pour lancer le service kerbérisé kadmind, il faut stocker sa clé dans un fichier (appelé fichier keytab) :

```
kadmin.local: ktadd -k /usr/.../kadm5.keytab kadmin/admin kadmin/changepw
```

La clé des principaux kadmin/admin et kadmin/changepw (kadmind gère aussi les changements de mots de passe) est stockée dans le fichier kadm5.keytab dont l'emplacement est précisé dans kdc.conf.

3 Lancement des services

```
kdc-master:~# krb5kdc
kdc-master:~# kadmind
```

On peut alors accéder à l'interface d'administration à distance et créer un utilisateur par exemple :

```
client-1:~# kinit admin/admin@TEST.FR (demande de ticket pour le principal admin/admin)
Password for admin/admin@TEST.FR: xxxxxxx
client-1:~# klist (affichage des tickets)
Ticket cache: FILE:/tmp/krb5cc_501
Default principal: admin/admin@TEST.FR
Valid starting Expires        Service principal
02/06/04 13:28:21 02/06/04 23:28:21 krbtgt/TEST.FR@TEST.FR
client-1:~# kadmin -r TEST.FR -s kdc-master:749 (si krb5.conf est bien renseigné, pas besoin de ces options)
kadmin: ank paul
Enter password for principal «paul@TEST.FR»: xxxxxxx
Re-enter password for principal «paul@TEST.FR»: xxxxxxx
Principal «paul@TEST.FR» created.
kadmin: quit
```

Par défaut, le principal paul@TEST.FR correspond à l'utilisateur dont le nom de login est paul.

Un KDC doit être une machine à la fois très sécurisée et redondante : il est indispensable de mettre en place au moins un 2ème KDC (esclave) sur le réseau ainsi que les sauvegardes (sécurisées) de ces derniers.

Lors de la configuration logicielle d'un KDC, il est notamment important de :

→ choisir les algorithmes de chiffrement supportés. Le repeuplement de la base Kerberos avec un nouvel algorithme de chiffrement est un processus long et dépend de la politique de changement de mots de passe mise en œuvre sur le réseau. Néanmoins, pour des problèmes d'interopérabilité, il peut être nécessaire d'enlever le support de certains algorithmes ;

→ choisir la durée de vie des tickets et la durée pendant laquelle ils sont renouvelables ;

→ mettre en place la pré-authentification par défaut.

Une machine peut gérer plusieurs royaumes : ce cas est très utile pour ne pas multiplier les KDC sur le réseau.

3.1.3 L'authentification système

La configuration d'un client est simple : il suffit d'avoir installé la distribution Kerberos en local de la machine et de renseigner un fichier de configuration (la vérification du bon fonctionnement se fait à l'aide de la commande kinit et par l'obtention d'un TGT). Il existe deux possibilités pour intégrer l'authentification Kerberos à un système Unix : l'utilisation d'un module PAM ou le changement de la commande login (ainsi que lockscreen, etc.).

La solution basée sur un module PAM est plus intéressante dans le sens où elle survit à l'installation d'un patch touchant la commande login. Dans ce cas, il est nécessaire de valider le TGT retourné par le KDC en demandant l'obtention d'un TS (option « validate=true » du module pam_krb5 [14]). Sans cette précaution, une attaque par usurpation du KDC est triviale.

Prenons l'exemple de la kerbérisation des r-commandes. Les services krshd et krlogind utilisent par défaut le principal « host/machine@ROYAUME ». Pour utiliser ces services sur la machine client-1, il suffit de créer le principal host/client-1@TEST.FR et d'exporter sa clé dans le fichier /etc/krb5.keytab de client-1 :

```
kadmin: ank -randkey host/client-1 (on crée le principal avec une clé aléatoire)
Principal «host/client-1@TEST.FR» created.
kadmin: ktadd host/client-1@TEST.FR (on exporte la clé dans le fichier keytab)
Entry for principal host/client-1@TEST.FR added to keytab
WRFILE:/etc/krb5.keytab.
kadmin: quit
client-1:~#
```

Il ne reste plus qu'à lancer le service via inetd, avec par exemple dans inetd.conf :

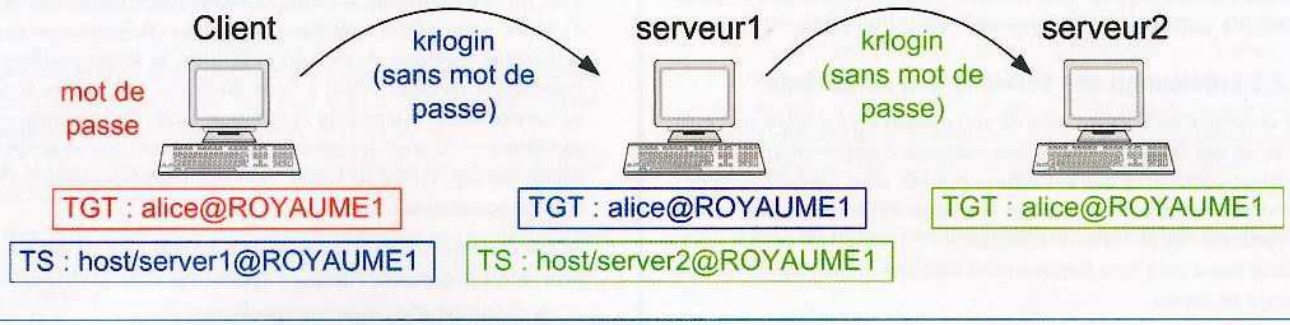
```
kshell stream tcp nowait root /usr/sbin/tcpd /usr/sbin/kshd -5ec
```

Puis Paul peut se connecter à client-1 en rsh « kerbérisé » :

```
paul@youki:~$ kinit (acquisition d'un TGT pour paul@TEST.FR)
Password for paul@TEST.FR: xxxxxxx
paul@youki:~$ klist (Paul a bien un TGT)
Ticket cache: FILE:/tmp/krb5cc_501
Default principal: paul@TEST.FR
Valid starting Expires        Service principal
02/06/04 15:29:18 02/07/04 01:29:18 krbtgt/TEST.FR@TEST.FR
paul@youki:~$ rsh -x client-1 id ('-x' pour une connexion chiffrée)
This rsh session is encrypting input/output data transmissions.
uid=501(paul) gid=501 groups=501
paul@youki:~$ klist (Paul a maintenant un TS en cache)
Ticket cache: FILE:/tmp/krb5cc_501
Default principal: paul@TEST.FR
Valid starting Expires        Service principal
02/06/04 15:27:26 02/07/04 01:27:26 krbtgt/TEST.FR@TEST.FR
02/06/04 15:27:30 02/07/04 01:27:26 host/client-1@TEST.FR
```

Que s'est-il passé ? Paul fait une demande de TGT à l'aide de la commande kinit en fournissant son mot de passe (étape 1 et 2, figure 1). Si la procédure de login avait été kerbérisée (avec un module PAM par exemple), cette étape n'aurait pas été nécessaire : Paul serait en possession d'un TGT de manière transparente dès

Figure 2



sa connexion. Paul accède ensuite au serveur `krshd` de client-1 : de manière transparente, le client kerbérisé `rsh` a obtenu grâce au TGT de Paul un TS pour le service `host/client-1@TEST.FR` (étape 3 et 4, figure 1). Ce ticket de service permet à Paul de s'authentifier auprès du service `krshd` de client-1 (étape 5 et 6, figure 1). Paul a maintenant en cache un TS pour `host/client-1`. Ce cache est stocké dans un fichier temporaire dans `/tmp` et est accessible à Paul seulement (et à `root`). Le nom de ce fichier est stocké dans la variable d'environnement `KRB5CCNAME` de Paul. Pendant toute la durée de vie de ce ticket de service, Paul pourra s'en servir pour accéder au service `host/client-1` (c'est-à-dire `krsh`, `krlogin`, `ktelnet` ou `ssh`). Seules les étapes 5 et 6 seront répétées (pas d'échange avec le KDC).

La puissance de Kerberos est ici : avec une procédure de login kerbérisée, un utilisateur entre son mot de passe à la console de sa machine, puis il est authentifié de manière transparente pour chaque accès à un service kerbérisé pendant toute la durée de validité de son TGT. Allié avec l'option « forwardable », qui permet à un TGT d'être dupliqué sur la machine cible, Kerberos autorise les utilisateurs à rebondir de machine en machine de manière authentifiée, sans donner de nouveau leur mot de passe. La figure 2 montre le contenu des *caches tickets* sur chaque machine après deux rebonds.

Depuis l'ajout du support de la GSSAPI dans OpenSSH, il est possible d'intégrer ce service dans le SSO Kerberos : avec un TGT valide, on peut s'authentifier auprès d'un serveur SSH sans entrer de nouveau son mot de passe (option `GSSAPIAuthentication` dans `sshd_config`).

3.1.4 L'authentification applicative

Le système d'exploitation n'est pas l'unique partie du système d'information où une authentification est requise. L'accès au mail, à une base de données, ou à une application Web, peut exiger une authentification supplémentaire ou simplement l'authentification système, mais renouvelée pour accorder son accès. Dans le cas d'une authentification renouvelée, c'est-à-dire où l'application utilise le même protocole que l'authentification système, le déploiement de Kerberos aura des conséquences sur le fonctionnement de l'application. Même si des standards, le plus souvent fondés sur la GSSAPI, voient progressivement le jour, leur intégration n'est jamais transparente. Cet impact est donc à prendre en compte dès l'origine du projet de déploiement.

3.2 Déploiement

Il est plus facile de créer un réseau kerbérisé que de kerbériser un réseau déjà existant, notamment pour des raisons liées au peuplement de la base Kerberos et à l'éradication des services non kerbérisés.

3.2.1 Peuplement de la base Kerberos

Au cours du déploiement de Kerberos se pose le problème du peuplement de la base Kerberos. Il n'y a pas de moyen de convertir un fichier `/etc/shadow` en base de mots de passe Kerberos.

Deux approches existent :

→ la réinitialisation des mots de passe : cette solution n'est pas forcément envisageable pour un grand nombre d'utilisateurs (forcer le mot de passe « toto » à tous et leur demander de le changer, alors qu'il y a des absences, des congés, etc., c'est quand même un peu risqué) ;

→ l'utilisation d'un module PAM de migration (`pam_migrate`) permettant de renseigner automatiquement la base Kerberos : cette méthode nécessite la création en local de chaque machine d'un fichier `keytab` permettant à la PAM d'accéder et de créer les différents utilisateurs.

Exemple de configuration :

```
login auth sufficient pam_krb5.so
login auth requisite  pam_unix_auth.so use_first_pass
login auth optional   pam_migrate.so keytab=/etc/krb5/keytab.migrate
login auth required   pam_krb5.so use_first_pass
```

Les points importants dans cet exemple sont :

→ le drapeau de contrôle du module `pam_unix_auth` est « requisite » : si l'authentification n'est pas autorisée par le mécanisme standard, le module `pam_migrate` n'est pas activé ;

→ après le peuplement de la base Kerberos, le module `pam_krb5` est de nouveau activé pour fournir un ticket à l'utilisateur dès sa première connexion suivant le début de la migration ;

Cette étape de peuplement doit être temporaire. Elle nécessite que le fichier `keytab` d'un principal « privilégié » soit stocké en local des machines. Il est préférable de créer un fichier `keytab` spécifique pour le module `pam_migrate` afin de pouvoir le supprimer plus facilement (un changement de configuration sur toutes les machines d'un réseau est suffisamment complexe comme cela).

À noter que cette solution (pratique) est non sûre dans le sens où elle importe dans Kerberos des mots de passe précédemment utilisés de manière non sûre. Elle doit, à tout le moins, être couplée avec une politique de changement de mot de passe rigoureuse.

3.2.2 Eradication des services non kerbérisés

La sécurité d'un SI valant celle de son maillon le plus faible, tant qu'il subsiste des services utilisant une méthode d'authentification faible, on peut considérer que ces efforts ont été vains. L'éradication des services non kerbérisés implique le traçage et l'adaptation de toutes les procédures et outils (`ufsdump`, `tar`, `cvs`,...) accédant ces services. Cette étape peut être fastidieuse et difficile à évaluer en termes de temps de travail.

3.3 Administration d'un réseau kerbérisé

L'administration d'un réseau kerbérisé fait notamment naître les problèmes suivants :

3.3.1 Installation d'une machine cliente

Pour un nombre réduit de machines, une solution non automatique est envisageable. Les fichiers de configuration sont facilement applicables par la procédure d'installation et il ne reste plus alors qu'à générer le fichier `keytab` en se connectant en local sur la machine et en utilisant la commande `kadmin` avec un principal privilégié.

Dans le cas d'un nombre important de machines, une installation complètement automatique peut être nécessaire. Le fichier `keytab` de la machine sera généré lors de la préparation de l'installation de ladite station. Le problème réside alors dans le transfert de ce fichier :

→ copie par NFS non kerbérisé ou équivalent ;

→ transfert par une application client/serveur : lors de la préparation de l'installation, le serveur pourrait vérifier l'adresse IP demandeur du transfert du fichier et écrire dans un journal l'action de transfert ;

Ce type de solution aboutit à une certaine exposition du fichier `keytab`.

3.3.2 Accès à un service sans mot de passe

Plusieurs tâches d'administration classiques nécessitent l'accès à un service sans fournir de manière interactive l'authentifiant de l'identité utilisée. De telles procédures sont bien évidemment difficilement compatibles avec une méthode d'authentification sûre.

→ **Dépannage des utilisateurs :** les administrateurs ont parfois besoin de prendre l'identité complète de l'utilisateur pour résoudre ses problèmes (au niveau UNIX et Kerberos). Or, ne connaissant pas le mot de passe de l'utilisateur, il est impossible d'obtenir un TGT à son nom pour prendre totalement son identité. On peut envisager plusieurs solutions :

→ utiliser le fichier `~/.k5login` (comparable au fichier `.rhosts`) ;

→ utiliser le cache de ticket d'une machine sur laquelle l'utilisateur est connecté ;

→ créer un sous-royaume d'administration avec une relation de confiance unilatérale.

→ **L'exécution de procédures automatiques :** il s'agit du cas, par exemple, d'un script exécuté par `cron` et souhaitant accéder à un service kerbérisé. Un moyen de résoudre ce problème est d'utiliser un principal spécifique possédant un fichier `keytab` sur la machine d'origine. Avant son exécution, le script acquiert les `credentials` de ce principal à l'aide du fichier `keytab` puis accède au service avec l'identité de l'utilisateur cible. Celui-ci aura dans son fichier `~/.k5users` les commandes exécutées par ce script. Ce fichier est équivalent au fichier `.k5login` mais n'autorise qu'une liste de commandes.

→ **Le fonctionnement en mode batch :** ce problème est proche du précédent. Plusieurs approches sont envisageables :

→ définition d'un royaume spécifique ;

→ utilisation des fichiers `.k5*` ;

→ génération au moment de la soumission, d'un ticket POSTDATED (cette solution n'est valable que dans des conditions bien spécifiques).

3.3.3 Autres

Cette partie dédiée aux difficultés spécifiques de l'administration Kerberos n'est pas exhaustive. Citons pour finir deux écueils classiques :

- **La gestion de l'autorisation :** Les possibilités de Kerberos dans ce domaine sont limitées. Il n'intègre pas de mécanisme d'autorisation générique et a donc du mal à satisfaire certains besoins. Par exemple, si vous interdisez à un groupe d'utilisateurs de se connecter à des machines spécifiques ou en dehors de plages horaires définies à l'aide d'un module PAM, le déploiement de `krlogind` fait « sauter » cette restriction.

- **Kerberos et le NAT :** par défaut, les tickets sont « adressés » c'est-à-dire que l'on vérifie qu'ils sont bien utilisés depuis la machine pour laquelle ils ont été émis (ces adresses sont visibles avec la commande `klist -a`). Dans certaines conditions et selon les implémentations, il est parfois difficile de faire cohabiter tickets adressés et translation d'adresse.

4 Clarifications et extensions du protocole

La RFC 1510, définissant le protocole Kerberos dans sa version 5 date de plus de dix ans (1993). Depuis, de nouveaux besoins (et défauts) sont apparus nécessitant de faire évoluer cette norme. Ces évolutions, coordonnées par un groupe de travail de l'IETF, sont regroupées en deux catégories : les clarifications et les extensions.

4.1 Les clarifications

Les clarifications rassemblent les spécifications des évolutions du protocole Kerberos pour lesquelles un consensus a été atteint. Elles apportent aussi des précisions sur certains points qui n'étaient pas complètement spécifiés dans la RFC 1510.

A l'inverse, certaines contraintes limitant de manière inutile les possibilités des développeurs ont été retirées. Les clarifications de Kerberos sont souvent référencées comme la RFC 1510bis car destinées à rendre la RFC 1510 obsolète.

Parmi les différences les plus notables avec la RFC 1510, on trouve :

- l'amélioration de l'utilisation des algorithmes de chiffrement et des calculs de `checksum`, les rendant plus conformes aux pratiques actuellement recommandées.
- la clé secrète d'un utilisateur n'est plus obligatoirement dérivée de son mot de passe. Par exemple, la pré-authentification matérielle est prévue (flag «HW-AUTHENT»).
- la possibilité d'inclure des données liées à l'autorisation dans un ticket Kerberos.
- la possibilité pour un client de déléguer son autorité à un service si celui-ci est estampillé comme « digne de délégation » par le KDC.

4.2 Les extensions

Les extensions sont un ensemble de documents, eux aussi sous forme de `drafts` de l'IETF. Elles constituent les options pour faire évoluer le protocole dont les spécifications sont encore en cours de définition. Par exemple, bien que mentionnée dans les clarifications, la possibilité d'utiliser des algorithmes de chiffrement à clé publique reste une extension [15]. Les extensions abordent aussi l'utilisation de mots de passe à usage unique [16] ou la standardisation du protocole de changement de mot de passe.

4.3 Kerberos et le chiffrement asymétrique : PKINIT

PKINIT (*Public Key Cryptography for Initial Authentication in Kerberos*) désigne l'extension du protocole Kerberos visant à intégrer les algorithmes de chiffrement à clés publiques dans la phase de pré-authentification. Les aspects touchant à la mise en œuvre d'une PKI ne seront pas discutés puisque ayant fait l'objet d'un dossier MISC [17]. On supposera donc que les utilisateurs possèdent une clé privée, protégée par un mot de passe et un certificat signé par une autorité de certification. Le certificat (éventuellement auto-signé) de cette autorité de certification est accessible à tous les principaux du royaume Kerberos.

4.3.1 Fonctionnement

- PKINIT intervient au niveau de la pré-authentification. Au cours de la requête de TGT, le client envoie un « pré-authentificateur » spécifique constitué de sa clé publique et d'une signature.
- Le KDC évalue cette requête en vérifiant que la clé publique du client est bien signée par l'une de ses autorités de confiance (dont il possède les certificats) et que cette clé publique permet de vérifier la signature.
- En cas de succès, le KDC répond avec le TGT et une clé de chiffrement symétrique signée avec la clé privée du KDC et chiffrée avec la clé publique du client.
- Le client déchiffre la réponse et procède ensuite normalement (demande de ticket de service, etc.).

Grâce à ce schéma, le partage d'un secret entre les clients et le KDC n'est plus nécessaire. Le fait d'introduire cette modification au niveau de la pré-authentification permet aussi de conserver le protocole intact au-delà de la requête de TGT.

4.3.2 Exemple

La seule implémentation de Kerberos supportant PKINIT de manière standard est celle fournie par Microsoft. Elle supporte le stockage des certificats et des clés dans une carte à puce, ainsi que l'utilisation du crypto-processeur. Cependant, cette implémentation se base sur une version ancienne du draft de l'IETF et ne suit donc pas les dernières spécifications. Néanmoins, il est possible de tester cette fonctionnalité avec les versions de développement de Heimdal par exemple.

Pour cela, il est nécessaire d'ajouter les lignes suivantes dans le fichier `krb5.conf` du KDC :

```
[kdc]
        enable-pkinit = yes
        pki-identity = FILE:/etc/MaCA/as.crt,/etc/MaCA/private/as.key
        # Certificat et clé privée du service AS du KDC
        pki-anchors = OPENSSL-ANCHOR-DIR:/etc/anchor
        # Répertoire contenant les certificats des autorités de certification
```

Et les lignes suivantes dans le fichier `krb5.conf` des clients :

```
[appdefaults]
        pkinit-anchors = OPENSSL-ANCHOR-DIR:/etc/anchor
        # Répertoire contenant les certificats des autorités de certification
```

La correspondance entre principal et certificat est donnée explicitement dans un fichier de configuration, par exemple `/var/heimdal/pki-mapping` :

```
paul@TEST.FR:/C=FR/ST=IDF/O=MISC Co/CN=paul/emailAddress=paul@test.fr
```

Si la clé secrète du KDC est protégée par un mot de passe, celui-ci est demandé au lancement du service :

```
kdc-master:~# /usr/local/sbin/kdc
Enter your private key passphrase:
```

Un client peut alors s'authentifier à l'aide de son certificat (`~/paul.crt`) et de sa clé privée (`~/private/paul.key`) :

```
paul@client:~$ kinit -C FILE:/home/paul/paul.crt,/home/paul/private/paul.key
Enter your private key passphrase:
kinit: NOTICE: ticket renewable lifetime is 1 week
paul@client:~$ klist
Credentials cache: FILE:/tmp/krb5cc_501
       Principal: paul@TEST.FR
  Issued           Expires          Principal
Jul 26 23:44:07  Jul 27 09:44:07  krbtgt/TEST.FR@TEST.FR
```

Dans les journaux du KDC, on peut voir :

```
2004-07-26T23:44:07 AS-REQ paul@TEST.FR from IPv4:192.168.1.1 for
krbtgt/TEST.FR@TEST.FR
2004-07-26T23:44:07 Looking for PKINIT pa-data - paul@TEST.FR
2004-07-26T23:44:07 PKINIT pre-authentication succeded - paul@TEST.FR using
/C=FR/ST=IDF/O=MISC Co/CN=paul/emailAddress=paul@test.fr
2004-07-26T23:44:07 Requested flags: renewable, proxiable, forwardable
2004-07-26T23:44:07 sending 2578 bytes to IPv4:192.168.1.1
```

5 Conclusion

Kerberos a longtemps eu une réputation d'usine à gaz difficile à administrer. La maturité des implémentations actuelles (surtout Open Source) permet de mitiger ce jugement. Même s'il est vrai que son déploiement peut représenter des efforts importants, une fois déployé, Kerberos permet d'accéder au saint Graal de l'authentification unifiée et sécurisée sur un réseau hétérogène. Il fournit une solution de *Single Sign On* complète : une fois authentifié, un utilisateur

peut rebondir de service en service (ou de machine en machine) de manière authentifiée sans avoir à fournir de nouveau son mot de passe. Allié à un service d'annuaire, Kerberos permet une gestion centralisée et extensible des utilisateurs. La portabilité de ce protocole est assurée par son adoption par plusieurs systèmes d'exploitation propriétaires (dont Windows) et la disponibilité de deux implémentations Open Source opérationnelles. Au niveau applicatif, son intégration est facilitée par le développement de standards construits sur la GSSAPI (NFSv4 [6], SPNEGO [19]), dont Kerberos est un mécanisme de sécurité sous-jacent.

Enfin, les évolutions à venir (PKINIT) associées au support des cartes à puce [16] laissent envisager un moyen (parmi d'autres) d'avoir une authentification forte avec Kerberos.

Références

[1] B. Clifford Neuman and T. Ts'o. *Kerberos : An Authentication Service for Computer Networks*, IEEE Communications, 32(9):33-38. September 1994

[2] J. Kohl, B. Clifford Neuman, and T. T'so. *The Evolution of the Kerberos Authentication System*. In Distributed Open Systems, pages 78-94. IEEE Computer Society Press, 1994

[3] J. Kohl and B. Clifford Neuman. *The Kerberos Network Authentication Service* (Version 5). RFC-1510. September 1993

[4] J. Linn. *The Kerberos Version 5 GSS-API Mechanism*. RFC 1964

[5] RFC 2203 : RFCSEC_GSS Protocol Specification

[6] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck. *Network File System (NFS) version 4 Protocol*. RFC 3530, Avril 2003

[7] **http://www.isi.edu/gost/info/kerberos/**

[8] Bill Bryant, Theodore Ts'o, *Designing an Authentication .System : a Dialogue in Four Scenes*

[9] Shumon Huque, ISC Networking & Telecommunications, University of Pennsylvania, *An Introduction to Kerberos*

[10] Needham, R., and M. Schroeder. *Using Encryption for Authentication in Large Networks of Computers, Communications of the ACM*, Vol. 21 (12), pp. 993-999, December 1978.

