misc
Multi-System & Internet Security Cookbook

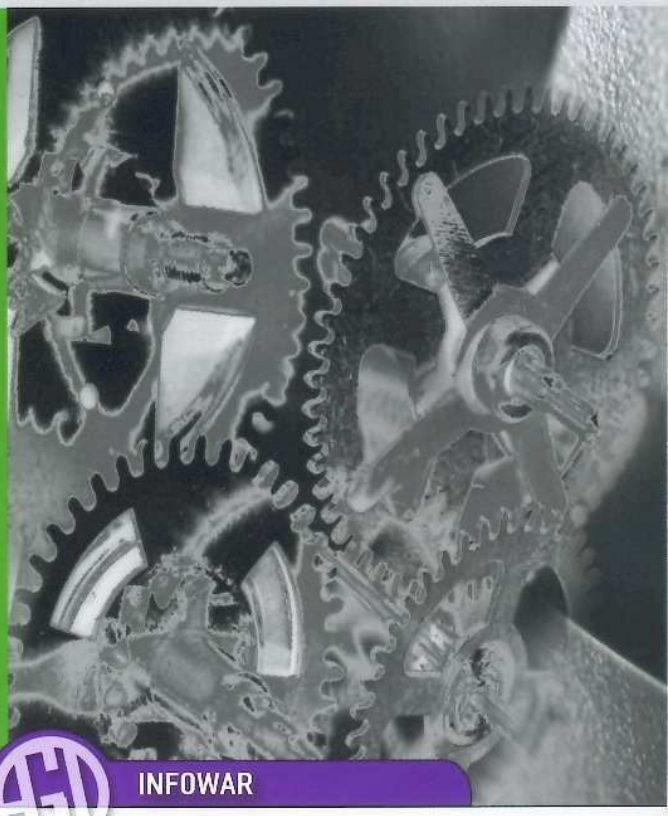
France Métro : 7.45 Eur - CH : 12.5 CHF
BEL. LUX. PORTCONT : 8.5 Eur - CAN : 13 $C
MAR : 75 DH

23 Janvier Février 2006

100 % SÉCURITÉ INFORMATIQUE

De la recherche de failles à l'exploit

Chercher, exploiter, prévenir les failles logicielles



- Recherche de vulnérabilités grâce à l'analyse différentielle de binaires
- Recherche de vulnérabilités à l'aide du fuzzing
- Fiabilisation d'exploits Windows
- Protection de l'espace d'adressage d'un processus

INFOWAR

La vision chinoise de la Guerre de l'Information

FICHE TECHNIQUE

Déployer une architecture DNS sécurisée

VULNÉRABILITÉ

Rooté par un iPod !

Édito

Âge de glace, âge de raison, âge de zoo (pour continuer les histoires d'eau de l'épisode précédent)

Bonne année, meilleurs vœux, blablabla... Bon, ça y est, ça, c'est réglé. Je préfère le mettre tout de suite pour être certain de ne pas l'oublier. Comme quelques mécréant(e)s irrespectueux(ses) me le rappellent souvent, avec l'âge, ma mémoire devient de moins en moins fiable. Les connaisseurs noteront l'habile transition entre mon âge – loin d'être encore vénérable, quoique – et un anniversaire : celui de la revue que vous tenez précisément entre vos mains à cet instant. Je vois bien au fond d'Internet les petits rigolos qui se demandent de quoi il parle puisqu'ils sont en pleine lecture sur notre site. Mais j'ai confiance, vous irez vous procurer la version papier aux couleurs si vives et au toucher si délicat.

Mais je m'éloigne du sujet : MISC a 4 ans ! Quand toute l'équipe s'est lancée dans l'aventure, nous n'avions pas vraiment de doute quant à son succès : nous étions jeunes (et le sommes encore ;), pleins d'énergie et avec une énorme envie de faire bouger plein de choses ici chez nous. Nous pensions naïvement que cela suffirait à faire vivre la revue. Malheureusement, les premiers numéros ont été bien plus délicats que ce à quoi nous nous attendions. « À vaincre sans péril, on triomphe sans gloire », comme dit Fidel Castro à chaque élection (à moins que ça ne soit un chanteur, son nom m'échappe... Corneille ;-). Bref, grâce à une mobilisation générale de tous, Diamond, les auteurs, et vous public chéri mon amour (ça faisait trop longtemps que je ne l'avais pas utilisé), la revue continue à vivre et se développe : **MERCI !!!** Enfin, je ne sais pas si je dois réellement remercier quiconque à cause des soirées et week-ends que je passe à corriger les articles, ou des copines des auteurs qui me maudissent, parfois sans même me connaître `<mode="auto-fleurs">`alors que je suis tellement charmant`</mode>`, ou encore les moult bières que je dois aux auteurs pour les motiver... En fait, MISC, c'est un véritable sacerdoce à ronger.

Mais puisque je radote sur l'âge, je voudrais revenir sur une « expérience » que j'ai vécue, il y a peu. J'ai pu participer à quelques réunions de travail (enfin, c'est le nom officiel parce que ma grand-mère est plus efficace quand elle fait ses mots croisés) sur la sécurité des systèmes d'information dans notre présipauté. Et j'ai été frappé d'une révélation quant à l'organisation de ce microcosme, une fois passée la surprise d'avoir appris qu'un anti-virus et un pare-feu, c'est pareil (oui, je sais, ça fait un choc). En bas de l'échelle de l'évolution, à peine accrochés au dernier barreau, il y a les « opérationnels ». Il s'agit souvent de jeunes, aussi appelés « pirates » ou « hackers », tout juste diplômés d'une école d'ingénieurs (ou pas, mais dans ce cas, ils ne sont pas accrochés au dernier barreau, ils restent au pied de l'échelle). Parfois, ils sont remplis d'assurance, certains de leurs compétences techniques et scientifiques. Cela peut les faire paraître arrogants aux yeux de ceux qui ne comprennent pas la grâce et la beauté d'un *remote control* via un *heap overflow* sur une machine dont la mémoire est « randomisée ». Ils mangent des pizzas et boivent du coca, sont mal rasés, voire ont les cheveux longs (et forcément les idées courtes qui vont avec).

Au milieu de l'échelle, les architectes et autres chefs de projets, conscients des enjeux tactiques. Ils ont de vagues restes techniques et scientifiques, étant donné qu'ils sont noyés par des réunions tout à fait passionnantes et utiles, qu'ils passent à boire du café en remplissant des sodokus en attendant que ça se termine. En revanche, ils ont une bonne vision des risques réels car ils sont capables de comprendre les jeunes. D'ailleurs, les jeunes les inquiètent car, bien souvent, ils préfèrent ignorer les problèmes pour ne pas avoir à les traiter, alors que ces derniers viennent les leur rapporter dans toute leur naïveté... Au sommet de l'échelle, le vieux singe, fin stratège, ne comprend rien à ce que racontent les autres : il croit être définitivement à l'abri derrière son pare-feu, ou alors s'il a un passé légèrement technico-scientifique, qu'il faut absolument qu'il y ait un `/bin/sh` pour exécuter du code arbitraire sur une machine compromise. Bien souvent, il a été mis à ce poste parce qu'on ne savait pas où le mettre ailleurs et sa vision du problème est uniquement économique. Il aime participer activement à des cocktails en buvant du champagne avant d'acheter le dernier IPS à la mode. Ensuite, il pourra le donner à ses équipes, ils sauront bien quoi en faire... alors qu'eux réclament juste un nouveau commutateur pour segmenter le réseau. Et pendant ce temps-là, le vieux singe continue à se regarder le nombril en s'auto-félicitant.

Certains sont enfermés dans des « doctrines » vieilles de 5 ans alors que la SSI n'a bien sûr pas évolué pendant tout ce temps. D'autres continuent de penser que la technique ou l'argent sont les seules réponses à tous les problèmes, et restent alors accrochés à leur barreau sans prendre le reste en considération. On s'est récemment rendu compte de la nécessité d'avoir, dans certains milieux en tout cas, des personnes dédiées à la sécurité des systèmes d'information. Malheureusement, les formations n'existent que depuis peu, alors on a propulsé qui on pouvait sur la scène.

Une des difficultés de la SSI vient de ce que c'est un problème multi-échelle (voire fractal), et qu'on doit jouer à tous les niveaux pour y parvenir : c'est un problème global qui demande des solutions locales. Personne ne peut le résoudre seul dans son coin.

C'est d'ailleurs bien là le cœur de ce qui motive la troupe des gentils organisateurs de SSTIC. Alors, si vous ne savez pas quoi faire les 31 mai et 1-2 juin, bloquez tout de suite ces dates dans votre agenda pour assister à ce Woodstock de la SSI : thés, conférences, festival off, etc. Pour ceux qui ne connaissent pas, ne prenez pas peur. Il s'agit simplement de mettre dans un grand amphi quelques passionnés, et de les laisser mariner pendant 3 jours. Comme peuvent en témoigner les actes des éditions passées (**http://actes.sstic.org**), le résultat est probant. Mais l'intérêt, pour moi, est que c'est la seule tribune qui regroupe en France tous les acteurs de la SSI : scientifiques, industriels, étatiques,... et tout cela dans une excellente ambiance, ce qui ne gâche rien, au contraire.

Il me reste à terminer sur mes bonnes résolutions pour l'année à venir. Parfois, mais ça dure moins d'une seconde en général, j'ai honte de mes vannes, surtout quand elles coulent de source comme ça. Alors, je vais peut-être arrêter d'en faire, mais ce serait comme arrêter de boire de l'eau : « un seul être vous manque, et tout est dépeuplé ». Non, ça n'ira pas, je serais dans le lac, comme dit Lamartine dans son *Isolement*. Tant pis, je continuerai, en espérant qu'elles seront « moins pires » avec l'âge.

Fred Raynal

Sommaire

GUERRE DE L'INFO 04 → 08
>Guerre de l'information en Chine

VULNÉRABILITÉ 10 → 14
>Un accès sans limite à la mémoire grâce au FireWire

DROIT 16 → 23
>Vulnérabilités logicielles et droit : responsabilité, recherche et divulgation

DOSSIER 24 → 61
De la faille à l'exploit
>Analyse différentielle de binaires : Application à la recherche de vulnérabilités / 24 → 33
>Recherche de vulnérabilités à l'aide du fuzzing / 34 → 41
>Fiabilisation d'exploits Windows / 42 → 47
>Protection de l'espace d'adressage : État de l'art sous Linux et OpenBSD / 48 → 61

SYSTÈME 62 → 67
>Évaluation de la sécurité des terminaux mobiles sous Windows CE

SCIENCE 68 → 72
>Comment Leurré.com observe l'Internet

FICHE TECHNIQUE 74 → 80
>De la sécurité d'une architecture DNS d'entreprise

> Abonnements et Commande des anciens Nos / 9/15/81

Guerre de l'information en Chine

Le concept de « Guerre de l'Information » (GI) s'est imposé en Chine dès le début des années 1990, essentiellement auprès des stratèges militaires qui en ont d'abord fait une analyse théorique, ont développé le concept et en assurent aujourd'hui la mise en œuvre dans le cadre de la modernisation des armées. L'approche du concept de GI peut ressembler aux théories occidentales [1], mais elle a su au cours de ces dernières années acquérir une identité propre, aux traits spécifiquement chinois.

I – Théories de la guerre de l'information en Chine

1.1 – Des théories nées de l'observation

Au milieu des années 1980, la Chine a commencé à développer une nouvelle génération de forces de combat. En 1991, avec la Guerre du Golfe, les stratèges de l'armée chinoise prennent toute la mesure de l'écart qui sépare la Chine du potentiel américain. L'armée iraquienne, en bien des points semblable à celle de la Chine à la même époque, a été défaite en 42 jours, succombant sous la puissance de feu américaine (rapidité de l'intervention, équipement high-tech). Toute victoire de la Chine apparaissait fortement improbable dans l'éventualité d'un conflit armé contre les États-Unis. Une modernisation rapide de l'armée et des systèmes de défense s'imposait. La guerre du Golfe permit également de prendre conscience de la nécessité d'obtenir la maîtrise de l'information, des systèmes d'information et de communication, pour agir plus vite et plus efficacement que l'adversaire. Désormais les guerres ne seraient plus comme avant. Le conflit du Kosovo (Opérations Forces Alliées, 1999) qui opposait de nouveau des pays de puissance très inégale, alimenta les réflexions des stratèges chinois, confirmant les conclusions tirées de la Guerre du Golfe. Dans ce conflit, la résistance serbe impressionna les Chinois. Le concept de guerre asymétrique [2], rendant possible la victoire sur un adversaire technologiquement supérieur, s'imposait. La Chine s'est alors intéressée de manière spécifique au concept de GI, à son rôle dans la stratégie et dans le développement de solutions de guerre asymétrique.

1.2 – Définitions de la Guerre de l'information en Chine

Dans la première moitié des années 1990, selon les stratèges chinois (dont les figures centrales sont le général Wang Pufeng, considéré comme le père du concept de GI en Chine, le colonel Wang Baocun, le général Yuan Banggen et Shen Weiguang), la GI a pour objectif le « contrôle » des flux d'information. Cette approche est alors assez différente de celle des États-Unis qui privilégie l'attaque et la destruction.

Liang Zhenxing, autre stratège, propose en 1997 une définition plus proche des critères internationaux : la GI est l'ensemble des activités d'affrontement impliquant l'exploitation, l'altération, la paralysie de l'information et des systèmes d'information de l'ennemi, ainsi que toute activité permettant de protéger sa propre information et ses propres systèmes d'information.

Les définitions proposées les années suivantes affinent le concept de GI, en déterminant les caractéristiques par temps de paix, de crise, de guerre, en situation offensive et défensive, stratégique [3], tactique. Le concept chinois de GI inclut la déception [4] militaire, la guerre psychologique, la guerre électronique, la destruction physique des infrastructures de Commande et Contrôle (C2), le renseignement, la guerre des réseaux informatisés. Il s'attache également à l'étude des processus cognitifs.

L'objectif de la GI est de contraindre l'ennemi à se rendre sans avoir combattu, grâce à la supériorité issue de la maîtrise de l'information.

En 2000, Wang Baocun définit la GI comme « une forme d'actions de combat qui s'attaque à l'information et aux systèmes d'information de l'ennemi tout en protégeant l'information et les systèmes d'information (SI) de son propre pays (...). Son but essentiel est de gagner et maintenir une domination de l'information » [5].

La GI apparaît comme une arme stratégique complémentaire, non conventionnelle, devant être utilisée dès le début d'un conflit, distincte de la guerre informatisée [6] qui elle sera peut-être dans l'avenir une nouvelle forme de guerre à part entière [7].

[1] On retrouvera des points communs avec les sept sous-catégories définies par l'Américain Martin Libicki (guerre de commandement et de contrôle, guerre du renseignement, guerre électronique, guerre psychologique, guerre des « hackers », guerre de l'information économique, cyber-guerre), ainsi qu'avec l'approche française (guerre par/contre/pour l'information, maîtrise de l'information).

[2] L'objectif stratégique de la guerre symétrique est la victoire par destruction de l'adversaire. Celui de la guerre asymétrique, l'effondrement d'un système. Voir : http://www.webzinemaker.com/admi/exec/print.php3?ident=geopolitique&rubr=4&id=177406

[3] La stratégie consiste à définir des buts fondamentaux à long terme et choisir les modes d'actions et ressources nécessaires à la réalisation de ces objectifs. La tactique est l'application de la stratégie, ce sont les moyens dont on se sert pour parvenir à un résultat, à court terme.

[4] Terminologie militaire désignant la tromperie, la falsification, visant à induire l'adversaire en erreur.

[5] « The Current Revolution in Military Affairs and its impact on Asia-Pacific Security », China Military Science, April 2000.

[6] « Informationized War » en anglais

[7] Wang Baocun. 1999.

Daniel Ventre
CNRS
daniel.ventre@gern-cnrs.com

1.3 – Une théorie « chinoise » de la Guerre de l'information

Les stratèges ont développé un concept de GI qui a au cours du temps acquis ses propres spécificités chinoises, en accord avec la culture, la pensée, la politique, les dangers perçus par le pays.

Selon le général Wang Pufeng (1995), les capacités de l'ère de l'information pourraient prolonger la théorie de « **Guerre du Peuple** » chère à Mao Zedong. Ce concept, inconnu en Occident, fait de chaque citoyen un combattant. Grâce au concept de guerre du peuple, la GI est forte d'une armée de plusieurs millions d'individus.

Le simple civil, depuis chez lui, à partir d'un simple ordinateur connecté aux réseaux, peut ainsi servir les intérêts de la nation en attaquant des cibles ennemies (*hacking*, piratage), civiles ou militaires. Les experts en informatique, réseaux, électronique, télécommunications, peuvent devenir les nouveaux héros de cette nouvelle forme de lutte.

Cet aspect contribue à la révolution du monde militaire chinois et à l'évolution « chinoise » du concept de GI. Le champ de bataille traditionnel n'existe plus, la guerre peut être partout, elle est donc l'affaire de tous. La fusion des deux concepts est facilitée par le développement rapide de l'Internet en Chine.

Les théories modernes de GI sont fortement influencées par **l'art militaire** chinois et ses traditions ancestrales. Depuis le début des années 1980, on assiste en Chine à un renouveau de la culture stratégique, renouant avec la tradition des écrits militaires classiques.

Grâce au concept de GI, les « 36 stratagèmes » [8] trouvent un sens et des applications nouvelles. Il en est de même des théories développées dans *L'Art de la Guerre* de Sun Tzu [9], le plus ancien traité militaire du monde. Les attaques anonymes via les réseaux, la vulnérabilité des systèmes face aux virus, redonnent un visage modernisé aux stratagèmes.

Nombre de stratagèmes (comment tromper la vigilance d'un ennemi, comment vaincre un ennemi plus fort, comment choisir son champs de bataille et le moment, comment affaiblir et déstabiliser un ennemi...) et de principes posés par Sun Tzu (toute guerre est basée sur la déception, quand on est prêt à attaquer il faut sembler ne pas l'être...) sont transposables à l'univers des TIC (dissimuler des virus dans des messages réguliers, lancer des attaques multiples et coordonnées et profiter du désordre pour lancer une attaque fatale, etc.).

La Chine est convaincue qu'une **stratégie** [10] peut venir à bout d'ennemis technologiquement supérieurs. Elle est d'ailleurs coutumière des guerres asymétriques ayant déjà affronté et vaincu par la stratégie, au cours des siècles passés, des ennemis qui lui étaient militairement et technologiquement supérieurs (lutte contre l'envahisseur japonais, lutte contre le Kuomintang...). Dans un contexte de GI, par «stratégies» il faut entendre : sabotage des systèmes d'information ennemis, affaiblissement des capacités de frappe informationnelles de l'ennemi, dispersion des forces ennemies, propagation de fausses informations, lancement d'attaques surprises contre les systèmes d'information.

Un autre changement radical dans l'approche chinoise des méthodes de guerre est dans la prédominance accordée à la **connaissance**. La manière de penser devient plus importante que la manière de faire. La confrontation de commandements est un nouveau mode de guerre qui implique la maîtrise de la fourniture de l'information, son contrôle, et son utilisation (traitement). La connaissance devient la ressource stratégique majeure, plus importante encore que les armes.

Pour Sun Tzu, la plus grande des **victoires** est celle que l'on remporte **sans combattre**. Ce but ambitieux implique une stratégie longuement mûrie et que les commandements soient en mesure de mener ce que les Chinois appellent une « **guerre d'acupuncture** », c'est-à-dire d'attaquer le cœur même de la défense ennemie, que sont aujourd'hui les points névralgiques des réseaux.

La GI est un outil clef de la guerre moderne « **sans contact** ». Nombre de responsables de l'armée chinoise ont vu en la guerre du Kosovo un modèle de guerre sans contact dans laquelle le contrôle de l'espace aérien et des systèmes d'information furent des facteurs décisifs. Les guerres à venir devraient être le théâtre d'affrontements terrestres de plus en plus courts, voire non existants, où l'information jouera un rôle essentiel.

Le concept de **défense active** est le modèle de défense prôné par la Chine. Elle affirme officiellement le caractère strictement défensif de ses politiques, ne se définissant pas comme un pays agresseur militairement : elle ne déclare pas une guerre ou des agressions, mais s'engage dans une guerre uniquement pour défendre la souveraineté nationale et l'intégrité territoriale [11], et n'attaque qu'après avoir été elle-même attaquée. Elle ne prend pas l'initiative, elle agit en réaction. Les stratèges chinois doivent s'efforcer de définir toutes les guerres menées par l'armée comme des guerres « défensives ».

[8] « L'Art Secret de la Guerre : les 36 stratagèmes » : compilation de proverbes chinois anciens, applicables à la politique, la diplomatie et l'espionnage.

[9] Comment établir un plan de bataille, attaque par stratagème, dispositions tactiques, points faibles et forts, tactique, espionnage... http://www.au.af.mil/au/awc/awcgate/artofwar.htm

[10] Général Dai Qingmin (ex-commandant du Centre de GI de l'armée à Wuhan), « Innovating and Developing Views on Information Operations », China Military Science, Août 2000.

[11] La violation du cyber-espace est pour la Chine aussi importante qu'une violation de souveraineté nationale.

En matière de défense, le Général Dai Qingmin recommande le modèle serbe (guerre du Kosovo), modèle de défense active (défense positive) ayant permis de tenir en échec les Américains, et non celui des Iraquiens, passif (défense négative). La défense serbe consista à mettre en œuvre diverses mesures telles que la dissimulation des avions et blindés, la dispersion des troupes dans la population, des moyens pour échapper aux outils de reconnaissance, leurres pour attirer les missiles, utilisation de l'internet pour diffuser de l'information sur les attaques de l'OTAN, tentatives d'attaques de type DoS contre les sites de l'OTAN [12].

II – De la théorie à la pratique

La Chine est en train de développer ses infrastructures militaires et civiles afin d'appliquer ses théories. La mise en œuvre de la GI est inscrite dans le cadre de la modernisation de l'armée (RMA) [13].

2.1 – La formation des militaires s'avère indispensable à la mise en pratique de la GI qui reste encore un concept relativement méconnu. De nombreux centres de formation des armées (Zhengzhou, Wuhan, Changsha...) dispensent désormais aux militaires des enseignements en GI, car mener une telle guerre nécessite de la part de ceux qui la dirigent de réelles capacités de commandement ainsi que des qualités psychologiques spécifiques.

2.2 – Sous la responsabilité de l'armée, des unités spéciales de réserve [14] chargées de mission de GI sont créées dans diverses provinces (Hubei, Fujian, Shanxi...). Ces unités de réserve, actives dans plusieurs villes, se spécialisent, développent des « sites d'excellence » [15] : télécommunications, réseaux, guerre électronique, intelligence, guerre psychologique... Les forces de réserve de la ville de Shanghai, par exemple, se spécialisent dans les réseaux de télécommunications sans fil et la cryptographie. Ces expertises « locales » peuvent être étendues, coordonnées, pour former un corps de « combattants des réseaux » capables de défendre la Chine, ses réseaux de télécommunications, de commandement et d'information et d'atteindre les vulnérabilités des systèmes d'information étrangers.

2.3 – Des stratèges appellent à la création d'une « Net Force » indépendante (à côté de l'Air Force, de la marine, etc.) capable de mener les nouvelles batailles de hautes technologies du futur. Le rôle de cette branche de l'armée serait de protéger la souveraineté (le cyberespace) et de s'engager dans des conflits. Elle utiliserait des technologies capables de lancer des attaques sur le net, utilisant des logiciels pour paralyser l'information, la bloquer, la falsifier, serait en mesure de prendre le commandement d'opérations ennemies par le jeu de la falsification d'identités, technologies d'attaques couplées à des capacités défensives pour déjouer les attaques ennemies.

2.4 – Depuis 1997, la presse fait état de nombreux exercices de GI [16] menés par les forces armées, preuve du passage progressif de la théorie à la pratique. Le premier exercice de bataille de type GI pour l'armée chinoise a eu lieu en octobre 1997 à Shenyang. Il s'agissait de contrer une attaque informatique paralysant ses systèmes.

En 1998, un exercice de plus grande envergure était réalisé, mettant en action plusieurs régions militaires du pays et coordonné par Pékin. Depuis lors, les exercices militaires de GI se poursuivent et se sont successivement complexifiés [17]: mesures défensives à l'affrontement de commandements et possibilité de lancer des attaques contre les C4I (Commandement, Contrôle, Communications, Computers, Intelligence) ennemis tout en assurant la défense des SI du pays (CNA [18] et CND [19]). La Chine a inclus les techniques d'attaque des pirates, attaques par virus et blocage de l'information, dans ses exercices militaires.

Pour réaliser ces exercices, l'armée collabore avec des instituts de recherche au développement de systèmes de simulation d'affrontement, d'applications défensives et offensives, en utilisant les réseaux, le multimédia, la réalité virtuelle. L'Académie militaire de Wuhan est apparue comme l'un des principaux centres de recherche en GI de Chine. En 1998, l'Académie créait le premier centre d'expérimentation et de simulation de GI du pays. Dans la province de Hubei, une base d'entraînement spéciale a été construite permettant à 500 personnes de s'entraîner simultanément. En avril 2004, le groupe Infoguerre de l'Armée, rattaché au commandement militaire de Pékin, a lancé une série d'exercices intitulés « Forces Rouges contre Forces Bleues », dont les résultats ont démontré que l'armée chinoise était en mesure de remporter un succès contre une armée occidentale.

2.1 – Des conflits potentiels comme champs d'application des théories

Théâtre des conflits annoncés : le Pacifique. La Chine, qui veut garder sous son sein Taiwan, pourrait intervenir militairement si cette dernière proclamait son indépendance. Dans ce cas, les États-Unis, et le Japon très probablement, s'interposeraient militairement. Pékin a promulgué le 14 mars 2005 une loi anti-sécession pour réaffirmer l'unité de la Chine et préparer l'éventualité de la proclamation de la sécession de Taiwan. La loi repose le principe de l'unité de la Chine (article 1) dont Taiwan n'est qu'une partie. Pékin n'en acceptera jamais l'indépendance. La question de Taiwan est une affaire intérieure qui n'autorise

[12] Mr. Tomithy L. Thomas, « Like adding Wings to the Tiger : Chine Information War Theory and Practice ». www.iwar.org.uk/iwar/resources/china/iw/chinaiw.htm

[13] La Révolution dans les Affaires Militaires.

[14] http://news.xinhuanet.com/mil/2003-06/12/content_916888.htm

[15] En anglais dans ce cas « pocket of excellence ».

[16] http://news.xinhuanet.com/mil/2003-06/12/content_916888.htm

[17] http://www.iwar.org.uk/pipermail/infocon/2003-July/000429.html

[18] Computer Network Attack

[19] Computer Network Defense

aucune ingérence de forces étrangères (article 2 et 3). L'article 6 énumère les moyens que Pékin veut consacrer à la réalisation et au maintien de l'unification : des moyens pacifiques (échanges culturels, économiques, scientifiques, etc.) ainsi que d'« autres moyens », sans précision. Des manœuvres relevant de la GI sont dans ce cadre totalement envisageables. Taiwan est sous tension craignant que la Chine ne tente une attaque informatique en lieu et place d'une invasion conventionnelle.

Dans son *Livre Blanc de la Défense* [20], la Chine évoque de possibles « conflits locaux sous contrainte de l'information » [21], ayant totalement conscience du rôle majeur et décisif qu'ont pris les technologies de l'information et de la connaissance dans un conflit moderne. La modernisation de l'armée se poursuit donc dans la perspective de conflits avec Taiwan, les États-Unis, le Japon. Selon un rapport du Département de la Défense américain [22] en date du 19 juillet 2005, la Chine se focalise sur les potentiels des technologies de l'information et œuvrerait activement au développement de ses capacités de GI : veille des avancées de la doctrine américaine, R&D pour mise en œuvre de la doctrine C4ISR (Commandement, Contrôle, Communications, Computers, Intelligence, Surveillance, Reconnaissance).

2.2 – Les hackers patriotes

Au cours des dernières années, des évènements perçus comme autant d'atteintes à l'intégrité et à la souveraineté nationale et l'identification d'adversaires majeurs ont déclenché des cyber-attaques massives, lancées sur l'initiative de *hackers* se faisant fort de défendre les intérêts de la Chine :

- Le bombardement de l'Ambassade de Chine par les forces de l'**OTAN** à Belgrade le 8 mai 1999 a entraîné une vague d'attaques (DoS, envois massifs d'e-mail, virus, intrusions...) contre les sites américains. On parlait alors d'une cyber-guerre sino-américaine. Le quotidien de l'Armée de libération chinoise écrivait le 27 juillet 1999 qu'une bataille des réseaux était engagée entre la Chine et les États-Unis.

- La déclaration du président de **Taiwan**, Teng-Hui Li, en août 1999, affirmant le principe des « deux États » (la Chine et Taiwan). Les affrontements revêtent davantage la forme de duels entre hackers individuels (défiguration de sites, vols de données, virus...) qu'entre vrais combattants depuis leurs bunkers à Pékin ou à Taipei.

- En 2001, les **États-Unis** sont victimes d'une attaque en règle contre leurs sites officiels en protestation contre la mort d'un pilote chinois lors d'une collision avec un avion espion américain près des côtes chinoises le 1er avril.

- En avril 2005, le **Japon** est confronté à un puissant sentiment anti-japonais (violentes manifestations de rue à Pékin ou Shanghai, militaires appelant à la guerre) accompagné d'une recrudescence des cyber-attaques contre les sites japonais (sites des ambassades, agences nationales, ministère de la Défense, des Affaires étrangères, universités, entreprises comme Sony...). Les attaques visent les systèmes vitaux, tentent de dérober des informations confidentielles.

- Des centaines d'ordinateurs du gouvernement **sud-coréen** ont été infectés au cours des derniers mois par des virus capables de dérober les mots de passe et autres informations sensibles. Selon la KISA (*Korean Information Security Agency*), durant les six premiers mois de 2004, un total de 10628 cas de piratage auraient été enregistrés, soit 30 fois plus que durant la même période de 2003.

- Le gouvernement **tibétain** en exil en Inde accuse également des groupes basés à Pékin de s'être introduits dans ses sites officiels et de l'avoir espionné. Les hackers procèdent par envois de mails accompagnés de Trojan, et usent de l'usurpation d'adresse pour rendre leurs messages plus authentiques. De telles activités on été rapportées depuis octobre 2003.

Qui sont donc ces hackers patriotes ? Leurs actes sont à la fois l'illustration du concept de « guerre du peuple » et une modalité de la « défense active » (ripostes à des attaques extérieures, atteintes aux systèmes adverses, protection des systèmes du pays). Mais les attaques en nombre croissant depuis la fin des années 1990, du fait de l'augmentation exponentielle des internautes en Chine, ne sont pas uniquement défensives. Elles prennent pour cibles des centres de recherches, des entreprises, des administrations d'Etat de l'ensemble des pays industrialisés. Les attaques ont plusieurs motifs : vol d'information, espionnage, revendications idéologiques, déstabilisation. Pour nombreuses que soient les accusations portées à l'encontre de la Chine, il n'est pas toujours possible de démontrer avec exactitude l'origine des attaques de type intrusion dans les systèmes, défiguration de sites, altération de données, vol de données, DoS, *flooding*, propagation de virus. S'agit-il de hackers chinois ou pro-chinois ? Exercent-ils en Chine ou depuis un autre pays ? Sont-ils des pirates isolés dont les actions peuvent rejoindre les intérêts de l'Etat, des pirates commandités par le gouvernement, des hackers des unités de l'armée aux méthodes d'attaques très professionnelles ?

Pékin réfute bien entendu l'implication de l'Etat dans les actes de piratage visant les intérêts étrangers, malgré les nombreuses allégations internationales. Rien ne prouve que les autorités chinoises commanditent ces activités. Mais en ne les réprimant pas, elles les cautionnent.

III – Les besoins technologiques

La démarche chinoise a consisté à créer des concepts, doctrines et stratégies préalablement à la mise en œuvre des moyens nouveaux, à l'acquisition et à l'intégration des évolutions technologiques. La Chine développe ses infrastructures civiles et militaires. Le pays a travaillé activement au développement de systèmes de positionnement global, réseaux de fibres optiques

[20] « China's National Defense in 2004 » (Livre Blanc de la Défense), Décembre 2004, Pékin. http://www.china.org.cn/e-white/20041227/

[21] « local wars under conditions of informationalization ». Cette terminologie a été introduite en 2004, remplaçant celle de « local wars under high-tech conditions » (guerres locales sous contraintes high-tech) qui mettait en valeur le rôle majeur des hautes technologies.

[22] « The Military Power of the People's Republic of China 2005 », Annual Report to Congress, Office of the Secretary of Defense. http://www.globalsecurity.org/military/library/report/2005/d20050719china.pdf

et infrastructures de communication, amenés à être complétés par des satellites, missiles balistiques, missiles de croisière, ainsi que des virus, bombes logiques, armes EMP [23], tous « made in China ».

Pour développer ses capacités de GI, la Chine a besoin de technologies de pointe qu'elle va acquérir de plusieurs manières.

Recherche et développement sont menés conjointement dans les centres civils et militaires. La Chine allouant un budget important à la défense (environ 9% du PIB) [24], elle peut se permettre d'investir dans le développement de la GI.

Déjà en 1991, le Centre Chinois des Sciences et Technologies de l'Information estimait que 80% des informations nécessaires à l'acquisition de technologies étaient disponibles dans des sources ouvertes. Au-delà de la stricte **veille technologique**, l'**espionnage** est un moyen supplémentaire, ancré dans la tradition des stratèges chinois. Le chapitre XIII de *L'Art de la Guerre* de Sun Tzu est lui-même entièrement dédié à l'espionnage.

Parallèlement au développement de solutions au sein de ses propres industries, la Chine est un important **importateur de technologies** à des fins de construction d'une infrastructure de GI. Sous l'administration Clinton, la Chine a importé directement des États-Unis des supercalculateurs. Combien auront terminé leur route dans des laboratoires militaires à des fins autres que celles initialement prévues ? Les exportations de l'UE vers la Chine sont, quant à elles, sous embargo depuis les évènements de Tiananmen de 1989. Lever cet embargo serait profitable à la Chine en lui permettant d'accéder à des technologies à double usage. Les technologies de cryptographie sont pour leur part sous le contrôle des pays membres des Arrangements de Wassenaar [25], dont la Chine n'est pas, ce qui lui rend difficiles les coopérations industrielles dans le domaine et l'acquisition des technologies occidentales.

Conclusion

La Chine veut maîtriser l'information et l'utiliser comme outil stratégique. Elle s'en donne les moyens. Le défi est alors double. Pour la Chine tout d'abord. L'émergence rapide des TIC constitue une (r)évolution sociale, économique, politique, qui appelle les stratèges à relever de nouveaux défis. Pour le reste du monde ensuite. Que fera la Chine de ce pouvoir que constitue la maîtrise de l'information ?

Références complémentaires

Qiao Liang, Wang Xiangsui, « *Unrestricted Warfare* », PLA *Literature and Arts Publishing House*, Février 1999, Pékin.
www.terrorism.com/documents/TRC-Analysis/unrestricted.pdf

Toshi Yoshihara, « *Chinese Information Warfare: A Phantom Menace or Emerging Threat ?* », Novembre 2001.
www.iwar.org.uk/iwar/resources/china/iw/chininfo.pdf

[23] Electro Magnetic Pulse, armes à impulsion électromagnétique.

[24] General Abe C. Lin., « Comparison of the Information Warfare Capabilities of the ROC and PRC ». 27 décembre 2000. http://cryptome.org/cn2-infowar.htm

[25] Les Arrangements (13/07/96) visent à limiter les exportations d'armes, de biens et de technologies à double usage.

Un accès sans limite à la mémoire grâce au FireWire

Le protocole FireWire est bien connu quand il s'agit de connecter des appareils vidéo numériques ou des disques durs externes à des ordinateurs. Mais on ignore le plus souvent qu'il permet de lire et d'écrire des données dans la mémoire physique des appareils connectés, et ce, sans intermédiaire logiciel (technologie DMA, ou Direct Memory Access). Cet article évoque les aspects du protocole qui permettent l'accès à la mémoire. Il montre, en outre, comment utiliser en pratique ces spécifications pour par exemple écraser la mémoire vidéo. Le protocole prévoit néanmoins les moyens d'empêcher cet accès, lesquels sont désormais mis en œuvre par les systèmes d'exploitation. Sur un serveur, il peut cependant être intéressant de permettre l'accès. Il devient ainsi possible, dans le cas où l'ordinateur serait compromis, d'obtenir une image complète de la mémoire centrale, et ce, sans modifier l'état du serveur (analyse post-mortem).

Contexte technique

Les possibilités décrites plus loin reposent sur la norme OHCI [1], *1394 Open Host Controller Interface*, une implémentation du protocole de la couche liaison du bus série IEEE-1394. L'IEEE-1394 se décline en trois versions : IEEE-1394 (1995), IEE1394a (2000) et IEEE-1394b (2002), chacune ayant permis de décupler la vitesse de transfert.

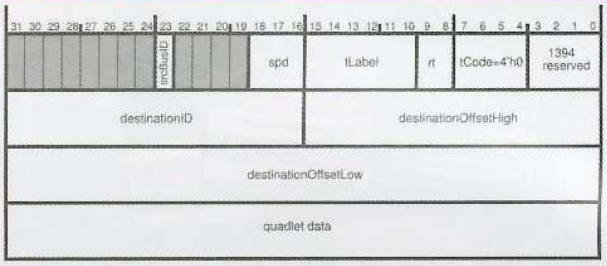
Le bus assure un débit de données de 800 Mb/s et permet d'établir des connexions directes entre les périphériques sans passer par un ordinateur hôte, contrairement à l'USB. Il est possible de relier 1023 ponts (reliant plusieurs bus entre eux), chacun pouvant recevoir 63 nœuds.

Parallèlement au mode de transfert asynchrone, il fonctionne également en mode isochrone, ce qui lui permet d'utiliser une bande passante large, si bien que cette interface est devenue courante dans le secteur de la vidéo numérique. Appelées i.Link par Sony et FireWire par Apple, ces implémentations par les deux constructeurs de la norme IEEE-1394 sont parfaitement compatibles. Mais on désigne globalement par le terme FireWire autant la norme OHCI que l'une des trois versions de la norme 1394.

Les appareils FireWire sont identifiés par un numéro GUID (*Global Unique ID*), qui est en fait un nombre codé en clair sur 64 bits. 24 bits sont réservés aux paramètres du fabricant et 40 bits à la puce d'identification.

La mémoire est alors répartie en « quadlets », soient 4 octets, selon l'appellation de l'IEEE. Pour profiter des possibilités décrites ci-après, seules deux commandes du mode de transfert asynchrone sont nécessaires, à savoir « *quadlet read request* » et « *quadlet write request* ». La requête « *write* » a la structure suivante figure 1.

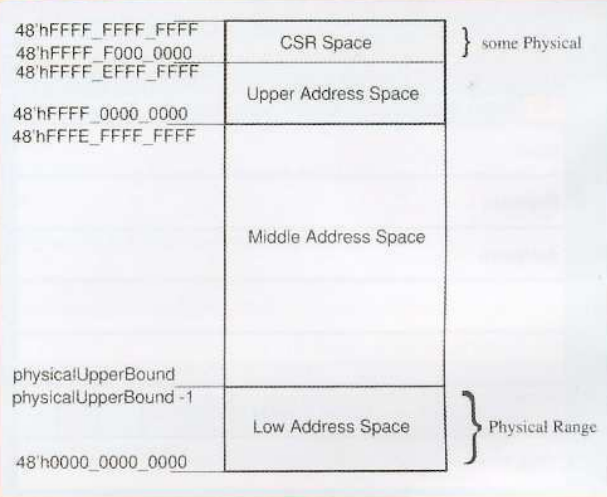
Figure 1



« Quadlet write request » (d'après la norme OHCI)

Les champs présentant un intérêt dans le paquet sont `destinationID`, `destinationOffset` et `quadlet data`. `destinationID` combine l'ID du bus sur 10 bits et l'ID du périphérique sur 6 bits. `quadlet data` comprend les données de l'instruction d'écriture, qui sont écrites sur l'adresse `destinationOffset` de 48 bits. L'adresse mémoire du périphérique est répartie comme suit :

Figure 2



Adressage selon la norme OHCI

Le paramètre « *low address space* » est intéressant puisqu'il représente la mémoire physique de l'appareil. La limite supérieure de la plage mémoire est déterminée en fonction du registre optionnel `physicalUpperBound` décrit dans la norme OHCI. Elle est fixée à 4 Go lors de l'initialisation de l'appareil. Elle peut cependant être réduite si le registre optionnel est implémenté par l'équipement. L'espace de stockage, réparti sur les 48 bits de l'adressage, peut s'étendre jusqu'à 256 téraoctets, mais cela ne présente pas d'intérêt.

Ces quelques connaissances préalables permettent de mettre en avant les caractéristiques intéressantes de l'OHCI :

Michael Becher
michael.becher@rwth-aachen.de,

Maximillian Dornseif
md@hudora.de

Norme OHCI – extrait 1

« Low Address Space is from 48'h0 up to physicalUpperBound. Asynchronous read and write requests into this range can be handled by the Physical Request/Physical Response units, providing an efficient mechanism for moving asynchronous data. »

(Traduction : "Le Low Address Space se situe entre 48'h0 (0000_0000_0000) et physicalUpperBound. Les unités Physical Request/Physical Response exécutent les requêtes de lecture et d'écriture asynchrones contenues dans cette plage et permettent ainsi de déplacer efficacement des données asynchrones.»)

Les unités *Physical Request/Physical Response* sont des unités fonctionnelles se trouvant sur la carte qui traitent les flux logiques à travers l'interface OHCI, et qu'on désigne par *Physical Requests*. (Leurs caractéristiques sont détaillées dans un chapitre de la norme).

Norme OHCI – extrait 2

« When a block or quadlet read request or quadlet write request is received, the 1394 Open HCI chip handles the operation automatically without involving software if the offset address in the request packet header meets a specific set of criteria. »

(Traduction : «Quand un bloc, une requête de lecture ou une requête d'écriture est reçue, la puce 1394 Open HCI exécute l'opération automatiquement sans passer par un mécanisme logiciel, si l'offset de l'adresse contenu dans l'en-tête du paquet satisfait certains critères spécifiques».)

Cela signifie donc que, si certaines conditions sont remplies, des requêtes d'écriture et de lecture sont exécutées directement par le matériel FireWire sans intervention du pilote. Lors d'une requête, un paramètre doit se situer dans les limites du *Physical Range* (voir Fig. 2), c'est-à-dire précisément dans les 4 Go de la mémoire centrale. Avant de répondre, le périphérique FireWire commence par vérifier si les bits requis sont correctement placés dans les deux registres qui permettent de filtrer les requêtes. Le pilote aurait pu faire varier ce second critère s'il avait écrit dans les registres en question. Pour plus de détails sur ce point, consultez la partie « Filtre OHCI » située plus bas. On suppose en plus que les registres contiennent les valeurs correctes, rendant ainsi possible la lecture et l'écriture.

Exploitation des possibilités

Un ordinateur met entièrement sa mémoire principale (en lecture et écriture) à disposition de tous les appareils FireWire auxquels il est connecté. Cela fonctionne indépendamment du processus auquel appartient la mémoire et des droits d'accès locaux. La mémoire centrale de la carte graphique, en général, se fond à la mémoire principale de l'ordinateur, ce qui permet d'accéder au contenu de l'écran, comme cela est expliqué ci-après. La mémoire centrale d'un ordinateur offre de nombreux points d'attaque potentiels. Elle permet par exemple de faire une recherche parmi des chaînes de caractères, lesquelles chaînes peuvent représenter des mots de passe qui auraient été gardés en mémoire par un gestionnaire de mots de passe. Une telle tentative, comme la recherche d'une clé privée d'un système de cryptographie à clé publique, est décrite en référence [2].

On peut retrouver efficacement une telle clé dans un grand volume de données, si on connaît la clé publique. Si la clé privée venait à se retrouver dans la mémoire pour une raison quelle qu'elle soit, elle pourrait être compromise. Il est en principe possible de récupérer de nombreuses informations via le système ; elles doivent cependant être correctement interprétées dans l'ordre physique incohérent de la mémoire.

Il est aisé d'écraser l'ensemble de la mémoire et de faire planter le système en écrivant directement dans les pages mémoire. Il est néanmoins plus intéressant de bloquer seulement un processus isolé ou d'écrire dans la mémoire vidéo. L'injection du code dans certains processus ou la production de nouveaux processus pourraient nuire au système en exploitant de manière plus ambitieuse certaines possibilités du FireWire.

Comment utiliser concrètement les informations précédentes ? Pour réaliser une programmation via le FireWire, il est possible de faire appel aux multiples fonctions des APIs, dont on utilisera uniquement les fonctions de base en lecture et écriture, pour exécuter un *quadlet read* ou *quadlet write*. Les lignes de code les plus importantes lors de l'écriture ressemblent à ceci :

```
MacOS :
IOCreatePlugInInterfaceForService(self->aDevice,kIOFireWireLibTypeID,
kIOCFPlugInInterfaceID, &cfPlugInInterface, &theScore);
(*cfPlugInInterface)->QueryInterface(cfPlugInInterface, CFUUIDGetUUIDBytes(kIOFir
eWireDeviceInterfaceID), (void **)&fwIntf);
(*fwIntf)->Open(fwIntf);
(*fwIntf)->Write(fwIntf, self->aDevice, &fwaddr, (void *) buffer, &bufsize, false,
0);

Linux (libraw1394) :
handle = raw1394_new_handle();
raw1394_set_port(handle, 0);
raw1394_write(handle, node_id, fwaddr, bufsize, (quadlet_t *) buf);
```

Ces quelques lignes prouvent donc que l'accès au Physical Range (Fig. 2) est possible sans intervention logicielle et étayent nos déclarations antérieures. Pour s'en rendre compte, dans le noyau Linux, le paramètre `Debug` doit cependant être activé pour le support FireWire (*Excessive debugging output* ou `CONFIG_IEEE1394_VERBOSEDEBUG`). Cette option permet en particulier d'écrire les en-têtes de chaque paquet reçu dans `Syslog`. En essayant différentes adresses mémoire situées en dessous et au-dessus du registre `PhysicalUpperBound` (et/ou 4 Go), on peut voir que toutes les demandes situées en dessous de `PhysicalUpperBound` ne seront pas journalisées. Elles sont donc directement prises en charge par le matériel FireWire.

Les codes ci-dessus permettant l'accès au bus FireWire ont été écrits avec l'environnement nécessaire à l'intérieur d'un module C sous Python. On peut considérer qu'ils sont facilement utilisables sous Linux et MacOS, si on fait abstraction des nombreuses étapes indispensables à la programmation sous C et l'exécution de scripts sous Python. Python offre les fonctions `read` et `write` pour lire et écrire des données, ainsi que la fonction `scanbus` qui permet de faire une liste de tous les appareils connectés au bus FireWire. Le module et des exemples de scripts sont développés dans **[3]**. Les informations qui suivent vous donnent une idée de la manière dont on peut lire et écrire directement sur la mémoire. Les codes donnés en exemple sont écrits en Python et utilisent le module Python.

Suppression du contenu d'un écran

La mémoire vive de la carte graphique va généralement être combinée à la mémoire centrale de l'ordinateur et permettre ainsi son accès direct. L'intérêt principal de cet accès est d'offrir la possibilité de pouvoir modifier l'affichage à l'écran. Pour un résultat immédiatement visible, il est possible de supprimer le contenu d'écran d'un autre ordinateur via le bus FireWire. Cela fonctionne avec le programme suivant :

```
addrs = { 0xd0c0000c2cf801:     (0xd8000000L, 1400, 1050, 2) }

devices = fw.scanbus()

for device in devices:
    if device.guid in addrs:
        base, xres, yres, bpp = addrs[device.guid]

        pos = base + (xres*bpp)
        while pos < base + (yres*xres*bpp):
                print device.write(pos, "\0"*(xres*bpp*4)),
                pos += xres*bpp*4
```

Les ordinateurs sont facilement identifiables par leur GUID. On peut ainsi assigner manuellement les valeurs importantes qui définissent la position et la taille de la mémoire vidéo. Le GUID du matériel FireWire se trouve facilement sous Linux, par exemple via `Syslog`, lorsqu'un appareil est connecté à l'ordinateur, et sur un Mac, dans la rubrique ***À propos de ce Mac -> Plus d'infos***. Le GUID présente un intérêt, dans la mesure où il est possible de paramétrer l'adresse à laquelle la mémoire vive de la carte va se trouver. Sous Linux, elle sera écrite et pourra être récupérée par exemple dans le fichier log du serveur X. Il est important de remarquer également que l'adresse ne dépend pas directement du matériel mais bien du système d'exploitation. Il suffit alors d'écraser la mémoire en écrivant des zéros. Cela se traduit par un écran devenant noir.