[11] E. Nemeth, G. Snyder, S. Seebass, and T. Hein. *Unix, Guide de l'administrateur*, CampusPress, 2001.

[12] **http://web.mit.edu/kerberos/www/**

[13] **http://www.pdc.kth.se/heimdal/**

[14] **http://sourceforge.net/projects/pam-krb5**

[15] PKINIT, **http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-20.txt**

[16] *Integrating Single-use Authentication Mechanisms with Kerberos*, INTERNET-DRAFT, draft-ietf-krb-wg-kerberos-sam-03.txt, Updates : RFC 1510, July 15, 2004, Ken Hornstein, Ken Renard, Clifford Newman, Glen Zorn

[17] Dossier PKI, MISC 13, mai-juin 2004

[18] Voir article sur PKCS#11 dans ce dossier

[19] E. Baize, D. Pinkas. *The Simple and Protected GSS-API Negotiation Mechanism*, RFC 2478, December 1998

Secure Remote Password Protocol

Ludovic FLAMENT
ludovic.flament@free.fr
Ingénieur R&D - NETASQ
Spécialiste cryptographie & PKI

Introduction

Le projet SRP est un projet Open Source qui a débuté en 1997 à l'université de Standford. L'initiateur du protocole, Thomas Wu [Wu01], voulait proposer un protocole d'authentification fondé sur des mots de passe et facilitant l'échange de clefs sur des réseaux non sécurisés tout en conservant les outils et protocoles réseaux existants (Telnet, FTP, etc.). Le protocole est également conçu pour résister aux attaques passives et actives sur les mots de passe.

En complément, il possède la propriété de *perfect forward secrecy*, qui « protège » les clefs de sessions en cas de compromission du mot de passe, et inversement, protège le mot de passe en cas de compromission d'une clef de session.

Enfin, le SRP génère une clef de session permettant la protection des échanges survenant après l'authentification (intéressant par exemple dans le cas d'une session Telnet).

Cette fonctionnalité très utile n'est pas nouvelle dans le monde des protocoles d'authentification. En effet, Bellovin et Merritt proposèrent les premiers une méthode d'authentification permettant de générer une clef de session. Ce protocole, connu sous le nom d'EKE « *Encrypted Key Exchange* », combine un algorithme de chiffrement symétrique et un algorithme de chiffrement asymétrique pour contrer les attaques par dictionnaire. Par la suite, Bellovin et Merritt proposèrent une version améliorée « *Augmented Encrypted Key Exchange* ». Nous renvoyons le lecteur à [BeMe] pour plus d'informations sur ce sujet.

Le protocole SRP décrit dans la RFC 2945 [RFC] est aujourd'hui l'un des meilleurs protocoles d'authentification à base de mot de passe disponibles. On peut noter que le SRP fait également partie du petit nombre d'algorithmes retenus dans les différents Drafts du standard IEEE-P1363.2 [IEEE].

Le protocole SRP s'appuie essentiellement sur l'algorithme de Diffie-Hellman. Nous allons faire un rappel sur ce dernier avant d'aborder en détail la mécanique du protocole.

Algorithme de Diffie-Hellman

Introduction

Whitfield Diffie et Martin Hellman sont les « inventeurs » de la cryptographie asymétrique. En 1976, ils sont les premiers à proposer l'utilisation d'un algorithme fondé sur un problème mathématique et disposant de deux clefs distinctes (clef privée & clef publique) liées entre elles par des propriétés mathématiques.

Ce problème mathématique qui se doit d'être « simple » lorsqu'on connaît un élément secret, mais très difficile dans l'autre cas, est le calcul du « logarithme discret » par rapport au calcul « d'exponentiation », les calculs étant faits dans un corps fini.

Il est ainsi facile dans un corps fini de calculer très rapidement la valeur Y définie par $Y = g^x \bmod n$ (g, x, n étant des entiers), mais il est très compliqué de retrouver x si on ne connaît que Y, g et n.

L'utilisation principale de cet algorithme est l'échange de clef. Il permet à deux entités de calculer une même clef de session (également appelée secrète), utilisable pour le chiffrement d'informations, sans la faire transiter sur le réseau.

Description de l'algorithme de Diffie-Hellman

Pour utiliser l'algorithme de Diffie-Hellman, les deux entités désirant s'échanger des clefs doivent tout d'abord s'accorder sur le corps servant à effectuer les calculs.

Ce corps est défini par deux nombres :

→ n le module du corps.

→ g un générateur du corps (une racine primitive modulo n).

Le choix des valeurs de n et g est important pour la sécurité du protocole. Il est impératif de respecter les points suivants :

→ n doit être un « grand » nombre premier (minimum recommandé : 1024 bits).

→ $(n-1)/2$ doit également être premier.

→ g doit être strictement supérieur à 1.

↘ **Remarque :** n et g sont des éléments publics et ne sont donc pas considérés comme sensibles.

Schéma

Regardons en détail le déroulement du protocole :

1. Alice génère une valeur aléatoire r_a, et envoie à Bob : $X = g^{ra} \bmod n$
2. Bob génère une valeur aléatoire r_b, et envoie à Alice : $Y = g^{rb} \bmod n$
3. Alice calcule sa clef de session : $S_a = Y^{ra} \bmod n$
4. Bob calcule sa clef de session : $S_b = X^{rb} \bmod n$

Alice et Bob disposent maintenant d'une même clef de session qu'ils utilisent pour échanger des messages chiffrés.

Prouvons qu'Alice et Bob possèdent la même clef de session, $(S_a = S_b)$:

$S_a = Y^{ra} \bmod n$	$S_b = X^{rb} \bmod n$
$S_a = (g^{rb} \bmod n)^{ra} \bmod n$	$S_b = (g^{ra} \bmod n)^{rb} \bmod n$
$S_a = g^{(rb * ra)} \bmod n$	$S_b = g^{(ra * rb)} \bmod n$

Pour plus d'informations sur l'utilisation de l'algorithme de Diffie-Hellman, nous renvoyons le lecteur à l'article paru dans un précédent numéro de MISC [Raynal].

Nous allons maintenant aborder en détail le protocole SRP.

Description du SRP

Le protocole SRP est construit sur les principes fondamentaux du protocole AKE (*Asymetric Key Exchange*). Dans ce protocole, la fonction primaire est d'échanger des clefs de session entre deux entités, et d'utiliser ces clefs pour vérifier que les deux parties connaissent le mot de passe.

A la différence du protocole EKE, on ne chiffre aucun des échanges dans le protocole AKE.

Les échanges chiffrés sont avantageusement remplacés par des échanges fondés sur les relations mathématiques liant les clefs éphémères engendrées et les mots de passe. Les avantages obtenus en ne chiffrant pas des échanges sont :

- Simplification du protocole en éliminant la nécessité de négocier un algorithme commun de chiffrement. La méthode alternative, fixant l'algorithme utilisé, entraîne une dépendance entre l'algorithme de chiffrement et le protocole.
- Si l'algorithme de chiffrement utilisé possède des faiblesses, alors le protocole risque d'en posséder également. De plus, si les mots de passe sont utilisés en tant que matériel pour la génération des clefs de chiffrement, le protocole est vulnérable à certains types d'attaques. Ce point sera développé plus loin.
- Dans certains cas, l'utilisation d'algorithmes de chiffrement engendre des limitations juridiques sur son utilisation, importation, exportation, taille de clefs utilisables, ... dans certains pays.

Les protocoles construits sur EKE imposent l'échange préalable de secrets. Ceci signifie que les deux parties gardent exactement la même chaîne de caractères secrète et l'emploient indirectement pour s'authentifier durant le protocole. Cela oblige les deux parties à être extrêmement prudentes dans la protection de ce secret.

Dans le cas d'AKE, chaque entité crée une valeur de vérification à partir de son mot de passe et l'envoie à l'autre entité. Il est important que l'attaquant n'entre pas en possession de cette valeur. Dans le cas contraire, même s'il n'est pas possible de s'authentifier vis-à-vis de l'autre partie directement, des attaques par dictionnaire restent possibles.

↘ **Remarque :** Dans le cas d'un serveur d'authentification, il est très avantageux que seule la valeur permettant la vérification soit gardée sur le serveur. Dans ce cas, le mot de passe n'est pas obligatoirement envoyé au serveur lors des procédures d'initialisation (création de la base des « logins/mots de passe » dans les autres protocoles). L'utilisateur peut créer la valeur de vérification sur son poste local et l'envoyer ensuite au serveur.

Etude « mathématique » du SRP

Comme nous l'avons indiqué précédemment, tous les aspects mathématiques du protocole SRP tournent autour de l'algorithme de Diffie-Hellman. Les calculs utilisés se font donc dans un corps de Galois : il faut choisir un module et un générateur définissant le corps sur lequel les calculs sont effectués.

Définitions

Nous allons dans le tabelau ci-dessous définir les notations utilisées lors des explications et de la description du protocole :

n	module du groupe de Galois, ***n*** est un grand nombre premier (minimum recommandé : 1024 bits).
g	générateur du groupe de Galois GF (***n***).
salt	valeur aléatoire permettant de dériver le mot de passe de l'utilisateur (cette valeur est appelée graine).
password	mot de passe de l'utilisateur.
P	fonction de vérification du mot de passe, définie par : $P(x) = g^x \bmod n$.
H	une fonction dérivée des algorithmes de hash (SHA1, MD5, etc.) et fournissant une clef de session de 320 bits.
u	valeur aléatoire utilisée pour contrer certains types d'attaques.
v	valeur de vérification, générée par l'utilisateur et connue du vérifieur, elle permet à ce dernier d'authentifier l'utilisateur.

Pour générer l'élément servant à l'authentifier, Alice calcule préalablement la valeur de vérification v :

$x = H(salt \,||\, H(login||password))$

$v = P(x) = g^x \bmod n$.

↘ **Remarques :**

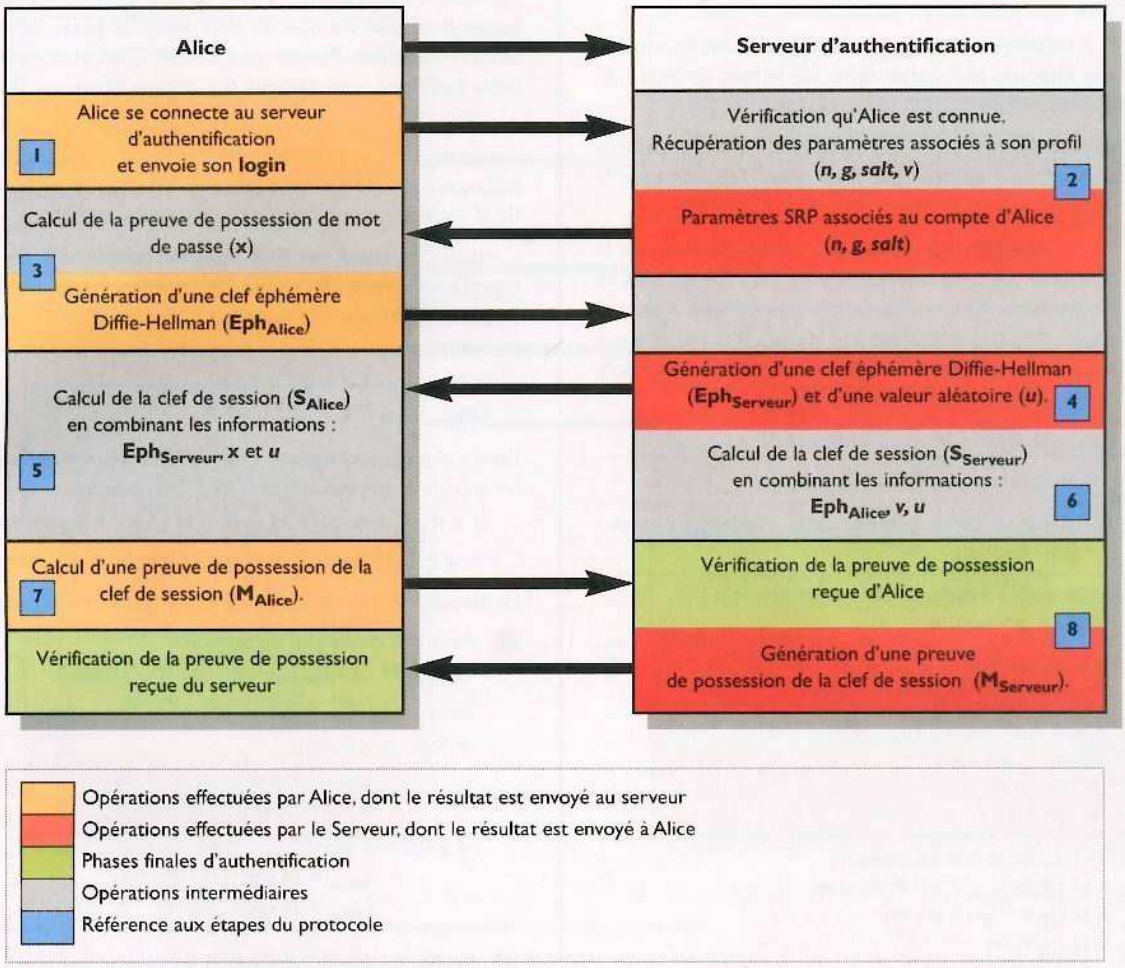
- Le serveur conserve le quadruplet (***n, g, salt, v***) spécifique à chaque utilisateur afin de l'authentifier durant le déroulement du protocole d'authentification.
- Il est possible qu'une communauté d'utilisateurs partage les valeurs « ***n*** » et « ***g*** » sans pour autant altérer la sécurité du protocole.

Déroulement du protocole SRP

Examinons en détail les différentes étapes du protocole. Nous supposons qu'Alice tente de s'authentifier sur le serveur de sa société : cf ci-contre **Schéma d'authentification SRP**

1. Alice envoie son login utilisateur au Serveur : Alice
2. Le serveur d'authentification vérifie qu'Alice fait partie des utilisateurs de la société et, dans ce cas, envoie les paramètres associés au compte d'Alice : ***salt, g, n***
3. Alice calcule sa preuve de possession du mot de passe : $x = H(salt \,||\, H(Alice \,||\, password))$
   Alice génère une valeur aléatoire « r_a » et une clef éphémère
   $Eph_{Alice} = g^{ra} \bmod n$, et l'envoie au serveur : Eph_{Alice}
4. Le serveur génère deux valeurs aléatoires « r_b » et « ***u*** », ainsi qu'une clef éphémère $Eph_{Serveur} = v + g^{rb} \bmod n$, il envoie à Alice : $Eph_{Serveur}$, ***u***.

↘ Schéma d'authentification SRP



5 Alice calcule sa clef de session :
$S_{Alice} = H((Eph_{Serveur} - g^x)^{ra + (u * x)} \bmod n)$

6 Le serveur calcule sa clef de session :
$S_{Serveur} = H((Eph_{Alice} * v^u)^{rb} \bmod n)$

7 Alice calcule une valeur de preuve de possession de la clef de session et envoie le résultat au serveur :
$M_{Alice} = H(Eph_{Alice} \| Eph_{Serveur} \| S_{alice})$

8 Le serveur vérifie que la valeur M_{Alice} est correcte et envoie à son tour une preuve de possession de la clef de session :
$M_{Serveur} = H(Eph_{Alice} \| M_{Alice} \| S_{serveur})$

9 Alice vérifie que la valeur $M_{Serveur}$ est correcte. Dans ce cas, le protocole est terminé, et les deux entités peuvent communiquer dans un canal sécurisé en utilisant comme clef de chiffrement :
$S = S_{Alice} = S_{Serveur}$

Démontrons que lors des étapes **5** et **6**, Alice et le serveur génèrent effectivement la même clef de session, c'est-à-dire que $S_{Alice} = S_{Serveur}$.

Nous partons du principe que tous les calculs sont effectués modulo n dans la démonstration.

$S_{Alice} = (Eph_{Serveur} - g^x)^{ra + (u * x)}$
$S_{Alice} = (v + g^{rb} - g^x)^{ra + (u * x)}$
$S_{Alice} = (g^x + g^{rb} - g^x)^{ra + (u * x)}$
$S_{Alice} = (g^{rb})^{ra + (u * x)}$
$S_{Alice} = g^{(rb * ra) + (rb * u * x)}$

$S_{Serveur} = (Eph_{Alice} * v^u)^{rb}$
$S_{Serveur} = (g^{ra} * v^u)^{rb}$
$S_{Serveur} = (g^{ra} * g^{(x * u)})^{rb}$
$S_{Serveur} = (g^{ra + (u * x)})^{rb}$
$S_{Serveur} = g^{(ra * rb) + (u * x * rb)}$
$S_{Serveur} = g^{(rb * ra) + (rb * u * x)}$

Sécurité du protocole

A l'étape **4**, le fait de combiner v et g^{rb} lors de la génération de la clef éphémère $Eph_{Serveur}$ sert à contrer les attaques par dictionnaire sur le mot de passe d'Alice.

Il serait effectivement envisageable de monter une telle attaque en obtenant la valeur ***v***, cette dernière étant dérivée d'information publique (login, ***g***, ***n***, ***salt***) et d'une information privée (le mot de passe du client).

La valeur « r_b » aléatoire n'étant jamais divulguée, il est impossible de monter des attaques par dictionnaire sur le mot de passe du client en interceptant $Eph_{Serveur}$ sur le réseau.

Regardons maintenant quel est le rôle de la valeur « ***u*** », présentée comme étant une valeur contrant certains types d'attaques sur le protocole.

La valeur « ***u*** » empêche les attaques intervenant lors de la compromission de la valeur de vérification « ***v*** » du client. Dans ce cas, et dans l'hypothèse où une attaque par dictionnaire n'est pas envisageable (mot de passe correctement choisi), si la valeur « ***u*** » n'est pas utilisée ou si elle était fixe, une attaque simple peut être menée par Estelle, notre espionne de service ;-)

- Estelle envoie le login d'Alice au Serveur (Alice)
- Le serveur envoie la graine d'Alice à Estelle (A_{salt})
- Estelle construit sa clef éphémère de la manière suivante :
  $Eph_{Estelle} = g^{ra}\ v^{-u} \bmod n$,
- Le serveur envoie à Estelle sa clef éphémère et ***u*** :
  $Eph_{Serveur} = v + g^{rb} \bmod n$, ***u***
- Estelle calcule la clef de session :
  $S_{Estelle} = H((Eph_{Serveur} - g^x)^{ra} \bmod n)$
  $S_{Estelle} = H((v + g^{rb} - g^x)^{ra})$
  $S_{Estelle} = H((g^x + g^{rb} - g^x)^{ra})$
  $S_{Estelle} = H((g^{rb})^{ra})$
  $S_{Estelle} = H(g^{rb * ra})$
- Le serveur calcule sa clef de session :
  $S_{Serveur} = H((Eph_{Estelle} * v^u)^{rb} \bmod n)$
  $S_{Serveur} = H((g^{ra} * v^{-u} * v^u)^{rb})$
  $S_{Serveur} = H((g^{ra})^{rb})$
  $S_{Serveur} = H(g^{ra * rb})$

> **Remarque :** Il est indispensable que le serveur attende d'avoir reçu la clef éphémère du client avant de lui envoyer la valeur « ***u*** ». Dans le cas contraire, l'attaque fonctionne également.

La sécurité du protocole repose également sur les opérations mathématiques effectuées durant le déroulement de ce dernier. Nous listons les contraintes devant être vérifiées par les deux entités sans entrer dans le détail :

→ Le module « ***n*** » est un nombre premier sûr, d'une taille suffisante pour résister à d'éventuelles « attaques mathématiques ».

→ (***n***-1)/2 est un nombre premier.

→ ***g*** est une racine primitive de GF (***n***).

→ Les clefs éphémères sont non nulles.

→ Les valeurs aléatoires générées sont plus grandes que $\log_{[g]}(n)$.

Malgré l'aspect très sécurisé du SRP, une attaque active a été découverte et une nouvelle version du protocole contrant cette dernière a vu le jour en octobre 2002. Cette version connue sous le nom de SRP-6 [Wu02] est celle actuellement utilisée (ou du moins celle qui devrait l'être !).

Attaque active par dictionnaire sur le SRP

L'attaque découverte sur le SRP est difficile à mettre en œuvre et réclame la contribution active de la « victime ». Elle permet de tester la validité ou non de deux mots de passe lors d'une seule tentative d'authentification de l'attaqué. C'est pour cette raison que nous qualifions cette attaque d'« attaque active par dictionnaire ».

Il est également à noter que l'attaque n'est réalisable que si les paramètres (***n***, ***g***, ***salt***) sont échangés durant le protocole et que l'attaquant réussit à se faire passer pour le serveur d'authentification de la société, ce qui n'est pas toujours évident !

L'attaque se situe sur le « point sensible » du protocole que représente l'étape 4. Durant cette phase, Estelle teste deux mots de passe d'Alice en une seule fois. Cela est dû au fait que l'équation utilisée pour générer la clef éphémère du serveur est symétrique :

$Eph_{Serveur} = v + g^{rb} \bmod n$
$Eph_{Serveur} = g^x + g^{rb} \bmod n$

Estelle, durant son attaque, construit les deux valeurs g^{p1} et g^{p2} et les substitue aux valeurs de ***v*** et g^{rb} en prenant :

→ $v = g^{p1}$ avec $p1 = H(salt \| H(Alice \| password1))$

→ $g^{rb} = g^{p2}$ avec $p2 = H(salt \| H(Alice \| password2))$

Le déroulement du protocole, à partir de l'étape 5, devient alors :

5 Alice calcule sa clef de session :
$S_{Alice} = H((Eph_{Serveur} - g^x)^{ra + (u * x)} \bmod n)$
$S_{Alice} = H((g^{p1} + g^{p2} - g^x)^{ra + (u * x)})$

→ Si $p1 = x$ alors :
$S_{Alice} = H((g^{p2})^{ra + (u * p1)})$
$S_{Alice} = H(g^{(p2 * ra) + (p2 * u * p1)})$

→ Si $p2 = x$ alors :
$S_{Alice} = H((g^{p1})^{ra + (u * p2)})$
$S_{Alice} = H(g^{(p1 * ra) + (p1 * u * p2)})$

6 Estelle calcule ses deux clefs de session comme suit :

→ $p1 = x$:
$S_{Estelle1} = H((Eph_{Alice} * g^{(p1 * u)})^{p2} \bmod n)$
$S_{Estelle1} = H((g^{ra} * g^{(p1 * u)})^{p2})$
$S_{Estelle1} = H(g^{(ra + (p1 * u))\ p2})$
$S_{Estelle1} = H(g^{(ra * p2) + (p1 * u * p2)})$

→ $p2 = x$:
$S_{Estelle2} = H((Eph_{Alice} * g^{(p2 * u)})^{p1} \bmod n)$
$S_{Estelle2} = H((g^{ra} * g^{(p2 * u)})^{p1})$
$S_{Estelle2} = H(g^{(ra + (p2 * u))\ p1})$
$S_{Estelle2} = H(g^{(ra * p1) + (p2 * u * p1)})$

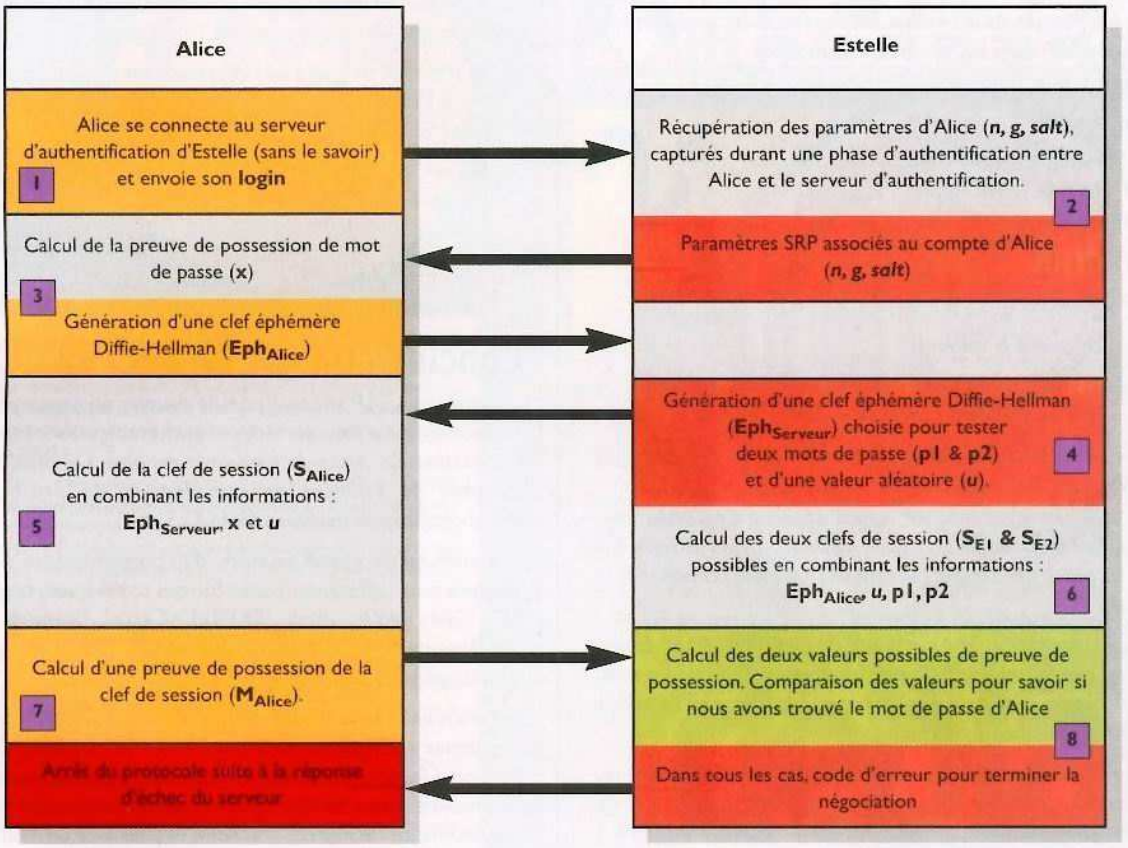
7 Alice calcule une valeur de preuve de possession de la clef de session et envoie le résultat à Estelle :
$M_{Alice} = H(Eph_{Alice} \| Eph_{Serveur} \| S_{Alice})$

8 Estelle vérifie si elle a trouvé le mot de passe d'Alice en testant :

→ Si $p1 = x$, alors :
$M_{Alice} = H(Eph_{Alice} \| Eph_{Serveur} \| S_{Estelle1})$

→ Si $p2 = x$, alors :
$M_{Alice} = H(Eph_{Alice} \| Eph_{Serveur} \| S_{Estelle2})$

↘ Schéma du protocole durant l'attaque par dictionnaire



Si Estelle trouve le mot de passe d'Alice, elle lui renvoie un échec d'authentification et stoppe l'attaque. Alice pourra alors s'authentifier sur le véritable serveur.