Lecture de la mémoire

Pour passer de l'écriture à la lecture sur la mémoire vidéo, il n'y a qu'un pas. Toutefois, nous avons un peu simplifié la procédure d'écriture : l'écran est simplement devenu noir. Mais lorsque l'on veut convertir la mémoire en images, il est alors indispensable de connaître avec précision l'organisation physique de la mémoire vidéo. Malheureusement, il s'avère que l'organisation de la mémoire vidéo ne dépend pas uniquement du matériel utilisé mais bien du système d'exploitation et du mode vidéo utilisé par l'interface graphique. En mode texte, sous FreeBSD, vous trouverez par exemple un caractère tous les 8 octets. Il est possible de lire la mémoire vidéo du mode texte grâce à une boucle simple :

```
data = device.read(addrs[device.guid], 80*25*8)
data = data[::8]
print "+%s+" % ("-"*80)
while data:
    print "|%s|" % data[:80]
    data = data[80:]
print "+%s+" % ("-"*80)
```

Les modes graphiques sont en revanche considérablement plus difficiles. On termine généralement le processus par une combinaison en bits dans Python et une étape de post-processing via ImageMagick.

Obtention des droits root dans un processus

Lorsqu'on peut écrire dans la mémoire, il est également possible de s'octroyer les droits *root*. En principe, bien sûr. Notre premier « Proof of Concept » a été de « dumper » le contenu complet de la mémoire d'un iBook dans un fichier. À l'aide d'un éditeur hexadécimal, nous avons ensuite recherché l'identifiant de l'utilisateur connecté à l'intérieur du fichier, pour retrouver ensuite les adresses correspondant à cet utilisateur. Nous avons alors programmé un script pour mettre la valeur de ces adresses à zéro. Cette tentative a immédiatement porté ses fruits : l'identifiant de l'utilisateur valant zéro, l'utilisateur obtenait alors les droits root sur la machine. Mais cela ne s'est pas fait sans engendrer des problèmes considérables : en effet, après le redémarrage, les fichiers utilisateurs situés sur le bureau s'étaient convertis en fichiers ayant les droits root. Une telle situation n'est guère étonnante, mais plutôt désagréable...

Pourtant, il est difficile de se satisfaire d'une procédure aussi peu fiable. Lors d'une conférence, les résultats d'analyse de fichiers à l'aide d'éditeur hexadécimal sont ennuyeux à présenter et d'autre part, le propriétaire de l'iBook ne souhaitait plus soumettre son appareil aux fins de cette expérience. Nous avons donc fini par « sacrifier » un ordinateur FreeBSD 5.3. La recherche de l'identifiant d'un utilisateur précis renvoyait beaucoup d'autres données ; elles avaient par ailleurs aussi été remises à zéro, engendrant une belle confusion. Nous avons alors tenté de construire une heuristique capable de reconnaître la structure du processus dans la mémoire et ne mettant les valeurs à zéro qu'à cet endroit-là. Dans la pratique, cela équivaut à rechercher, à l'aide d'expressions régulières, les valeurs typiques contenues dans la structure de description d'un processus. Il s'avéra malgré tout que de nombreuses données avaient été remises à zéro, données qu'on aurait préféré ne pas toucher, sans oublier l'instabilité postérieure de l'ordinateur. Mais il est évident que pour une première tentative, cela est satisfaisant et permet au pirate ayant un accès physique de récupérer un shell root sans rien faire ou presque.

Un iPod pour pénétrer dans la mémoire

L'iPod commercialisé par Apple n'est pas seulement un lecteur MP3, mais également un appareil disposant d'une interface FireWire et d'un disque dur interne. De plus, une distribution Linux a même été développée pour l'iPod. De taille compacte,

alimenté grâce à une batterie, l'iPod se révèle un outil parfait pour transposer les concepts décrits plus haut, mis en pratique sur d'autres machines. L'installation d'iPodLinux est très bien décrite sur les pages du projet [4]. Il existe toutefois une différence par rapport à la distribution normale : le module noyau `raw1394` doit être compilé. C'est pourquoi il est nécessaire d'aménager un peu le système si vous installez Linux sur un iPod.

L'iPod doit être démarré dans le mode Disque à l'installation ; il suffit pour cela d'appuyer simultanément sur les touches « Back » et « Forward » pour un iPod de la troisième génération ou sur « Menu » et « Select » pour un modèle de la quatrième génération. La quatrième génération d'iPod n'est cependant pas officiellement supportée par iPodLinux, et n'est pas compatible FireWire. C'est pourquoi nous poursuivons notre exemple avec un iPod de troisième génération. En mode Disque, l'iPod sera reconnu comme appareil SCSI grâce au module SBP2 et associé à `/dev/sda`. Le firmware devra par ailleurs être sauvegardé grâce à `dd if=/dev/sda1 of=ipod_os_partition_backup`, de manière à pouvoir revenir au mode normal baladeur. On partitionne ensuite le disque dur pour générer un nouveau système de fichiers (n'oubliez pas de sauvegarder préalablement les données importantes contenues dans votre iPod). L'iPod peut alors être monté sur ce système de fichiers avec `mount -t ext3 /dev/sda3 /mnt/ipod`.

Une première étape pour la personnalisation du noyau sera d'installer les GNU Toolchains, qui comprennent un compilateur et des outils de cross-compilation pour l'architecture ARM du iPod. Vous trouverez une version binaire du projet Linux/Microcontroller aux liens indiqués en [4]. Ce projet repose sur un noyau 2.4.24, lequel est fourni avec des patches spéciaux pour l'iPod et des fichiers de configuration également spécifiques pour l'iPod. À ce stade, la procédure est différente des instructions données parce que le noyau doit contenir le module `raw1394`. On appelle alors par exemple `make menuconfig` et on choisit, dans le menu « IEEE1394 (FireWire) Support », l'option « Raw IEEE 1394 I/O support » en tant que module. On termine enfin en supprimant la fonction `raw1394_mmap` et sa référence dans la structure `file_ops` qui apparaissent dans le fichier `linux-2.4.24/drivers/ieee1394/raw1394.c`. Cette dernière n'est pas utile et pourrait engendrer une erreur au chargement du module sur l'iPod. On continue ensuite avec la « construction du noyau » (c'est-à-dire le paramétrage de `my_sw.bin` avec `make_fw` et la copie sur l'iPod via `dd`). Si l'iPod est disponible sous `/mnt/ipod`, les modules peuvent ensuite être installés depuis le répertoire principal du noyau fraîchement compilé par la commande `make modules_install INSTALL_MOD_PATH=/mnt/ipod`.

Il faut maintenant procéder à l'installation du reste du système d'exploitation sur l'iPod (section « Userland installation »), en adaptant certains fichiers et installant la version en cours de l'interface utilisateur graphique Podzilla (section « Finishing up »). À cause de l'installation du module `raw1394`, les lignes suivantes doivent être ajoutées au fichier `/etc/rc` :

```
modprobe tsb43aa82
mknod -m 600 /dev/raw1394 c 171 0
modprobe raw1394
```

Si vous ne souhaitez pas faire le travail vous-même, vous pouvez aussi télécharger notre version d'installation depuis [5] et importer les images des partitions `sda1` et `sda3` sur votre iPod.

Figure 3



Filtre OHCI

Parallèlement à la lecture et à l'écriture directe sur l'unité Physical Response, la norme OHCI offre la possibilité d'empêcher cet accès. Les registres de 64 bits `AsynchronousRequestFilter` et `PhysicalRequestFilter` sont prévus à cet effet. Dans les deux cas, un bit représente un appareil du bus FireWire. Ce bit mis à zéro dans le registre `AsynchronousRequestFilter` empêche généralement tout appareil d'accéder à la mémoire physique du système local. Cela permet ainsi de couper la liaison FireWire si nécessaire. Mais cela vous fait cependant passer à côté d'autres utilisations.

Plus important, le registre `PhysicalRequestFilter` interdit tout accès direct à un nœud (si le bit est remis à 0) sur l'unité Physical Response du nœud local, et remet la responsabilité au pilote au lieu du nœud en question. Le pilote identifie chaque tentative d'accès à la mémoire physique et peut décider au cas par cas s'il accepte la tentative d'accès ou la rejette. Pour plus de détails sur la norme OHCI, voir [1], paragraphe 5.14. Cette possibilité d'empêcher l'accès à la mémoire grâce au registre `PhysicalRequestFilter` est prévue dans les pilotes Linux et Mac OS. Elle peut être activée facilement :

Linux : ohci1394.c

```
                /* Accept Physical requests from all nodes. */
                reg_write(ohci,OHCI1394_AsReqFilterHiSet, 0xffffffff);
                reg_write(ohci,OHCI1394_AsReqFilterLoSet, 0xffffffff);
                /* Turn on phys dma reception.
                 *
                 * TODO: Enable some sort of filtering management.
                 */
                if (phys_dma) {
                        reg_write(ohci,OHCI1394_PhyReqFilterHiSet, 0xffffffff);
                reg_write(ohci,OHCI1394_PhyReqFilterLoSet, 0xffffffff);
                reg_write(ohci,OHCI1394_PhyUpperBound, 0xffff0000);
                   } else {
                reg_write(ohci,OHCI1394_PhyReqFilterHiSet, 0x00000000);
                reg_write(ohci,OHCI1394_PhyReqFilterLoSet, 0x00000000);
                   }

                DBGMSG("PhyReqFilter=%08x%08x",
                       reg_read(ohci,OHCI1394_PhyReqFilterHiSet),
                       reg_read(ohci,OHCI1394_PhyReqFilterLoSet));
```

MacOS X : IOFireWireController.cpp

```
                IOFWSecurityMode mode = kIOFWSecurityModeNormal;
                OSString * securityModeProperty = OSDynamicCast(
OSString,options->getProperty("security-mode") );
```

```
if( securityModeProperty != NULL &&
strcmp("none", securityModeProperty->getCStringNoCopy()) != 0 )
{
        // set security mode to secure/permanent
        mode = kIOFWSecurityModeSecurePermanent;
}
et ensuite :
// shut them all down!
fFWIM->setNodeIOPhysicalFilter( kIOFWAllPhysicalFilters, false );
```

Pour empêcher l'accès sous Linux, les paramètres de la variable `phys_dma` du fichier `ohci1393.C` doivent être définis en chargeant le module, par exemple avec la commande `modprobe ohci1394 phys_dma=0`. Sur un système MacOS X 10.3, il existe un équivalent avec le fichier `IOFireWireController.cpp` si le « security mode » de l'Open Firmware actuel renvoie une autre valeur que « none ». C'est dans ce fichier que les registres `PhysicalRequestFilter` seront remis à 0. Mais cette fonctionnalité n'est pour l'instant pas documentée. Ce dont nous sommes certains c'est que la version actuelle de MacOS X ne permet plus cet accès. NetBSD reconnaît bien le matériel FireWire et les appareils FireWire connectés, mais ne permet cependant aucun accès. Les registres sont directement placés dans un mode sûr. On peut le vérifier dans le code source avec `fwohci.c`.. Sous Windows, l'accès n'est pas possible non plus. Sur les anciennes versions de Windows 2000, cela venait tout simplement du fait que le système se bloquait dès qu'il était connecté à un Mac via une liaison FireWire. Sur Windows XP, la connexion ne fonctionne pas, et donc l'accès est par le fait même impossible. Est-ce parce que les registres sont initialisés en mode sûr ou parce que ceux-ci demeurent dans leur état de démarrage lors de l'initialisation du matériel ? Impossible d'examiner la situation dans la mesure où aucun code source ni documentation ne sont disponibles.

Analyses post-mortem

Bien que les filtres soient finalement implémentés partout, il est possible de permettre à certains systèmes l'accès à la mémoire, et d'effectuer ainsi des analyses sur un ordinateur sous tension. Jusqu'à présent, il existe deux possibilités pour une analyse : éteindre l'ordinateur et réaliser une analyse post-mortem, ce qui détruit les informations dans les processus en cours, ou récolter des informations sur un système sous tension, ce qui peut engendrer la disparition de preuves importantes. On peut résoudre le problème en réalisant un vidage complet de la mémoire via la liaison FireWire, sans pour autant modifier le système. Le défi de cette solution repose, entre autres, sur l'assignation de la mémoire physique par rapport à la disposition logique de la mémoire, car il s'agit de faire ressortir des données utilisables de la mémoire qui a été vidée. Pour l'instant, il n'existe pas de logiciel qui permette une analyse post-mortem précise d'un programme à partir d'une image mémoire brute. Cela sous-entend que l'on peut certes rechercher des chaînes dans ce type d'image mémoire, mais que pour réaliser des recherches plus approfondies, comme par exemple, l'examen des connexions réseau ouvertes ou autres, les outils font actuellement défaut.

Conclusion

Quelle conclusion peut-on tirer de l'histoire du FireWire si ce n'est une évidence ? Les ports FireWire font partie du Trusted Computing Base (TCB). On ne devrait les relier qu'à des appareils dignes de confiance, ou le cas échéant, les bloquer ou les verrouiller. On peut supposer que d'ici *peu on fera état de* tel ou tel hack via FireWire, comme par exemple un piratage de fond d'écran, ou des manipulations sur des terminaux publics via des ports FireWire, tel qu'il en existe déjà sur les appareils des laboratoires photos. Mais l'aspect le plus intéressant est en fait le manque de communication entre les différentes communautés. Pour les développeurs de pilotes, il était connu que le FireWire permettait un accès direct à la mémoire. Cet accès est même utilisé depuis longtemps pour le débogage. Mais les implications liées à la sécurité n'ont jamais été prises au sérieux. Et, pendant longtemps, la communauté dite de sécurité ignorait tout de ces « caractéristiques ». Aussi, aujourd'hui, nous souffrons d'un manque cruel de communication. Des filtres OHCI sont installés par défaut ça et là, mais ils ne sont jamais documentés et il n'est pas aisé de deviner à quelle version d'OS s'adressent les règles de filtres OHCI. C'est pour cette raison qu'il semble urgent que la communauté de sécurité informatique et les fabricants de matériel collaborent plus activement sur ce thème mais également sur d'autres sujets, comme par exemple, l'USB.

Remerciements

Nous souhaitons remercier Christian Klein qui ne s'est pas contenté d'être à l'origine de l'accès à la mémoire via FireWire, mais qui a aussi contribué à nos premières tentatives d'implémentations sur Mac OS.

Références

[1] Norme OpenHCI version 1.1 datant de janvier 2000 : http://developer.intel.com/technology/1394/download/ohci_11.htm

[2] Adi Shamir, Nicko van Someren, Playing hide and seek with stored keys : http://www.ncipher.com/resources/downloads/files/white_papers/keyhide2.pdf

[3] Consultez la bibliothèque Python et essayez les scripts : http://c0re.23.nu/c0de/pyfw/

[4] Page sur le projet iPodLinux avec les conseils pour installer Linux et construire un kernel : http://www.ipodlinux.org/Installation_from_Linux, http://www.ipodlinux.org/Kernel_Building

[5] Toutes les infos importantes pour supprimer le contenu de l'écran d'un ordinateur à l'aide d'un iPod : http://md.hudora.de/presentations/firewire/ipod-linux-pyfw.tar.bz2

Vulnérabilités logicielles et droit : responsabilité, recherche et divulgation

Les vulnérabilités logicielles ou applicatives sont monnaie courante, à tel point que certains ironisent en évoquant ces programmes « qui tombent en marche », et pour lesquels les fournisseurs et éditeurs refusent de ne donner aucune garantie contractuelle [1]. Pourtant, malgré les différentes sources de responsabilité reconnues par la loi (1), ceux-ci continuent trop souvent encore de rester sourds aux doléances des utilisateurs. Dès lors, la politique de la « sécurité par la transparence », qui procède de techniques de recherche de vulnérabilités dont la légalité est strictement encadrée (2), trouve auprès de l'ensemble des protagonistes (y compris les juges [2] ?) un écho favorable, sans pour autant justifier certaines pratiques de divulgation irresponsable des failles de sécurité (3).

1. Des programmes qui « tombent en marche » : quelle responsabilité légale des fournisseurs/éditeurs au regard de la qualité et de la performance de leurs logiciels

L'actualité récente a été riche en pannes informatiques [3]. Si leur origine n'a pas toujours été explicitée avec force de détails, il s'agissait cette fois encore des conséquences d'un « *bug* », c'est-à-dire une erreur de conception dans le produit qui impacte la fiabilité du système (voire peut mettre en cause sa sécurité dès lors que la vulnérabilité est découverte et devient exploitable par un tiers non autorisé).

Pourtant, bien que ces dysfonctionnements deviennent de plus en plus dommageables au fur et à mesure que les systèmes d'information touchent le cœur même de l'activité de l'entreprise, on continue d'observer chez les « particuliers », usagers de ces produits, comme un « seuil de tolérance » vis-à-vis des fournisseurs/éditeurs concernés. Se protégeant derrière une clause d'exonération générale de responsabilité, les fournisseurs de solutions de sécurité et les éditeurs de programmes continuent parfois encore de travailler suivant des standards de qualité inférieurs à ceux des autres industries [4], sans être inquiétés du fait du manque de fiabilité de leurs produits et de l'insécurité logicielle qui en découle. En effet, en présence d'une clause d'exonération totale de responsabilité (c'est-à-dire que le logiciel est fourni « tel que » [5]), c'est le licencié (le cas envisagé ici étant celui du particulier utilisateur) [6] qui prend la totalité du risque quant à la qualité et aux performances du logiciel.

D'une manière générale, le recours aux clauses limitatives ou élusives de responsabilité est autorisé pour limiter la responsabilité *contractuelle* [7] (entre cocontractants [8]) aussi bien en France (article 1150 du code civil) qu'aux Etats-Unis [9] (*Uniform Commercial Code*, article 2-316 - **http://www.law.cornell.edu/ucc/2/2-316.html**). Néanmoins, elles font l'objet de certaines restrictions, qui sont très souvent imposées par le législateur dans un souci de protection du consommateur

[1] A titre d'illustration : Extrait du Contrat de Licence Utilisateur Final – CLUF © 2004 Microsoft Corporation. « 12. La garantie. - Le Produit est conçu et proposé comme étant un produit à usage général et n'est pas destiné à répondre aux besoins particuliers d'un quelconque utilisateur. Vous reconnaissez que le Produit n'est pas exempt d'erreurs et que nous vous avons fortement recommandé de sauvegarder régulièrement vos fichiers. Sous réserve que vous disposiez d'une licence valable, Microsoft garantit que a) le fonctionnement du Produit sera conforme, pour l'essentiel, à la description qui figure dans la documentation écrite qui accompagne le Produit, pendant une durée de quatre-vingt-dix (90) jours à compter de la date de réception de votre licence d'utilisation du Produit ou pendant la durée la plus courte autorisée par la réglementation applicable, (...). » Par ailleurs, les logiciels libres ne sont pas en reste sur ce point : Extrait de la licence GNU GPL « 11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expresses or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction. »

[2] Commentant le délibéré du 8 mars dans l'affaire « Guillermito » et son verdict façon « relaxe Canada Dry », Me Eolas (http://maitre.eolas.fr/journal/...) relève, très justement selon nous, que « le tribunal n'a donc [pas eu] l'hostilité du parquet envers les Arsène Lupin et les Robins des Bois (...) ». Pour plus de détails sur cette affaire, voir plus loin l'encadré... De même, dans l'affaire Tati/Kitetoa (où une faille de sécurité avait été révélée après information et non correction par le responsable du site), le Parquet lui-même avait interjeté appel de la décision du Tribunal correctionnel qui avait condamné l'animateur du site pour accès frauduleux à un système ; la Cour d'appel avait alors infirmé la décision de première instance en considérant que l'accès avait été fait « par des moyens réguliers »....

[3] 30/10/2004 : perturbation des communications vers les téléphones fixes ; 17/11/2004 : panne sèche pour le réseau de téléphonie mobile de Bouygues Telecom ; 3/12/2004 : « vendredi noir » pour la SNCF dont près de 1000 terminaux de vente de billets sont paralysés (Source : 01net)

[4] Quelques exceptions notables cependant, parmi lesquelles les secteurs de l'aéronautique, du nucléaire ou de la défense qui ont imposé pour leur informatique une démarche industrielle de la qualité.

[5] Clause « AS IS » ou bien du type « WITH ALL FAULTS »... dans les contrats américains.

[6] Pour une approche plus orientée « logiciel spécifique » et contrat d'entreprise, lire l'article : «Le traitement juridique des failles de sécurité», par Me d'Haultfoeuille - http://www.journaldunet.com/juridique/juridique051108.shtml

[7] ..., à l'exclusion de la responsabilité civile délictuelle qui est d'ordre public en France et ne peut être écartée par une stipulation entre les parties.

[8] Incidemment, rappelons en effet que les clauses limitatives de responsabilité ne lient que les parties au contrat et elles ne pourraient empêcher d'agir le tiers qui a subi des dommages du fait du produit.

Marie Barel, mbarel@legalis.net
Juriste spécialisée TIC/PI/SSI

Article rédigé à la suite d'une conférence donnée à EUROSEC'05 : « Recherche de vulnérabilités et exploitation », **Marie Barel** et **Frédéric Raynal**.

ou de préservation de la loyauté et de l'équilibre des relations commerciales. Ainsi, en droit français, plusieurs fondements légaux peuvent permettre, en principe, de faire échec aux clauses qui nous intéressent ici.

(1)Limitations de portée générale

D'abord, pour être reconnues valides, les clauses limitatives ou élusives de responsabilité doivent avoir été légalement constituées et exécutées de bonne foi [10] (articles 1134 alinéa 3 et 113 code civil). Ensuite, selon une jurisprudence constante, ces clauses sont jugées en principe valables, sauf « en cas de dol [11] ou de faute lourde [12] » de la partie qui invoque le bénéfice de la clause (Cass.Civ. 1ère, 24 février 1993 – JCP.G.1993.II.22166). La Cour de Cassation n'admet pas non plus la validité des clauses limitatives de responsabilité dont l'effet est de plafonner l'indemnisation à un montant trop dérisoire. Enfin, conformément à la jurisprudence dite « Chronopost » [13], les juges sanctionnent également les clauses qui, en raison du manquement du débiteur à une obligation essentielle du contrat, ont pour effet de priver l'engagement de toute portée car le contrat n'a plus alors de raison d'être.

(2)Clauses abusives dans les contrats conclus avec un consommateur ou un « non professionnel »

Dans les contrats conclus entre professionnels et non professionnels [14] ou consommateurs, sont considérées abusives et dès lors réputées « non écrites », les clauses « *qui ont pour objet ou pour effet de créer, au détriment du non professionnel ou du consommateur, un déséquilibre significatif entre les droits et les obligations des parties au contrat* » (article 132-1 du Code de la consommation [15]). Ainsi, en matière de contrats proposés par les éditeurs ou distributeurs de logiciels ou progiciels destinés à l'utilisation sur micro-ordinateurs [16], la Commission des clauses abusives a émis une recommandation [17] par laquelle elle considère comme abusives les clauses qui excluent toute garantie sur le logiciel ou son support. Dès lors, la doctrine estime en général que « les chances de survie d'une clause limitative opposée à un consommateur sont très faibles ».

(3)Responsabilité de plein droit du fait des produits défectueux

Un autre texte, la directive du 25 juillet 1985 [18] sur la responsabilité du fait des produits défectueux, serait de nature à rendre responsable les éditeurs de programmes malgré la clause d'exonération insérée dans les contrats conclus avec des consommateurs. Il s'agit d'un texte impératif qui prévoit l'application d'un régime de responsabilité *sans faute* des fabricants et distributeurs dès lors qu'un produit fini cause des dommages corporels ou matériels du fait d'un défaut. Précisons qu'en application de l'arrêt C-52/00 du 25 avril 2002 de la Cour de justice des Communautés européennes, le projet de loi du 16 juin 2004 relatif à la garantie de la conformité du bien au contrat due par le vendeur au consommateur et à la responsabilité du fait des produits défectueux, prévoit une modification de certains des articles 1386-1 à 1386-18 CC. L'une de ces modifications, de caractère exclusivement technique, consiste à rétablir, conformément à la directive 85/374 susvisée, un seuil d'indemnisation des dommages réparables et dont le montant doit être fixé par décret à 500 € [19].

[9] Référence obligée étant donné que de nombreuses licences en matière de logiciel sont soumises au droit américain et contiennent souvent des disposition particulièrement favorables aux professionnels. Pour plus de précisions sur la validité des clauses limitatives de responsabilité en droit américain, voir : Sabine Lipovetski, Les clauses limitatives de responsabilité et de garantie dans les contrats informatiques – Approche comparative France/Etats-Unis, Expertises, mai 2000 p. 143, http://www.kahnlaw.com/usa/newsjob/publications/clauses_limitatives_sl.htm.

[10] Ce qui n'est pas le cas par exemple lorsque l'inexécution totale ou partielle des engagements pris dans le contrat est délibérée.

[11] Par dol, on entend une tromperie ou toute manœuvre employée de manière à induire une personne en erreur en vue d'obtenir son consentement. (Source : Lexique des termes juridiques, éditions Dalloz)

[12] La faute lourde est considérée constituée par « un comportement d'une extrême gravité confinant au dol et dénotant l'inaptitude du débiteur de l'obligation à l'accomplissement de la mission contractuelle ». (Source : idem)

[13] Cass.Com., 22 octobre 1996 – JCP.G.1997.II.22881, note D. Cohen

[14] Exclut les contrats conclus entre professionnels de « même spécialité » ou encore les contrats qui ont un « rapport direct avec l'activité professionnelle exercée par le co-contractant, même en dehors de sa sphère de compétence ». En revanche, un utilisateur averti (un informaticien par exemple) achetant un produit logiciel pour ses besoins personnels peut être admis au bénéfice de la législation sur les clauses abusives en qualité de « consommateur ».

[15] Article qui a été inséré par la loi du 1er février 1995, transposant en droit français la directive 93/13/CEE du 5 avril 1993 (JOCE 21 avril 1993, n°L95) concernant les clauses abusives dans les contrats conclus avec les consommateurs.

[16] Les logiciels ainsi visés sont seulement les « logiciels standards » et non les logiciels spécifiques, réalisés suivant des spécifications client et destinés en règle générale à des professionnels…

[17] Recommandation n°95-02 adoptée le 7 avril 1995 : http://www.clauses-abusives.fr/recom/95r02.htm

[18] Transposée en droit français par la loi du 19 mai 1988 dont sont issus les articles 1386-1 à 1386-18 du code civil.

[19] Lire le texte : http://www.senat.fr/leg/pjl03-358.html.

Cependant, bien que la Commission européenne ait déclaré que la directive s'appliquait aux logiciels, le champ d'application concret de ces dispositions paraît plutôt ténu en la matière. En effet, il a été précisé à l'occasion d'une réponse ministérielle [20], que « *les seuls dommages dont ladite loi assure la réparation sont les atteintes physiques à la personne et les dommages matériels causés aux biens. L'application de ce texte aux logiciels ne vise donc que les situations où ceux-ci seraient à l'origine directe d'une atteinte à la sécurité des personnes ou des biens, hypothèses pour le moins résiduelles*». Or, même dans les situations où un logiciel est intégré dans un système dont dépend directement la sécurité des personnes (instruments de bord dans un avion, appareils de contrôle sur un site industriel Seveso, ...), il est vraisemblable, ainsi que le soulignent plusieurs auteurs, que les responsables de la partie logicielle pourraient invoquer le bénéfice de l'article 1386-11 dernier alinéa, qui stipule que « *le producteur de la partie composante n'est pas non plus responsable s'il établit que le défaut est imputable à la conception du produit dans lequel cette partie a été incorporée ou aux instructions données par le producteur de ce produit* ».

Remarque finale : nous ne développerons pas ici le principe de garantie des vices cachés qui existe dans les contrats de vente et où les clauses limitatives de responsabilité ne sont valables qu'entre professionnels « de même spécialité ». En effet, la nature juridique des contrats de licence d'utilisation d'un logiciel nous paraît (comme à d'autres auteurs spécialistes du sujet) incompatible avec la qualification de contrat de vente et ne permettrait donc pas l'application de cette garantie [21].

En définitive, les fondements légaux existent bien, et pas seulement ceux de nature à faire échec spécifiquement aux clauses limitatives de responsabilité. Par exemple, les « consommateurs de logiciels » pourraient également agir, suivant les cas, sur le fondement du manquement à l'obligation d'information, de la publicité mensongère [22], de l'obligation de délivrance conforme [23] ou bien encore de la tromperie [24] ... Aussi la faiblesse du contentieux tendant à faire reconnaître la responsabilité des fournisseurs/éditeurs quant à la fiabilité et la sécurité de leurs programmes ne trouve pas vraiment d'explication dans une prétendue insuffisance de la loi et serait plutôt à rechercher du côté de la difficulté de la preuve [25], du seuil de juridicité ou de l'intérêt à agir. Gageons [26] dès lors que, si elle parvient à surmonter les obstacles juridiques à son intégration dans l'ordre juridique français [27], la transposition des principes de la « *class action* » [28] en droit français, telle que souhaitée par le Président de la République, M. Jacques Chirac, à l'occasion de ses vœux aux forces vives le 4 janvier dernier, viendra relancer la dynamique de ces actions judiciaires...

2. Recherche et divulgation de vulnérabilités : quelles limites ?

On vient de l'envisager, la voie judiciaire se révèle tantôt inexploitée, tantôt difficile à faire prospérer de façon à forcer les fournisseurs/éditeurs de programmes à adopter de nouveaux standards de qualité. Aussi d'autres voies ont-elles été privilégiées, et en particulier une approche de la sécurité « par la transparence ». Cette démarche s'appuie sur la recherche (2.1) et la divulgation (2.2) de vulnérabilités, ce qui comporte là encore des risques juridiques importants et nécessite de prendre des précautions.

2.1 Recherche de vulnérabilités : à propos du « reverse légal »

D'abord, parmi les méthodes de recherche de vulnérabilités, l'audit de code occupe une place de choix, qu'il s'agisse d'analyser

[20] Rép. Min. n° 15677, JOANQ 24 août 1998.

[21] La jurisprudence sur ce sujet étant par ailleurs divisée et non constante...

[22] Ainsi, selon le procureur en charge du dossier dans l'affaire Viguard contre Guillermito, il eut été plus sage pour le prévenu d'adopter cette voie légale pour dénoncer les défauts du produit Viguard, présenté comme l'« anti-virus absolu »... tandis que les failles découvertes par le chercheur ont été qualifiées de « pertinentes » dans le rapport d'expertise. Pour un compte-rendu d'audience : http://maitre.eolas.online.fr/journal/index.php?2005/01/05/37-affaire-guillermito-compte-rendu-daudience

[23] Notons à cet égard que les consommateurs disposent, depuis l'ordonnance du 17 février 2005, d'une nouvelle action en « garantie de conformité du bien au contrat ». Pour plus d'informations, voir par exemple : Garantie de conformité du bien, par le cabinet Chamaillard - http://www.village-justice.com/articles/Garantie-conformite-bien-contrat,1564.html ; texte de l'ordonnance à l'adresse : http://www.legifrance.gouv.fr/WAspad/UnTexteDeJorf?numjo=JUSX0500005R

[24] C'est sur ce fondement en particulier que plusieurs affaires en matière de « CD anti-copie » ont abouti à la condamnation de sociétés de production musicale... Voir : http://www.clic-droit.com/web/editorial/article.php?art_id=302

[25] Preuve par exemple, s'agissant d'un composant logiciel intégré dans un système complexe, que le dysfonctionnement provient effectivement de ce programme ou pour le cas d'un progiciel, preuve de la non-conformité de la livraison en l'absence de cahier des charges pouvant servir de référence...

[26] Difficile d'être vraiment affirmative à ce stade. En effet, on peut noter qu'il existe déjà en droit français une action réservée aux associations de consommateurs pour la suppression des clauses abusives dans les contrats avec des professionnels (article 421-6 du Code de la Consommation). Appliquée avec succès dans d'autres domaines (ex. laboratoires de développement photos se dégageant de toute responsabilité en cas de perte de la pellicule ; établissement de crédit s'octroyant le droit d'augmenter discrétionnairement ses primes dans les contrats d'assurance chômage), force est de s'interroger pourquoi les associations n'ont pas fait preuve de la même pugnacité à l'encontre des éditeurs de logiciels ?...

[27] Notons que, sans même attendre les conclusions du groupe de travail formé par le Ministère de le Justice (rapport à venir très prochainement, annoncé initialement pour le mois d'octobre 2005), une plateforme en ligne « de gestion de procédure avec intervenants multiples ») a déjà été réalisée pour permettre le mise en relation de groupes de plaignants avec des avocats...
Indépendamment de la question de l'intégration possible des class action en droit français, ce site a déjà subi deux revers judiciaires (pour plus de détails, actualité du 9/12/2005 sur http://www.legalis.net/).

[28] Selon ses promoteurs, ce type d'action (qui existe en droit américain et été utilisée avec succès notamment pour combattre les pratiques anti-concurrentielles de la société Microsoft) doit « permettre à des groupes de consommateurs et à leurs associations d'intenter des actions collectives contre les pratiques abusives observées sur certains marchés».Notons cependant que le *recours collectif* déposé en 2003 en Californie *à l'encontre de la société Microsoft* et *visant à voir sa responsabilité civile engagée du fait des vices affectant la sécurité de ses logiciels* et en particulier leur sensibilité aux infections virales - l'action ayant été introduite après les ravages provoqués au mois d'août par la propagation du virus SoBig F et du ver MSBlaster (LoveSan), n'a pas pu prospéré, comme pouvaient l'augurer plusieurs avocats américains spécialisés en ce domaine. Sur ce sujet, lire notamment : /A legal fix for software flaws ?/ par Declan Mc Cullagh - http://news.com.com/2100-1002_3-5067873.html. Texte du recours introduit devant la Cour suprême de Californie : http://www.computerbytesman.com/security/hamilton_v_microsoft_complaint.htm

le code source d'un programme « ouvert » ou encore le code décompilé d'une application dont les sources ne sont pas disponibles. Cependant, la rétroconception de programmes ou « *reverse engeneering* » peut s'entendre plus largement comme « l'analyse d'un système ou d'une solution technique afin d'en retrouver les principes de conception, dans le but de résoudre une difficulté ou de développer une solution plus performante » [29]. Ainsi, procède de l'ingénierie inverse non seulement la technique d'analyse de code source et binaire (on est alors dans une approche de type « boîte blanche »), telle qu'envisagée ci-dessus, mais aussi celle dite « d'injection de fautes » (approche de type « boîte noire »), les outils de recherche du rétroconcepteur étant à la fois un désassembleur (exemple incontournable : IDA Pro de la société Datarescue [30]), un déboggeur [31] et des logiciels de surveillance système [32]. Considérant que, suivant les finalités recherchées et les contextes d'application, on peut prétendre à un recours légitime aux techniques concernées, dans quels cas peut-on ainsi « regarder à l'intérieur d'un logiciel » ?

– La décompilation à des fins d'interopérabilité

En droit français, c'est la loi du 10 mai 1994 sur la protection juridique des programmes d'ordinateur qui a introduit au bénéfice de l'utilisateur un « droit » de décompilation (les guillemets semblent plutôt appropriés ici car la décompilation n'est pas en réalité autorisée de plein droit – cf art. L.122.6-1.IV. 2 du Code de la Propriété intellectuelle – CPI). Le droit de reproduire, sans l'autorisation de l'auteur [33], le code du logiciel ou « d'en traduire la forme » [34] est donc enfermé dans des limites strictement définies. En particulier, les actes de décompilation doivent avoir pour seule finalité « *l'interopérabilité* d'un logiciel créé de façon indépendante *avec d'autres logiciels* » et ils ne doivent porter que sur les « *seules parties du logiciel d'origine nécessaires à l'interopérabilité* » ... L'exercice du « droit de décompilation » en dehors des limites prévues par la loi [35] est constitutif d'un acte de contrefaçon.

A l'évidence, l'audit de code décompilé (ou désassemblé) en vue de la recherche de vulnérabilités ne ressort nullement de l'objectif imparti par le législateur. Dès lors, faut-il en conclure que la rétroconception de programmes n'est pas possible en dehors de la finalité d'interopérabilité ?

– A des fins de maintenance ou d'adaptation du logiciel : une étape potentiellement nécessaire

Sauf lorsque l'auteur s'en est réservé le droit (ce qu'il ne manque jamais de faire en pratique [36], et excepté bien sûr pour les développements obéissant au modèle du « libre » et autres modèles voisins [37]), l'utilisateur légitime d'un logiciel peut en principe, si nécessaire, « reverser » un logiciel pour les besoins de sa maintenance. Le droit (on devrait plutôt écrire ici, la « faculté ») de correction accordé à l'utilisateur par l'article L.122-6-1, (I.) CPI peut donc justifier la décompilation du code (ou son désassemblage), en vue par exemple de « déboguer » le programme ... La doctrine considère également que le droit d'adaptation peut permettre à l'utilisateur légitime de faire évoluer un logiciel (toutefois avec la même réserve pratique que pour le « droit » de correction), ce qui pourrait l'autoriser dès lors à accéder aux sources d'un logiciel, par exemple pour l'adapter à une évolution de la réglementation administrative ou fiscale et permettre ainsi que l'utilisateur continue de disposer d'une utilisation conforme à sa destination [38].

– En vertu du « droit de comprendre »

Ensuite, il existe bien, en matière de logiciel, un « droit de comprendre » les idées et principes de base qui sous-tendent n'importe quel élément d'un programme. Ce droit, qui a été reconnu en même temps que le « droit de décompilation » susvisé, est celui prévu à l'article L.122-6-1 (III.) du CPI. Celui-ci confère au titulaire de la licence le droit (sans l'autorisation de l'auteur) « *d'observer, d'étudier et de tester le fonctionnement* » d'un logiciel.

Ce droit doit s'exercer dans la limite des opérations « *de chargement, d'affichage, d'exécution, de transmission ou de stockage du logiciel que [l'utilisateur] est en droit d'effectuer* ».

[29] Définition extraite du Lefebvre Informatique et télécoms, sous la direction d'Alain Bensoussan.

[30] Pour une introduction aux fonctionnalités d'IDAPro, lire : Nicolas Brulez, Introduction au Reverse Engeneering – IDA, l'arme absolue pour l'analyse de code, MISC 14 (juillet-août 2004)

[31] Permet d'étudier pas à pas le comportement d'un programme pendant son exécution.

[32] Ces outils vont permettre de surveiller, en temps réel, l'activité de la base de registres ou l'activité réseau, et donc d'avoir une information externe au programme et d'identifier les interactions qu'un programme peut avoir avec son environnement de fonctionnement. Exemple du logiciel Monitor de la société Sysinternals cité dans l'article de M. Serge Lefranc, La rétroconception : application à l'analyse logicielle – Actes du symposium SSTIC'03

[33] Celui-ci n'ayant pas fourni les informations suffisantes à l'interopérabilité : art. L. 122.6-1.IV. 2° précité.

[34] La formule est en réalité suffisamment générique pour viser non seulement les actes techniques de décompilation au sens strict : traduction du code binaire ou code machine en code source (langage haut niveau), mais aussi plus largement toute traduction du code binaire en langage plus bas niveau, proche du code machine, du type ASM (assembleur)... Ainsi, contrairement à ce qui est suggéré dans le rapport d'audience du procès Guillermito (Cf. op. cit.), le désassemblage de code est tout aussi répréhensible (au sens de la loi) que la décompilation et tous ces actes sont en général expressément interdits dans les contrats de licence des logiciels commerciaux.

[35] Pour plus de détails, voir : Décompilation, interfaçage et interopérabilité, par Xavier Linant de Bellefonds – Semaine Juridique, n° 12 du 18 mars 1998

[36] Là encore, on voit bien, comme en matière de copie privée, qu'il s'agit seulement d'une exception à un droit (celui de l'auteur), et non à proprement d'un « droit de correction » de l'utilisateur ! Deux optiques somme toute fort divergentes... - :)

[37] Par exemple : Creative Commons - http://www.creativecommons.org - et IANG (IANG Ain't No Gnu) – http://iang.fr/info/index.php.

[38] Sur ce point de vue, lire Mes Jérôme Perlemuter et Paul Hebert (Cabinet Salans Hertzfeld & Heilbronn : L'utilisateur d'un logiciel peut-il accéder aux codes sources ? – http://www.jdn.com/juridique/juridique020409.shtml. En particulier, sur la définition de l'usage conforme à la destination du logiciel, les auteurs rappellent que : « faute de définition légale de la notion de «destination», la doctrine précise que le logiciel est conforme à sa destination lorsqu'il est capable de traiter les informations en vertu des «fonctionnalités voulues» par l'auteur et programmées comme telles. »

La méthode d'injection de fautes, qui permet éventuellement de déduire les instructions mêmes du code d'un programme, entre-t-elle dans le champ de cette exception ? Sans doute partiellement. Ainsi la technique de « fuzzing » (on parle aussi de *stress testing*), utilisée lors de la recherche de vulnérabilités sur des applications en *closed source*, consiste, par exemple, à « *générer de façon automatique, des chaînes de caractères de tailles importantes et croissantes. Ces chaînes sont ensuite utilisées pour tester différents protocoles (FTP, HTTP, SMTP, etc.) et pour permettre de découvrir rapidement si les applications testées contiennent des débordements de buffer* ». [39] Dans ce cas, il semble bien que les limites du droit d'observer, d'étudier et tester le fonctionnement du logiciel soient respectées et que la technique soit autorisée.

Rien n'est moins sûr dans d'autres cas : ainsi, lorsque l'opération envisagée consisterait à tenter de faire lire et exécuter au programme un type de fichier pour lequel il n'est pas conçu en principe (ex. : injecter un fichier de format *vidéo* sur un lecteur de fichiers *audio* du type MP3 [40]), le « droit de comprendre » se heurterait dans cette hypothèse à une autre limitation : l'« usage conforme » du logiciel, c'est-à-dire la destination du logiciel stipulée dans le contrat de licence. Dès lors, le droit d'observer, d'étudier et de tester le fonctionnement du logiciel ne porterait que sur les éléments non protégés par le droit d'auteur, ce qui, ramené à l'essentiel, conduisait Michel Vivant à considérer que ce texte permet simplement de « regarder … ce qu'il est permis de voir » [41]. En effet, il est permis de se montrer réservé sur la portée d'une telle disposition sachant d'une part, que la reprise des idées et principes d'un programme est « de libre parcours » (sauf à entrer sur le terrain des agissements parasitaires et de la concurrence déloyale) et que, d'autre part, la seule analyse, sans acte de reproduction, d'un logiciel, ne peut en elle-même être constitutive d'un acte de contrefaçon…

– En vertu du « droit à l'analyse critique »

Comme maître Iteanu dans sa plaidoirie en faveur du prévenu « Guillermito » [42] (voir encadré *Actualités*), on peut déplorer néanmoins que la précédente disposition n'ait jamais reçu aucune application devant les tribunaux, car couplée avec **l'exception d'analyse critique**, elle pourrait conforter la défense des auteurs de la presse informatique spécialisée qui dénoncent régulièrement des défauts de sécurité de certains programmes.

Ainsi, suivant l'article L.122-5 du CPI : « Lorsque l'œuvre a été divulguée, l'auteur ne peut interdire :

3° Sous réserve que soient clairement indiqués le nom de l'auteur et la source :

a) les analyses et courtes citations justifiées par le caractère critique, polémique, pédagogique, scientifique ou d'information de l'œuvre à laquelle elles sont incorporées. »

Nonobstant ce droit, il est important pour le journaliste ou le pigiste faisant usage de cette liberté d'expression, de bénéficier de la protection bienveillante (et pas simplement velléitaire) d'un directeur de publication, pour se trouver encore un peu plus à l'abri de revers judiciaires à l'occasion de la divulgation de vulnérabilités exploitables (cf *infra*).

Actualités - L'affaire « Guillermito » en quelques mots

En résumé : Guillaume Tena est poursuivi par l'éditeur Tegam International pour avoir rendu publique une partie du code du logiciel antivirus Viguard, une publication réalisée dans le cadre d'un test tendant à mettre en exergue des faiblesses de sécurité du programme.

Fondements de la plainte : contrefaçon par reproduction, décompilation et diffusion gratuite non autorisées.

Réquisitions du Procureur de la République **à l'audience du 4 janvier 2005 :** 4 mois de prison avec sursis et 6.000 euros d'amende (auxquelles s'ajoute la demande en dommages-intérêts de la partie civile, à hauteur de 900.000 euros !).

Délibéré du 8 mars 2005 : 5.000 euros d'amende avec sursis et aucune inscription au casier judiciaire.

Jugement sur les intérêts civils du 7 juin 2005 **(malgré l'appel de la décision au pénal !*) :** 14.300 euros de dommages-intérêts et frais de justice de la partie civile.

Leçon de cette affaire ? : présentée par beaucoup comme la décision qui pèsera sur l'avenir du *full disclosure* en France, le jugement au fond du procès Guillermito est resté sur le terrain de la contrefaçon, malgré les tentatives de la défense de déplacer le débat sur celui de la primauté de la liberté d'expression sur les droits de propriété intellectuelle. Bref, une montagne (médiatique) qui accouche d'une souris **…

(*) Selon l'adage en effet, *« le pénal tient le civil en l'état »* et donc le tribunal aurait dû surseoir à statuer. A défaut, si la cour relaxe Guillermito en appel, il y aura contrariété entre le jugement sur intérêt civil et l'arrêt de la cour ; pour autant, il appartiendra à Guillermito de faire appel de la décision au civil…

(**) *« Il est vrai que les conséquences sont nulles pour Guillaume Tena »* (commentaire de Me Olivier Iteanu au sortir du verdict rendu le 8 mars dernier)… Sur le pénal certes, mais pas sur le plan civil … du moins jusqu'à la décision en appel (action pendante au jour de la présente publication) !

09/2005. - Dans une autre affaire, qui rappelle les méthodes de démonstration technique utilisées en son temps par Serge Humpich (condamné pour introduction frauduleuse dans un système et contrefaçon de cartes bancaires), un récent dépôt de plainte pour contrefaçon et escroquerie en bande organisée vise cette fois deux informaticiens qui dénoncent des « trous dans la carte Vitale » : http://news.tf1.fr/news/multimedia/0,,3246224,00.html.

[39] Recherche de vulnérabilités par désassemblage, Nicolas Brulez – MISC 12

[40] Attention, l'exemple cité est purement hypothétique et sans doute pas vraiment réaliste. Il a simplement valeur d'illustration quant à l'idée d' «usage (non) conforme». Tester un programme en lui soumettant un fichier vidéo à la place d'un fichier audio présenterait en pratique peu d'intérêt car, a priori, on imagine mal que cela puisse marcher et produire un résultat exploitable ! En ce sens, voir plus loin dans le dossier de la présente édition, l'article de Thorsten Holz et Ilja van Sprundel, Recherche de vulnérabilités à l'aide du fuzzing – MISC 23, p.34 : « Si l'on envoyait au système des données totalement fantaisistes, il rejetterait certainement ces entrées dès qu'il remarquerait qu'elles ne correspondent pas à ce qui est attendu. Nous choisissons plutôt des données semi-valides, qu'un parser ne refusera pas tout de suite, (…)». Tester un programme en lui soumettant un fichier vidéo à la place d'un fichier audio présenterait en pratique peu d'intérêt car, a priori, on imagine mal que cela puisse marcher et produire un résultat exploitable ! :)

[41] Voir Logiciel 94 : tout un programme ?, Michel Vivant, JCP 1994.I.3792.