Sinon, elle lui renvoie également un échec d'authentification, et continue l'attaque avec deux nouveaux mots de passe. Cela n'étant possible que si Alice s'obstine même après plusieurs échecs.

Contre-mesure

L'attaque est uniquement réalisable parce que l'équation de l'étape 4 est symétrique. La parade consiste simplement à casser cette symétrie. Thomas Wu propose comme solution :

→ remplacer l'équation de l'étape 4 par :
$Eph_{Serveur} = 3 * v + g^{rb} \pmod{n}$.

→ remplacer l'équation de l'étape 5 par :
$S_{Alice} = H((Eph_{Serveur} - 3 * g^{x})^{ra + (u * x)} \pmod{n})$.

↘ Remarques :

- Le facteur 3 est le plus petit facteur utilisable, mais il n'est pas le seul, tout facteur répondant aux conditions suivantes est utilisable :
  - être un résidu quadratique de *n*.
  - ne pas être la puissance d'un entier.
- Dans la version SRP-6, Thomas Wu propose également des modifications dans l'ordonnancement des différents échanges afin d'améliorer les performances du protocole.

Utilisation du protocole

Le protocole SRP est aujourd'hui de plus en plus utilisé en complément sécuritaire dans divers outils standards. De nombreux Drafts ont été rédigés dans ce sens, on peut citer les propositions d'utilisation du SRP dans les protocoles suivants :

→ Serveur Telnet: telnetd

→ Client Telnet :
- Secure NetTerm
- The Java Telnet Applet
- DataComet

→ FTP: ProFTPD

→ SSL/TLS : gnutls

→ SSH : LSH client & server

Pour obtenir les références de ces exemples et bien d'autres, vous pouvez consulter le site Web consacré au SRP [SRP01].

Lors d'une authentification via le protocole SRP, les éléments de sécurité suivants sont respectés :

- Aucune information utile sur le mot de passe n'est transmise. Il est impossible pour un attaquant de monter une attaque sur le mot de passe par simple capture des paquets constituant le protocole.
- Aucune information utile sur la clef de session n'est transmise. En effet, la clef de session est une clef cryptographique forte au lieu d'un simple mot de passe (souvent faible et peu résistant aux attaques par dictionnaire).
- Même si un attaquant possède la capacité de modifier ou de créer ses propres paquets et de les faire passer pour ceux de l'une des deux entités, la sécurité du protocole interdit à l'attaquant de s'authentifier sur le serveur ou de déduire des informations sur les mots de passe ou les clefs de session.
- Le protocole dispose d'une « contre-mesure » vis-à-vis des DoS. En effet, les calculs « lourds » sont toujours effectués par le client en premier. Le serveur n'effectue pas de calculs lourds si le client n'est pas apte à prouver qu'il possède le secret.
- Si la base contenant les mots de passe utilisateurs est compromise, c'est-à-dire que l'attaquant dispose entre autres de la valeur de vérification « *v* », cela ne devrait pas aider l'attaquant à s'authentifier sans effectuer une attaque par dictionnaire sur la valeur afin de récupérer le mot de passe originel.
- Si une clef de session est compromise, elle n'aide pas l'attaquant à trouver le mot de passe de l'utilisateur.
- Si le mot de passe de l'utilisateur est compromis, il ne permet pas à l'attaquant d'en déduire les valeurs des clefs de session passées.
- Une attaque dite de « l'homme au milieu » est uniquement possible dans le cas où l'attaquant possède le mot de passe d'Alice.

Conclusion

Le protocole SRP est en passe de devenir un protocole de référence pour les services d'authentification, lorsque l'utilisation de mots de passe est requise. De plus, il est possible de l'utiliser en complément de protocoles d'authentification existants (SSH, TLS, etc.).

A noter qu'un grand nombre d'implémentations libres existent pour différentes plates-formes et différents langages (C, C++, JAVA, etc.) [SRP02]. Cette diversité des implémentations contribue également à l'utilisation et au développement du SRP.

Signalons enfin qu'une version du protocole à base de courbes elliptiques est intégrée au Draft IEEE-P1363.2 [IEEE]. Cette version a l'avantage d'employer des clefs plus courtes pour un niveau de sécurité identique, et une consommation moins importante en ressources (mémoire et puissance processeur).

Les courbes elliptiques sont aujourd'hui très appréciées dans les environnements nomades disposant de peu de ressources (PDA, Téléphone Portables, etc.). De ce fait, il est envisageable que cette version du SRP, à base de courbes elliptiques, soit massivement utilisée dans les applications développées pour ces environnements nomades et nécessitant de l'authentification mutuelle.

Références

[BeMe] : *Augmented Encrypted Key Exchange*, S. M. Bellovin and M. Merritt (1993).

[IEEE] : *IEEE-P1363.2 :Specifications for Public Key Cryptography: Password-based Techniques.*

[Raynal] : *Echange de clé selon Diffie-Hellman.* Frédéric Raynal MISC, 10 : p17-19 (Novembre/Décembre 2003).

[RFC] : *RFC 2945*, **http://www.ietf.org/rfc/rfc2945.txt**

[SRP01] : **http://srp.stanford.edu/links.html**

[SRP02] : **http://srp.stanford.edu/download.html (API C)**
http://www.cryptix.org (API JAVA)
http://srp.stanford.edu/demo/demo.html (JavaScript)

[Wu01] : *The Secure Remote Password Protocol.* Thomas Wu (Novembre 1997), **http://srp.stanford.edu/doc.html**.

[Wu02] : *SRP-6:Improvements and Refinements to SRP Protocol.* Thomas Wu (Octobre 2002), **http://srp.stanford.edu/doc.html**.

Heap de Windows : structure, fonctionnement et exploitation

Introduction

Force est de remarquer que les vulnérabilités des systèmes d'exploitation Microsoft sont nombreuses (et de plus en plus recherchées) et leur exploitation a souvent des conséquences désastreuses pour la sécurité d'un réseau ou la santé d'Internet en général. Il est tout de même nécessaire de mitiger ce constat par une remarque simple : la très grande majorité des vers ayant ciblé dernièrement certaines versions de Windows reposait sur une vulnérabilité de type débordement de buffer dans la pile (« stack overflow »), citons par exemple Blaster (MS03-026), Slammer (MS02-061), Sasser (MS04-011), Witty.

Qu'en est-il de la seconde vulnérabilité RPC/DCOM (MS03-039), de la vulnérabilité du Windows Messenger Service (MS03-043), la vulnérabilité ASN.1 (MS04-007) ? Certes, des programmes permettant de compromettre des systèmes non mis à jour existent, mais leur fiabilité est douteuse : ils nécessitent en général une connaissance au *service pack* près (voire au *patch*) des systèmes à attaquer, dépendent de la régionalisation de l'OS et par conséquent peuvent difficilement être utilisés dans le cadre d'outils automatisés de compromission ou de virus. La raison ? Ces vulnérabilités sont de type débordement de buffer dans le tas (« heap overflow »), un domaine largement parcouru dans le monde UNIX, mais qui reste encore très obscur en ce qui concerne Windows.

Quelques papiers ont été publiés couvrant les techniques de base utilisables lorsqu'est impliqué le heap de Windows [LITCHFIELD] [AITEL] [HALVAR] et les connaissances à en tirer sont somme toute limitées. Cependant, durant la conférence CanSecWest 2004 [RUIU], une analyse poussée du fonctionnement du heap de Windows a été présentée par Oded Horovitz et Matthew Conover [HOROVITZ], ainsi que des façons possibles d'exploiter de manière fiable des « heap overflows » sous Windows de façon indépendante du Service Pack ou de la régionalisation (elle justifiait à elle seule d'être présent à la conférence...).

Les implications sont nombreuses, puisque s'ouvre aux plus motivés un pan encore peu exploré de l'exploitation de certains types de débordement de tampon sous Windows.

Cet article, très largement fondé sur le travail de Horovitz et Conover, développera des aspects de la gestion des tas de Windows [**note** : les plates-formes concernées sont les Windows à noyau NT : Windows NT 4.0, Windows 2000, Windows XP (hors SP2), de légères différences pouvant apparaître d'un environnement à un autre - Windows XP SP2 et Windows 2003 ont des protections supplémentaires], et des moyens de les détourner pour aboutir à des exploitations fiables de « heap overflows ».

La structure des tas sous Windows

Attention : en mode *debug*, les tas ont une structure différente ; pour appliquer correctement ce qui est présenté dans cet article, il vous faudra attacher des processus en cours d'exécution, et non les exécuter directement sous un *debugger*. Je n'aborderai pas les différences en question, libre au lecteur de creuser ces aspects.

Tas et processus

Tout processus voit coexister dans son espace mémoire un ou plusieurs tas qui accueilleront les données allouées dynamiquement au cours de son exécution. Deux fonctions exportées par la bibliothèque `kernel32.dll` permettent d'obtenir les *handles* (correspondant aussi à l'adresse de base) des différents tas du processus :

→ `GetProcessHeap()` renvoie le handle du tas par défaut ;

→ `GetProcessHeaps()` renvoie un tableau de handles de tas dont le premier sera toujours le tas par défaut du processus.

En regardant un peu plus en détail leur contenu et celui de `RtlGetProcessHeaps()` dans `ntdll.dll`, on notera que lesdites fonctions se contentent de lire des pointeurs et entiers situés dans le Process Environnement Block. Le PEB est une structure spécifique au processus courant (voir notre encadré « Structure du PEB » page 65), gérée par le système d'exploitation, et contenant des variables nécessaires à son exécution. Parmi les éléments pertinents, le handle du tas par défaut situé à l'offset 0x18 du PEB, le nombre de tas à l'offset 0x88, l'adresse du tableau des tas existants à l'offset 0x90.

Ceci peut être illustré par les quelques lignes de C suivantes :

```
int i;
PDWORD PPEB = (PDWORD)0x7ffdf000;
      // Adresse statique du PEB
HANDLE ProcessHeap = (HANDLE)(*(PPEB + 0x18/4));
      // Tas par défaut du processus
DWORD NumberOfHeaps = *(PPEB + 0x88/4); // Nombre de tas
PHANDLE ProcessHeapsListBuffer = (PHANDLE)(*(PPEB + 0x90/4));
      // Tableau des tas du processus
fprintf(stdout, «ProcessHeap :%08x\nNumberOfHeaps : %d\nProcessHeapsListBuffer :
%08x\nHeaps : «,
   ProcessHeap,NumberOfHeaps, ProcessHeapsListBuffer);
for (i = 0; i < NumberOfHeaps; i++)
   fprintf(stdout, «%08x «, *(ProcessHeapsListBuffer + i));
fprintf(stdout, «\n»);
```

❑ *Exemple d'exécution*

```
ProcessHeap :00140000
NumberOfHeaps : 3
ProcessHeapsListBuffer : 77fb5a80
Heaps : 00140000 00240000 00250000
```

A titre de rappel, l'adresse du PEB, fixée à 0x7ffdf000 dans les versions de Windows étudiées, se retrouve via les instructions assembleur :

Kostya Kortchinsky
kostya.kortchinsky@renater.fr
Responsable du CERT RENATER

Figure 1 *Structure de gestion du tas*

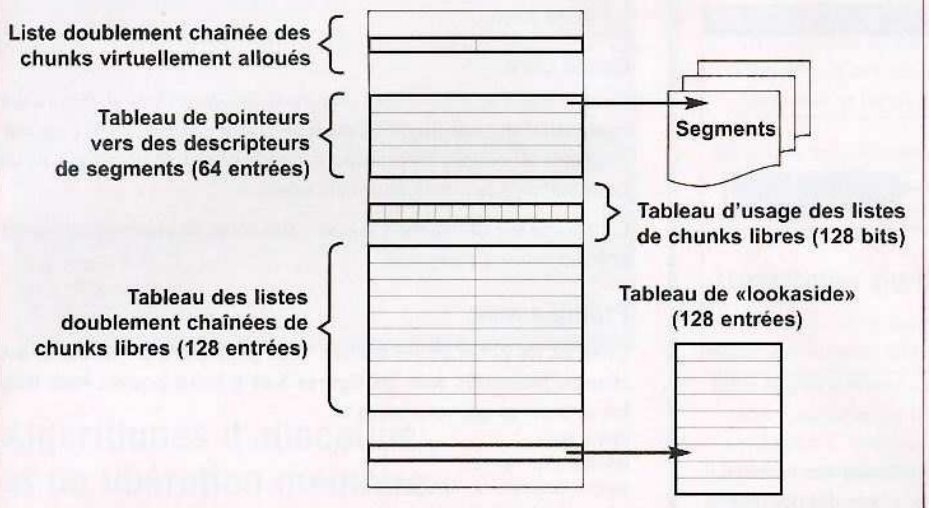


Figure 2 *Tableau des listes doublement chaînées de chunks libres*

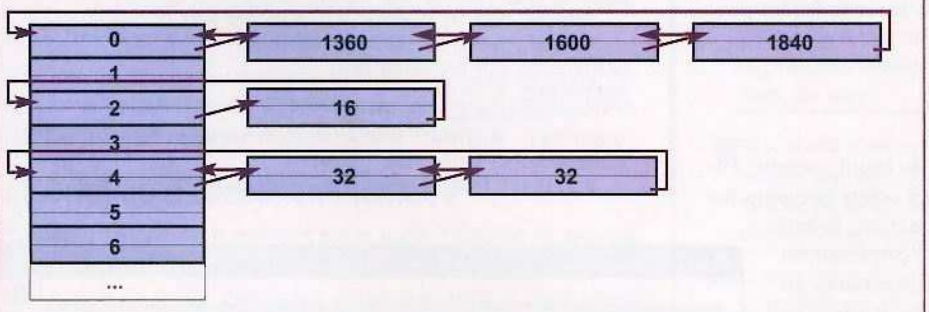


Figure 3 *Tableau d'usage des listes de chunks libres*


```
mov     eax, large fs:18h
mov     eax, [eax+30h]
```

ou bien par exemple grâce à la fonction `RtlGetCurrentPeb()` exportée par `ntdll.dll` sous Windows XP et ultérieur.

Il est possible de créer de nouveaux tas dans le contexte du processus courant via l'API `HeapCreate()` de `kernel32.dll`, les variables du PEB étant alors modifiées en conséquence, et bien évidemment de détruire un tas via l'API `HeapDestroy()`.

Structure du tas

Lors de la création d'un tas, un gros segment (64 * 4096 octets par défaut) [**note** : comprenez des zones de mémoire virtuelle contiguës destinées à être utilisées par le tas] est tout d'abord réservé via l'API `ZwAllocateVirtualMemory()` et seulement une petite partie (4096 octets par défaut) est allouée. C'est dans cette dernière que sera stockée la structure de gestion du tas, comprenant de nombreuses variables nécessaires aux allocations, accès et libérations de mémoire. L'ensemble n'étant pas documenté, il est assez complexe d'en établir un tableau exhaustif suite à une analyse superficielle des bibliothèques dynamiques de Windows, néanmoins, certains de ces éléments (dont l'organisation est présentée en Figure 1) méritent que l'on s'attarde sur eux.

- la liste des *chunks* virtuellement alloués : lorsque la taille d'allocation demandée est supérieure à un seuil déterminé (d'autres conditions peuvent ou doivent être respectées, mais je ne m'attarderai pas là-dessus), un nouvel espace mémoire est virtuellement alloué afin de satisfaire la demande, celui-ci étant ajouté par la suite à la liste en question ; il s'agit d'une liste dynamique doublement chaînée.

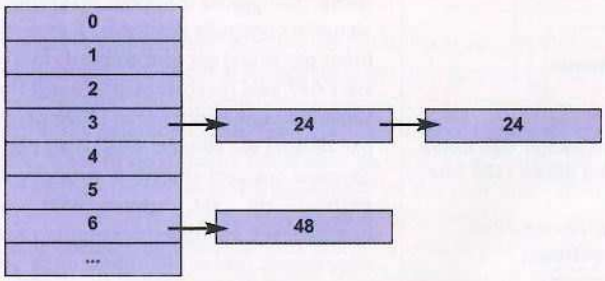
- le tableau des segments du tas : d'une taille fixe (64 entrées), il regroupe les informations nécessaires à la gestion de chaque segment : adresse de base, nombre de pages, première entrée dans le tas, dernière entrée dans le tas, etc.

- le tableau des listes de chunks libres : 128 listes doublement chaînées référençant les chunks libres disponibles ; la taille des chunks en octets dans chaque liste correspond à l'indice de la liste dans le tableau multiplié par 8, à l'exception de l'indice 0 qui contient une liste de chunks libres dont la taille est supérieure ou égale à 1024, et strictement inférieure au seuil d'allocation virtuelle, triée du plus petit chunk au plus gros (Figure 2).

- le tableau d'indication d'usage des listes de chunks libres : d'une taille de 128 bits, il implémente un moyen rapide de déterminer si des chunks libres sont disponibles dans les listes précédemment explicitées pour la taille désirée (Figure 3) ; lorsqu'un bit du tableau est positionné à 1, cela signifie que la liste correspondante est non vide.

■ la table de « lookaside » : élément des plus intéressants (qui présente l'inconvénient de ne pas être créé dans tous les cas), elle contient des listes simplement chaînées de chunks « occupés », à savoir récemment libérés et pouvant être réalloués immédiatement. Le tout est optimisé pour un accès rapide (la profondeur est limitée) et représente la méthode privilégiée de libération et d'allocation mémoire comme sera détaillé par la suite (Figure 4).

Figure 4 *Tableau de « lookaside »*



Structure d'un chunk

Les éléments essentiels de gestion du tas maintenant connus, il convient de regarder plus en détail ce qui est à la base des opérations d'allocation et de libération de mémoire : le « chunk ». Il se présente sous deux formes (on met de côté le cas des chunks alloués virtuellement), l'une correspondant à un morceau de mémoire alloué, l'autre à un morceau de mémoire libéré, la différence résidant dans la présence de deux pointeurs dans la seconde forme, pour des tailles respectives de 8 et 16 octets.

Chunk alloué

Descriptif des champs :

■ « `Size` » - USHORT, 2 octets : taille du chunk courant. Elle correspond à un nombre de cellules de 8 octets occupées par le chunk. Lors de la demande d'allocation, la taille demandée est arrondie à un multiple de 8 (supérieur ou égal), auquel s'ajoutent les 8 octets nécessaires au stockage des champs du chunk. La taille stockée ici répond donc à l'opération (tailledemandee + 7) >> 3 + 1, soit 65 pour 512 octets, 18 pour 129, etc. ;

■ « `PreviousSize` » - USHORT, 2 octets : taille du chunk précédent, possède les mêmes caractéristiques que « `Self size` » ;

■ « `SegmentIndex` » - UCHAR, 1 octet : indice du segment auquel appartient le chunk, en référence au tableau des segments du tas ;

■ « `Flags` » - UCHAR, 1 octet : indicateurs des propriétés du chunk, chaque bit positionné à 1 de cet octet faisant état d'une caractéristique particulière du chunk, ils peuvent être trouvés sur le Web (les plus utiles seront détaillés par la suite) :

- `0x01, HEAP_ENTRY_BUSY`
- `0x02, HEAP_ENTRY_EXTRA_PRESENT`
- `0x04, HEAP_ENTRY_FILL_PATTERN`
- `0x08, HEAP_ENTRY_VIRTUAL_ALLOC`
- `0x10, HEAP_ENTRY_LAST_ENTRY`
- `0x20, HEAP_ENTRY_SETTABLE_FLAG1`
- `0x40, HEAP_ENTRY_SETTABLE_FLAG2`
- `0x80, HEAP_ENTRY_SETTABLE_FLAG3`

■ « `UnusedBytes` » - UCHAR, 1 octet : nombre d'octets non utilisés dans le buffer alloué, égal à la taille allouée à laquelle est soustraite la taille demandée - information d'une utilité très relative ;

■ « `SmallTagIndex` » - UCHAR, 1 octet : uniquement utilisé en mode debug.

Chunk libre

Dans le cas d'un chunk libre (Figure 6), le début de la structure est rigoureusement identique à celle d'un chunk occupé ; vient s'ajouter à celle-ci un couple de pointeurs vers le chunk libre suivant et un pointeur vers le chunk libre précédent.

Ce couple est utilisé dans le cadre des listes doublement chaînées précédemment évoquées.

Pratiquement

L'extrait de programme suivant vous permettra de visualiser les champs présentés dans les Figures 5 et 6 (vous pouvez vous faire les structures qui vont bien) :

```
HANDLE hHeap;
unsigned char *buffer;
hHeap = HeapCreate(0, 4096, 65536);
      // Création d'un tas
fprintf(stdout, «*** HeapAlloc ***\n»);
buffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, 512);
      // Allocation de 512 octets sur ce tas
fprintf(stdout, «Self size : %d\nPrevious size : %d\nSegment index : %d\nFlags :
%d\nUnused bytes : %d\nTag index : %d\n»,
   *(unsigned short int *)(&buffer[-8]), *(unsigned short int *)(&buffer[-6]),
   buffer[-4], buffer[-3], buffer[-2], buffer[-1]);
fprintf(stdout, «*** HeapFree ***\n»);
HeapFree(hHeap, 0, buffer);
      // Libération du tampon alloué
fprintf(stdout, «Self size : %d\nPrevious size : %d\nSegment index : %d\nFlags :
%d\nUnused bytes : %d\nTag index : %d\n»
   «Next chunk : 0x%08x\nPrevious chunk : 0x%08x\n»,
```

Figure 5 *Schéma d'un chunk occupé*

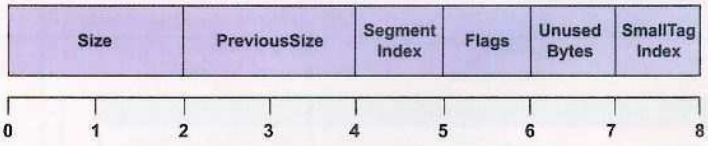
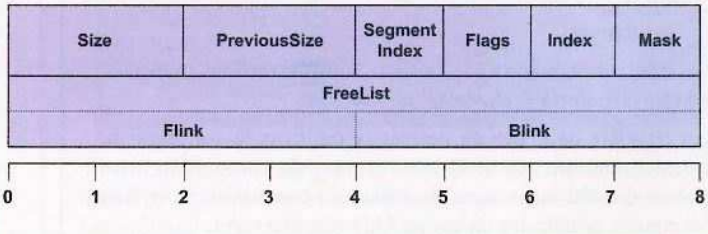


Figure 6 *Schéma d'un chunk libre*



```
   *(unsigned short int *)(&buffer[-8]), *(unsigned short int *)(&buffer[-6]),
   buffer[-4], buffer[-3], buffer[-2], buffer[-1],
   *(unsigned long int *)(&buffer[0]), *(unsigned long int *)(&buffer[4]));
HeapDestroy(hHeap);
     // Destruction du tas
```

❑ *Exemple d'exécution*

```
*** HeapAlloc ***
Self size : 65
Previous size : 8
Segment index : 0
Flags : 1
Unused bytes : 8
Tag index : 0
*** HeapFree ***
Self size : 304
Previous size : 8
Segment index : 0
Flags : 16
Unused bytes : 8
Tag index : 0
Next chunk : 0x00320178
Previous chunk : 0x00320178
```

Algorithmes d'allocation et de libération mémoire

Les quelques structures nécessaires à la compréhension du fonctionnement du tas de Windows ayant été introduites, nous allons maintenant nous attaquer aux algorithmes d'allocation et de libération de la mémoire. Ces algorithmes sont bien évidemment loin d'être simples (surtout lorsqu'étudiés en assembleur...) et il serait très long de les développer en détail ici.

Néanmoins, une compréhension minutieuse de ces derniers est absolument nécessaire afin d'appréhender les nouvelles techniques d'exploitation de « heap overflows » sous Windows.

Algorithme d'allocation mémoire

L'allocation d'un espace mémoire sur le tas de Windows se déroule en suivant les étapes ci-après :

1. La taille du buffer demandée est ajustée suivant l'opération décrite plus haut.

3. Si le tas comporte un tableau de « lookaside », que ce dernier n'est pas verrouillé, et que la taille demandée est inférieure à 1024 octets (afin de ne pas dépasser les limites du tableau), alors on regarde dans ce tableau si un buffer de taille exacte est disponible. Si tel est le cas, on le retire du tableau et on le retourne au demandeur, sinon, on poursuit.

3. Si la taille demandée est inférieure à 1024, on peut regarder dans le tableau des listes de chunks libres (sans tenir compte de l'entrée 0) :
   - 1. Si la liste correspondant à la taille exacte du buffer n'est pas vide, on enlève le premier chunk disponible de ette liste et on le retourne à l'utilisateur ;
   - 2. Sinon, on utilise le tableau d'usage des listes de chunks libres afin de trouver la première liste de chunks libres (de taille supérieure à celle demandée bien entendu) non vide. Si une telle liste est trouvée, on en extrait le premier élément pour le retourner au demandeur. Si la taille du chunk retourné dépasse de plus de 8 octets (strictement) la taille demandée - ce qui est des plus probables, on est amené à casser ce chunk en deux morceaux, l'un de la taille demandée, le morceau de chunk restant étant retourné dans la liste de chunks libres correspondant à sa taille.

4. Si la taille demandée est inférieure au seuil d'allocation virtuelle, on peut alors chercher dans l'entrée 0 du tableau des listes des chunks libre. Dans le cas où aucun chunk ne satisfait à notre requête, le tas est étendu (je ne vais pas m'étendre ici sur la méthode) et un chunk fraîchement créé est renvoyé à l'utilisateur.

5. Sinon, une allocation est effectuée (si le tas est indiqué comme étant extensible), le chunk résultant est ajouté à la liste des chunks virtuellement alloués.

Algorithme de libération mémoire

La libération d'un espace mémoire alloué sur le tas de Windows observe le cheminement suivant :

1. Si le chunk n'est pas indiqué comme étant occupé (*flag* « `Busy` »), que l'adresse du buffer n'est pas un multiple de 8, ou que le « `Segment index` » du chunk est supérieur ou égal à 64 (le maximum autorisé), on sort en indiquant une erreur dans les paramètres fournis.

2. Si le tas comporte un tableau de « lookaside », qu'il n'est pas verrouillé, que le chunk n'est pas indiqué comme étant virtuellement alloué (flag « `Vitrual alloc` »), que la taille du chunk est inférieure à 1024, alors on tente de libérer le chunk vers le tableau de « lookaside ». Cela consiste à laisser le chunk marqué occupé, et à ajouter une entrée dans le tableau si ce dernier n'est pas plein.

3. Si le chunk n'est pas virtuellement alloué, on le fusionne éventuellement avec des chunks libres situés avant et/ou après (procédé développé après), et le chunk résultant ira s'ajouter à la liste des chunks libres correspondant à sa taille.

4. Sinon, on retire le chunk en question de la liste des chunks virtuellement alloués, et on libère la mémoire anciennement occupée vers le système d'exploitation.

Fusion de chunks libres

Le principe de fusion des chunks libres est assez intuitif. Puisque les chunks libres sont régulièrement séparés, le tas pourrait être amené à se retrouver dans un état très fragmenté rendant son utilisation peu efficace. Ainsi, lorsqu'un chunk est libéré, Windows va vérifier la présence de chunks libres avant et après celui-ci afin de les fusionner en un seul et même chunk, rétablissant par là même une certaine homogénéité dans le tas.

La Figure 7 montre comment la libération d'un buffer situé entre deux chunks libres va entraîner la fusion des trois buffers entre eux pour qu'il n'en reste qu'un au final (procédé illustré sur la Figure 7 page suivante) :

1. On détermine l'emplacement du chunk précédent (grâce au champ `PreviousSize`), si ce chunk existe, qu'il n'est pas signalé

comme étant occupé (flag « Busy »), et que la somme des tailles des deux chunks ne dépasse pas la taille maximale d'un chunk (0xfe00 octets), alors on peut les fusionner :

- 1. retrait du chunk précédent de sa liste de chunks disponibles ;
- 2. retrait du chunk courant de sa liste de chunks disponibles ;
- 3. mise à jour de l'information de dernière entrée du chunk précédent si le chunk courant en était doté (flag « Last entry ») ;
- 4. mise à jour de la taille du chunk précédent pour refléter la fusion, et éventuellement le champ PreviousSize du chunk suivant s'il existe (absence du flag « Last entry »).

2 On détermine ensuite l'emplacement du chunk suivant le chunk fusionné (grâce au champ Size), si ce chunk existe, n'est pas signalé comme étant occupé, et que la somme des tailles des deux chunks ne dépasse pas la taille maximale d'un chunk, alors on peut les fusionner :

- 1. si cela n'a pas encore été fait, on retire le chunk courant de sa liste de chunks disponibles ;
- 2. retrait du chunk suivant de sa liste de chunks disponibles ;
- 3. éventuelle mise à jour de l'information de dernière entrée ;
- 4. mise à jour de la taille du chunk courant, et éventuellement du champ PreviousSize du chunk situé après le chunk fusionné s'il existe.

3 Le chunk résultant de la fusion est alors retourné.

La fusion ne peut donc pas se faire dans certains cas (très logiques) :

→ lorsque le chunk est indiqué comme ne pouvant fusionner (flag « Don't coalesce », 0x80) ;

→ lorsque le chunk est le premier, il ne peut fusionner avec un chunk situé avant ;

→ lorsqu'il est le dernier, il ne peut fusionner avec un chunk situé après ;

→ les chunks précédents ou suivants sont indiqués comme étant occupés (par exemple s'ils ont été libérés vers le tableau de « lookaside ») ;

→ lorsque la taille du chunk résultant de la fusion dépasse le seuil d'allocation virtuelle.

Si la quantité de mémoire libérée dépasse un certain seuil, elle pourra être directement libérée vers le système d'exploitation plutôt que d'être introduite dans les listes de chunks libres.

Pour résumer...

Les structures du tas particulièrement intéressantes sont le tableau de « lookaside », les listes de chunks libres, la liste de chunks libres d'indice 0, le tableau des segments, le tableau d'usage des listes de chunks libres.

Les structures impliquées dans l'allocation et la libération de mémoire le sont toujours dans le même ordre : le tableau de « lookaside », les listes de chunks libres, la liste de chunks libres d'indice 0.

Enfin, la mémoire du tas est entièrement recyclable : les chunks sont cassés pour satisfaire aux demandes d'allocation, et fusionnés suite à leur libération.

Les Heap Overflows

Le principe

Contrairement aux *stack overflows* qui donnent - en général - un contrôle immédiat du registre EIP et donc du flux d'exécution, les possibilités offertes par un débordement de tampon dans le tas de Windows sont plus « limitées ». Les outils d'exploitation de « heap overflows » et articles circulant sur le sujet aboutissent tous à un résultat similaire dit ***4 byte overwrite***, qui correspond au fait d'écrire 4 octets à n'importe quel emplacement mémoire. Cela nous permet d'écraser une adresse par une autre et potentiellement aboutir à de l'exécution de code lorsqu'on sait ce que l'on fait.

Comment est-ce possible ?

Vous l'aurez certainement compris, la réponse se situe au niveau des deux pointeurs situés dans la structure d'un chunk libre. Lors d'un débordement de tampon dans le tas, nous allons dans un premier temps remplir l'espace normalement alloué pour le buffer - jusqu'ici tout va bien - puis nous allons écraser la structure du chunk situé immédiatement après le nôtre dans le tas, quel que soit son état.

En conséquence, nous prenons le contrôle de sa taille, de la taille du chunk précédent (à savoir le nôtre), de l'indice de son segment,

Figure 7 *Exemple de fusion*

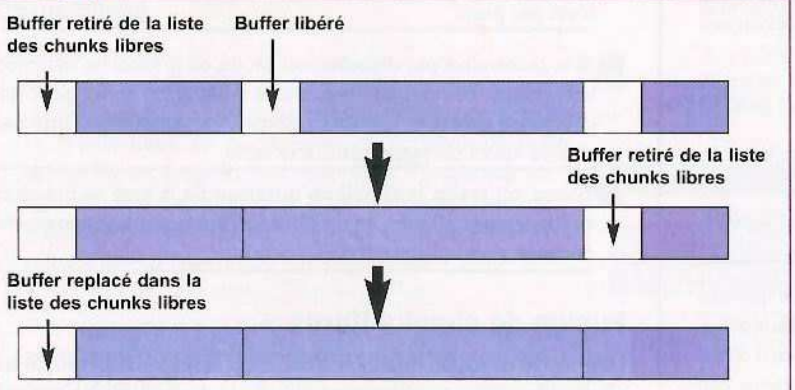


Figure 8 *Schéma de l'overflow*

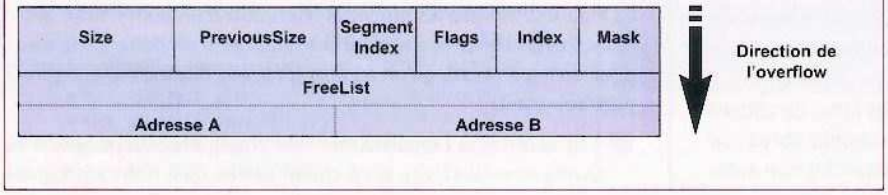


Tableau 1

Adresse B	Adresse A	Commentaire
Unhandled Exception Filter	call [edi + XX] ou équivalent	Bon taux de succès mais dépendant du SP
Vectored Exception Handler	Emplacement sur la pile pointant vers notre buffer	Bon taux de succès mais dépendant du SP
Fonctions de verrouillage du PEB	Adresse devinée ou spécifique à l'application	Taux de réussite moyen mais indépendant du SP

de ses options, de l'index et du masque, et des adresses A et B des chunks libres suivant et précédent (voir Figure 8) : même si le chunk sur lequel nous débordons était un chunk alloué, peu nous importe, puisque nous pouvons en faire un chunk libre :)

Plusieurs cas de figure peuvent alors se présenter, entraînant des manipulations de mémoire au niveau des adresses A et B. Je vais maintenant entrer dans les détails pour l'un de ces cas.

Le couple « Adresse A », « Adresse B » est en fait une structure de type `LIST_ENTRY` qui s'inscrit dans le contexte de gestion des listes doublement chaînées par Windows :

```
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY,*PLIST_ENTRY;
```

Lorsque l'on désire enlever un élément d'une liste doublement chaînée, il est nécessaire de mettre à jour le pointeur `Flink` (élément suivant) de l'élément qui le précède dans la liste, ainsi que le pointeur `Blink` (élément précédent) de l'élément qui lui succède. Cela ressemble *grosso modo* aux quelques lignes de codes ci-dessous, en supposant lorsque l'on désire retirer l'élément e de la liste :

```
PLIST_ENTRY b;
PLIST_ENTRY f;
f = e->Flink;
b = e->Blink;
b->Flink = f;
f->Blink = b;
```

Puisque e->Flink = AdresseA et que e->Blink = AdresseB, on en déduit que le retrait de 'e' correspond aux opérations :

```
*(AdresseB) = AdresseA
*(AdresseA + 4) = AdresseB
```

Cela s'applique relativement bien au cas présent : lorsque notre chunk reconstruit devra être retiré d'une liste, les 4 octets de AdresseA seront écrits à l'emplacement pointé par AdresseB, et les 4 octets de AdresseB seront écrits à l'emplacement pointé par (AdresseA + 4), et nous avons un des cas menant à un double ***4 byte overwrite*** !

Rappelez-vous des opérations pouvant mener au retrait de notre faux chunk d'une liste :

→ allocation de notre chunk s'il est référencé comme libre ;

→ fusion de notre chunk avec un chunk en train d'être libéré qui le suit ou le précède immédiatement ;

→ libération de notre faux chunk si ce dernier est indiqué comme virtuellement alloué...

Que de possibilités à exploiter. Mais attention, en fonction du cas de figure que vous désirez exploiter, il faudra bien évidemment adapter le contenu des différents champs du chunk que vous construisez via le débordement. Certes, le couple d'adresses constituant le champ `FreeList` est d'importance, mais les 4 premiers champs de la structure le sont tout autant pour aboutir à une exploitation correcte. Il est notamment indispensable que les champs de taille de l'enregistrement soient corrects lors de la fusion de blocs, que le champ d'indice de segment soit inférieur au nombre maximal possible de segments (64), et que les options du chunk soient en concordance avec la situation (présence du flag « `Last entry` » ou non, présence du flag « `Busy` » ou non, présence du flag « `Virtually allocated` » ou non).

Les couples populaires

Nous sommes maintenant plus ou moins en mesure d'écrire 4 octets à un emplacement mémoire déterminé, il s'en suit une question évidente : comment choisir ce couple de nombre de 32 bits pour prendre le contrôle du flux d'exécution du processus vulnérable ?

Ce problème n'est pas des moindres puisqu'il a empêché jusqu'à présent toute exploitation fiable et portable de « heap overflows » sous Windows, un mauvais choix du couple entraînera au choix une violation d'accès mémoire, une corruption de données, un crash du programme.

Plusieurs couples d'adresses sont déjà connus et utilisés couramment dans des programmes publics d'exploitation de « heap overflows » sous Windows : cf Tableau 1 ci-dessus.

Développons cela :

→ Le « Unhandled Exception Filter » correspond à l'adresse d'un jeu d'instructions exécutées par le système d'exploitation lorsqu'une exception se déclenche et n'est pas gérée. Il est défini grâce à la fonction `SetUnhandledExceptionFilter()` exportée par la bibliothèque `kernel32.dll` :

```
.text:77E5E5A1        mov     ecx, [esp+arg_0]
.text:77E5E5A5        mov     eax, dword_77EB73B4
.text:77E5E5AA        mov     dword_77EB73B4, ecx
```

Ici, l'adresse du UEF est 0x77EB73B4. Lorsque le noyau prend en charge une exception non gérée, il exécute la portion de code suivante, située dans `UnhandledExceptionFilter()` :

```
.text:77E73114        mov     eax, dword_77EB73B4 ; POINTEUR VERS L'UEF
.text:77E73119        cmp     eax, esi
.text:77E7311B        jz      short loc_77E73132
.text:77E7311D        push    edi
.text:77E7311E        call    eax ; APPEL A NOTRE CODE
```

Le registre EDI est poussé sur la pile, et à 0x78 octets de EDI se situe une adresse pointant vers le tas - à un endroit que l'on peut contrôler dans le cas d'un débordement. Le principe est donc de remplacer l'UEF par l'adresse d'un `call [edi+0x78]` ou équivalent et de provoquer une exception afin de mener à l'exécution de notre code. L'exception est en général assez facile

à déclencher, il suffit de choisir Adresse A dans une page mémoire n'offrant pas les droits en écriture, le second ***4 byte overwrite*** provoquant alors une violation d'accès. On peut aboutir à des résultats similaires avec des adresses relatives aux registres EBP ou ESI dans certaines versions de Windows.

L'inconvénient ? Le UEF et l'adresse du call dépendent étroitement de la version de l'OS, de son SP et de sa régionalisation. A moins de trouver la version exacte (quasiment au patch près) du système vulnérable, l'exploitation a de grandes chances d'échouer.

→ Le « Vectored Exception Handler » est une des nouveautés de XP. Contrairement aux structures usuelles d'administration des gestionnaires d'exception situées sur la pile, le VEH se trouve dans le tas. Un pointeur vers le premier VEH est situé à une adresse fixe dans la bibliothèque `ntdll.dll` :

```
.text:77F5CD0E              mov     esi, dword_77FB4880
        ; POINTEUR VERS LE VEH
.text:77F5CD14              jmp     short loc_77F5CD24
.text:77F5CD16
.text:77F5CD16 loc_77F5CD16:
        ; CODE XREF: sub_77F5CCDE+48
.text:77F5CD16              lea     eax, [ebp+var_8]
.text:77F5CD19              push    eax
.text:77F5CD1A              call    dword ptr [esi+8]
        ; APPEL A NOTRE CODE
.text:77F5CD1D              cmp     eax, 0FFFFFFFFh
.text:77F5CD20              jz      short loc_77F5CD3C
.text:77F5CD22              mov     esi, [esi]
.text:77F5CD24
.text:77F5CD24 loc_77F5CD24:
        ; CODE XREF: sub_77F5CCDE+36
.text:77F5CD24              cmp     esi, edi
.text:77F5CD26              jnz     short loc_77F5CD16
```

La technique consiste à remplacer ce dernier par un pointeur vers une fausse structure de VEH que nous avons construite sur la pile. Lorsqu'une exception se produira, nous gagnerons le contrôle du flux d'exécution via le pointeur situé en [esi+8].

L'inconvénient ? L'adresse du VEH est spécifique aux versions de Windows - encore une fois au SP près, l'adresse sur la pile n'est pas forcément fiable.

→ Il existe deux pointeurs, situés dans le PEB, qui référencent deux fonctions. Ces deux fonctions sont appelées dans `RtlAcquirePebLock()` et `RtlReleasePebLock()` au sein de `ntdll.dll`, elles-mêmes susceptibles d'être appelées à différentes reprises. Dans la mesure où l'adresse du PEB est fixe dans les versions de Windows considérées, les deux pointeurs vers les fonctions sont aussi situées à des emplacements fixes (PEB+0x20 pour le premier, PEB+0x24 pour le second, se référer à l'encadré page 65), et représentent donc une cible de choix à écraser par une adresse pouvant aboutir à notre code.

L'inconvénient ? Le saut vers le pointeur écrasé se fera à un moment où il n'existera plus de référence à notre code aisément atteignable (à savoir pas moyen d'accéder à notre buffer grâce à une adresse relative à un registre), il faudra donc utiliser des adresses devinées ou spécifiques à l'application - ce qui nuit fortement à la fiabilité de l'exercice.

Au final, pas de méthode miracle, et des débordements assez complexes à exploiter de façon convenable. Y a-t-il un moyen d'améliorer cela ?

Le tableau de « lookaside » : le saint Graal de l'exploitation des « heap overflows » sous Windows

Nous avons vu précédemment que le tableau de « lookaside », lorsqu'il existe, est la première option retenue pour le choix de chunks lors de l'allocation et la libération d'espace mémoire de taille réduite (à savoir inférieure à 1024 octets). Approfondissons cela.

Ses caractéristiques

Le tableau de « lookaside » est créé à la fin de la fonction `RtlCreateHeap()` de `ntdll.dll` si les conditions suivantes sont réunies :

1. le tas supporte les accès sérialisés (absence de l'indicateur `HEAP_NO_SERIALIZE` dans les propriétés du tas) ;
2. le tas est extensible (présence de l'indicateur `HEAP_GROWABLE` dans les propriétés du tas) ;
3. l'utilisation de tableau de « lookaside » n'est pas désactivée ;

Le tableau est la première structure à être allouée sur le tas, d'une taille de 6144 octets : 128 fois la taille d'une entrée de « lookaside » soit 0x30 octets, et par conséquent commence toujours au même emplacement relativement à l'adresse de base du tas (à 0x688 octets de la base). Bien que l'adresse soit fixe relativement au tas, le tableau de « lookaside » est référencé par un pointeur dans la structure de gestion du tas à l'offset 0x580 : si ce pointeur est nul, le tas ne possède pas de tableau de « lookaside », s'il n'est pas nul, il pointe vers la première entrée du tableau. Un autre moyen de déterminer si un tas possède un tableau de « lookaside » est l'utilisation de la fonction `HeapQueryInformation()` apparue avec Windows XP qui renvoie 1 si le tas le supporte.

Attention, La fonction `RtlCreateHeap()` est très rarement utilisée directement pour la création d'un tas, au profit de la fonction `HeapCreate()` de `kernel32.dll`. Or cette fonction filtre l'indicateur `HEAP_GROWABLE` dans le champ `flOptions` : le seul moyen d'obtenir un tas extensible et possédant donc un tableau de « lookaside » est de spécifier une taille minimale et maximale égale à 0, la fonction s'occupera du reste. Notez que le tas par défaut d'un processus est extensible, et que le « lookaside » commence vierge de toute entrée.

Illustrons tout cela grâce au programme suivant :

```
typedef BOOL (*MYPROC)(HANDLE, HEAP_INFORMATION_CLASS, PVOID,SIZE_T, PSIZE_T);
...
HANDLE hHeap;
MYPROC ProcAdd;
HINSTANCE hInstance;
hHeap = HeapCreate(0, 4096, 65536);
fprintf(stdout, «%08x %08x\n», hHeap, *(unsigned int *)((unsigned int)hHeap + 0x580));
hHeap = HeapCreate(0, 0, 0);
fprintf(stdout, «%08x %08x\n», hHeap, *(unsigned int *)((unsigned int)hHeap + 0x580));
hHeap = GetProcessHeap();
fprintf(stdout, «%08x %08x\n», hHeap, *(unsigned int *)((unsigned int)hHeap + 0x580));
if ((hInstance = LoadLibrary(«kernel32»)) == NULL)
    goto Fin;
if ((ProcAdd = (MYPROC)GetProcAddress(hInstance, «HeapQueryInformation»)) == NULL)
// Seulement sous XP
    goto Fin;
(ProcAdd)(hHeap, HeapCompatibilityInformation, &ulHeapInformation,
sizeof(ulHeapInformation), &ReturnLength);
fprintf(stdout, «Heap base address: 0x%x\nHeap type: %u\n\n», hHeap,
ulHeapInformation);
Fin:
```

❑ *Exemple d'exécution*

```
00320000 00000000
00330000 00330688
00140000 00140688
Heap base address: 0x140000
Heap type: 1
```

Dans le premier tas créé : pas de « lookaside », dans le second il y en a un, dans le tas par défaut du processus aussi. Notez que la présence de « lookaside » ou non est rarement mise en avant dans les exemples d'exploitation de « heap overflows » publiés jusqu'à présent (comme par exemple dans [LITCHFIELD]) alors qu'il s'agit d'un élément déterminant dans le déroulement des procédés étudiés.