[42] Pour un compte-rendu d'audience, voir : http://bricablog.net/index.php/2005/01/04/359-viguard-cest-de-la-daube ; http://maitre.eolas.online.fr/journal/index.php?2005/01/05/37-affaire-guillermito-compte-rendu-daudience

- En vue d'évaluer la dangerosité et la nocivité des programmes malveillants (malwares) et de déterminer les moyens de les neutraliser : l'activité de recherche en sécurité informatique

Concernant plus spécifiquement le domaine de la sécurité informatique, il est courant pour les professionnels de la sécurité (éditeurs d'anti-virus, fournisseurs de solutions de détection d'intrusion, CERTs, etc.) d'avoir recours à des techniques d'audit de code pour évaluer la dangerosité et la nocivité de programmes malveillants (*malwares*) dont les sources ne sont pas disponibles (et dans ce cas, ils utilisent généralement la technique du désassemblage). Les finalités poursuivies sont diverses : au-delà du débogage et du développement de correctifs, on peut citer l'analyse de binaires compilés sur une machine compromise comme la neutralisation d'un programme malveillant (ver ou virus) se propageant sur un réseau [43] ... Dans ces hypothèses, il est permis de penser que les programmes concernés (*exploit*, virus, vers, etc.) ayant une cause illicite (permettre la commission d'une infraction pénale : accès frauduleux à un système, atteinte aux données ou encore entrave au fonctionnement d'un système), ils ne peuvent prétendre à la protection par le droit d'auteur [44]. En pratique, on voit mal d'ailleurs l'auteur d'un programme malveillant attrait devant les tribunaux se défendre en opposant que la victime a violé ses droits d'auteurs sur le programme ! :-)

Ainsi [45], la protection des systèmes d'information contre les risques d'infection virale ou de propagation de programmes malveillants doit permettre de justifier une atteinte au droit *théorique* de leurs auteurs. La défense des réseaux et systèmes d'information [46] constitue, que ce soit dans un cadre professionnel ou privé, un « motif légitime » qui doit permettre de recourir aux techniques de recherche de vulnérabilités, selon les cas, soit en réaction à un incident de sécurité, soit à l'occasion d'une « anomalie » révélée au cours des opérations d'audit ou d'administration du système. Dans tous les cas, le recours à ces techniques, et notamment aux techniques d'audit de code, doit être strictement nécessaire au but poursuivi : en l'occurrence, la neutralisation ou la mitigation de risques *avérés* [47] et l'endiguement des effets dommageables du *malware* sur le système cible (atteinte au secret ou à la confidentialité, atteinte à la vie privée, etc.).

- Pour « l'inspiration » : le reverse comme méthode d'ingénierie

En dernier lieu, comme l'indique la définition citée plus haut, le « *reverse engeneering* » reste encore légal lorsque l'analyse du système ou de la solution technique sert à en retrouver les principes de conception « *dans le but de développer une solution plus performante* ». En effet, un principe du droit d'auteur est que « *les idées restent de libre parcours* »... Lorsqu'elle est ainsi utilisée pour accéder à moindre frais à un savoir ou une technologie, les limites qui s'appliquent à cette forme d'« ingénierie » résident alors dans la définition de la frontière du plagiat avec la contrefaçon... Cependant, nous ne développerons pas davantage ce point qui nous éloigne de notre sujet principal, qui est la recherche de

En résumé, le reverse « légal » ...

- Interopérabilité : le droit de décompilation (122-6-1 IV CPI)
  - De nombreuses conditions difficiles à mettre en œuvre
  - Ne peut servir à la recherche de vulnérabilités
- Correction d'erreurs et évolution du logiciel : le droit de correction et d'adaptation (122-6-1 I CPI)
  - La décompilation du code comme étape nécessaire au débogage
  - L'accès aux sources pour permettre un usage conforme à la « destination du logiciel »
  - Dans la pratique, droit de réserve toujours exercé par l'auteur (voir conditions de la licence pour les logiciels propriétaires)
- Analyse
  - Le droit à l'analyse et de courte citation (art. L. 122-5 CPI) :
    - Caractère critique, polémique, pédagogique, scientifique ou d'information de l'œuvre analysée ou citée
    - L'œuvre analysée ou citée est utilisée comme support à une discussion et sert de base à un commentaire critique et des appréciations personnelles de l'analyste
    - Une disposition qui n'a jamais été mise à l'épreuve des tribunaux dans ce domaine
  - Le droit de comprendre (art. L.122-6-1 III CPI) :
    - Champ d'application restreint : permet seulement d'« observer ce qui est permis de voir ».
    - Ne permet pas le recours aux techniques de décompilation ou de désassemblage pour l'audit de code ; pourrait être utilisé dans une approche de type « boîte noire »
- Protection du système d'information
  - Opérations strictement nécessaires à l'objectif poursuivi : au-delà du débogage et du développements de correctifs, neutralisation d'un programme malveillant (*malware*) se propageant sur un système (ex. : ver, virus ...)
  - En réaction à un incident de sécurité (risque avéré) ou à l'occasion d'une « anomalie » révélée au cours des opérations d'audit ou d'administration du système
- Inspiration
  - Les idées et principes sont de libre parcours
  - Distinction entre les éléments protégeables d'un logiciel et ses éléments non protégeables
  - Hors du champ de recherche de vulnérabilités

[43] Opinions tirées de : L'analyse de binaires, Kostya Kortchinsky – Sécurité informatique, décembre 2004, page 6 : http://www.sg.cnrs.fr/fsd/securite-systemes/revues-pdf/num51.pdf

[44] De même qu'une obligation (un contrat) fondée sur une cause illicite est nulle (articles 1131 du Code civil) ; article 1133 C.civ. : « La cause est illicite quand elle est prohibée par la loi, quand elle est contraire aux bonnes mœurs ou à l'ordre public ».

[45] De même que la chambre sociale de la Cour de cassation a admis comme seule exception à la consultation de fichiers personnels sur le PC d'un salarié la justification d'un « risque ou évènement particulier » – Arrêt du 17 mai 2005, Philippe K. / Cathnet-Science

[46] Prise non pas au sens d'« active defense » mais bien de mesures de protection statiques (Cf L. Oudot – MISC 18, pp. 56-64) : mise à jour des bases de données de signatures des anti-virus et IDS, application de correctifs ...

[47] De simples « soupçons » seraient selon nous insuffisants dans ce cadre : en effet, certains voudraient prétendre que les utilisateurs pourraient recourir aux techniques de reverse pour lever leurs soupçons quant à la présence de portes dérobées dans les programmes ou bien encore pour vérifier que les briques de crypto implémentées dans les systèmes sont bien « saines » (exemple de Skype). Notre réponse sur ce sujet est que la confiance des utilisateurs dans les produits qu'ils installent ne peut légalement s'acquérir en procédant à un audit du code. Cette confiance est nécessairement donnée a priori, tout en suivant un principe général de précaution, celui des « reputable sources ».

vulnérabilités. En définitive, comme le montre le résumé ci-contre, il existe de nombreuses motivations à la rétroconception de programmes, qui sont elles-mêmes très dépendantes des différents acteurs qui vont la réaliser : entreprises, particuliers, groupes de pirates ou organisations criminelles [48] De même, chacune de ces catégories de personnes peut être diversement motivée à divulguer ses résultats concernant plus particulièrement les vulnérabilités découvertes.

2.2 Divulgation de vulnérabilités

Sécurité par la transparence, oui ...

La publication des failles de sécurité dans les systèmes informatiques et des vulnérabilités logicielles ou applicatives en général, procède de ce qu'on appelle « la sécurité par la transparence » dont on admet aujourd'hui couramment qu'elle augmente davantage la sécurité desdits systèmes que la politique inverse de « sécurité par l'obscurité » (*bug secrecy*).

En effet, la transparence représente une incitation forte pour les éditeurs de logiciels à fournir plus rapidement des correctifs de sécurité et à adopter des méthodes de développement sécurisées de façon à conserver une bonne image de marque et améliorer la qualité de ses produits.

Surtout, à travers le partage de l'information et la diffusion des connaissances, elle tend à rétablir dans le domaine de la lutte informatique, l'équilibre entre les professionnels de la sécurité d'une part et les délinquants informatiques d'autre part (« *White hats know what black hats know* » [49]).

Cette « victoire » n'est cependant pas sans connaître une rançon !

Ainsi la divulgation de failles de sécurité emporte à l'inverse certains inconvénients, au premier rang desquels on peut citer l'expansion du nombre des personnes susceptibles d'exploiter la vulnérabilité publiée, ouvrant ainsi dans la théorie du « *full disclosure* » exposée par Bruce Schneier [50] une « fenêtre d'opportunité » (*Window of Opportunity*) où le risque d'exploitation s'achève seulement après la publication et la mise en œuvre *effective* du correctif ou des mesures de contournement identifiées.

De plus, la divulgation de vulnérabilités s'accompagnant souvent de la diffusion d'*exploits*, la diminution du niveau d'expertise et de compétence requis explique le phénomène des « *script kiddies* [51] »...

Néanmoins, de l'avis de nombreux professionnels de la sécurité, l'approche par la transparence est une nécessité et les exploits du type *0-day*, qui permettent une exploitation silencieuse ou concomitante de la vulnérabilité par des programmes malveillants, sont pires encore que les risques de cette politique. Il n'en demeure pas moins que la démarche de publication doit s'entourer de nombreuses précautions.

... mais de manière responsable : démarche idéale et facteurs de risque ?

En effet, le risque juridique d'une action judiciaire introduite par un fournisseur est loin d'être théorique (exemple des affaires « Guillermito » et « Serge Humpich » déjà citées), la mauvaise publicité ainsi faite autour de ses produits pouvant déterminer l'éditeur de logiciel ou le fournisseur à agir, selon les cas, en contrefaçon [52], en accès frauduleux à un système, en dénigrement de produit, en violation de confidentialité (affaire « Cisco contre Mickaël Lynn » - BH US 2005 [53]) ou encore pour fourniture de moyens de piratage (article 323-3-1 nouveau du Code pénal [54], introduit par la LCEN)...

Dès lors, idéalement, la démarche de publication d'une vulnérabilité devrait toujours être accomplie moyennant information préalable et collaboration (bénévoles) avec l'éditeur et après la publication d'un correctif et d'un bulletin de sécurité par celui-ci.

S'éloignant plus ou moins de ce « processus idéal » [55], les autres scénarii de divulgation comprennent différents facteurs de risques qu'il convient d'envisager pour pondérer l'hypothèse d'un revers judiciaire. Parmi ces facteurs, on relèvera, de manière non exhaustive, les indices suivants :

- Publication de données permettant d'exploiter directement la vulnérabilité

[48] Pour une présentation détaillée de ces motivations, voir l'article de M. Lefranc (précité)

[49] Jennifer Stisa Granick, Legal risks of vulnerability disclosure – BlackHat, USA 2004. Présentation .PPT : http://www.blackhat.com/html/bh-media-archives/bh-archives-2004.html

[50] Full Disclosure and the Window of Exposure – [CRYTPO-GRAM], Nov. 15, 2001 : http://www.wounterpane.com/crypto-gram-0111.html#1

[51] Définition extraite de l'encyclopédie Webopedia : "A person, normally someone who is not technologically sophisticated, who randomly seeks out a specific weakness over the Internet in order to gain root access to a system without really understanding what it is s/he is exploiting because the weakness was discovered by someone else. A script kiddie is not looking to target specific information or a specific company but rather uses knowledge of a vulnerability to scan the entire Internet for a victim that possesses that vulnerability." Sur search Security.com : "The typical script kiddy uses existing and frequently well-known and easy-to-find techniques and programs or scripts to search for and exploit weaknesses in other computers on the Internet - often randomly and with little regard or perhaps even understanding of the potentially harmful consequences."

[52] La première précaution à prendre étant bien entendu d'être « utilisateur légitime » du produit testé pour pouvoir prétendre ensuite au bénéfice des « droits » de l'utilisateur (droit de décompilation, droit de « comprendre », droit de correction, droit d'analyse critique – Cf supra).

[53] http://www.securityfocus.com/news/11259

[54] Cette nouvelle infraction vise « le fait, sans motif légitime, d'importer, de détenir, d'offrir, de céder ou de mettre à disposition un équipement, un instrument, un programme informatique ou toute donnée conçus ou spécialement adaptés pour commettre une ou plusieurs des infractions prévues par (les articles 323-1 à 323-3). » Pour plus de détails, lire notre article paru dans [MISC 14] : Nouvel article 323-3-1 du code pénal : le cheval de Troie du législateur ?, pp.14-17.

[55] Pour plus de détails, lire : Jean-Baptiste Marchand, Vulnérabilités : de la découverte à l'exploitation, conférence présentée à Networld+Interop Paris 2004, session Vulnérabilités et gestion des correctifs de sécurité – Disponible sur le site HSC, Hervé Schauer consultants : http://www.hsc.fr/ressources/presentations/ni04-vuln/index.html.fr

- Publication de données techniques détaillées permettant de reproduire la vulnérabilité
- Faible/haut niveau d'expertise requis
- Système/environnement/logiciel standard (répandu) ou plutôt exotique ou complexe
- Nombre d'utilisateurs potentiellement impactés
- Criticité ou sévérité de la vulnérabilité
- Publication avec/sans « solutions de contournement » (palliatifs)
- Publication restreinte ou au contraire très large (mesure d'audience forums/web/..., nombre de tirages presse papier)
- Public visé : grand public / professionnels, communauté scientifique ...
- Publication précédée de l'information de l'éditeur / avec ou sans délai [56] / avec ou sans assistance / à titre gratuit ou onéreux
- Publication à titre individuel (« white hat ») / professionnels (laboratoires de recherche, département R&D, presse spécialisée ...)
- Publication par le biais d'un réseau / instance professionnelle (par l'intermédiaire d'un CERT [57] par exemple).
- (...)

A ce jour, conscients des différents enjeux et respectueux des intérêts de chacun, les professionnels de la sécurité comme les éditeurs semblent avoir mis en place les conditions d'une diffusion responsable de l'information, adoptant pour les uns des politiques de divulgation empreintes à la fois d'obligations de coopération et de diligence [58], et organisant, pour les autres, les moyens de communication appropriés (bulletins de sécurité, hotline sécurité, adresse mail spécifique du type **secure@company.com** ...).

Conclusion

Cycles de développement trop courts, insuffisance des tests, impératif marketing de lancement du produit..., l'industrie informatique a produit à ce jour des milliards de lignes de code dont une part irréductible d'erreurs (*bugs*) qui engendrent quotidiennement des anomalies diverses dans les traitements effectués par les utilisateurs. Pour combattre l'insécurité logicielle et inciter les fournisseurs/éditeurs à prendre leurs responsabilités, l'adoption d'une démarche de « sécurité par la transparence » s'appuyant notamment sur les techniques d'analyse de code n'est pas la panacée, d'autant qu'elle se heurte à des contraintes juridiques importantes. Pire encore, l'écriture et la fourniture d'*exploits* pour faire la démonstration des vulnérabilités que l'on a identifiées, sont devenues au regard du droit l'antichambre de la cybercriminalité. Ainsi, en l'absence d'intention spécifique prévue par le législateur, la répression nouvelle des « attitudes d'amont » [59] semble faire encore plus reculer la limite de ce qui est possible de faire pour tenter de forcer les fournisseurs et éditeurs à améliorer la sécurité et la qualité de leurs produits.

En définitive, on peut affirmer que la loi ne peut être le substitut au génie logiciel (méthodes de développement sécurisé, phases de test, évaluation ...), tout au plus un « recours de la dernière chance ». De plus, le lecteur l'aura compris, la sécurité par la transparence ne doit pas être entendue comme un *full disclosure* radical : la divulgation se limitant à une critique non constructive et concourant à accélérer l'exploitation d'une vulnérabilité en abaissant notablement le niveau d'expertise requis est selon nous (juridiquement) inacceptable. Quant à la solution..., on peut noter que l'assurance tendrait à être désignée outre-Atlantique comme un vecteur de responsabilisation des éditeurs de logiciels, à travers l'effet conjugué de produits labellisés et de la variation des primes d'assurance. De même, l'obligation de report d'incident pour les organismes victimes et de signalement des failles de sécurité pour les éditeurs est réclamée par diverses organisations... Dans tous les cas, la solution à la problématique de la qualité et de la sécurité des logiciels n'est tout simplement pas à voie unique (et certainement pas de nature législative, le droit français étant suffisamment pourvu [60]) et nécessite l'implication de l'ensemble des acteurs dans ce domaine.

[56] Là encore, on peut penser à différents délais : délai laissé avant publication, délai pour corriger, délai laissé pour l'application des correctifs – cf. en ce sens : Upcoming Advisories d'eEye [http://www.eeye.com/html/research/upcoming/index.html] et l'approche préconisée par l'OIS (Organization for Internet Safety) : http://www.oisafety.com/adopters.html

[57] Il y a en France trois CERTs. Chacun dessert un secteur économique et coopère étroitement avec les autres en échangeant quotidiennement des informations sur l'actualité et les techniques. Ce sont :
- le CERT RENATER, pour le secteur Universités – Recherche : http://www.renater.fr/Securite/CERT_RENATER.htm
- le CERTA, pour le secteur des administrations : http://www.ssi.gouv.fr/rubriq/certa.html,
- le CERT-IST, pour les secteurs de l'industrie, des services et du tertiaire : http://www.cert-ist.com/.

[58] Voir par exemple la politique de divulgation d'iDEFENSE

[59] Référence à l'adoption de l'article 323-3-1 du code pénal dans le cadre de la Loi pour la Confiance dans l'Economie Numérique, du 21 juin 2004 (loi n°2004-575 ; JO n° 143 du 22 juin 2004 page 11168). En téléchargement sur : http://www.legifrance.gouv.fr/

[60] Les éditeurs doivent-ils être responsables des failles de sécurité?, par Lisa M. Bowman, CNET News, Christophe Guillemin - ZDNet France, 24 janvier 2002 : http://www.zdnet.fr/actualites/informatique/0,39040745,2103069,00.htm

Analyse différentielle de binaires : Application à la recherche de vulnérabilités

L'analyse des différences et des similarités de binaires est une problématique complexe en pleine expansion qui trouve de nombreux usages. Les applicatifs uniquement disponibles sous forme compilée tiennent une place considérable dans l'informatique actuelle : que ce soit le système d'exploitation le plus répandu au monde, les systèmes de messagerie ou de gestion de base de données les plus robustes ou simplement les vers et virus variés en perpétuelle propagation.

Dans ce contexte, des interrogations émergent quant aux modifications apportées par un éditeur ou constructeur à son produit lors de mises à jour (correctifs de sécurité comme nouvelles versions), mais aussi au niveau de l'utilisation de certaines portions de code (violation de la propriété intellectuelle, degré de ressemblance). Autant une réponse immédiate peut être fournie lorsque les sources des applicatifs sont disponibles, autant c'est délicat dans le cas contraire. Un recoupement octet à octet est inexploitable, une comparaison textuelle des lignes désassemblées ne prend pas en compte la logique du programme, bref, il fallait introduire un niveau d'abstraction supplémentaire.

La solution réside dans la représentation d'un binaire sous forme de graphes, facilitant à la fois l'analyse et l'interprétation des résultats : le problème de comparaison de binaires s'apparente alors à un sous-ensemble très restreint de celui de l'isomorphisme de graphes. Nous abordons dans cet article les méthodes actuelles d'analyse différentielle de binaires et les principes sur lesquels elles se fondent. Nous montrons ensuite l'application possible de ces procédés à la recherche de vulnérabilités, point des plus passionnants.

Pourquoi ne pas appliquer une comparaison textuelle ?

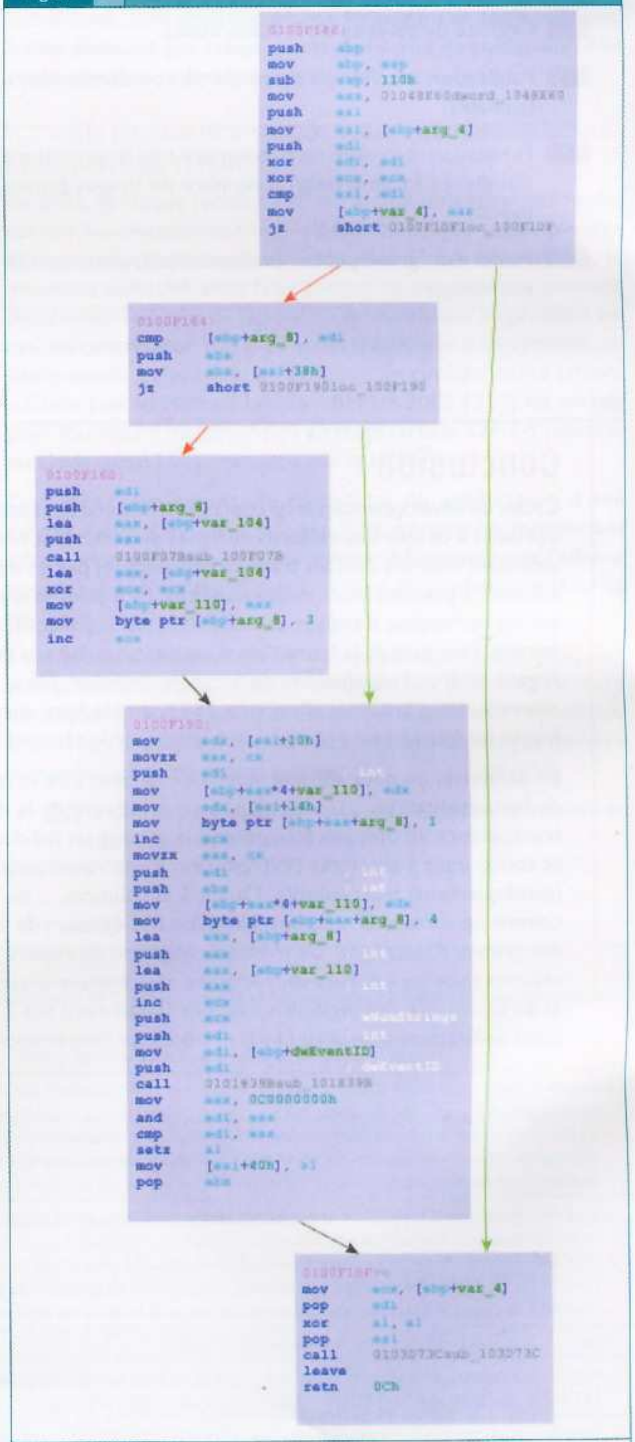
Comparer des variantes différentes du même objet exécutable (ou juste deux objets exécutables arbitraires partageant une certaine quantité de similarités) conduit à opposer des représentations peut-être complètement dissemblables au niveau assembleur. Il existe cependant un certain nombre de changements communs qui se produisent entre deux versions d'un objet exécutable :

L'allocation de registres différents
En fonction des paramètres d'optimisation du compilateur et des changements du code, des registres différents seront assignés à des instructions identiques ;

La réorganisation d'instructions
Compte tenu du modelage effectué par le compilateur pour le *pipelining* du CPU, certaines instructions individuelles seront réorganisées ;

L'inversion des branchements
Dans plusieurs cas, le compilateur essaie d'optimiser l'alignement

Figure 1



Flowchart d'une fonction

Kostya Kortchinsky
Ingénieur chercheur – EADS/CCR – kostya.kortchinsky@eads.net

Thomas Dullien
CEO and Head of Research – Sabre Security GmbH – thomas.dullien@sabre-security.com

des blocs basiques (que l'on définira comme étant une suite d'instructions consécutives exécutées linéairement, prenant généralement fin avec une instruction de branchement ou un retour d'appel) par une inversion de la condition d'un branchement et par un échange des deux blocs basiques auxquels ce branchement peut mener.