Contrôle du « lookaside »

Puisque le « lookaside » est généralement présent dans le tas et que son emplacement reste fixe avec les versions de Windows, il est intéressant d'étudier les possibilités offertes par un ***4 byte overwrite*** dans ce tableau, ou en rapport avec ce tableau.

Si on provoque l'allocation d'un buffer de taille inférieure à 1024 octets, celui-ci va être libéré vers le « lookaside » : le chunk va rester occupé et intact (sauf les 4 premiers octets mis à 0), et son adresse placée dans l'entrée correspondante du tableau de « lookaside ». Puisque nous connaissons 1) l'adresse de base du tableau, 2) la taille d'une entrée et que 3) le numéro de l'entrée est déductible de la taille d'allocation demandée, nous connaissons un emplacement fixe pointant sur notre buffer !

En pratique, on calcule le numéro de l'entrée à partir de la taille ajustée du buffer à allouer i = (taille + 7) / 8 + 1, on retrouve l'adresse grâce à l'opération base + i * 0x30. En l'occurrence, pour une adresse de base du tas de 0x140000 (et donc du « lookaside » de 0x140688) et une taille à allouer de 922, on retrouve le pointeur vers notre buffer après libération à l'adresse 0x140000 + 0x688 + ((922 + 7) / 8 + 1) * 0x30 = 0x140688 + 0x75 * 0x30 = 0x141C78.

En C, on illustre ça avec :

```
HANDLE hHeap;
unsigned char *buffer;
hHeap = GetProcessHeap();
fprintf(stdout, «*** HeapAlloc ***\n»);
buffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, 922); // Allocation de 922 octets
memset(buffer, 0x42, 922); // On remplit de buffer de 0x42
fprintf(stdout, «Buffer address : %08x\nBuffer content : %08x\nLookaside pointer
: %08x\n», buffer, *(unsigned int *)(buffer), *(unsigned int *)((unsigned
int)hHeap + 0x1c78));
fprintf(stdout, «*** HeapFree ***\n»);
HeapFree(hHeap, 0, buffer); // Libération du buffer
fprintf(stdout, «Buffer address : %08x\nBuffer content : %08x\nLookaside pointer
: %08x\n», buffer, *(unsigned int *)(buffer), *(unsigned int *)((unsigned
int)hHeap + 0x1c78));
```

❑ *Exemple d'exécution*

```
*** HeapAlloc ***
Buffer address : 00145620
Buffer content : 42424242
Lookaside pointer : 00000000
*** HeapFree ***
Buffer address : 00145620
Buffer content : 00000000
Lookaside pointer : 00145620
```

Une fois le buffer libéré, ses 4 premiers octets sont mis à 0, et la valeur du pointeur dans le « lookaside » correspondant à la taille de 922 octets est mise à l'adresse de notre buffer.

Maintenant que se passe-t-il si on redemande l'allocation d'un espace mémoire de 922 octets ? Windows va tout simplement renvoyer le pointeur se trouvant dans l'entrée correspondante du « lookaside ». Imaginons que ce pointeur ait été modifié entre-temps (par un ***4 byte overwrite*** par exemple :), l'adresse retournée par `HeapAlloc()` sera en notre contrôle et nos 922 octets seront écrits à un emplacement arbitraire ! Notre ***4 byte overwrite*** devient un ***922 byte overwrite***, les possibilités ouvertes sont énormes...

```
HANDLE hHeap;
unsigned char *buffer;
hHeap = GetProcessHeap();
fprintf(stdout, «*** HeapAlloc ***\n»);
buffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, 922);
memset(buffer, 0x42, 922);
fprintf(stdout, «*** HeapFree ***\n»);
HeapFree(hHeap, 0, buffer);
/* Simulation du 4 byte overwrite : l'adresse 0x7ffdf130 remplace le pointeur
vers notre buffer dans le «lookaside»
C'est équivalent à Adresse B = 0x141c78 et Adresse A = 0x7ffdf130 (ça tombe bien
pour Adresse B, on a le droit à un 0x00 terminal, et les deux adresses sont dans
des pages avec droit en écriture, donc pas d'exception) */
*(unsigned int *)((unsigned int)hHeap + 0x1c78) = 0x7ffdf130;
fprintf(stdout, «*** HeapAlloc ***\n»);
buffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, 922);
memset(buffer, 0x43, 922);
fprintf(stdout, «Buffer address : %08x\nBuffer content : %08x\nLookaside pointer
: %08x\n», buffer, *(unsigned int *)(buffer), *(unsigned int *)((unsigned
int)hHeap + 0x1C78));
```

❑ *Exemple d'exécution*

```
*** HeapAlloc ***
*** HeapFree ***
*** HeapAlloc ***
Buffer address : 7ffdf130
Buffer content : 43434343
Lookaside pointer : 00000000
```

Nous venons d'écrire 922 octets dans le PEB à l'adresse 0x7ffdf130 !

Application à l'exécution de code

En se fondant sur les constats précédemment énoncés, on peut en déduire plusieurs méthodes utilisant le « lookaside » sous une forme ou une autre pour provoquer l'exécution de code arbitraire de façon fiable, je m'attarderai sur quelques-unes seulement :

1. Le grand désavantage de l'utilisation du couple faisant intervenir les fonctions de verrouillages du PEB est l'absence quasi totale de visibilité quant à l'emplacement du shellcode en mémoire. On peut grandement améliorer cette technique grâce à l'utilisation du « lookaside », en suivant les étapes ci-après :
   1. On provoque l'allocation d'un buffer d'une taille < 1024 octets et sa libération afin de peupler l'entrée du « lookaside » correspondante ;
   2. On provoque un ***4 byte overwrite*** afin d'écraser le pointeur de l'entrée en question (calculable bien évidemment) par une adresse fixe (pourquoi pas 0x7ffdf130 dans le PEB ? :) ;

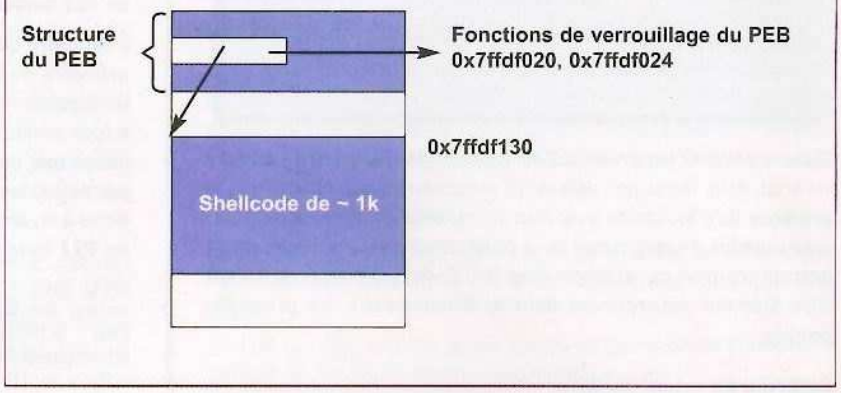
■ 3. On provoque l'allocation d'un buffer de taille identique à celui de l'étape 1. afin que notre shellcode soit écrit à l'adresse fixée précédemment ;

■ 4. On provoque un ***4 byte overwrite*** afin d'écraser l'une des deux fonctions de verrouillage du PEB (connue) par l'adresse de notre shellcode (aussi connue) ;

■ 5. On attend que soit appelée la fonction en question !

Voir figure 9.

Figure 9 *Illustration de la première méthode*



2 La méthode précédente est séduisante et assez intuitive, mais nécessite deux overflows à la suite, ce qui se révèle la plupart du temps hasardeux dans la mesure où le tas peut se retrouver dans des états quelque peu instables après ce type d'opération. Pour la seconde méthode, il nous faudra trouver un emplacement mémoire fixe contenant une ou plusieurs adresses de fonctions susceptibles d'être appelées dans la suite du programme. Puisque nous pouvons maintenant écrire un peu moins de 1024 octets de données n'importe où, nous pouvons écraser le pointeur vers ladite fonction par une adresse pointant sur notre shellcode : si nous connaissons l'adresse du pointeur en question, notre shellcode sera quelques octets plus loin en mémoire.

■ 1. On provoque l'allocation d'un buffer d'une taille < 1024 octets et sa libération afin de peupler l'entrée du « lookaside » correspondante ;

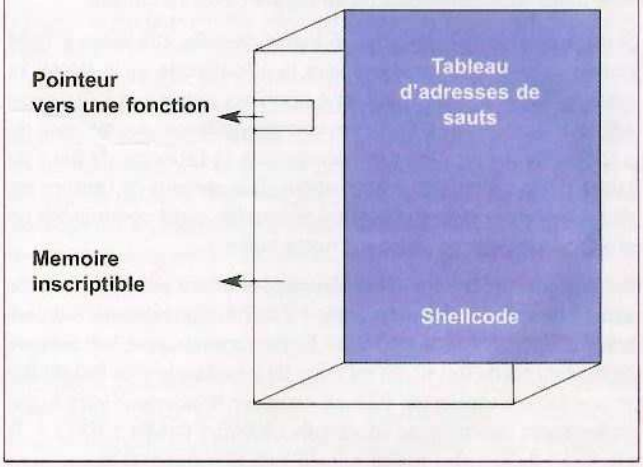
■ 2. On provoque un ***4 byte overwrite*** afin d'écraser le pointeur de l'entrée en question par l'emplacement mémoire contenant le pointeur vers la fonction qui sera exécutée - cette zone doit bien sûr être inscriptible ;

■ 3. On provoque l'allocation d'un buffer de taille identique à celui de 1. afin que notre buffer contenant l'adresse du shellcode et notre shellcode (le tout minutieusement arrangé) soit écrit à l'adresse précédemment déterminée ;

■ 4. On attend que soit appelée la fonction en question !

Voir figure 10.

Figure 10 *Illustration de la seconde méthode*



3 La troisième méthode consiste simplement à trouver une variable globale (un chaîne de caractères par exemple) que nous pourrons remplacer, par exemple celle utilisée par l'API `GetSystemDirectory()`. Cette dernière, qui renvoie normalement la chaîne `c:\winnt\system32\`, pourrait maintenant renvoyer `\\1.2.3.4\backdoors` :)

Dès lors, il vous suffira de mettre à ladite adresse une bibliothèque « backdoorée » (`kernel32.dll`, `user32.dll`, peu importe), elle sera automatiquement chargée par les exécutables sur la machine compromise si ces derniers en importent des fonctions.

Personnellement, mon coup de cœur va à une autre technique qui consiste à déplacer le pointeur d'une « Dispatch Table » vers un endroit du « lookaside » afin que la prochaine fonction à être requise pointe en fait sur notre buffer : c'est très efficace et portable entre les SP d'une même version de Windows. La méthode diffère un peu des précédentes dans la mesure où cette fois-ci, on n'écrase pas une adresse du « lookaside », mais on modifie l'adresse d'une zone mémoire vers ce dernier afin de bénéficier de l'adresse connue d'un buffer libéré contenant notre shellcode (ses 4 premiers octets ayant été mis à 0).

■ 1. On provoque l'allocation d'un buffer d'une taille < 1024 octets (contenant notre shellcode) et sa libération. Les 4 premiers octets de ce buffer seront mis à 0 et son adresse ira dans le « lookaside » dans l'entrée associée à sa taille ;

■ 2. On provoque un ***4 byte overwrite*** afin d'écraser le contenu de la `User32Dispatch` située à l'offset 0x2c dans le PEB (ce tableau est utilisé dans le cadre d'opérations graphiques). La nouvelle valeur de cette variable pointera dans le tas de sorte que User32Dispatch + (4 * Numéro de la fonction) = Adresse du buffer dans le « lookaside ». Le numéro de la fonction se trouve en analysant le flux d'exécution du programme afin de déterminer quelle sera la prochaine appelée ;

■ 3. Il ne reste plus qu'à attendre que la fonction soit appelée, ou bien forcer le destin en provoquant son appel :)

Et cette liste ne se veut pas exhaustive...

Conclusion

Un exploit pour le Messenger Service de Windows qui fonctionne convenablement sans avoir à identifier de façon exacte la version du système d'exploitation vulnérable, vous en rêviez, c'est maintenant possible (et fait). Les compromissions de systèmes Microsoft ont encore de beaux jours devant elles, même si les techniques exposées ici ne semblent pas encore avoir fait leur chemin jusqu'aux exploits publics.

Il est à mettre au crédit de Microsoft des efforts conséquents pour tenter d'arranger un peu les choses avec le Service Pack 2 de Windows XP et avec Windows 2003 Server : l'adresse du PEB est rendue plus aléatoire, un cookie est introduit dans les chunks des tas afin de se prémunir contre tout overflow, le retrait d'un élément d'une liste doublement chaînée est plus sûr.

Un point important n'a pas été abordé ici, faute de temps et d'espace, il s'agit bien évidemment de la stabilisation de l'environnement d'exécution du programme. Il n'est pas rare que l'exploitation d'un « heap overflow » provoque des dysfonctionnements au niveau du processus, ou pire, du système d'exploitation ; il est nécessaire de s'assurer qu'une telle chose ne surviendra pas, notamment en reconstruisant les structures modifiées, ce qui inclut par exemple le PEB et ses fonctions de verrouillage, ou bien le tas lui-même qu'il faudra réparer. Gageons que ces sujets seront abordés dans un prochain numéro de MISC !

Encore une fois, j'adresse mes salutations et mes sincères remerciements à Oded Horovitz et Matthew Conover pour leur travail sur le tas de Windows.

Références

[LITCHFIELD] Windows Heap Overflows :
http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt

[AITEL] Microsoft Heap Overflows :
http://www.immunitysec.com/downloads/msrpcheap.pdf & http://www.immunitysec.com/downloads/msrpcheap2.pdf

[HALVAR] Third Generation Exploitation :
http://www.blackhat.com/presentations/win-usa-02/halvarflake-winsec02.ppt

[RUIU] CanSecWest/Core04 Conference :
http://www.cansecwest.com/

[HOROWITZ] Reliably Exploiting Windows Heap Overflows :
http://cansecwest.com/csw04/csw04-Oded+Connover.ppt

[NTDLL] ntdll.h : **http://cvs.kldp.net/cgi-bin/cvsweb.cgi/moguasrc/include/ntdll.h?cvsroot=mogua&rev=1.3**

Structure du PEB

Voici la structure définissant le PEB telle qu'elle peut apparaître après étude des bibliothèques dynamiques de Windows. N'étant pas documentés, certains champs sont d'une exactitude toute relative (voire non définis).

```
/*
 * Process Environment Block, situé à l'adresse 0x7FFDF000.
 */
typedef struct _PEB {
/*000*/    BOOLEAN InheritedAddressSpace;
/*001*/    BOOLEAN ReadImageFileExecOptions;
/*002*/    BOOLEAN BeingDebugged;
   BYTE b003;
   DWORD d004;
/*008*/    PVOID SectionBaseAddress;
/*00C*/    PPROCESS_MODULE_INFO ProcessModuleInfo;
/*010*/    PPROCESS_PARAMETERS ProcessParameters;
/*014*/    DWORD SubSystemData;
/*018*/    HANDLE ProcessHeap;
/*01C*/    PCRITICAL_SECTION FastPebLock;
/*020*/    VOID (*AcquireFastPebLock)(PCRITICAL_SECTION);
/*024*/    VOID (*ReleaseFastPebLock)(PCRITICAL_SECTION);
   DWORD d028;
/*02C*/    PPVOID User32Dispatch;
   DWORD d030;
   DWORD d034;
   DWORD d038;
/*03C*/    DWORD TlsBitMapSize;
/*040*/    PRTL_BITMAP TlsBitMap;
/*044*/    DWORD TlsBitMapData[2];
   PVOID p04C;
   PVOID p050;
/*054*/    PTEXT_INFO TextInfo;
/*058*/    PWCHAR InitAnsiCodePageData;
/*05C*/    PWCHAR InitOemCodePageData;
/*060*/    PSHORT InitUnicodeCaseTableData;
/*064*/    DWORD KeNumberProcessors;
/*068*/    DWORD NtGlobalFlag;
   DWORD d06C;
/*070*/    LARGE_INTEGER MmCriticalSectionTimeout;
/*078*/    DWORD MmHeapSegmentReserve;
/*07C*/    DWORD MmHeapSegmentCommit;
/*080*/    DWORD MmHeapDeCommitTotalFreeThreshold;
/*084*/    DWORD MmHeapDeCommitFreeBlockThreshold;
/*088*/    DWORD NumberOfHeaps;
/*08C*/    DWORD AvailableHeaps;
/*090*/    PHANDLE ProcessHeapsListBuffer;
   DWORD d094;
   DWORD d098;
   DWORD d09C;
/*0A0*/    PCRITICAL_SECTION LoaderLock;
/*0A4*/    DWORD NtMajorVersion;
/*0A8*/    DWORD NtMinorVersion;
/*0AC*/    WORD NtBuildNumber;
/*0AE*/    WORD CmNtCSDVersion;
/*0B0*/    DWORD PlatformId;
/*0B4*/    DWORD Subsystem;
/*0B8*/    DWORD MajorSubsystemVersion;
/*0BC*/    DWORD MinorSubsystemVersion;
/*0C0*/    KAFFINITY AffinityMask;
/*0C4*/    DWORD ad0C4[35];
   PVOID p150;
   DWORD ad154[32];
/*1D4*/    HANDLE Win32WindowStation;
   DWORD d1D8;
   DWORD d1DC;
/*1E0*/    PWORD CSDVersion;
   DWORD d1E4;
} PEB, *PPEB, **PPPEB;
```

L'ASN.1 par l'exemple dans les certificats X.509

Souvent méconnu des informaticiens et malgré un sigle rébarbatif, l'ASN.1 (« Abstract Syntax Notation One ») tient un rôle important dans la définition de structures de données. A ce titre, en sécurité, on le retrouve notamment dans la moindre signature RSA [PKCS1], dans chaque certificat numérique [X.509], dans les informations biométriques [X9.84], pour la transmission de messages sécurisés [CMS,PKCS7], ou encore la définition de protocole d'horodatage sécurisé [RFC3161]. Les exemples ne manquent pas, et si l'ASN.1 semble souvent rester tapi dans l'ombre, l'objectif de cet article consiste à l'amener sous le feu des projecteurs, et plus particulièrement à comprendre où il intervient en sécurité. Afin de rendre cette présentation plus attrayante, nous illustrerons au fil de l'article le cas des certificats X.509. On retrouvera notamment dans ces derniers la valeur précise de la clé publique et la signature du certificat.

A quoi donc l'ASN.1 sert-il ?

De nos jours, les réseaux informatiques ont pris une importance capitale dans nos systèmes, et savoir transmettre des données d'une machine à une autre est une nécessité absolue. Cependant, l'hétérogénéité des parcs informatiques soulève de nombreux problèmes techniques :

- Comment transmettre des données binaires d'une machine Big Endian à Little Endian ?
- Comment représenter une structure de données indépendamment d'un langage de programmation ? Par exemple, comment faire pour qu'un certificat numérique soit compréhensible par des programmes Java, C ou Ada ?
- Comment indiquer à une machine qu'on envoie une chaîne de caractères UTF8, et non pas Unicode (par exemple) ?

L'ASN.1 résout ces problèmes de communications et de compatibilités. C'est une **norme qui permet de décrire des types de données abstraits, indépendamment de tout langage de programmation et de toute spécificité de plate-forme** . Pour décrire les structures de données, l'ASN.1 offre un certain nombre de types prédéfinis comme INTEGER, OCTET STRING, BOOLEAN, UTFString, PrintableString (chaîne avec caractères imprimables uniquement), UTCTime (heure universelle). Ces types prédéfinis servent de briques de base pour construire les nouveaux types dont vous avez besoin. Par exemple, on définit le nouveau type abstrait MISCMag comme étant composé d'un titre, numéro, et d'une date de parution.

```
MISCMag ::= SEQUENCE {
  titre   UTFString,
  numero  INTEGER,
  date    GeneralizedTime
}
```

Des exemples plus complexes sont disponibles notamment dans le module ASN.1 de [PKCS7].

Attention, l'ASN.1 ne permet de décrire que le **type** des données, mais pas leur **valeur**. Aussi l'ASN.1 est-il généralement utilisé de paire avec un **encodage** de données tel que le BER, CER ou le DER [X.690]. Ces encodages ont pour rôle de décrire la transformation de valeurs de données en chaîne d'octets, et réciproquement. En sécurité, c'est essentiellement le DER (« *Distinguished Encoding Rules* ») qui est utilisé.

Création d'un certificat de test

Tout d'abord, on rappelle qu'un certificat (de clé publique) permet de garantir qu'une clé publique donnée appartient à telle personne (ou organisation). Un certificat contient donc au minimum une clé publique et l'identité du propriétaire de la clé. L'ensemble est signé par l'émetteur du certificat (l'entité qui garantit que la clé appartient au propriétaire).

Plusieurs types de certificats sont possibles, mais dans la pratique, le format le plus utilisé est de loin le format [X.509]. Un dossier complet sur les PKI est disponible dans [MISC13] et pourra vous apporter plus de détails sur le sujet.

Au fil de cet article, nous allons essayer de comprendre plus en détail le contenu de tels certificats. Nous générons d'abord une paire de clés RSA (2048 bits), puis un certificat de test auto-signé.

En exemple, voici les commandes nécessaires pour OpenSSL :

1. D'abord, générer une paire de clés RSA 2048 bits, sauvegardées dans `keys.pem`. Puis émettre un certificat auto-signé valide pour 50 jours, dans `cert.pem`. Les fichiers sont sauvegardés au format PEM.
```
$ openssl req -x509 -newkey rsa:2048 -keyout keys.pem -out cert.pem -
days 50 -outform PEM
```
2. Ensuite, transformer les clés et le certificat au format DER
```
$ openssl rsa -in keys.pem -out keys.der -outform DER
$ openssl x509 -in cert.pem -out cert.der -outform DER
```
3. Enfin, pour importer un certificat, certains navigateurs demandent un format [PKCS12]. Autant donc en créer un :
```
$ openssl pkcs12 -export -in cert.pem -inkey keys.pem -out certkey.p12
```

Sans trop détailler, le Tableau 1 ci-contre récapitule les différents formats utilisés. Les formats DER et PEM sont similaires. Le DER consiste en une suite d'informations binaires, tandis que le PEM est un format texte « lisible ». Ni l'un ni l'autre ne se préoccupent de sécuriser les données qu'ils contiennent, ils servent uniquement à représenter les données.

Le format `PKCS#12`, lui, opère à un niveau au-dessus : il permet d'empaqueter des données (représentées en DER par exemple) de manière publique (non protégées) ou privée (protégées par mot de passe). Typiquement, un fichier `PKCS#12` contiendra une chaîne de certificats et une clé privée, protégée par mot de passe.

Axelle Apvrille
axellec@netcourrier.com

Tableau 1 — *Comparaison entre formats PEM, DER et PKCS#12*

Nom	Format	Protection des données
DER	Binaire	Non
PEM	Texte (encodage Base64 du DER, avec ajout d'en-tête et fin de texte)	Non
PKCS#12	Binaire (au-dessus de DER)	Oui

Examen du certificat X.509 créé

L'examen détaillé du certificat X.509 précédemment créé réserve quelques surprises avec Internet Explorer :

- Pourquoi la clé publique RSA parle-t-elle d'une clé RSA 2048 bits, en affichant ensuite dans la zone détaillée 270 octets (=2160 bits) ?
- On ne trouve pas la valeur de la signature du certificat, seulement l'algorithme employé (MD5+RSA).
- Les champs `thumbprint algorithm` et `thumbprint` nous sont inconnus (X.509 n'en parle pas).

Introduction au DER

Pour répondre à ces questions, il est nécessaire d'approfondir un peu le format DER. Plus haut, nous avons dit que le DER était utilisé de paire avec l'ASN.1. En effet, le DER permet d'encoder les valeurs des structures de données représentées en ASN.1. Pour couper court à toute récrimination éventuelle, on signale au passage qu'un encodage n'est pas un algorithme de chiffrement et qu'il n'assure aucune confidentialité des données. En DER, les données sont représentées sous forme d'une suite d'octets, en utilisant un format dit « TLV » : Tag (étiquette) - Longueur - Valeur.

L'étiquette décrit le type de valeur véhiculée (un entier, un booléen, etc.) et est codée sur un octet (cf. figure 2) :

- les bits 7 et 8 (poids forts) codent la catégorie du type. En effet, il existe des types dits universels, des types propres à une application, spécifiques à un contexte, ou encore privés. Le plus utilisé - universel - se code 00.
- le bit 6 indique s'il s'agit d'un type primitif (0) ou construit (1). Un type est construit s'il peut contenir d'autres types. Par exemple, les ensembles (SET) sont des types construits (ils peuvent contenir entre autres des entiers), mais un booléen est un type primitif (il ne contient rien d'autre que lui-même).
- enfin, les bits 5 à 1 représentent le numéro du type. Le Tableau 2 page suivante indique le numéro des types les plus courants.

Après l'étiquette, la longueur représente le nombre d'octets contenus dans le champ valeur. Si la longueur des données est inférieure ou égale à 127, elle peut être codée sur un seul octet, avec le bit 8 (de poids fort) à 0. Pour des tailles strictement supérieures à 127, on utilise plusieurs octets. Le premier octet comporte toujours le bit 8 à 1, et les 7 autres bits indiquent le nombre d'octets utilisés pour représenter la taille des données. Cette dernière est obtenue en mettant bout à bout tous les bits des octets suivants. Par exemple, la taille 2003 peut être représentée par 10000010 00000111 11010011 en binaire : 10000010 indique

Figure 1 — *Détails du certificat X.509*

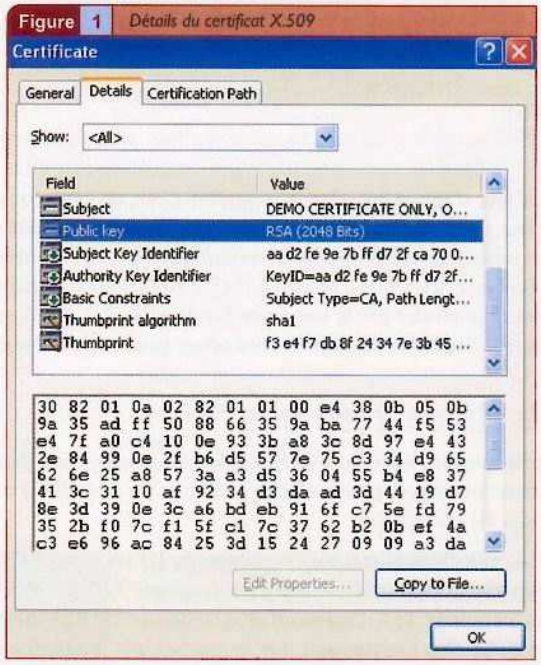
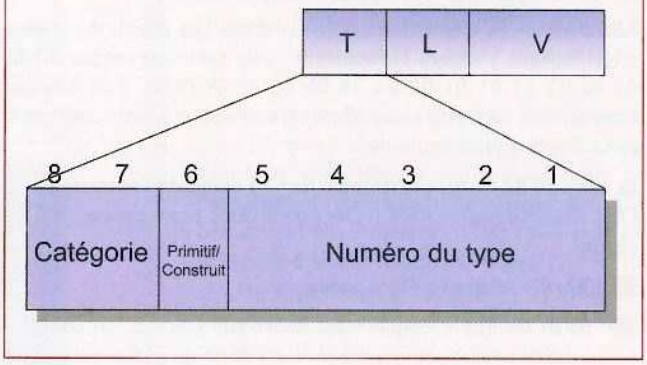


Figure 2 — *Encodage de l'étiquette du type (poids fort sur le bit 8).*



qu'il faut deux octets pour coder la taille, et la juxtaposition 0000 0111 1101 0010 donne le décimal 2003. Enfin, la valeur contient tout simplement la donnée en elle-même.

Pour récapituler, l'encodage DER d'un entier de valeur 7 donnerait le résultat suivant :

→ le type entier (INTEGER) est un type prédéfini en ASN.1 par la valeur « universelle » 2. C'est un type primitif, donc la valeur de l'étiquette est 0000 0010, soit 02 en hexadécimal.

→ la valeur à coder est l'entier 7, donc 07 en hexadécimal.

→ enfin les données tiennent sur 1 octet, donc la taille à mentionner est 01.

Nous avons réalisé notre premier encodage DER : 02 01 07 signifie « entier de valeur 7 ».

Procédons maintenant à l'encodage du type construit suivant, avec a=VRAI, et b=7.

```
ConstructedTrial ::= SEQUENCE {
  a BOOLEAN,
  b INTEGER
}
```

Il nous faut d'abord encoder la séquence. C'est un type construit, de valeur universelle 16. Son encodage est donc de 0011 0000, soit 30 en hexadécimal. En guise de valeur, la séquence contient le booléen vrai (01 01 01), et de l'entier 7 (02 01 07). La taille des données contenues par la séquence est donc de 3 + 3 = 6 octets. La figure 3 détaille le résultat de l'encodage général de la structure.

Une clé de 2160 bits ?!

Les principes de l'encodage DER étant connus, essayons d'élucider pourquoi Internet Explorer affiche 2160 bits (et non 2048) dans le champ « clé publique ».

En RSA, une clé publique est composée de (1) un exposant public (généralement choisi relativement petit, comme 3, 17 ou 65536), et (2) un modulo (grand). Lorsqu'on dit détenir une clé RSA 2048 bits, cela signifie plus exactement que le modulo est de 2048 bits. Le champ « clé publique » de notre certificat contient donc forcément une structure avec le modulo d'une part et l'exposant public d'autre part : il n'est pas étonnant qu'il fasse plus de 2048 bits.

Retrouvons ces éléments dans le certificat. Les détails du champ « clé publique » (figure 1) montrent qu'ils commencent par 30 82 01 0a 02 82 01 01 00 e4 38 0b 05 0b. À l'aide d'un éditeur hexadécimal, on ouvre le certificat `cert.der` pour y retrouver cette suite d'octets (voir figure 4).

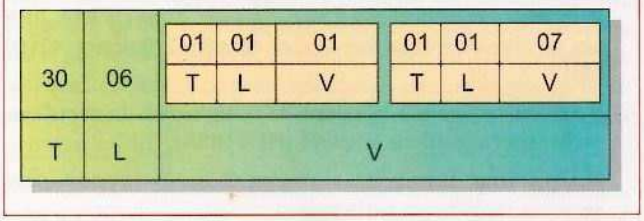
La suite 30 82 01 0a..., ressemble fort au début de l'encodage DER d'une séquence ASN.1. Décodons donc pas à pas ce champ :

1. 30 : (T) étiquette d'une séquence.
2. 82 01 0a : (L) la longueur est codée sur 2 octets : 01 0a. La longueur totale de la séquence est donc de 266 octets. Si on ajoute l'étiquette et les octets de longueur, cela fait bien 270 octets, soit 2160 bits.
3. 02: (t1) on lit maintenant un entier.
4. 82 01 01 : (l1) la longueur est codée sur 2 octets : 01 01. Un entier de 257 octets va donc suivre.

Tableau 2 — *Numéros des types pré-définis les plus courants en ASN.1*

Catégorie et numéro	Type	Construit/ Primitif
UNIVERSEL 1	BOOLEAN	Primitif
UNIVERSEL 2	INTEGER	Primitif
UNIVERSEL 3	BIT STRING	Primitif
UNIVERSEL 4	OCTET STRING	Primitif
UNIVERSEL 5	NULL	Primitif
UNIVERSEL 6	OBJECT IDENTIFIER	Primitif
UNIVERSEL 16	SEQUENCE ou SEQUENCE OF	Construit
UNIVERSEL 17	SET ou SET OF	Construit

Figure 3 — *Encodage DER d'un type construit.*



5. 00 E4 38 0B 05 0B ... FC 69 : (v1) si on ôte le 00 de tête, on a donc là un entier de 256 octets (2048 bits). Pas de doute, c'est bien le modulo de notre clé publique.

Nous n'avons lu que 257 + 3 + 1 = 261 octets, or la structure générale (étapes 1 et 2) nous en a promis 266 : il nous reste donc encore 5 octets à lire : 02 03 01 00 01 (t2-l2-v2) : il s'agit d'un entier (02), de taille 3 octets, et de valeur hexadécimale 01 00 01, soit 65537. Nous venons de retrouver l'exposant public utilisé. Au total, nous avons donc décodé une structure de données qui contenait d'abord un entier (le modulo) puis un autre (l'exposant public), soit quelque chose comme :

```
PublicKey ::= SEQUENCE {
  modulus  INTEGER,
  exponent INTEGER
}
```

Or ceci correspond exactement à la structure RSAPublicKey spécifiée dans [PKCS1].

Dans ce paragraphe, nous avons donc pu (1) retrouver exactement les composantes de notre clé publique, (2) comprendre pourquoi le champ clé publique affiche 2160 bits, alors que la clé est dite « 2048 bits », et enfin (3) montrer que la structure décodée correspond effectivement à la manière « standard » de transmettre des clés publiques RSA.

Où est la signature du certificat ?

Tout certificat numérique est signé par son émetteur, nous allons maintenant retrouver sa signature dans `cert.der`. Plutôt que de chercher au hasard dans le fichier, la lecture de [X.509] (ou, si vous n'y avez pas accès, [RFC2459]) s'avérera intéressante. Les

informations du certificat sont contenues dans la structure ASN.1 `Certificate`. Notamment, on trouve la version du certificat, le propriétaire (`subject`), l'émetteur (`issuer`), les dates de validité du certificat... :

```
Certificate ::= SIGNED { SEQUENCE {
  version         [0] Version Default v1,
  serialNumber    CertificateSerialNumber,
  signature       AlgorithmIdentifier,
  issuer          Name,
  validity        Validity,
  subject         Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo,
  issuerUniqueIdentifier  [1] IMPLICIT UniqueIdnetifier OPTIONAL,
  subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL,
  extensions              [3] Extensions OPTIONAL
}
```

Concernant la signature du certificat, on note que la structure `Certificate` contient un champ signature, qui indique le nom de l'algorithme de signature utilisé (`AlgorithmIdentifier`), et que l'ensemble du certificat est signé (`SIGNED { ... }`). Au passage, on remarque que c'est une excellente idée de signer le nom de l'algorithme de signature utilisé, sinon un attaquant pourrait le modifier et ainsi rendre le certificat invérifiable, ou modifier à sa guise le certificat, en s'arrangeant pour que la signature corresponde à une signature correcte avec le nouvel algorithme choisi (attaque probablement plus difficile à mettre en pratique, mais tout à fait possible en théorie).

La structure `SIGNED` est un raccourci désignant une séquence contenant les données à signer (i.e. la structure `Certificate`), suivies du contenu du raccourci `SIGNATURE`. À son tour, une `SIGNATURE` est une séquence constituée de l'algorithme de signature, et d'un « hash chiffré ». Ce dernier représente tout simplement la valeur de la signature : par construction, une signature numérique est réalisée en deux étapes : on hashe le message, puis on « chiffre » le résultat (signature par annexe).

```
ENCRYPTED-HASH { ToBeSigned } ::= BIT STRING (
 CONSTRAINED BY { -- must be the result of applying
 -- a hashing procedure to the DER-encoded octets
 -- of a value of -- ToBeSigned -- and then applying
 -- an encipherment procedure to these octets } )
SIGNATURE { ToBeSigned } ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier,
  encrypted            ENCRYPTED-HASH { ToBeSigned }
}
SIGNED { ToBeSigned } ::= SEQUENCE {
  toBeSigned      ToBeSigned,
  COMPONENTS OF   SIGNATURE { ToBeSigned }
}
```

Figure 5 *Signature du certificat X.509 dans le fichier cert.der*

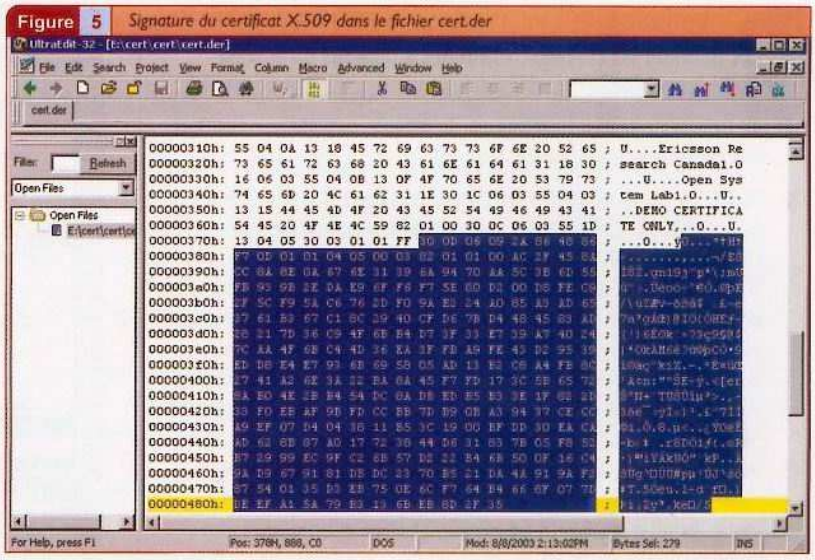
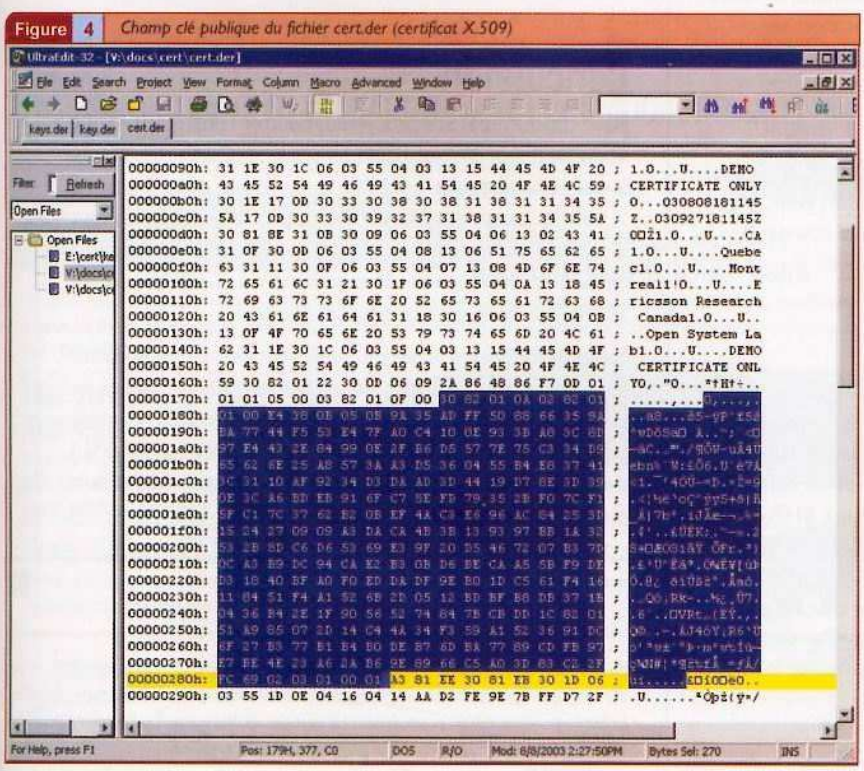


Figure 4 *Champ clé publique du fichier cert.der (certificat X.509)*



Le type `AlgorithmIdentifier` sert à référencer les algorithmes utilisés. Il correspond à un *object identifier*, suivi de paramètres optionnels. Sans rentrer trop dans les détails, il nous suffit de savoir qu'un object identifier (abrégé OID) est une suite d'identifiants organisée sous forme hiérarchique, et maintenue par des organisations telles que l'IANA. Par exemple, l'OID de l'algorithme de signature MD5+RSA est : `{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1 4 }`.

Il en résulte donc qu'un certificat X.509 devrait être composé, dans l'ordre, des données du certificat (`toBeSigned`), suivies du nom de l'algorithme de signature utilisé et ses paramètres (`algorithmIdentifier`), puis enfin de la signature en elle-même (champ `encrypted`). La signature se trouve donc à la fin du fichier, et puisque notre certificat est auto-signé (c'est un certificat de test), on sait que la clé qui a servi à réaliser la signature est la nôtre, i.e. une clé RSA 2048 bits. La signature comportera donc 2048 bits (256 octets). La figure 5 montre la fin du fichier `cert.der` chargé dans un éditeur hexadécimal. Les parties qui nous intéressent sont surlignées.

Il nous faut à nouveau décoder le DER correspondant à la signature :

1. Les données du certificat se situent au-dessus de la zone surlignée en bleu.
2. 30 0D : il s'agit d'une séquence de 13 octets (elle correspond à la structure `AlgorithmIdentifier`)
3. 06 09 2A 86 48 86 F7 0D 01 01 04 : l'étiquette 06 fait référence à un OBJECT IDENTIFIER comportant 9 octets de valeur. L'encodage DER d'un OID est relativement complexe, et je vous invite à lire les détails dans [X.690] (paragraphe 8.19). On en retiendra que l'encodage d'un OID correspond (à peu près, car je simplifie !) à la valeur de chacun des identifiants (sauf pour les deux premiers, et sauf si la valeur ne peut être codée sur un octet). Dans la réalité, la plupart des implémentations ne calculent pas l'encodage DER d'OIDs, mais utilisent des valeurs pré-encodées, en « dur » dans le code. Ainsi, [PKCS1] nous précise-t-il que l'encodage de l'OID 1.2.840.113549.1.1 correspond à 2A 86 48 86 F7 0D 01 01. On en déduit alors facilement que 2A 86 48 86 F7 0D 01 01 04 correspond à l'OID 1.2.840.113549.1.1.4, soit `md5WithRSAEncryption`. Nous venons de retrouver l'algorithme de signature employé.
4. 05 00 : l'étiquette 05 correspond au type NULL, et indique qu'il n'y a pas de paramètres spécifiques à fournir pour cet algorithme de signature.
5. 03 82 01 01 00 AC 2F ... 2F 35 : l'étiquette 03 désigne un BIT STRING, contenant 257 octets (01 01). Cela correspond donc au raccourci ENCRYPTED-HASH. Si on ôte le premier 00 d'en-tête, on lit bien une signature sur 256 octets (AC 2F .. 2F 35).

Nous avons ainsi retrouvé la valeur de la signature du certificat. Il s'agit d'une signature MD5+RSA, effectuée avec une clé RSA 2048 bits, et ayant pour valeur AC 2F ... 2F 35. Curieusement, la valeur de la signature n'est pas disponible dans les détails du certificat X.509 sous Internet Explorer, alors qu'elle fait partie du fichier `cert.der`.

En revanche, le navigateur Mozilla vous permettra de vérifier qu'on ne s'est pas trompés (***Edit/Preferences/Privacy & Security/ Certificates*** puis ***Manage Certificates***, et enfin bouton ***View***, onglet ***Details***, ***Certificate Signature Value*** — oui, il faut le vouloir !) : cf. figure 6.

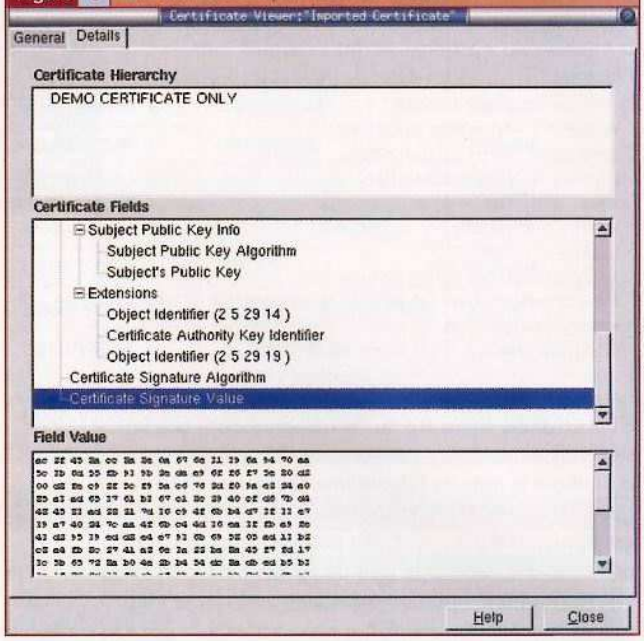
Thumbprint (empreinte) et certificat X.509

Enfin, reste le cas de l'empreinte affichée à la figure 1 : f3 e4 f7 db .. f0 5f. Puisque l'algorithme d'empreinte (*thumbprint algorithm*) mentionné au-dessus est SHA-1, l'empreinte est très certainement le résultat d'un hashage effectué avec l'algorithme SHA-1. On recherche alors la chaîne f3 e4 f7 db .. f0 5f dans le fichier `cert.der` : rien !

Cette chaîne ne fait donc pas partie du certificat, elle est donc probablement calculée par Internet Explorer. Pas besoin de beaucoup d'intuition pour essayer (à tout hasard) de hasher le fichier certificat `cert.der` :

```
$ shalsum cert.der
f3e4f7db8f24347e3b45ad97899318697e27f05f  cert.der
```

Figure 6 *Vue détaillée de certificat X.509 sous Mozilla*



C'est bien ça ! L'empreinte mentionnée est donc le hash SHA-1 du fichier du certificat X.509. L'idée est probablement de permettre à l'utilisateur de vérifier rapidement qu'il vient d'importer le certificat qu'il voulait, via comparaison de hash. Pourquoi pas, mais il ne faut surtout pas s'y méprendre : la sécurité d'un certificat tient à la vérification de sa signature, et non pas à une simple comparaison de hash.

Conclusion

Pour conclure, cet article a voulu montrer l'utilité de l'ASN.1 et du DER aux développeurs en sécurité. Ceci a été illustré à travers une étude détaillée du contenu des certificats X.509. En particulier, nous avons pu localiser précisément le modulo et l'exposant public de la clé certifiée, la signature digitale du certificat, et enfin comprendre à quoi le *thumbprint* affiché par Internet Explorer correspond.

Nous laisserons maintenant au lecteur le soin de décoder tout un certificat X.509 ou un fichier `PKCS#12` ! Et si un soupçon de paresse vous guette, des outils comme [CRYPTIX], [OBJSYS] ou tout simplement [OPENSSL] devraient vous aider (d'autres liens sont également disponibles sur [ASN1INF] et [SNMP]).

Pour cet article, je tiens à remercier Ericsson Research Canada, et notamment Simon Jubinville et David Gordon pour leurs remarques et relecture détaillée. Enfin et pour une fois, je souhaiterais remercier mon mari, relecteur anonyme sans égal (et sans pitié !) de la plupart de mes articles.

Références

[ASN1INF] Site d'information sur l'ASN.1 - **http://asn1.elibel.tm.fr/fr/index.htm**

[ASN1ORG] ASN.1 Consortium - **http://www.asn1.org**

[CMS] R. Housley, *Cryptographic Message Syntax*, Network Working Group, RFC 2630, June 1999, **http://www.ietf.org/rfc/rfc2630.txt**

[CRYPTIX] Cryptix ASN.1 Kit - **http://sourceforge.net/projects/cryptix-asn1**

[MISC13] MISC n°13, *PKI : Public Key Infrastructure, Ou comment organiser et gérer votre cryptographie*, mai/juin 2004

[RFC2459] R. Housley, W. Ford, W. Polk, D. Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, Network Working Group, RFC 2459, January 1999, **http://www.ietf.org/rfc/rfc2459.txt**

[RFC3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*, Network Working Group, RFC 3161, August 2001, **http://www.ietf.org/rfc/rfc3161.txt**

[PKCS1] PKCS#1 - ASN Module for PKCS#1 v2.1 - RSA Laboratories, **ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.asn**

[PKCS7] PKCS#7 - Cryptographic Message Syntax Standard - Version 1.5 - RSA Laboratories, **http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/**

[PKCS12] PKCS#12 - Personal Information Exchange Syntax - Version 1.0 - RSA Laboratories, **http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/**

[SNMP] SNMP Link - **http://www.snmplink.org/ASN1.html**

[OBJSYS] ASN.1 Viewer - Objective Systems Inc. - **http://www.obj-sys.com/freesoftware.shtml**

[OPENSSL] OpenSSL - **http://www.openssl.org**

[X.509] *Information technology - Open Systems Interconnection - The Directory: Authentication framework* - ITU-T Series X: Data Networks and Open System Communications - X.509

[X.680] *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation* - ITU-T Series X: Data Networks and Open System Communications - X.680

[X.690] *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)* - ITU-T Series X: Data Networks and Open System Communications - X.690

[X9.84] *Biometric Information Management and Security* - ANSI X9.84 - **http://www.x9.org**

IPSec et Linux 2.6

Cette fiche pratique concerne l'utilisation du protocole IPSec dans le noyau Linux 2.6. En effet, dans le dossier VPN de MISC 10 et sa partie consacrée à IPSec [1], la précédente fiche pratique [2] s'intéressait au noyau 2.4 et à FreeS/Wan. Celle-ci détaille les solutions et leurs mises en œuvre dans un environnement Linux 2.6 : nous verrons que beaucoup de choses ont changé !