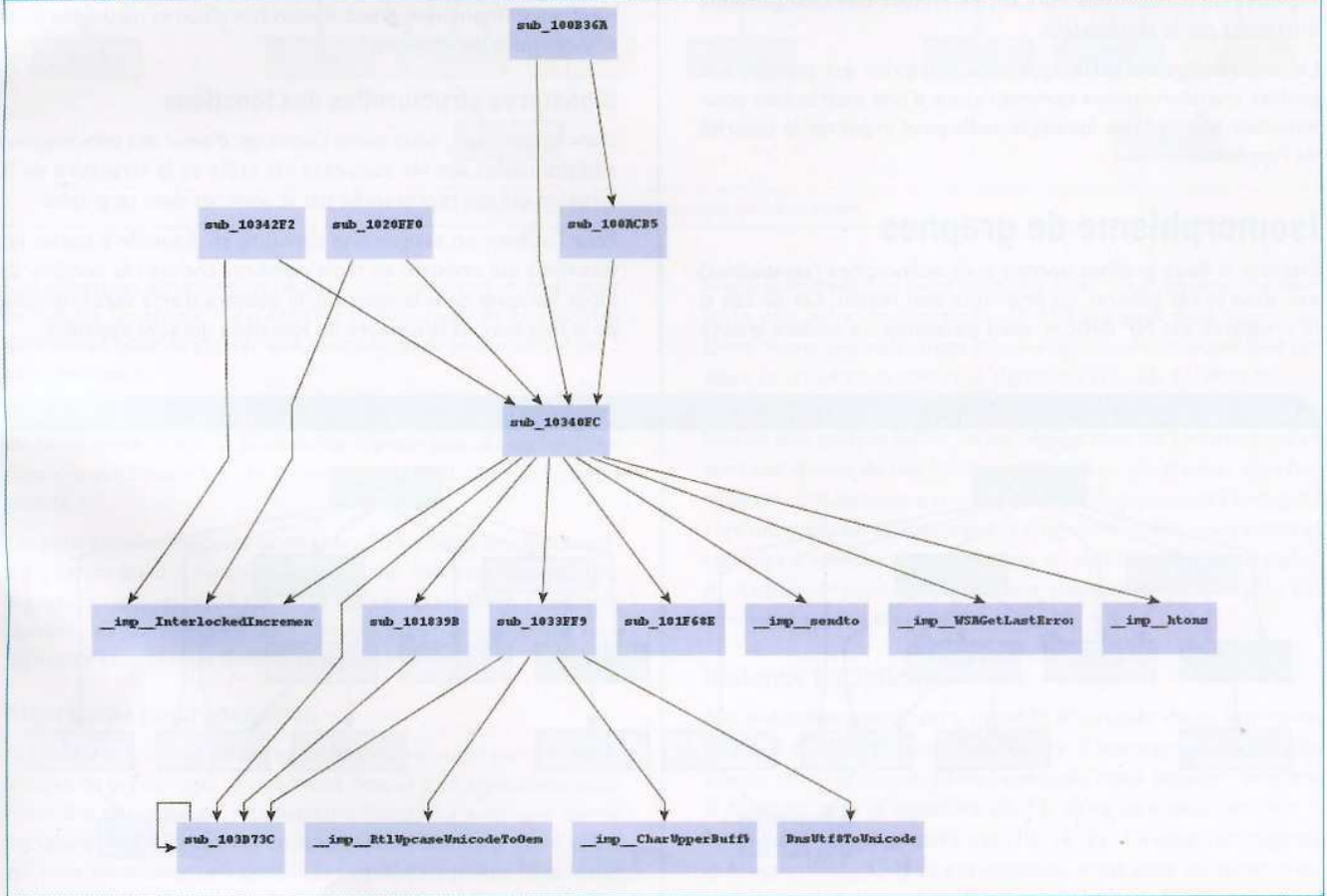
Évidement un nombre important de changements majeurs peut se produire. L'observation principale sur laquelle se fondent certaines méthodes présentées ici, est que le graphe d'appels (défini par la suite) d'un objet exécutable reste en majeure partie identique, même lorsqu'il est compilé avec un compilateur différent et pour une architecture différente. Au lieu de se cantonner aux instructions assembleur obtenues par désassemblage, une approche est de se concentrer sur les propriétés structurelles de l'objet exécutable, surtout sur l'abstraction essentielle des fonctions, des blocs basiques et aussi sur leurs relations.

Représentation des binaires sous forme de graphes

Généralement, on peut considérer un applicatif comme un graphe de graphes. Tout d'abord, un exécutable peut être vu comme un grand graphe dont les sommets sont les fonctions du logiciel. Si une fonction du binaire `'foo'` fait un appel à une fonction `'bar'`, un arc du sommet `'foo'` rejoindra le sommet `'bar'` dans ce graphe. Nous appelons ce graphe « le graphe d'appels » (aussi appelé *callgraph* par la suite). Nous pouvons ensuite aussi envisager chaque fonction du logiciel comme un graphe – que nous appellerons le *flowchart* du code de la fonction. Dans le flowchart, les sommets sont les blocs basiques, et les arcs sont les relations entre les blocs basiques, induites par les instructions de branchement, conditionnel ou non.

Afin d'expliciter davantage les deux types de graphes, nous vous proposons les exemples suivants dans les figures 1 & 2.

Figure 2



Callgraph d'un logiciel

L'idée d'une comparaison fondée sur les graphes d'un logiciel vient d'une observation simple, selon laquelle presque toutes les modifications appliquées au logiciel produisent des changements au niveau du callgraph ou du flowchart, parmi lesquels figurent :

L'ajout d'un contrôle de gamme
L'ajout d'un contrôle de gamme implique sur presque toutes les plateformes une comparaison supplémentaire, ainsi qu'une instruction de branchement en plus. Ces changements modifient le nombre de sommets et le nombre d'arcs dans le flowchart d'une fonction ;

Le remplacement d'un appel par un autre
Changer un appel à une fonction dangereuse (comme `strcpy()`) dans une fonction `'foo'` en un appel à une fonction différente (comme `strncpy()`) produit un changement dans le callgraph : un arc de `'foo'` à `'strcpy'` disparaît, et en même temps un arc de `'foo'` à `'strncpy'` apparaît.

Parallèlement, les changements d'optimisation du compilateur n'altèrent pas la structure essentielle du programme (avec quelques exceptions, voir ci-dessous) : remplacer des instructions par d'autres, inverser des branchements et réorganiser des instructions sont des opérations qui laissent les graphes du binaire invariants.

Il est évident qu'une comparaison fondée sur la structure du programme a bon nombre d'avantages : si on pouvait comparer les graphes des deux binaires, on pourrait voir les changements logiques du programme en s'affranchissant des changements introduits par le compilateur.

Le désavantage de cette approche est qu'on est aveugle aux petites transformations comme l'ajout d'une instruction pour initialiser une variable locale, laquelle peut impacter la sécurité de l'applicatif.

Isomorphisme de graphes

Décider si deux graphes donnés sont isomorphes (semblables) est, dans le cas général, un problème non résolu. On ne sait si le problème est NP difficile, mais personne n'a encore trouvé un algorithme de complexité polynomiale pour le solutionner. Heureusement, nous n'avons pas besoin de résoudre le problème général de l'isomorphisme de graphes : les logiciels consistent en des graphes orientés avec une structure claire et prédictible.

En plus, nous savons que les graphes ont changé et nous voulons simplement trouver l'emplacement exact où les graphes ne sont plus identiques, ce qui restreint le champ d'étude de façon conséquente.

Précisément, nous voulons construire un **isomorphisme optimal** de sous-ensembles de ces deux graphes. Cela signifie que nous voulons associer toutes les fonctions et blocs basiques qui n'ont pas changé.

Points fixes

L'algorithme que nous utilisons est un algorithme itératif pour construire une approximation d'isomorphisme : nous commençons avec les callgraphs. Dans ce graphe, il y a quelques sommets que nous pouvons aisément associer :

- Les deux sommets d'**entrée** du logiciel ;
- Les sommets des fonctions importées (comme `GetWindow`, etc.).

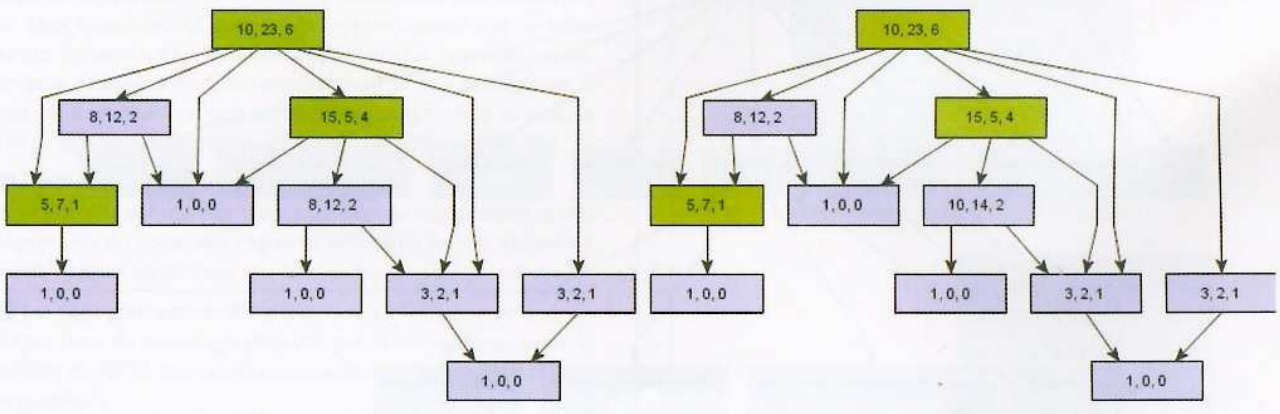
Nous allons appeler une paire de sommets que nous avons associés *fixed point*, ou point fixe de l'isomorphisme. En pratique, le nombre de sommets que nous pouvons initialement associer n'est pas suffisamment grand. Il nous faut d'autres méthodes pour associer plus de sommets.

Signatures structurelles des fonctions

Dans le callgraph, nous avons l'avantage d'avoir des informations additionnelles sur les sommets : la taille et la structure de la fonction qui est représentée par le sommet dans ce graphe.

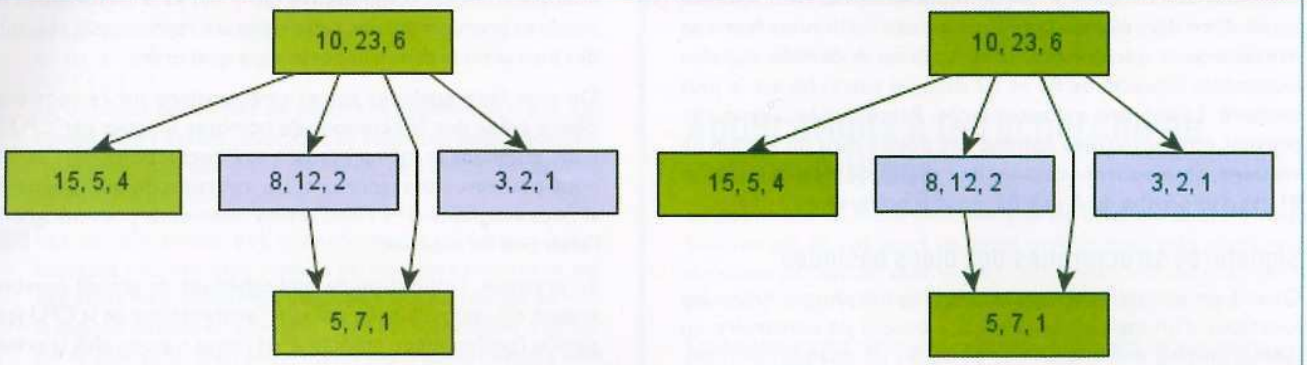
Pour l'utiliser, on assigne une signature structurelle à toutes les fonctions qui consiste en trois nombres entiers : le nombre de blocs basiques dans la fonction, le nombre d'arcs vers le graphe de la fonction, et le nombre de fonctions qui sont appelées.

Figure 3



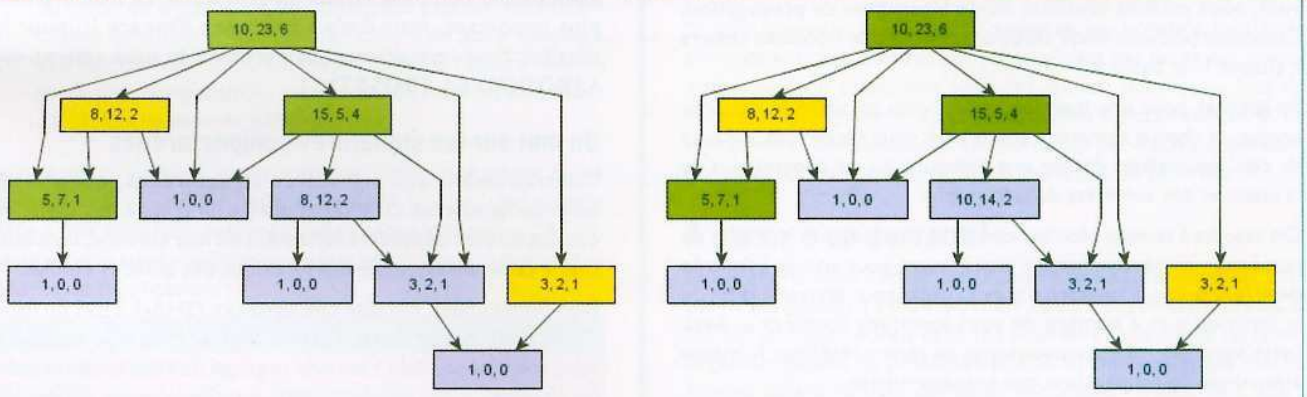
Signatures structurelles des fonctions

Figure 4



Analyse locale d'un binaire : graphe réduit

Figure 5



Analyse locale d'un binaire : sommets ajoutés au graphe

Ces signatures structurelles ne sont clairement pas uniques pour toutes les fonctions – il n'existe pas beaucoup de possibilités différentes pour organiser une fonction avec seulement 4 ou 5 blocs basiques.

Néanmoins, on gagne pas mal de points fixes : pour les fonctions de taille importante, la probabilité d'avoir plus d'une fonction dans chaque binaire avec la même signature est beaucoup moins grande.

On peut associer de cette façon entre 20% et 30% des fonctions : s'il n'existe qu'une fonction dans chacun des binaires avec une signature unique, nous pouvons les associer. Dans l'exemple suivant, les sommets en vert ont été associés parce que leur signature était unique dans chaque binaire.. (fig. 3)

Algorithme de propagation

N'avoir que 20% ou 30% d'associations ne suffit pas pour une analyse en profondeur : nous avons besoin d'un algorithme pour résoudre le problème de plusieurs fonctions avec une même signature. Nous avons cependant déjà un isomorphisme initial qui peut nous aider : en connaissant quelques points fixes, nous pouvons appliquer ces résultats, issus d'une analyse « globale » du logiciel, à une analyse « locale ».

Nous commençons avec un point fixe : c'est essentiellement juste une paire de fonctions (f1, f1') qui sont déjà associées. Dans notre exemple, nous commençons avec le point fixe des deux fonctions possédant la signature (10, 23, 6). Maintenant, nous ne regardons plus le binaire dans sa globalité, mais nous faisons une analyse locale : si on néglige tous les sommets qui ne sont pas voisins de nos fonctions associées, les graphes qu'il faut analyser sont beaucoup moins grands, et logiquement l'ambiguïté sera moindre. En se restreignant au graphe réduit, nous sommes capables d'associer deux sommets en plus dans le graphe global. A chaque itération, nous pouvons associer de plus en plus des sommets, jusqu'à finaliser l'analyse. (fig 4 & 5)

Distance Euclidienne

Nous désirons aussi être capable d'associer deux fonctions, même si elles ont changé, c'est-à-dire si leur signature n'est plus exactement identique. Dans l'exemple, nous pouvons voir que la fonction avec la signature (8, 12, 2) va être associée avec la fonction dont la signature est (10, 14, 2) : lorsque l'on regarde la fonction (15, 5, 4) et ses voisines, il est clair qu'après avoir associé les fonctions dont les signatures sont identiques, il n'y a plus d'autre choix !

Parfois la situation est beaucoup plus obscure, par exemple si deux sous-fonctions d'une fonction sont modifiées dans le même patch. Que devon-nous faire dans ce cas-là ? Il nous faut une **mesure** pour décider pour une fonction A donnée dans un exécutable laquelle, de B1 et B2 dans un autre, lui est le plus similaire. La solution est assez facile. Parce que les signatures peuvent être envisagées comme des points dans un espace de vecteurs, nous pouvons calculer leur **distance euclidienne** : si B1 est plus proche de A que B2, nous la préférerons.

Signatures structurelles des blocs basiques

Quand on est arrivé à construire l'isomorphisme entre les fonctions d'un logiciel, on a encore besoin de construire un isomorphisme entre les blocs basiques de chaque fonction. Le problème est similaire au problème de la construction de l'isomorphisme entre les callgraphs, mais malheureusement nous n'avons plus de signature structurelle évidente.

Puisque nous sommes paresseux (ou élégants selon le point de vue), nous voulons réutiliser notre algorithme de propagation. Comment pouvons-nous associer un triplet de nombres entiers à chaque bloc basique ?

En général, pour une fonction, nous avons un sommet d'entrée unique, et chaque sommet a soit un soit deux fils directs. A cause de cela, nous allons choisir une signature qui ne dépend plus de la position des sommets dans le graphe.

On associe à chaque bloc basique dans une fonction le triplet de nombres « nombre minimal d'arcs à suivre pour arriver à l'entrée de la fonction », « nombre d'arcs à suivre pour arriver à la fin de la fonction » et « nombre de sous-fonctions appelées ». Avec cette signature, nous pouvons plus ou moins réutiliser le même algorithme de propagation que précédemment.

Un inconvénient de notre signature est qu'elle est moins stable : l'insertion de quelques blocs basiques dans la fonction peut changer la signature d'autres blocs. Pour stabiliser la signature, nous utilisons d'autres informations pour la génération de points fixes :

- Références communes aux chaînes de caractères ;
- Références communes à des sous-fonctions ;
- Même SPP (voir ci-dessous)

Small primes product (SPP)

Le changement que le compilateur introduit le plus souvent est le réarrangement des instructions. Pour être capable d'associer deux blocs basiques qui sont identiques mais ont leurs instructions réarrangées, nous avons besoin d'une signature concise, qui peut identifier une séquence d'instructions, tout en étant invariante aux permutations d'instructions.

Quand la théorie des codes (comme la construction du code ISBN) essaie de construire un code qui détecte les changements, comme la permutation d'éléments, nous voulons construire un code simple qui détecte des changements de la séquence, mais pas les permutations :)

La solution est une fois de plus relativement immédiate : on assigne un petit nombre premier unique à chaque instruction assembleur. Pour calculer la signature d'un bloc basique, on calcule le produit de toutes les instructions dans ce bloc basique. Parce que la multiplication est commutative (ab = ba) et la factorisation en nombres premiers unique, cette signature représente la séquence des instructions dans n'importe dans quel ordre.

On peut faire quelques autres améliorations sur ce code si on désire créer des affectations de nombres uniques par CPU. Si nous assignons le même nombre aux instructions `'jnz'` et `'jz'`, nous pouvons alors ignorer cette inversion de branchements; si nous assignons `'1'` à l'instruction `'nop'`, nous pouvons ignorer l'insertion de *nops*, etc.

En pratique, l'utilisation de bibliothèques de grands nombres entiers est impossible. En utilisant l'arithmétique de la CPU (cela signifie l'arithmétique modulo 2^64) nous n'avons plus la preuve que la factorisation est unique, mais nous gagnons beaucoup en vitesse (et la probabilité de trouver deux séquences de code valides avec la même signature n'est pas grande).

D'un point de vue algorithmique, il y a beaucoup plus de petites astuces que l'on peut utiliser pour obtenir un isomorphisme plus important, mais il n'y a pas assez d'espace ici pour les détailler. Pour un traitement plus profond du sujet, référez-vous à **[BDDIMVA]**, **[BDSSTIC]**.

Un mot sur les signatures conjoncturelles

Contrairement aux signatures structurelles, qui utilisent essentiellement les caractéristiques d'un graphe, les signatures conjoncturelles dépendent fortement de leur contenu, dans notre cas, le code désassemblé des fonctions des binaires étudiés.

Bien évidemment, calculer une signature (SHA-1, MD5 ou même CRC32) de la séquence d'octets derrière un bloc basique ne vous mènera pas loin : le code regorge de références relatives ou absolues à des emplacements en mémoire qui seront bien évidemment déplacés suite à recompilation du source modifié. Cette méthode ne résistera pas davantage aux réarrangements des registres, aux inversions des instructions de branchement.

L'alternative réside dans un traitement plus réfléchi du code assembleur. Il sera avisé de prendre en considération les mnémoniques assembleur, la taille des opérandes de ces derniers (8, 16, 32 ou 64 bits), leur type (registre, valeur immédiate, phrase, etc.) tout en vous affranchissant de détails plus précis, tels les valeurs exactes, le nom du registre, l'adresse des sauts et appels.

Il ne vous restera plus qu'à calculer votre signature sur la séquence d'octets obtenus, comme le montre l'extrait d'un *plugin* IDA cité dans **[0DAYS]**. Cependant, ces seuls éléments, même s'ils sont passables, seront insuffisants à l'encontre de binaires plus complexes.

Parmi ce qu'il est nécessaire de prendre en compte lors du calcul de signatures dites « conjoncturelles », sur un bloc basique ou une fonction, figurent :

- L'inversion de deux instructions, l'une prenant la place de l'autre et vice versa. Le procédé de calcul de signature pourra donc être indépendant de l'ordre des instructions.
- Le remplacement de certaines instructions par des instructions équivalentes. Comme tout langage de programmation, l'assembleur permet d'effectuer des

actions comparables sous bien des formes différentes, par exemple :

```
xor eax, eax
mov eax, 0
push 0 & pop eax
```

auront tous pour effet de mettre le registre `eax` à 0. Les compilateurs savent jouer de ces subtilités pour optimiser l'exécution du code qu'ils génèrent, en vitesse d'exécution ou en taille de code machine généré.

- Les modifications des branchements. Certains blocs basiques peuvent être scindés en plusieurs morceaux par des sauts inconditionnels : situation qui ne change en rien la logique du bloc en question, mais est susceptible d'influer sur la signature. Bien souvent, les sauts conditionnels sont aussi réorganisés, la condition inversée. Attention ! La modification d'un saut signé en non signé peut cacher une faille.
- Les corrections de constantes. Souvent mises de côté lors du calcul de la signature, les constantes peuvent cependant cacher un correctif de sécurité : taille d'un buffer passée à une fonction d'allocation mémoire, taille de l'espace réservé sur la pile, comparaison, sont autant de facteurs potentiels de déclenchement de débordement.

Le figures ci-dessous montrent une différence d'une seule instruction dans un bloc basique de deux binaires découverte grâce à une signature conjoncturelle reposant sur une CRC32. Cette modification implique de potentielles répercussions sur la sécurité de l'application.

Figure 6

```
77ea0203 (fd6cea47):
77ea0203 lea     eax, [eax+eax+2]
77ea0207 add     eax, 3
77ea020a and     eax, 0FFFFFFFCh
77ea020d call    __alloca_probe
77ea0212 mov     esi, esp
77ea0214 push    esi
77ea0215 push    edi
77ea0216 call    ?DecodeEscapedString@@YGKPAG0@Z
77ea021b test    eax, eax
77ea021d jnz     short loc_77EA01D8
```

5.1.2600.2180 (Windows XP professionnel SP2 français)

Figure 7

```
78046d7f (ee2ee7ef):
78046d7f add     eax, eax
78046d81 add     eax, 3
78046d84 and     eax, 0FFFFFFFCh
78046d87 call    __alloca_probe
78046d8c mov     esi, esp
78046d8e push    esi
78046d8f push    edi
78046d90 call    ?DecodeEscapedString@@YGKPAG0@Z
78046d95 test    eax, eax
78046d97 jnz     short loc_78046D54
```

5.1.2600.1106 (Windows XP professionnel SP1 français)

Au final, la signature conjoncturelle peut donner de très bons résultats tout en fonctionnant à un niveau de granularité élevé, ce qui permet de détecter des modifications au niveau de l'instruction assembleur et non du graphe. Sa dépendance au langage assembleur utilisé la rend difficilement portable à un grand nombre d'architectures. Enfin, elle est relativement mal adaptée dans le cas où deux binaires diffèrent grandement.

Applications à la recherche de vulnérabilités

La possibilité de mettre en exergue les différences entre deux binaires est de nos jours un atout majeur pour tout chercheur de vulnérabilités ou tout expert technique désirant qualifier la portée d'un correctif et la gravité des failles qu'il corrige.

Les bulletins sont rarement explicites d'un point de vue technique (avec le vain espoir de complexifier le processus d'exploitation) et déterminer l'impact réel d'une vulnérabilité relève du tour de force, notamment lorsqu'il s'agit d'éclaircir les points suivants : la facilité d'exploitation, sa fiabilité, sa dépendance à la régionalisation du système, à la version du système.

Si bien souvent le correctif apporte un nombre limité de changements aux binaires analysés, ce n'est pas le cas des *service packs*, qui non seulement sont des cumuls de mises à jour de sécurité mais ajoutent aussi de nouvelles fonctionnalités et de nombreuses améliorations. Un bon outil permettra cependant d'y détecter les rustines réparant des trous de sécurité.

À partir d'un correctif

Nous allons vous conter une histoire amusante à propos d'un patch particulier : MS04-001. En 2003, l'organisation NISCC utilisait un fuzzer ASN.1 et H.323 pour faire l'analyse de certaines implémentations du protocole H.323. Ils trouvèrent pas mal de bogues différents dans des implémentations variées, de Cisco à Microsoft.