Rappels sur l'environnement de tests

Le but de cette fiche pratique est de configurer un client Linux avec IPSec, afin de sécuriser les communications réseau entre ce client (dans le sous-réseau 192.168.1.0/24) et une passerelle. L'environnement de tests est le même que celui de l'article sur la sécurisation des communications sans fil de MISC 10 [3]. Dans ce cas, la passerelle est une machine OpenBSD (mise à jour en OpenBSD 3.5 pour cette fiche) et les configurations IPSec doivent sécuriser les communications entre les clients sans fil (Slackware 9.1 et Debian unstable avec un noyau 2.6.6) et la passerelle. Enfin, la passerelle doit permettre l'accès au réseau se situant derrière elle. Tout ceci est résumé de manière plus visuelle sur la figure 1.

Les fonctionnalités IPSec dans le noyau Linux 2.6

Le noyau Linux 2.6 (à la différence du 2.4) comporte les fonctionnalités permettant d'utiliser IPSec. Il n'y a donc plus besoin de le patcher. En revanche, une recompilation sera peut-être nécessaire pour inclure ces fonctions. Elles sont fondées sur la CryptoAPI (***Cryptographic options***). Cette interface intègre des algorithmes de chiffrement, de signature et de hash afin de fournir les éléments cryptographiques de bases . Pour utiliser IPSec, la CryptoAPI et ses différents algorithmes doivent être validés. Les composants IPSec proprement dits sont dans les options du réseau (***Device drivers -> Networking support -> Networking options***) et sont au nombre de 5 :

- ***PF_key socket*** : interface de communication permettant de définir les politiques de sécurité et les associations de sécurité propres à IPSec ;
- ***AH transformation*** : implémentation du protocole AH de IPSec ;
- ***ESP transformation*** : implémentation du protocole ESP de IPSec ;
- ***IPComp transformation*** : implémentation du protocole IPComp (compression) de IPSec ;
- ***IPSec user configuration interface*** : interface avec les outils de configuration en mode utilisateur.

Il faut donc sélectionner ces 5 options (en module, par exemple) ainsi que les composantes de la CryptoAPI (en module, également), puis recompiler le noyau et les modules, installer le tout (configuration du *boot loader*) et rebooter.

Les IPSec-Tools (Racoon)

Disposer d'un noyau Linux intégrant toutes les fonctionnalités pour faire de l'IPSec, c'est bien, mais encore faut-il pouvoir configurer ces fonctionnalités. C'est ce que vont permettre les commandes en mode utilisateur. Deux solutions existent : `isakmpd` et les IPSec-Tools. C'est cette dernière solution que nous allons présenter.

Les IPSec-Tools [6] constituent le package des composantes IPSec en mode utilisateur. Il est censé devenir le package standard de toute distribution Linux incluant IPSec (déjà disponible dans toutes les bonnes distrib).

Il contient plusieurs choses :

- l'exécutable `setkey` : pour créer les associations de sécurité et les politiques de sécurité ;
- l'exécutable `racoon` : pour implémenter le protocole IKE et faisant partie du projet Kame [9] ;
- la librairie `lfbipsec` : pour l'interfaçage avec le noyau ;
- des pages de man : pour les exécutables et les fichiers de configuration ;
- des fichiers de configuration de base : `racoon.conf` et `psk.txt`.

Pour les distributions Linux n'intégrant pas un package IPSec-Tools, on peut récupérer les sources sur Sourceforge [6].

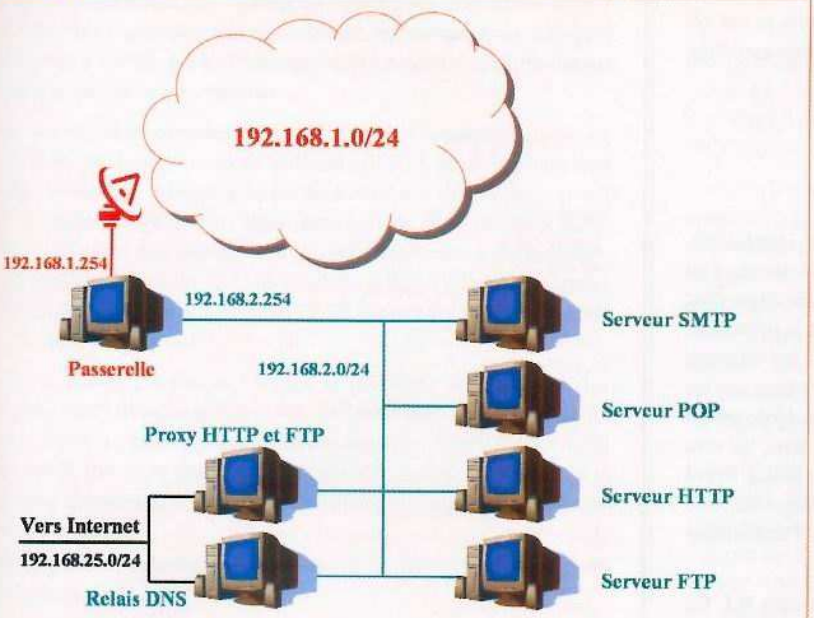
Configuration manuelle des clés

Ce type de configuration consiste à fixer, de manière explicite, les associations de sécurité et les définitions de politique de sécurité. Une association de sécurité (SA) précise, pour une communication réseau donnée (adresse source et destination), le protocole IPSec utilisé (ESP ou AH), son index (un numéro appelé SPI, qui doit être le même à chaque bout du VPN), le mode (tunnel ou transport), puis les algorithmes de chiffrement et de signature, ainsi que les clés associées. Une définition de politique de sécurité (SPD) fixe ce que le noyau doit faire d'un paquet qui entre ou qui sort de la pile IP. Ainsi, pour une communication réseau donnée (adresse source et destination, port et protocole : UDP, TCP, ICMP), une SPD explicitera comment le paquet devra être traité : transmis en clair, chiffré par IPSec en mode transport en utilisant ESP, chiffré par IPSec en mode tunnel (on fournira les adresses IP de l'entrée et de la sortie du tunnel) en utilisant ESP... Une SPD dit ce que l'on doit faire d'un paquet, tandis que la SA précise comment chiffrer le paquet. Ainsi, une SPD s'appuie sur une SA pour savoir comment traiter un paquet.

C'est l'utilitaire `setkey` qui configure les SA et les SPD. Cette commande prend un fichier en paramètre du connecteur `-f`. Ce

Frédéric Combeau
fred@rstack.org
Ingénieur-Chercheur en Sécurité des Systèmes d'Information au Commissariat à l'Energie Atomique.

Figure 1



fichier contient les commandes permettant de définir les SA et les SPD. La commande `setkey -D` permet de visualiser les SA, tandis que `setkey -DP` permet de visualiser les SPD.

Le fichier suivant permet de créer une configuration simple, adaptée à notre environnement présenté à la figure 1 (le client a comme adresse IP 192.168.1.1 et la passerelle 192.168.1.254, le réseau derrière la passerelle est 192.168.2.0/24) :

```
# /usr/local/etc/racoon/setkey.conf
## Configuration des SAs
### On efface toutes les SAs
flush;
### Sens client-passerelle
add 192.168.1.1 192.168.1.254 esp 0x201 -m tunnel
    -E 3des-cbc 0x0000000000000000000000000000000000000000000000000000
    -A hmac-md5 0x00000000000000000000000000000000;
### Sens passerelle-client
add 192.168.1.254 192.168.1.1 esp 0x301 -m tunnel
    -E 3des-cbc 0x0000000000000000000000000000000000000000000000000000
    -A hmac-md5 0x00000000000000000000000000000000;
## Configuration des SPDs
### On efface toutes les SPDs
spdflush;
### Le trafic entre la passerelle et le client nécessite IPSec
spdadd 192.168.1.1/32 192.168.2.0/24 any -P out
      ipsec esp/tunnel/192.168.1.1-192.168.1.254/require;
spdadd 192.168.2.0/24 192.168.0.7/32 any -P in
      ipsec esp/tunnel/192.168.1.254-192.168.1.1/require;
```

La SA de SaPI 0x201 de type tunnel configure le sens client-passerelle (la SA de SPI 0x201 sur la passerelle devra définir la même chose, dans le même sens, avec les mêmes algorithmes et les mêmes clés), en mode ESP avec l'algorithme `3des-cbc` pour le chiffrement avec la clé donnée en hexadécimal (il est possible de la fournir sous forme d'une chaîne de caractères) et l'algorithme `hmac-md5` pour la signature avec la clé donnée en hexadécimal (qui peut être, aussi, une chaîne de caractères). La SA de SPI 0x301 configure le sens passerelle-client.

Il est à noter que les algorithmes peuvent être différents, ainsi que les différentes clés. Mais tout ceci doit être cohérent avec la configuration de la passerelle.

De plus, les clés présentées ne sont là que pour la présentation, prenez autre chose que des clés à 0 (`/dev/urandom` est votre ami pour trouver des clés). Enfin, selon les algorithmes choisis, les longueurs de clés peuvent être fixées. Il faut donc prendre une clé avec la bonne longueur, sous peine de se voir insulter par `setkey`.

Les SA fixées, vient le tour des SPD. La première explicite que tout paquet provenant du client à destination du réseau 192.168.2.0/24 devra utiliser IPSec en mode ESP en créant un tunnel entre le client et la passerelle et que ceci est obligatoire. Les SPD, comme les SA, n'étant pas bi-directionnelles, il faut toujours préciser les deux sens. Pour une communication en mode transport (de machine à machine), on aurait écrit « transport » à la place de tunnel, sans renseigner le champ suivant (puisque ne servant à rien) et terminer par « require », pour signifier que l'utilisation d'IPSec est obligatoire.

Avant d'utiliser la commande `setkey`, il faut charger les modules noyau assurant les fonctionnalités IPSec. Ces modules ne sont pas chargés automatiquement par `setkey` (c'est un bogue en cours de correction). Il faut donc les charger à la main.

Pour cela, on fera : `modprobe esp4` ; `modprobe ah4` ; `modprobe ipcomp`.

Pour appliquer la configuration, on lancera la commande `setkey -f /usr/local/etc/racoon/setkey.conf`. À partir de là, tout paquet qui matchera une SPD sera traité selon la SA correspondante et envoyé sur le réseau. La passerelle recevant ce paquet procédera de la même manière et déchiffrera et authentifiera le paquet avant de l'envoyer sur le réseau interne. L'utilisation sur le client d'un `ping` vers ce réseau avec un `tcpdump` permettra de vérifier que le paquet est bien encapsulé dans un paquet IPSec.

En mode tunnel, un `tcpdump` sur l'interface réseau montrera un paquet entrant sur la machine, en clair, juste après le paquet IPSec correspondant. En fait, le paquet en clair ne circule pas sur le réseau. Il n'y a donc pas à s'inquiéter. Sur le réseau, tous les paquets qui circulent sont bien des paquets IPSec. Cependant, le paquet entrant est visible sur le client.

Ce comportement peut être un problème si le client est multi-utilisateur et que chaque utilisateur doit être cloisonné au niveau de la vision qu'il a du trafic réseau entrant. Il est présent quel que soit le mode d'authentification utilisé, à partir du moment où l'on utilise le mode tunnel. Ceci s'explique par l'implémentation du mode tunnel qui désencapsule le paquet en clair à partir du paquet IPSec reçu, et le réinjecte, tel quel, dans la pile IP du client, comme si IPSec n'avait pas été utilisé.

Il faudra prendre en compte cette implémentation dans le cas où on utilise des règles Netfilter (Iptables) sur le client, pour autoriser le paquet en clair entrant, en plus du flux IPSec.

Utilisation des IPsec-tools avec un secret partagé

Même si la configuration manuelle a des avantages (simplicité, interopérabilité), les inconvénients sont importants, surtout au niveau de la sécurité, puisque les clés ne changent pas. Afin de profiter de clés changées périodiquement, il convient d'utiliser le protocole IKE. Dans une phase 1, il va authentifier les machines qui désirent communiquer via IPSec. Dans une phase 2, il va négocier les algorithmes de chiffrement et de signature, puis les clés. Ainsi, celles-ci sont régulièrement changées puisqu'une phase 2 (avec les clés correspondantes) n'est valide que durant un temps limité. Passé cette période de validité, une nouvelle phase 2 est négociée avec de nouvelles clés. Le secret partagé, quant à lui, permet d'authentifier les machines au cours de la phase 1.

Avec les IPSec-Tools, c'est Racoon qui fournit le protocole IKE. Ce sera donc le processus racoon qui réalisera les phases 1 et 2, et générera les SA qui seront utilisées pour chiffrer et authentifier les paquets IPSec.

Les SPD devront être fixées par la même méthode que pour la configuration manuelle, c'est-à-dire avec `setkey`. Il faudra donc avoir un fichier `setkey.conf`, comme nous l'avons vu à la section précédente, mais sans la partie concernant les SA.

Racoon possède deux fichiers de configurations :

- → `racoon.conf` qui définit le comportement du processus ;
- → `psk.txt` qui contient les secrets partagés.

`racoon.conf` est décomposé en plusieurs parties. Seules les parties « `remote` » et « `sainfo` » sont intéressantes, les autres étant plus statiques. La partie « remote » précise le comportement de `racoon` pour la phase 1 d'authentification pour une machine distante donnée (le mot-clé « `anonymous` » permet de donner des paramètres par défaut à utiliser pour toutes les machines non explicitement renseignées). La partie « `sainfo` » définit le comportement de `racoon` pour la phase 2 (négociation des algorithmes et des clés et création des SA correspondantes).

Un fichier de configuration adapté à notre environnement de test, pourrait être le suivant :

```
# /usr/local/etc/racoon/racoon.conf

# Ce fichier contient les secrets partagés
path pre_shared_key «/usr/local/etc/racoon/psk.txt» ;

# Le niveau de log du processus
log debug;

# Paramètres de padding. Rien à modifier
padding
{
 maximum_length 20;
 randomize off;
 strict_check off;
 exclusive_tail off;
}

# Sur quelle adresse IP, racoon doit-il écouter ?
listen
{
 isakmp 192.168.1.1 [500];
}

# Paramètres des timers
timer
{
 counter 5;
 interval 20 sec;
 persend 1;
 # Temps d'attente pour compléter une phase
 phase1 30 sec;
 phase2 15 sec;
}

# Comportement de racoon vis-à-vis de la passerelle pour la phase 1
remote 192.168.1.254
{
 # on commence par le mode normal, puis le mode agressif
 exchange_mode main,aggressive;
 # Je m'identifie par mon adresse IP
 my_identifier address;
 # La passerelle fait de même
 peers_identifier address;
 # Je vérifie que le ID payload appartient bien à la passerelle
 verify_identifier on;
 # Durée de vie proposée durant la négociation de la phase 1
 lifetime time 1 min;
 # Le client initie le contact avec la passerelle
 initial_contact on;
 # La passerelle obéit aux propositions de la passerelle pour la durée de vie
 # et la politique d'échange de clé de la phase 2
 proposal_check obey;

 # Nos propositions concernant les algorithmes de la phase 1
 proposal
 {
  # 3des pour le chiffrement (AES n'est pas supporté par racoon)
  encryption_algorithm 3des;
  # Sha1 pour le hashage
  hash_algorithm sha1;
  # Authentification en clé partagée
  authentication_method pre_shared_key;
  # Fixe le groupe utilisé par l'exponentiation Diffie-Hellman
  dh_group 2;
 }
}

# Définition de l'association de sécurité à utiliser en Phase 2
# Depuis le client vers la passerelle
sainfo address 192.168.1.1 any address 192.168.1.254 any
{
 # Groupe utilisé par l'exponentiation Diffie-Hellman
 pfs_group 1;
 # Durée de vie proposée pour les SA de la phase 2
 lifetime time 30 sec;
 # Proposition des algorithmes de chiffrement
 # D'abord AES, puis 3des si AES n'est pas supporté par la passerelle
 encryption_algorithm rijndael, 3des;
```

```
 # Proposition des algorithmes de signature
 authentication_algorithm hmac_sha1;
 # Proposition d'utilisation de la compression des paquets IPsec
 compression_algorithm deflate;
}
```

Le fichier de configuration étant abondamment commenté, sa compréhension ne devrait pas trop poser de problèmes, sachant que la page de man de `racoon.conf` est assez exhaustive. On pourra également consulter [10] qui est un *howto* pour configurer `racoon` sur FreeBSD. Cet article donne un fichier de configuration, quelques explications d'IPSec, parle d'interopérabilité avec le monde Windows et donne des liens intéressants.

Une petite chose, cependant : `racoon`, à la différence de `isakmpd`, ne supporte pas l'algorithme de chiffrement AES pour la phase 1. Il faudra donc faire attention au paramétrage par défaut de `isakmpd`, qui souhaitera négocier cet algorithme lors de cette phase. Si 3DES vous semble un peu faible, il est possible de passer à du Blowfish, supporté par `racoon` et `isakmpd` (nommé « BLF »). En phase 2, tout va bien car les deux supportent l'AES (nommé « rijndael » pour `racoon` et « AES » pour `isakmpd`).

Enfin, le fichier de secret partagé (à protéger en adoptant des permissions adéquates, `chmod 600`, par exemple) est composé de deux colonnes. La première définit la machine à authentifier (son adresse IP, son nom qualifié ou une adresse e-mail, selon ce que la machine distante envoie pour s'identifier) et la seconde le secret partagé utilisé.

Comme pour la configuration manuelle, n'utilisez pas les secrets partagés donnés ici :

```
# /usr/local/etc/racoon.psk.txt
# IPv4/v6 addresses
192.168.1.254           mekmitasdigoat
# USER_FQDN
fred@rstack.org         mekmitasdigoatdefred
# FQDN
client1.rstack.org      mekmitasdigoatdefclient1
```

Comme nous utilisons une identification par adresse IP, le secret partagé utilisé sera la chaîne de caractères « mekmitasdigoat ». Cette chaîne sera utilisée pour une authentification réciproque avec la passerelle.

Maintenant que tout est défini, il faut lancer `racoon`. Dans les premières configurations, on préférera lancer `racoon` avec les connecteurs `-d` pour activer le mode *debug* et `-F` pour laisser `racoon` attaché au terminal (pas de lancement en mode démon). On pourra alors tester la configuration en vérifiant que les communications s'effectuent bien dans un tunnel IPSec.

Utilisation des IPSec-Tools avec des certificats x509

Dans la section précédente, le secret, qui permettait d'authentifier une machine, était « partagé ». Ce n'est pas ce que l'on fait de mieux pour lui conserver sa qualité de « secret », puisqu'il sera présent sur différentes machines (au moins le client et la passerelle). Le risque de vol est donc multiplié.

Avec des certificats, il n'y a plus de partage de secret : une partie secrète reste uniquement sur la machine et une partie publique est diffusée. De plus, une autorité de certification (CA) signe la partie publique, certifiant que les informations contenues dans le certificat appartiennent bien à la machine définie par ce certificat.

La création de certificats nécessite la mise en place préalable d'une CA, puis chaque certificat de machine devra être signé par ce CA. La création de CA et de certificats est décrite dans [8] (qui contient, également, de précieux conseils sur l'utilisation d'IPSec sur différentes plates-formes). Ces certificats doivent être au format x509 et OpenSSL est tout à fait capable de remplir cet office.

`isakmpd` nécessite d'enrichir ces certificats, car les informations de base stockées ne lui sont pas suffisantes. Il a besoin d'une extension, nommée `SubjectAltName` contenant un identifiant de la machine. Cet identifiant peut être une adresse IP, un nom totalement qualifié (FQDN) ou une adresse e-mail (UFQDN).

`racoon` n'a pas besoin de cette extension, mais il peut l'utiliser. Par contre, il ne supporte pas comme extension l'adresse IP. En fait, il l'ignore purement et simplement, ce qui conduit à ne pas pouvoir authentifier la passerelle en utilisant cette extension. Les documentations ne mentionnent pas ce problème, mais nos tests nous ont appris que l'authentification ne s'effectuait pas si on voulait vérifier le certificat de la machine distante et sa chaîne de certification. Par contre, si l'extension utilisée est la FQDN ou le UFQDN, tout fonctionne parfaitement.

Pour ajouter cette extension aux certificats, deux solutions sont possibles : on l'ajoute au moment de la création du certificat (la configuration de OpenSSL, par défaut, doit être modifiée) ou on l'ajoute après la création du certificat. Pour ce dernier cas, il faut utiliser l'utilitaire `certpatch` (disponible avec les sources d'`isakmpd` ou sur une machine OpenBSD).

Au niveau de la configuration de `racoon`, peu de choses changent. Le fichier `setkey.conf` ne bouge pas et précise toujours les mêmes SPD. Son utilisation reste nécessaire. Le fichier de configuration de racoon subit quelques changements :

- Définition d'un répertoire contenant les différents certificats à utiliser (certificats des machines, certificats des CA, clé secrète associée à la machine), le fichier `psk.txt`, par contre, ne sert plus à rien ;
- Définition du mode d'identification de la machine locale et distante (`asn1dn`) ;
- Définition du fichier de certificat de la machine locale et du fichier contenant la clé secrète ;
- Définition du fichier de certificat de la machine distante ;
- Directive de vérification du certificat de la machine distante par la chaîne d'autorité des CA (`verify_cert on`) ;
- Mode d'authentification de la phase 1 : `rsasig` (pour une authentification par certificats).

Le répertoire de stockage des certificats (ici `/usr/local/etc/racoon/certs`) doit avoir un accès limité puisqu'il contient la clé secrète de la machine locale (le fichier contenant cette clé secrète devra être, lui aussi, limité en lecture). Ce répertoire contient également les certificats des machines à authentifier, le ou les certificat(s) des CA et une liste de révocation (liste de certificats pour un CA donné ne devant plus être utilisés pour authentification).

La liste de révocation est à créer (grâce à OpenSSL) lors de la construction des certificats et doit être mise à jour régulièrement.

Les fichiers des CA et de la CRL (liste de révocation) peuvent avoir n'importe quel nom de fichier, mais des liens devront pointer sur ces fichiers.

Ces liens se nommeront `"hash du certificatCA".0` pour le lien pointant sur un fichier contenant un certificat de CA et `"hash du certificatCA".r0` pour un lien pointant sur un fichier contenant une CRL d'un CA. Le hash est donné par la commande `openssl x509 -noout -hash -in cacert.pem` où `cacert.pem` est le fichier contenant le certificat d'une CA.

Tout ceci fait, on obtient le fichier de configuration suivant :

```
# /usr/local/etc/racoon/racoon.conf

# Ce répertoire contient les différents certificats et la clé privée locale
path certificate «/usr/local/etc/racoon/certs» ;

#
# Idem que pour le secret partagé
#

# Comportement de racoon vis-à-vis de la passerelle pour la phase 1
remote 192.168.1.254
{
 # on commence par le mode normal, puis le mode agressif
 exchange_mode main,aggressive;
 # Je m'identifie par mon nom qualifié
 my_identifier asn1dn;
 # La passerelle fait de même
 peers_identifier asn1dn;
 # Je vérifie que le ID payload appartient bien à la passerelle
 verify_identifier on;
 # Mes certificats de type x509, la partie publique puis la clé privée
 certificate_type x509 «client.pem» «client.key»;
 # Le certificat de la machine distante
 peers_certfile «passerelle.pem»;
 # On vérifie le certificat de la machine distante et la chaîne des CA
 verify_cert on;
 # J'envoie mon certificat public
 send_cert on;

 #
 # Idem que pour le secret partagé
 #

 # Nos propositions concernant les algorithmes de la phase 1
 proposal
 {
  # Idem que pour le secret partagé sauf :
  # Authentification par certificat (remplace pre_shared_key)
  authentication_method rsasig;
 }
}

#
# Idem que pour le secret partagé
#
```

Le lancement de `racoon` reste le même que celui présenté pour une authentification par secret partagé.