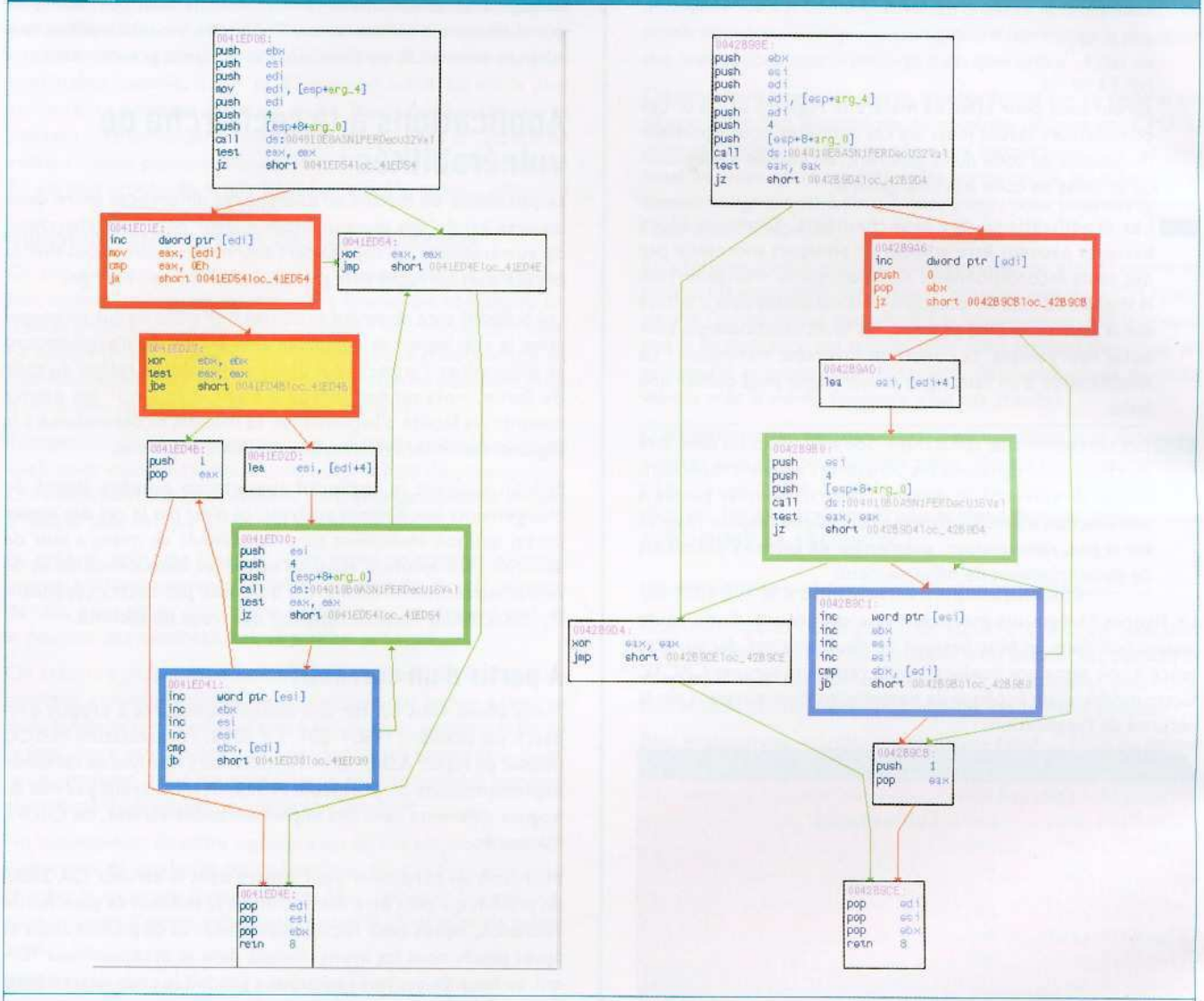
Microsoft en particulier était affecté dans le serveur ISA 2000, un produit qui peut être décrit comme la solution de pare-feu de Microsoft. Après avoir reçu les deux binaires du produit avant et après patch, nous les avons chargés dans le désassembleur IDA qui, au bout de quelques minutes, a produit le code désassemblé des applicatifs.

Nous avons alors pu utiliser les algorithmes décrits dans cet article pour construire un isomorphisme entre les deux binaires. Notre algorithme nous donne 6 fonctions associées avec des fonctions de signature différente – cela signifie qu'il nous signale 6 fonctions modifiées.

Nous regardons la première fonction changée, et voyons que Microsoft a introduit un contrôle de gamme dans le bloc basique à l'adresse `0x41ed1e`. La fonction en question prenait simplement une valeur N non signée de 32 bits, retournée par un appel à la fonction `ASN1PerDecU32Val()` et accessoirement contrôlable par un attaquant.

Elle copiait ensuite 2 * N valeurs non signées de 16 bits dans un buffer de taille fixe, celles-ci provenant d'appels répétés à `ASN1PerDecU16Val()` et étant également contrôlées par l'attaquant. Ce dernier est alors en mesure de choisir une valeur trop grande de N et provoquer un débordement puis de l'exécution de code arbitraire grâce à des données qu'il contrôle. (voir fig. 8 page 30)

Figure 8



Patch M504-001 : fonctions chargées

Ce patch apporta aussi d'autres changements moins didactiques : il introduisit quelques contrôles de gamme devant tous les appels à une fonction du système d'exploitation, `ASN1PERDecZeroTableCharStringNoAlloc`.

Cette information était d'une grande valeur, car indiquait l'absence de contrôles sur quelques paramètres de cette fonction. Cela se révéla bien utile pour trouver des failles 0-day dans d'autres applications l'utilisant, comme NetMeeting ! L'analyse a été menée avec **[BDIFF]** et a pris moins de deux heures.

Indépendance de l'architecture

En général, les algorithmes que nous avons abordés ici sont plus ou moins indépendants de l'architecture ou système d'exploitation – on peut sans modifications les utiliser sur des systèmes x86 embarqués (si on peut avoir un désassemblage propre dans IDA) ou même des systèmes non-x86.

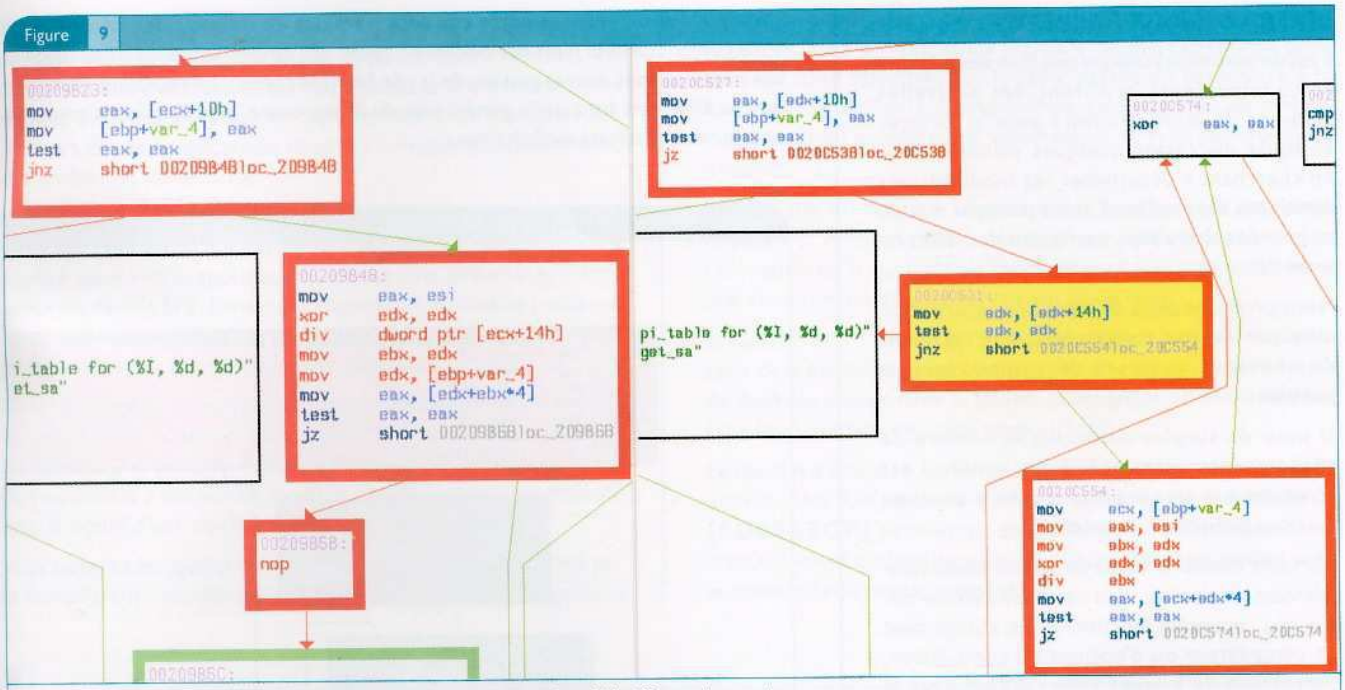
Par exemple, il est trivial de *differ* deux variantes du Cisco PIX ou d'autres systèmes embarqués. Le code suivant vient d'un *diff* de Cisco PIX entre les versions 6.34 et 6.33. Ici, les développeurs ont juste introduit une comparaison pour ne pas avoir de division par zéro.

Mais au total, il y a plus de 200 fonctions corrigées dans la mise à jour, et il bien possible qu'on puisse y trouver des changements plus enrichissants. (fig. 9)

Parce que nous n'avons pas restreint les algorithmes, qui sont indépendants de l'assembleur, nous avons aussi la capacité d'examiner les différences sur des plateformes non-x86.

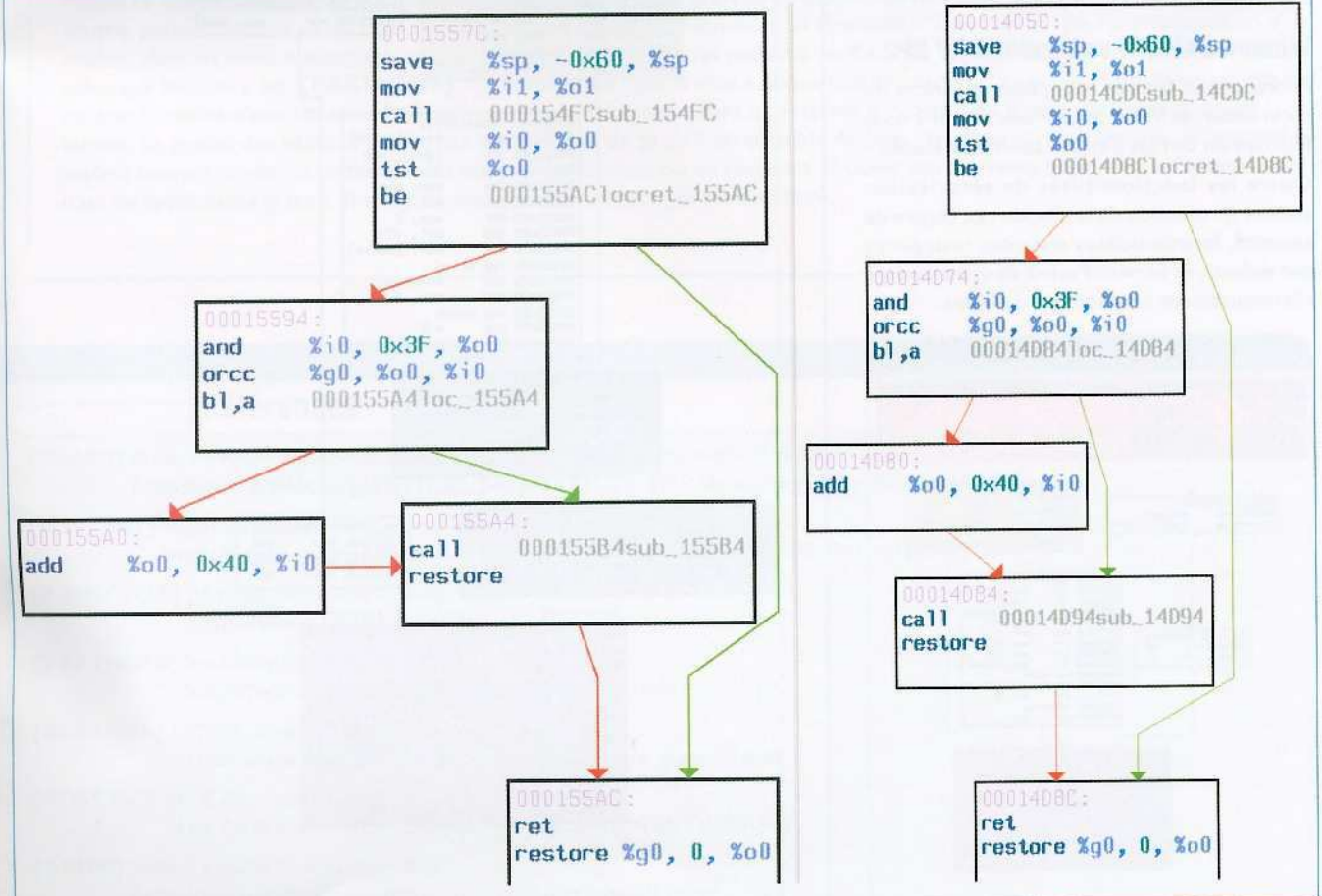
La seule chose que nous devrons adapter pour chaque CPU est le code de génération des graphes. Voici un exemple de deux fonctions identiques sur SPARC. (fig. 10)

Figure 9



Ciseo PIX, diff entre deux versions

Figure 10



Deux fonctions identiques sur SPARC

Entre versions majeures

Il est de notoriété publique que bien des éditeurs et constructeurs profitent des nouvelles versions de leurs produits pour y corriger en toute discrétion quelques vulnérabilités. En cherchant à déterminer les modifications apportées dans celles-ci, nous pouvons mettre au jour des altérations corrigeant des failles de sécurité.

Nous profitons alors du travail de sécurisation mené par l'éditeur, non divulgué, pour découvrir de nouveaux vecteurs de compromission possibles.

Si pour de simples correctifs le nombre de changements occasionnés est limité, il est généralement plus important dans la situation qui nous préoccupe maintenant.

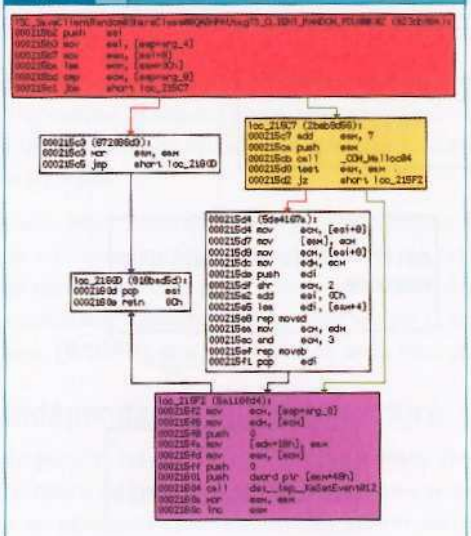
Certains éléments bouleverseront fondamentalement les binaires, mais ne nous intéresseront que peu : nouvelles fonctionnalités, changement de compilateur ou d'options de compilation, correction de bogues sans rapport avec la sécurité, nouvelles protections génériques. L'analyse différentielle n'en sera que plus complexe.

Windows XP SP1 vs Windows XP SP2

Microsoft ne fait pas exception et l'effort de sécurisation de Windows XP avec son SP2 nous fournira un terrain d'exploration convenable.

Outre les fonctionnalités de sécurisation variées (protection de la pile, du tas, centre de sécurité, fonctionnalités distantes restreintes par défaut), le Service Pack 2 de XP a corrigé silencieusement bon nombre de failles.

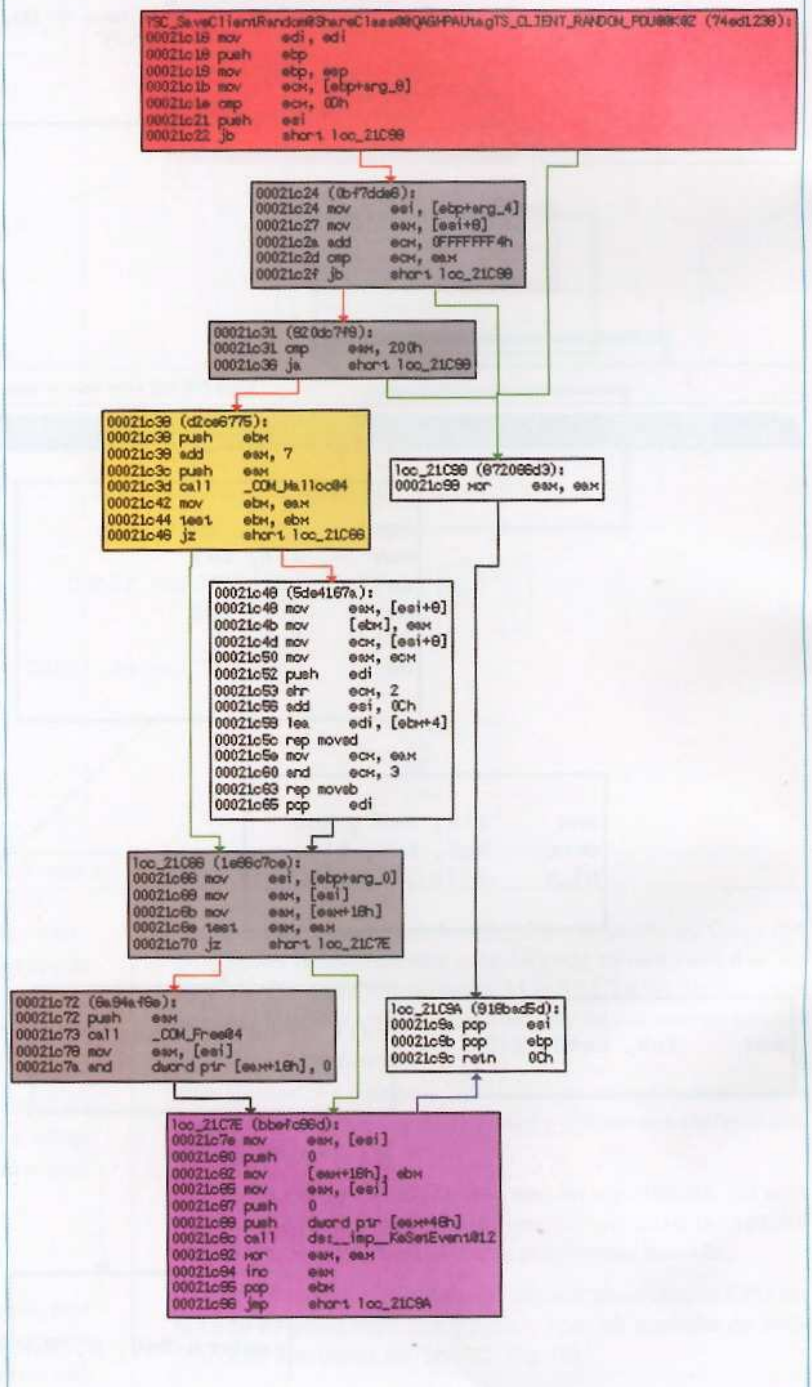
Figure 11



5.1.2600.1106 (Windows XP professionnel SP1 français)

L'analyse des différences entre ces deux versions en utilisant des signatures de code conjoncturelles n'est pas évidente à cause des optimisations de compilation et des modifications dans la gestion de la pile (dont le *cookie*). En réduisant l'influence de ces éléments lors de la génération de la signature, nous pouvons toutefois obtenir des résultats exploitables.

Figure 12



5.1.2600.2180 (Windows XP professionnel SP2 français)

Bureau à distance

Implémenté sous forme de driver système dans Windows XP (`rdpwd.sys` entre autres), le bureau à distance a été lourdement modifié avec la publication du Service Pack 2 de Windows XP. Plusieurs de ces modifications visaient à corriger, silencieusement, des problèmes de sécurité.

Lors de l'édition du correctif *Remote Desktop Protocol* **[MS05-041]** qui concernait un tout autre problème, Microsoft en a profité pour rétro-appliquer quelques-uns des changements apportés dans le SP2. L'analyse différentielle du binaire concerné a pu révéler à certaines personnes ces failles bien avant qu'elles ne fassent l'objet d'une mise à jour publique.

Dans les graphes ci-contre, les nœuds ayant un fond blanc sont inchangés (dans un sens relativement large) d'une version à l'autre, les nœuds gris sont de nouveaux nœuds, et ceux de couleur correspondent à des nœuds modifiés à mettre en correspondance avec le nœud d'une couleur similaire dans l'autre graphe.

Vous noterez les quelques modifications mineures induites par le compilateur : modification du cadre de pile, réarrangement de l'organisation des blocs de code, utilisation différente des registres. Une modification majeure réside cependant dans la vérification effectuée sur la taille passée en paramètre à la fonction `COM_Malloc()`. Cette comparaison permet de mettre rapidement en évidence une vulnérabilité de type débordement d'entier.

Le contenu du registre `eax` passé à la fonction est issu de la requête et donc contrôlé par le client, qui peut provoquer l'allocation de la quantité de mémoire qu'il souhaite et mener à une condition de déni de service sur le serveur.

Si `0xfffffff9 <= eax <= 0xffffffff`, alors `0 <= eax + 7 <= 6` et la taille de la mémoire allouée sera insuffisante pour la quantité (`eax`) de données copiée dans la foulée, provoquant un débordement type *heap overflow* en mode noyau (`ring0`).

Le résultat à attendre : un écran bleu (l'exécution de code est **très** improbable). Un patch de quelques lignes du logiciel rdesktop **[RDESKTOP]** permettait de déclencher le débordement à distance avant authentification du client, problème résolu avec le correctif cité précédemment.

Conclusion

Malgré ce que peuvent penser éditeurs et constructeurs, les correctifs binaires sont souvent aussi explicites que des `diff` de code source, pourvu que l'on ait les bons outils, et des compétences suffisantes en assembleur. Il en résulte que l'implémentation d'un exploit, dans les milieux autorisés, est une question d'heures : nous pouvons vérifier cela régulièrement grâce à des entreprises telles que Immunity, Inc. **[PARTNERS]**. Comprenez bien que la mise à disposition de correctifs au public rapidement est loin de refléter la réalité d'une situation bien sombre : leur déploiement dans un système d'information d'envergure sera toujours trop tardive. La qualité des exploits publics est bien en deçà de ce qu'il est possible de faire : les résultats des analyses différentielles restent souvent privés, de même que les outils d'exploitation qui en résultent. Trouver des nouvelles failles, dites « 0days », dans d'autres applications grâce à des bogues similaires à ceux corrigés est commun.

Références

[0DAYS] *0-Days – Recherche et exploitation de vulnérabilités en environnement Win32* :
http://actes.sstic.org/SSTIC05/0day/SSTIC05-article-Kortchinsky-0day.pdf

[MS05-041] *Vulnerability in Remote Desktop Protocol Could Allow Denial of Service* :
http://www.microsoft.com/technet/security/Bulletin/MS05-041.mspx

[RDESKTOP] *rdesktop : A Remote Desktop Protocol Client* :
http://www.rdesktop.org/

[PARTNERS] Immunity Partner's Page :
http://www.immunitysec.com/partners-index.shtml

[BDDIMVA] DIMVA paper on BinDiff :
http://www.sabre-security.com/files/dimva_paper2.pdf

[BDSSTIC] SSTIC05 paper on BinDiff :
http://www.sabre-security.com/files/BinDiffSSTIC05.pdf

[BDIFF] SABRE BinDiff Comparison Tool :
http://www.sabre-security.com/products/bindiff.html

Recherche de vulnérabilités à l'aide du fuzzing

Dans cet article, nous vous proposons une courte introduction au fuzzing, un moyen de découvrir relativement rapidement des erreurs dans des programmes. La méthodologie est simple : des données « quasi » conformes au protocole employé sont générées aléatoirement et envoyées au système à tester, permettant ainsi de jauger la réaction du programme à ces entrées inhabituelles. Nous vous présentons le fuzzing grâce à quelques exemples de base, tout en donnant un aperçu de la manière de l'employer pour détecter des faiblesses dans des programmes.