Conclusion

Linux est un système d'exploitation qui s'est vu doté de la fonctionnalité IPSec assez tardivement. Avant la version 2.6 du noyau, IPSec était disponible uniquement sous forme de patch, voire de plusieurs si on voulait utiliser l'authentification par certificats [2]. A partir de la version 2.6, IPSec est intégré au noyau. Il n'y a plus besoin d'appliquer de patch. De plus, deux solutions sont disponibles. Il s'agit de portages de solutions déjà utilisées sur d'autres systèmes d'exploitation : Racoon sur FreeBSD et Mac OS X et Isakmpd sur OpenBSD ([5]). Racoon, qui est la solution présentée dans cette fiche pratique, permet de profiter d'une solution mature, disposant de documentation et de fichiers de configuration réutilisables entre différents systèmes d'exploitation. Elle fonctionne parfaitement, même si l'interopérabilité entre des solutions différentes présente quelques difficultés.

Chaque solution peut avoir ses propres exigences et ses « secrets » de configuration. Alors qu'il n'y a pas de grosse difficulté à faire fonctionner deux machines utilisant la même solution, même sur des systèmes d'exploitation différents, l'interopérabilité doit être soigneusement testée avant tout déploiement pour obtenir une configuration satisfaisant les deux solutions.

Un grand merci à VG **<vg@rstack.org>**.

Ingénieur-Chercheur en Sécurité des Systèmes d'Information
au Commissariat à l'Energie Atomique.

Références

[1] : Dossier IPSec, *Sécurisation de communications sans fil avec IPSec*, MISC 10

[2] : Fiche pratique *IPSec et Linux*, MISC 10

[3] : Article *Sécurisation des communications WiFi avec IPSec*, MISC 10

[4] : Article *Introduction aux VPN (Virtual Private Network)*, MISC 10

[5] : Fiche pratique *IPSec sur OpenBSD, FreeBSD et Mac OS X*, MISC 10

[6] : **http://ipsec-tools.sourceforge.net** : sources des IPSec-Tools

[7] : **http://www.spenneberg.org/VPN/Kernel-2_6_IPsec** : packages rpm compilés des IPSec-Tools et isakmpd

[8] : **http://www.ipsec-howto.org** : article très intéressant sur l'utilisation de IPSec sur différents Unix (inclus la création des certificats)

[9] : **http://www.kame.net/racoon** : page du projet Racoon avec documentation, liens, RFCs et sources CVS

[10] : **http://www.daemonnews.org/200101/ipsec-howto.html** : howto IPSec FreeBSD avec Racoon

IPSEC et routeur CISCO

Marie Bordesoules
mary@rstack.org
Ingénieur-Chercheur
au Commissariat à L'Energie Atomique

Cette fiche technique décrit les différentes phases de configuration d'un routeur Cisco pour la mise en place d'IPSEC. Le routeur Cisco peut être vu comme un client potentiel de la passerelle sous OpenBSD de l'article « Sécurisation des communications WIFI avec IPSec » dans le MISC 10 [1], où une station (192.168.1.1) venait se connecter à un réseau local (10.0.2.0) via une passerelle OpenBSD (192.168.1.254 / 10.0.2.254) configurée pour échanger avec ce client du trafic IPSec.

Dans un premier temps, cet article décrit quelques commandes de base de Cisco utilisées avec IPSEC. Puis il montre, dans un deuxième temps, un exemple simple de configuration d'un routeur destiné à échanger du trafic IPSEC avec un hôte distant.

Installer IPSEC sur un routeur Cisco

Plusieurs modèles de routeurs Cisco (de 800 à 7200) peuvent supporter un système d'exploitation avec le protocole IPSEC. Tous les IOS (*Internetworking Operating System*) n'ont pas forcément l'implémentation d'IPSEC en natif. Prenons l'exemple d'un cisco 2600, de version 12.3. L'IOS est contenu dans la flash du routeur Cisco. Au boot du routeur, IPSEC est donc utilisable immédiatement. Des informations supplémentaires à ce sujet se trouvent sur le site Cisco [2].

Le fichier de configuration du IOS de Cisco

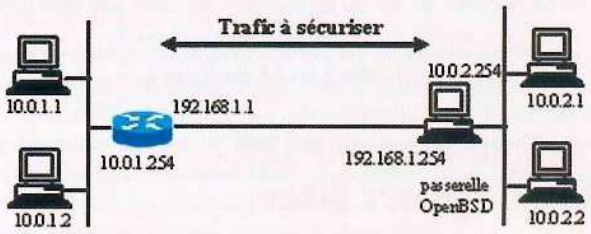
La configuration du routeur se trouve dans la NVRAM (RAM non volatile) et peut être visualisée grâce à la commande `#show config` dans le mode privilégié (`enable`) après avoir tapé le mot de passe correspondant.

Ce fichier contient toute la configuration du routeur, aussi bien les paramètres généraux du routeur (mot de passe « enable », `hostname` ...) que les configurations des différentes interfaces, la table de routage et les lignes VTY (interfaces virtuelles).

La future configuration d'IPSEC s'inscrira aussi ici :

- les paramètres de configuration d'ISAKMP (si on utilise ce protocole de gestion de clés), utiles pour négocier la politique de sécurité de la phase 1 (algorithmes, clés, durée de vie du tunnel ISAKMP) se trouveront à la suite de la ligne de configuration commençant par `crypto isakmp` ;
- les paramètres de configuration d'IPSEC nécessaires pour la négociation de la politique de sécurité de la phase 2 (protocoles, algorithmes, durée de vie du tunnel IPSec) se trouveront à la suite de la ligne de configuration commençant par `crypto ipsec` ;

Figure 1



- enfin, la description d'une *crypto map* rassemblant les paramètres des deux phases, l'extrémité distante du tunnel et la définition du trafic à sécuriser se trouvera à la suite de la ligne de configuration commençant par `crypto map`.

Ces configurations doivent être cohérentes avec la configuration de la passerelle.

Les commandes sous IOS

Les paramètres par défaut du routeur Cisco ne sont pas visibles dans la configuration en tapant simplement `#show config`.

Il est bon de vérifier la configuration exacte ISAKMP/IPSEC à l'aide des commandes suivantes :

→ `#show crypto isakmp policy` affiche la configuration complète d'ISAKMP ;

→ `#show crypto ipsec transform-set` affiche les algorithmes, les protocoles et le mode d'IPSEC ;

→ `#show crypto ipsec security-association lifetime` affiche la durée de vie du tunnel IPSEC ;

→ `#show crypto map` affiche le trafic à sécuriser et les paramètres utilisés.

Une fois que les configurations sont entrées (ce qu'on détaillera plus loin), avant d'établir la communication entre les deux hôtes, il est recommandé de lancer le mode `debug` :

→ `#debug crypto isakmp` (dans le cas où on utilise la négociation ISAKMP) ;

→ `#debug crypto ipsec`.

Le mode `debug` permet de visualiser les différentes étapes de la mise en place d'ISAKMP puis d'IPSEC et ainsi de mettre en évidence des problèmes de configuration.

Une fois la connexion établie entre la passerelle OpenBSD et le routeur Cisco (en lançant un `ping` par exemple), on peut vérifier les paramètres de la liaison IPSEC ainsi construite en utilisant les commandes suivantes :

→ `#show crypto isakmp sa` permet de visualiser la SA (*Security Association*) bidirectionnelle de phase 1 : on y trouve l'hôte distant et le numéro de séquence associé à ce tunnel de phase1

→ `#show crypto ipsec sa` permet de visualiser les SA unidirectionnelles de phase 2 : on y trouve le nombre de paquets encapsulés et décapsulés par le tunnel, les identités locale et distante, la crypto map utilisée, le SPI, etc.

Les commandes suivantes permettent d'effacer les SA sans attendre la fin de la durée de vie du tunnel (lors de tests par exemple) :

→ `#clear crypto isakmp` efface la SA de phase 1;

→ `#clear crypto sa` efface les SA de phase 2.

Configuration d'IPSEC sur un routeur Cisco

Le trafic à sécuriser se trouve entre le routeur Cisco (192.168.1.1) et la passerelle (192.168.1.254) (cf le schéma précédent).

Le protocole IPSEC est basé sur des algorithmes utilisant des clés. Pour communiquer, les deux hôtes doivent connaître les clés : soit les clés sont entrées manuellement dans la configuration des machines, soit elles sont échangées à l'aide du protocole de gestion de clés ISAKMP.

1-Configuration manuelle des clés

Ici, les clés des algorithmes utilisées dans les protocoles d'authentification et de chiffrement (AH et/ou ESP) sont prédéfinies et entrées manuellement dans la configuration des deux extrémités du tunnel, le routeur Cisco et la passerelle.

Il faut d'abord définir une transformation qui explicite les protocoles utilisés (AH et/ou ESP) et les algorithmes nécessaires pour la mise en place du tunnel IPSEC. Le nom de la transformation est suivi de la commande `crypto ipsec transform-set`. Ici, il s'agit de la transformation `chiff_auth` qui fait du chiffrement et de l'authentification avec le protocole ESP à l'aide de l'algorithme de chiffrement DES et de l'algorithme d'authentification MD5.

En configuration manuelle, le routeur Cisco ne permet pas d'utiliser le protocole 3DES. Il faut donc obliger l'autre extrémité du tunnel à fonctionner en DES.

De plus, c'est ici que le mode d'utilisation d'IPSEC est défini. Par défaut dans IOS, il s'agit du mode tunnel.

```
RoutCisco#config t
RoutCisco(config)# crypto ipsec transform-set chiff_auth esp-des esp-md5-hmac
RoutCisco(cfg-crypto-trans)#mode tunnel
```

On configure ensuite une crypto map qui lie les SA (*Security Association*) et la SP (*Security Policy*). Ici, on utilise `ipsec` en mode de gestion de clé manuelle (`ipsec-manual`). La crypto map porte un nom (`cisco_pass`) et un numéro de séquence (`10`) qui la définissent de manière unique. La crypto map permet de définir l'extrémité du tunnel IPSEC (`set peer`), le protocole utilisé (`ESP`), deux SPI (*Security Parameter Index*) suivant le sens du trafic (`inbound` et `outbound`) qui doivent être les mêmes que sur la passerelle (on peut cependant sur un Cisco utiliser le même SPI pour les deux sens), la clé utilisée pour le chiffrement dans le protocole ESP (`cipher`), la clé utilisée pour l'authentification md5 dans le protocole ESP (`authenticator`), la transformation définie plus haut dans laquelle on trouve les algorithmes choisis et le mode d'IPSEC (`set transform-set`) et enfin le trafic à sécuriser (`match address`) qui correspond à une access-list définie plus loin.

```
RoutCisco(config)#crypto map cisco_pass 10 ipsec-manual
RoutCisco(config-crypto-map)#set peer 192.168.1.254
RoutCisco(config-crypto-map)#set session-key inbound esp 300 cipher
74697469746F746F authenticator 31323333435363738393031616263646566
RoutCisco(config-crypto-map)#set session-key outbound esp 301 cipher
74697469746F746F authenticator 31323333435363738393031616263646566
RoutCisco(config-crypto-map)#set transform-set chiff_auth
RoutCisco(config-crypto-map)#match address 110
```

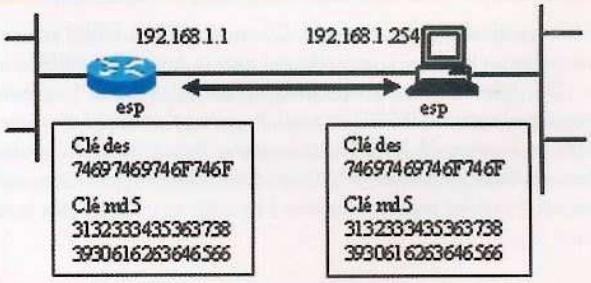
On configure alors l'access-list qui définit le trafic à sécuriser. Il s'agit de tout le trafic IP entre le routeur Cisco 192.168.1.1 et la passerelle 192.168.1.254. Ici, on a choisi le trafic entre deux entités (*host*), mais il est également possible de sécuriser toutes les communications entre deux sous-réseaux.

```
RoutCisco(config)# access-list 110 permit ip host 192.168.1.1 host 192.168.1.254
```

Il ne reste plus qu'à lier cette crypto map ainsi définie à une interface du routeur Cisco. Ici, il s'agit de l'interface FastEthernet0/0. Ainsi, tout le trafic arrivant ou sortant de cette interface est comparé avec le trafic défini dans l'access-list de la crypto map : s'il y a correspondance, le trafic est chiffré.

```
RoutCisco(config)#interface FastEthernet 0/0
RoutCisco(config-if)# ip address 192.168.1.1 255.255.255.0
RoutCisco(config-if)# crypto map cisco_pass
```

Figure 2



Comme les configurations par défaut dans les Cisco ne sont pas affichées en faisant la commande `#show config`, il est bon de vérifier les configurations entières avec `#show crypto ipsec transform-set`, `#show crypto map`.

```
RoutCisco#sh crypto ipsec transform-set
Transform set chiff: { esp-des esp-md5-hmac }
   will negotiate = { Tunnel, },


RoutCisco#sh crypto map
Crypto Map "cisco_pass" 10 ipsec-manual
        Peer = 192.168.1.254
        Extended IP access list 110
            access-list 110 permit ip host 192.168.1.1 host 192.168.1.254
        Current peer: 192.168.1.254
        Transform sets={
                chiff_auth,
        }
        Inbound esp spi: 300,
         cipher key: 74697469746F746F,
         auth_key: 31323334353637383930616263646566,
        Inbound ah spi: 0,
            key: ,
        Outbound esp spi: 301
         cipher key: 74697469746F746F,
```

```
      auth key: 313233343536373839308616263646566 ,
      Outbound ah spi: 0,
         key: ,
      Interfaces using crypto map cisco_pass:
            FastEthernet0/0
```

A ce stade de la configuration, le tunnel IPSEC est défini. Les données échangées entre les deux hôtes 192.168.1.1 et 192.168.1.254 seront encapsulées dans ce tunnel. On peut donc lancer un `ping` entre les deux machines et visualiser certaines informations de l'échange, comme le nombre de paquets encapsulés et décapsulés à une interface, les SPI, etc.

```
RoutCisco#sh crypto ipsec sa

interface: FastEthernet0/0
    Crypto map tag: cisco_pass, local addr. 192.168.1.1

   local  ident (addr/mask/prot/port): (192.168.1.1/255.255.255.255/0/0)
   remote ident (addr/mask/prot/port): (192.168.1.254/255.255.255.255/0/0)
   current_peer: 192.168.1.254
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 10, #pkts encrypt: 10, #pkts digest 10
    #pkts decaps: 10, #pkts decrypt: 10, #pkts verify 10
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
    #send errors 5, #recv errors 0

     local crypto endpt.: 192.168.1.1, remote crypto endpt.: 192.168.1.254
     path mtu 1500, media mtu 1500
     current outbound spi: 12D

     inbound esp sas:
      spi: 0x12C(300)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2001, flow_id: 1, crypto map: cisco_pass
              no sa timing
        IV size: 8 bytes
        replay detection support: Y

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x12D(301)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2000, flow_id: 2, crypto map: cisco_pass
        no sa timing
        IV size: 8 bytes
        replay detection support: Y

     outbound ah sas:

     outbound pcp sas:
```

Les « send errors » sont les premiers `ping` qui n'ont pas eu le temps de passer dans le tunnel car celui-ci était en train de se mettre en place.

Pour effacer ce tunnel, on peut lancer la commande `#clear crypto sa` :

```
01:10:06: IPSEC(delete_sa): deleting SA,
  (sa) sa_dest= 192.168.1.254, sa_prot= 50,
    sa_spi= 0x12D(301),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2000
01:10:06: IPSEC(delete_sa): deleting SA,
  (sa) sa_dest= 192.168.1.1, sa_prot= 50,
    sa_spi= 0x12C(300),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2001
```

Cette méthode de configuration manuelle des clés a deux inconvénients majeurs : il faut trouver des clés suffisamment sûres ; et une SA IPSec établie manuellement n'expire pas, c'est-à-dire que la SA ainsi que les clés ne sont pas renouvelées (ce qui est logique puisque les SPI et les clés sont entrées « en dur » dans la configuration). Pour contrer ces inconvénients, on peut utiliser le protocole ISAKMP de gestion des clés.

2-Utilisation de l'IOS de Cisco avec le protocole ISAKMP

Dans la configuration précédente, les clés des algorithmes de chiffrement et d'authentification étaient entrées « à la main ». Ici, les clés ne se trouvent pas directement dans le fichier de configuration du routeur. Elles sont dérivées d'une clé de session partagée par les deux hôtes, mais non échangée sur le réseau. Cette clé de session est issue du protocole de Diffie-Hellman et d'une authentification mutuelle initiale des deux hôtes.

Soit cette authentification est faite par un secret pré-partagé, soit elle est effectuée par un échange de certificats. Dans ce premier cas, nous n'exposerons que le secret pré-partagé.

Dans les deux cas, la gestion des clés est faite par le protocole ISAKMP (*Internet Security Association and Key Management Protocol*) ou IKE (*Internet Key Exchange*) phase1.

Utilisation de l'IOS de Cisco avec un secret pré-partagé

Ici, l'authentification mutuelle des hôtes (phase 1) se fait par un secret partagé initial et entré « en dur » dans la configuration de chaque hôte.

On configure donc les paramètres définissant la politique de sécurité de la phase1 reposant sur le protocole ISAKMP.

On précise d'abord la clé pré-partagée entre les deux entités, puisqu'il s'agit d'une authentification *pre-share*. Ceci est fait à l'aide de la commande `crypto isakmp key`.

```
RoutCisco(config)#crypto isakmp key mekmitasdigoat address 192.168.1.254
```

Puis on définit les algorithmes de chiffrement (ex. : `encr 3DES`) et d'authentification (ex : `hash md5`), la méthode d'authentification (ici `pre-share`), le groupe Diffie-Hellman (`group 1` ou `group 2`, suivant la taille choisie de la clé de l'algorithme de Diffie-Hellman pour établir une clé de session), la durée de la SA de phase1 (par défaut 86000 secondes). Lorsque la durée de vie de la SA est atteinte, la SA expire et est renégociée.

```
RoutCisco(config)#crypto isakmp policy 10
RoutCisco(config-isakmp)#encr 3des
RoutCisco(config-isakmp)#hash md5
RoutCisco(config-isakmp)#authentication pre-share
RoutCisco(config-isakmp)#group 2
```

A l'issue de cette négociation, on a un tunnel sécurisé (phase1 du protocole IKE). Désormais, la politique de sécurité de phase2 sera négociée à travers ce tunnel ISAKMP pour établir un tunnel final chiffré IPSEC dans lequel circuleront les paquets de données. On définit alors, comme dans le paragraphe précédent, une *transform-set* (algorithmes de chiffrement et d'authentification, mode d'IPSEC), une crypto map (extrémité du tunnel, transformation associée et trafic à sécuriser) et une access-list (trafic à sécuriser). La différence avec la première configuration est qu'au niveau de la crypto map,

les clés des algorithmes ne figurent pas en clair dans le fichier de configuration puisqu'elles sont négociées par ISAKMP (ipsec-isakmp) à l'aide de l'authentification mutuelle (pre-share) et du protocole Diffie-Hellman.

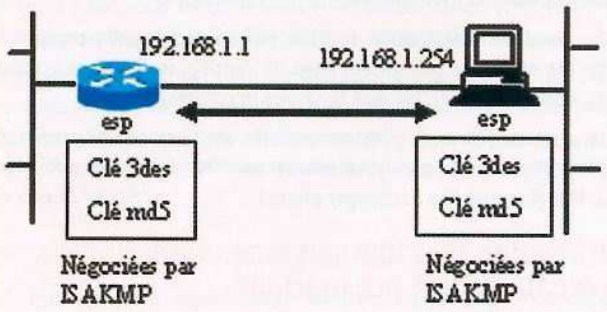
```
RoutCisco(config)#crypto ipsec transform-set chiff_auth esp-3des esp-md5-hmac

RoutCisco(config)#crypto map cisco_pass 10 ipsec-isakmp
RoutCisco(config-crypto-map)#set peer 192.168.1.254
RoutCisco(config-crypto-map)#set transform-set chiff_auth
RoutCisco(config-crypto-map)#match address 110

RoutCisco(config)#access-list 110 permit ip host 192.168.1.1 host 192.168.1.254

RoutCisco(config)#interface FastEthernet0/0
RoutCisco(config-if)# ip address 192.168.1.1 255.255.255.0
RoutCisco(config-if)#crypto map cisco_pass
```

Figure 3



↘ Remarques :

→ On peut définir plusieurs politiques de sécurité ISAKMP grâce à des numéros de priorité différents. La négociation conclura sur une politique identique des deux côtés.

→ On ne peut définir qu'une seule crypto map par interface. Si on souhaite établir deux tunnels IPSEC différents (extrémité différente, trafic différent, donc crypto map différente) à une seule interface donnée d'un routeur Cisco, il est possible de définir deux crypto map de même nom avec deux numéros de séquence différents. Le numéro de séquence le plus faible correspond à la priorité la plus élevée. Ainsi, le trafic arrivant à l'interface est comparé en premier au trafic associé à la crypto map de priorité la plus élevée.

→ Il est également possible de paramétrer la durée de vie du tunnel IPSEC en temps (exprimé en secondes) et en volume de données (exprimé en kilo-octets) par la commande `crypto ipsec security-association lifetime`. Par défaut, ces valeurs sur un routeur Cisco sont de 3600s et de 4608000 Ko. Lorsque ces paramètres sont dépassés, les SA IPSEC expirent et sont donc renégociées. Lorsqu'une SA expire, une autre est renégociée sans que le flux de données soit interrompu.

Pour vérifier les configurations, il est bon de lancer, comme dans la première partie, les commandes pour visualiser la configuration d'IPSEC (`#show crypto ipsec transform-set`, `#show crypto ipsec security-association lifetime`, `#show crypto map`) mais aussi la configuration d'ISAKMP (`#show crypto isakmp policy`) :

```
RoutCisco#sh crypto isakmp policy
Protection suite of priority 10
        encryption algorithm:   Three key triple DES
        hash algorithm:         Message Digest 5
        authentication method:  Pre-Shared Key
        Diffie-Hellman group:   #2 (1024 bit)
        lifetime:               86400 seconds, no volume limit
Default protection suite
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Secure Hash Standard
        authentication method:  Rivest-Shamir-Adleman Signature
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               86400 seconds, no volume limit
```

Lançons un `ping`. Il est possible de voir comme précédemment les SA IPSEC (`#show crypto ipsec sa`) et la SA ISAKMP ainsi crée (`#show crypto isakmp sa`) :

```
RoutCisco#sh crypto isakmp sa
dst             src             state          conn-id      slot
192.168.1.254   192.168.1.1     QM_IDLE           2          0
```

Conclusion

L'objectif de cette fiche pratique n'est pas de décrire de façon très détaillée la mise en place d'IPSec avec toutes ces configurations possibles, mais juste de donner quelques premières bases techniques pour utiliser IPSec sur un routeur Cisco. Cet article montre qu'il est possible d'utiliser IPSEC entre un routeur Cisco et une machine OpenBSD, en mode manuel de gestion de clés ou en mode automatique avec un secret partagé.

Il existe d'autres paramétrages d'IPSEC, comme par exemple l'utilisation de certificats X509. Ceci fera l'objet d'une prochaine fiche technique qui envisagera une liaison IPSEC entre deux réseaux locaux via deux routeurs Cisco.

Merci à Mathieu BLANC (**moutane@rstack.org**)
et à Laurent OUDOT (**oudot@rstack.org**)
ainsi qu'à mes collègues du laboratoire
Conception Développement et Système.

Références

[1] Dossier *IPSec Sécurisation de communications WIFI avec IPSec*, MISC 10

[2] **http://www.cisco.com/**

[3] **http://www.cisco.com/univercd/home/home.htm**

[4] **http://www.cisco.com/univercd/cc/td/doc/pcat/index.htm**

[5] **http://www.cisco.com/warp/public/707/nle_toc.html**

[6] Fiche pratique *IPSec sur OPenBSD, FreeBSD et Mac OS X*, dans MISC 10

[7] Fiche pratique *IPSec et Linux*, dans MISC 10