Introduction

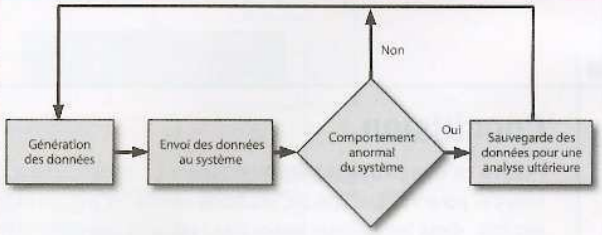
Le *fuzzing* est une méthode par laquelle on peut assez rapidement traquer des vulnérabilités dans des programmes. La pratique en est simple : en premier lieu, on génère des données aléatoires, qui sont à peu près conformes au protocole. Ces données peuvent être par exemple de très longues entrées (telles que 2342 x «A»), des indications de format (comme «%n%n%n%n»), des limites de valeurs d'entier (-1 ou 0xffffffff) ou des caractères Unicode. Celles-ci ne sont pas totalement aléatoires, mais correspondent *grosso modo* aux spécifications du protocole. Elles doivent obéir en gros à la logique de programmation du système à tester, en particulier ses limites dans les opérations de *parsing*. Si l'on envoyait au système des données totalement fantaisistes, il rejetterait certainement ces entrées dès qu'il remarquerait qu'elles ne correspondent pas à ce qui est attendu. Nous choisissons plutôt des données *semi-valides*, qu'un *parser* ne refusera pas tout de suite, mais qui pourront cependant provoquer assez de problèmes. Comme l'ont démontré de nombreux exemples par le passé, le parser d'un système est souvent le maillon faible :

- Ethereal (**http://www.ethereal.com/**) a déjà détecté des erreurs dans les nombreux parsers pour paquets réseaux. Par exemple, avec la version 0.10.13, vingt vulnérabilités furent en tout résolues, qui concernaient le parser du protocole réseau ;
- Dans le parser de Snort, une vulnérabilité critique BackOrifice a été découverte à la mi-octobre 2005, qui autorisait une prise de contrôle à distance (*Remote Command Execution*) **[Snort05]** ;
- Beaucoup de navigateurs plantent à cause de pages web complexes, car le parser provoque des erreurs lors du rendu. Mais nous y reviendrons...

Lors de la deuxième étape, on envoie ces données au système à tester. On entend ici *système* au sens large du terme: On peut tester, à l'aide du fuzzing, une application client ou serveur, mais un système d'exploitation ou un matériel peuvent également être mis à l'épreuve.

On observe après coup comment le système réagit. Au mieux, il plante et l'on a découvert un bogue. Mais il se peut aussi que le système ait un énorme et soudain besoin de ressources mémoire ou que la charge du processeur bondisse dans le rouge. Tous deux prouvent que l'interprétation de nos données semi-valides nous permet d'avoir une influence sur son bon fonctionnement même s'il ne peut s'agir « que » d'une faiblesse de type déni de service. Dans tous les cas, on sauvegarde les données envoyées pour une analyse ultérieure et on recommence du début. On effectue la démarche aussi si le système n'a pas réagi anormalement à nos entrées. La figure 1 clarifie le processus complet.

Figure 1



Principe du Fuzzing

Typiquement, on trouve à l'aide du fuzzing des faiblesses dans les fonctions de parsing ou de formatage. Mais l'on peut aussi, à l'aide de cette technique, découvrir des erreurs dans la gestion de la mémoire ou dans d'autres parties des programmes. Les vulnérabilités ainsi découvertes peuvent être alors utilisées pour acquérir le contrôle complet du programme testé. La création d'un code malicieux (typiquement un exploit) fonctionnel est l'étape suivante du fuzzing.

L'article de K. Kortchinsky de ce dossier aborde plus précisément la création de codes fiables. Bien sûr, toutes les faiblesses découvertes par fuzzing ne sont pas des vulnérabilités critiques, mais souvent certaines vulnérabilités rencontrées pourraient être exploitées par un individu malveillant. Un simple crash est déjà intéressant en lui-même, en particulier s'il s'agit par exemple de celui d'un système de détection d'intrusion ou d'un scanner anti-virus...

Dans la suite de cet article nous allons d'abord expliquer plus en détail la procédure fondamentale du fuzzing, ceci à l'aide d'un *fuzzer* pour client IRC: Nous vous présentons chaque étape et décrivons, à l'aide d'un exemple, comment l'on applique avec un programme les principes de la recherche de vulnérabilités.

Les faiblesses découvertes par nos soins dans le client IRC sont bien entendu présentées ; cependant nous n'indiquons pas si elles sont exploitables, ni comment elles le seraient. Ensuite, nous vous montrons grâce à quelques exemples supplémentaires de quelle manière le fuzzing peut être encore utilisé.

Thorsten Holz
thorsten.holz@miscmag.com

Ilja van Sprundel
ilja@netric.org

Cet article se conclut par une courte introduction à un *framework* qui permet de simplifier le fuzzing. À propos de **Protos [Protos]**, certainement le projet le plus connu dans ce domaine, un article spécifique vous livrera des informations de fond tout en vous permettant de jeter un coup d'œil au travail de l'équipe.

Le fuzzing de clients IRC

Comme exemple didactique des principes du fuzzing, nous allons étudier un fuzzer pour clients IRC. Pour cela, observons tout d'abord rapidement le protocole IRC. Comme mentionné plus haut, le fuzzing suit globalement les spécifications du protocole pour coller en gros à la logique du programme. *Alors qu'est exactement l'IRC et comment fonctionne ce protocole ?* IRC sont les initiales de « Internet Relay Chat » ; celui-ci est développé depuis 1988.

C'est un protocole d'échange de messages texte et les premières spécifications sont fixées dans la RFC 1459. La plupart des commandes et mécanismes qui y sont décrits sont encore valables ; cependant, les RFCs 2810-2813 proposent une version actualisée du protocole. Sans grande importance dans la pratique, puisque aucune implémentation IRC ne les supporte pleinement.

La structure du protocole IRC est comparable à celle des programmes de messageries instantanées comme Jabber, AOL Instant Messenger (AIM) ou ICQ, tout en montrant quelques différences. IRC est caractérisé par une architecture client-serveur : il y a donc un serveur central (ou un réseau de serveurs), auquel se connectent les clients.

La communication entre serveur et client est basée sur des *messages* et cela en clair ; un codage SSL est optionnel. Après le contact TCP 3-Way-Handshake s'échangent entre le client et le serveur des messages d'un maximum de 510 caractères (en excluant le retour chariot et le *Line Feed*) et qui doivent avoir cette structure : tout d'abord l'expéditeur (*préfixe*), ensuite la commande à proprement parler, puis d'éventuels paramètres supplémentaires.

La réponse du serveur à ce message d'un client est constituée de son préfixe, d'un code d'état de trois chiffres et du nom du client. L'utilisation du code d'état est similaire à d'autres protocoles en clair comme POP3, HTTP ou encore SMTP. Le client C envoie le premier message au serveur S :

```
C -> S: NICK user
C -> S: USER user 0 0 :realname
```

Le client s'enregistre à l'aide de ce message avec le pseudo désiré et donne également des informations sur l'utilisateur. Dans l'exemple suivant, nous voyons trois messages qui s'échangent lors du contact initial. Le préfixe de ce serveur dans ce cas est `:irc.matrix.org`, un serveur du réseau `Rizon`. Suivent ensuite les différents codes d'état et le pseudo du client, de même que les informations relatives aux codes.

```
S -> C: :irc.matrix.org 001 user :Welcome to the Rizon Internet Relay Chat
Network user
S -> C: :irc.matrix.org 002 user :Your host is irc.matrix.org[irc.matrix.
org/6667], running version PleXusIRCd-2.0.9p1
S -> C: :irc.matrix.org 003 user :This server was created Sun Sep 25 2005 at
01:33:15 GMT
```

Le serveur transmet d'autres informations au client, comme **251** (RPL_LUSERCLIENT – le nombre d'utilisateurs (in)visibles sur le serveur) ou **254** (RPL_LUSERCHANNELS – le nombre actuel de canaux). Cette phase s'achève normalement avec le code **376** (RPL_ENDOFMOTD), qui montre la fin du *Message of the Day*.

Jusque-là, le serveur a envoyé plus de dix messages au client, que celui-ci doit *parser*, seulement afin de comprendre l'état du serveur. Ensuite, il peut également envoyer des données au client, que celui-ci devra encore assimiler.

Et c'est entre ces deux extrémités que nous nous plaçons avec notre fuzzer : nous envoyons à un client IRC des messages conformes au protocole, mais qui contiennent néanmoins de quoi l'amener à un plantage.

Dans ce qui suit, nous vous présentons la technique du fuzzing à l'aide du programme **ircfuzz.c**, de Ilja van Sprundel [ircfuzz05]. Ce programme montre très clairement comment un fuzzer est construit et quel est son principe de fonctionnement.

Comment donc allons-nous faire ? Tout d'abord, nous ouvrons une socket TCP auquel le client IRC peut se connecter, en principe le port TCP 6667 ou le port TCP 7000. Ensuite, nous lançons une boucle d'une durée infinie, le fuzzing comme tel.

Dans cette boucle sont envoyées au client des données aléatoires pour tester si sa fonction de parsing tient bon, même avec des entrées inhabituelles. Nous utilisons ici le fait que le programmeur d'un client s'attend normalement à ce que le serveur lui transmette des données conformes au protocole – mais pourquoi n'y aurait-il pas de serveur mal intentionné ?

Avec la fonction `accept()` nous démarrons une connexion et nous empêchons la socket de se bloquer avec `fcntl()`. Cela sert en particulier à ne pas laisser troubler la communication avec le client à tester par des fonctions de blocage d'entrée/sortie. Après le traitement des messages NICK et USER envoyés par le client commence véritablement le fuzzing à proprement parler.

En tout, nous testons cinq différentes sortes d'entrées :

- Cas 0: Une longue chaîne aléatoire, tant par son contenu que dans sa longueur
- Cas 1: Une chaîne typique pour utiliser la vulnérabilité Formatstring
- Cas 2: Un grand nombre aléatoire d'arguments
- Cas 3: Une très longue chaîne alphanumérique
- Cas 4: Une longue chaîne aléatoire de caractères quelconques

Le code ressemble à ceci :

```
void fuzz (int fd) {
  char buf[100000];
  int raw = rand () % 1000, end, i, a, al;
  if (!(rand () % 10))
    raw = -raw;

  switch (rand () % 5) {
    case 0:                         // long string
      end = (rand () % 12000) + 10;
      for (i = 0; i < end; i++) {
            do { buf[i] = rand () % 256; }
            while (buf[i] == '\n' || buf[i] == '\r' || buf[i] == '\0');
          }
      break;
    case 1:                         // fmtbug
      strcpy (buf, "%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n");
      break;
    case 2:                         // lots of arguments
      a = (rand () % 230) + 20;
      al = (rand () % 750) / a;
      buf[0] = '\0';
      for (i = 0; i < a; i++) {
            char t[100];
            int j = 0;
            for (j = 0; j < al; j++)
              t[j] = arr[rand () % sizeof (arr)];
            t[j] = '\0';
            strcat (buf, t);
            strcat (buf, " ");
          }
      break;
    case 3:                         // long alpha str
      end = (rand () % 12000) + 10;
      for (i = 0; i < end; i++) {
            buf[i] = arr[rand () % sizeof (arr)];
          }
      buf[i] = '\0';
      break;
    case 4:                         // random stuff, random length
      end = (rand () % 2500);
      sprintf (buf, ":aaa %d ", raw);
      al = strlen (buf);
      for (i = 0; i < end; i++)
            buf[i + al] = rand () % 256;
      buf[i + al] = '\r';
      buf[i + al + 1] = '\n';
      write (fd, buf, i + al + 2);
      return;
  }

  if (rand () % 5)
    netprintf (fd, ":%s %03d %s :%s%s", gen_hostname (), raw, Nnick, buf,
               (rand () % 100) ? "\r\n" : "");
  else if (rand () % 3)
    netprintf (fd, ":%s %s %s :%s%s", gen_hostname (), gen_command (), Nnick,
               buf, (rand () % 100) ? "\r\n" : "");
  else
    netprintf (fd, ":%s %s %s :\x01%s %s%s%s", gen_hostname (), "PRIVMSG",
               Nnick, gen_submsg (), buf, (rand () % 20) ? "\x01" : "",
               (rand () % 100) ? "\r\n" : "");
  return;
}
```

La fonction `netprintf()` transmet les données remises par `send()` à la socket et suit les indications de formatage. Les fonctions `gen_hostname()`, `gen_command()` et `gen_submsg()` créent les entrées de fuzzing pour les différentes parties de message prévues par le protocole IRC.

En particulier, nous voulons mettre en valeur l'implémentation des fonctions `gen_submsg()` et `gen_hostname()`, par lesquelles seront générées aléatoirement des commandes typiques d'IRC ou un nom d'hôte :

```
char *submsg[] = {"DCC SEND", "DCC CHAT", "DCC XMIT", "DCC OFFER", "XDCC LIST",
"XDCC SEND", "FINGER", "VERSION", "USERINFO", "CLIENTINFO", "TIME", "FINGER",
"PING", "CDCC SEND", "CDCC LIST", "CDCC XMIT", "XDCC XMIT", "DCC LIST", "XDCC
OFFER", "XDCC CHAT", "CDCC OFFER", "CDCC OFFER", "DCC", "XDCC", "CDCC", NULL };

char * gen_submsg () {
  if (rand () % 26) {
      return submsg[rand () % 25];
  }
  else {
      return "";  // test for empty string
  }
}

char * gen_hostname () {
  static char b[1500];
  int len = rand () % 1500 / ((rand () % 3) + 1);
  memset (b, 'a', len);
  b[len] = '\0';
  return b;
}
```

Maintenant, nous disposons de tous les éléments pour assembler notre fuzzer : dans une boucle infinie, nous recevons toutes les communications entrantes et préparons avec nos routines de fuzzing les messages que nous allons retourner au client.

Nous attendons à chaque fois que le client nous ait envoyé les messages NICK et USER, puis nous lui transmettons les codes de connexions IRC. Au cas où le client résiste lors de cette phase, d'autres données de notre fuzzer tentent de le mener au plantage. Le pseudo-programme ressemble à ceci :

```
if (nick && user  {
  init++;
  netprintf (cfd, ":aaa 001 %s :a\r\n"
                  ":aaa 002 %s :a\r\n"
                  ":aaa 003 %s :a\r\n"
                  ":aaa 004 %s :a\r\n"
                  ":aaa 005 %s :a\r\n"
                  ":aaa 251 %s :a\r\n"
                  ":aaa 252 %s :a\r\n"
                  ":aaa 253 %s :a\r\n"
                  ":aaa 254 %s :a\r\n"
                  ":aaa 255 %s :a\r\n"
                  ":aaa 375 %s :a\r\n"
                  ":aaa 372 %s :a\r\n"
                  ":aaa 376 %s :a\r\n", Nnick, Nnick, Nnick,
                  Nnick, Nnick, Nnick, Nnick, Nnick, Nnick,
                  Nnick, Nnick, Nnick, Nnick);
}
if (init) {
  fuzz(cfd);
}
```

La façon dont un client IRC réagit à des entrées erronées d'un serveur IRC peut être testée avec l'aide de ce programme. Pour cela, on démarre le client IRC et on le configure de telle manière qu'il utilise `localhost` bzw `127.0.0.1` comme serveur.

Le client doit essayer de se connecter à notre fuzzer et ainsi débute le processus de fuzzing en lui-même : notre fuzzer génère des entrées aléatoires et les envoie au client. Celui-ci tente de parser les données entrantes et dérape parfois : soit un plantage, ou bien un comportement bizarre, comme une consommation anormalement élevée de ressources.

Le tableau suivant démontre à quel point ces tentatives sont efficaces. Tous les clients IRC ci-contre ont pu être crashés par

cette méthode et ont montré diverses sortes de vulnérabilités. Seul **Irssi** a résisté jusqu'ici à toutes les tentatives de fuzzing. On a seulement observé chez ce client IRC quelques fuites de mémoire, de même qu'une augmentation constante de la consommation de ressources pendant le fuzzing.

Opera et d'autres encore sous Mac OS X n'ont guère résisté longtemps à des tentatives de fuzzing. Il semblerait que l'implémentation sous Mac OS X éprouve le besoin d'être revue. On a pu, grâce à ces utilitaires, détecter également des vulnérabilités parmi les navigateurs disponibles sous Linux :

TABLEAU 1

BitchX (1.1-final)	mIRC (6.16)	xchat (2.4.1)	kvirc (3.2.0)	ircii(ircii-20040820)	eggdrop (1.6.17)
epic-4 (2.2)	ninja (1.5.9pre12)	emech (2.8.5.1)	Virc (2.0 rc5)	TurboIRC (6)	leafchat (1.761)
iRC (0.16)	conversation (2.14)	colloquy (2.0 (2D16))	snak (5.0.2)	ircle (3.1.2)	ircat (2.0.3)
darkbot (7f3)	bersirc (2.2.13)	Scrollz (1.9.5)	IM2	pirch98	Trillian (3.1)
Microsoft Comic Chat (2.5)	icechat (5.50)	centericq (4.20.0)	uirc (1.3)	weechat (0.1.3)	rhapsody (0.25b)
kmyirc (0.2.9)	bnirc (0.2.9)	bobot++ (2.1.8)	kwirc (0.1.0)	nwirc (0.7.8)	kopete (0.9.2)

Aperçu des clients IRC « fuzzés » avec succès

Certains clients IRC cibles ont un comportement intéressant pendant leur fuzzing. **Xchat** par exemple a provoqué ces messages d'erreur :

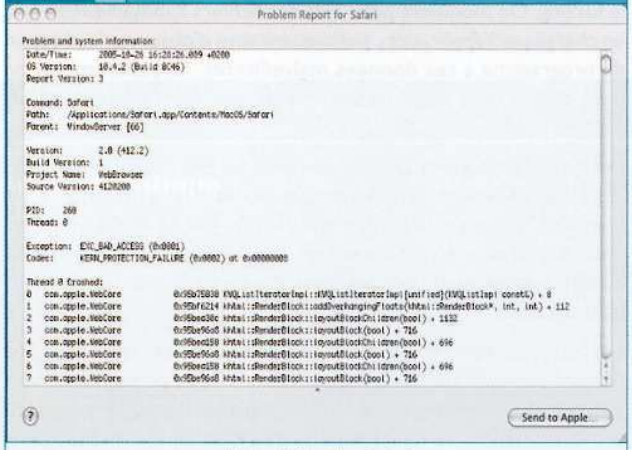
```
$ xchat
*** XCHAT WARNING: Buffer overflow - shit server!
*** XCHAT WARNING: Buffer overflow - shit server!
[...]
*** XCHAT WARNING: Buffer overflow - shit server!
Speicherzugriffsfehler (core dumped)
```

En cas de crash du client IRC, nous voulons bien sûr savoir quelles entrées l'ont provoqué. Il y a en principe deux possibilités pour le faire : ou nous journalisons tout en interne, le programme de fuzzing rédige lui-même un logfile. Ou bien nous procédons à un enregistrement externe, en l'occurrence ici avec l'utilitaire **tethereal**. Celui-ci a l'avantage dans ce cas de pouvoir se servir des données qui ont amené un client IRC à planter pour tester un autre client IRC sur cette même vulnérabilité. Ce qui peut être réalisé avec un utilitaire comme **tcpreplay**. Il faut néanmoins s'assurer de faire journaliser **tethereal** en mode Ringbuffer, car sinon les fichiers logs peuvent devenir énormes...

Le fuzzing d'autres applications clientes

En plus du fuzzing de clients IRC, on peut employer cette technique pour débusquer les vulnérabilités d'autres sortes de programmes. Un exemple connu est **MangleMe** de Michael Zalewski **[mangleme04]**. Ce programme conçoit de complexes sites Web, comprenant des erreurs typiques. Une implémentation en Python (**htmler.py** de nd**[htmler.py04]**) existe également. Nous l'avons un tantinet modifié dans l'optique de couvrir une zone d'entrées plus large. Ce programme réalise de manière aléatoire une page HTML difficile, qui sera ouverte dans un navigateur. Ces utilitaires ont permis de découvrir de nombreuses erreurs dans Internet Explorer ainsi que dans d'autres navigateurs. Ils sont encore utiles aujourd'hui... Ainsi, Safari, le navigateur standard de Max OS X, présente quelques vulnérabilités, comme le prouve un rapide test. (fig. 2)

Figure 2



Vulnérabilités dans Safari

Firefox 1.0.7, la dernière version de Firefox (Firefox 1.5 Beta 2), Mozilla 1.7.12, Konqueror 3.4 et Epiphany 1.7.6 ont été vaincus par le fuzzing.

Les erreurs provoquées ainsi vont du plantage (typiquement par SIGSEV) aux boucles sans fin jusqu'à la consommation exagérée de ressources mémoire. Voici, à titre d'exemple, le journal strace d'une tentative réussie de fuzzing sur Firefox :

```
writev(3, [{"H\2a\nL\1\200\0M\1\200\0\361\0\v\0\1\0\2\0\0\30\374\377"..., 24},
{"\377\0\0\0\0\377\377\377\0\377\377\377\0\0\377\377\377\0"..., 10604}], 2) = 10628
stat64("/usr/lib/mozilla-firefox/components/necko_cache.xpt", {st_mode=S_
IFREG|0644, st_size=2180, ...}) = 0
open("/usr/lib/mozilla-firefox/components/necko_cache.xpt", O_RDONLY|O_LARGEFILE) = 33
read(33, "XPCOM\nTypeLib\r\n\32\1\2\0\16\0\0\0\0\204\0\0\0\"\0\0\1"..., 2180) = 2180
close(33)                               = 0
gettimeofday({1132532058, 201260}, NULL) = 0
gettimeofday({1132532058, 201708}, NULL) = 0
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
unlink("/home/tho/.mozilla/firefox/ufyrgv4t.tho/lock") = 0
rt_sigaction(SIGSEGV, {SIG_DFL}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [SEGV], NULL, 8) = 0
tgkill(17875, 17875, SIGSEGV)           = 0
```

```
--- SIGSEGV (Segmentation fault) @ @ (@) ---
+++ killed by SIGSEGV +++
```

On peut rapidement débusquer de nouvelles erreurs même avec un fuzzer légèrement modifié. Pour atteindre en pratique de semblables résultats, il suffit simplement de télécharger **htmler.py** de **http://felinemenace.org/~nd/htmler.py** et de le modifier, par exemple en y rajoutant des tags ou des extensions, de tester par fuzzing d'autres caractères spéciaux, ou bien encore d'augmenter les valeurs de durée. Des possibilités supplémentaires de « fuzzer » un navigateur HTML sont, par exemple :

- « Fuzzer » l'en-tête HTTP, i.e. produire un en-tête HTTP défectueuse ;
- « Fuzzer » le traitement du JavaScript, i.e. créer des applications JavaScript aléatoires ;
- « Fuzzer » le traitement du FTP (ftp://), de Usenet (news://) ou de l'à propos (about:) ;
- « Fuzzer » les plugins, comme Java, RealPlayer, Flash ou Shockwave.

D'autres applications peuvent également être mises à l'épreuve par le fuzzing. On modifiera pour cela des fichiers habituellement pris en charge par l'application, permettant ainsi d'observer la réaction du programme à ces données malveillantes. Grossièrement, on peut différencier deux formes de fuzzer : les *intelligents* et les *simples*.

Les fuzzer intelligents savent comment est constitué le format du fichier et peuvent suivre les particularités d'un protocole. Cela peut être le cas quand le format de fichier enregistre la longueur d'un champ donné comme entier – nous essaierons par fuzzing d'avoir précisément une influence sur ce champ et les données qu'il contient.

Les fuzzers simples au contraire ne connaissent pas les formats de fichiers et modifient aléatoirement des séries d'octets dans leurs entrées. Ce procédé relativement simple peut malgré tout, bien souvent, mener rapidement au succès escompté... Nous allons maintenant observer de plus près quelques-uns de ces fuzzers simples.

Comme lors du fuzzing des clients IRC, nous créons des données semi-valides, qui ressemblent beaucoup aux entrées normales, mais qui cependant diffèrent légèrement des spécifications. Cela peut être la modification aléatoire de quelques octets dans l'en-tête du fichier ou l'introduction de longues séquences d'octets dans le corps du fichier.

Nous allons, à titre d'exemple, employer la technique de la recherche locale : on démarre par un fichier d'entrée valide et modifie au hasard quelques octets au sein du fichier. Dans un deuxième temps, nous testons si le fichier manipulé amène le programme à des réactions inhabituelles. Si ce n'est pas le cas, une nouvelle série aléatoire d'octets est modifiée.

Un nombre déterminé d'essais est d'abord effectué – puis l'on recommence du début avec un fichier de départ valide. Si l'on découvre lors de la deuxième étape un comportement anormal du programme, alors le fichier est sauvegardé pour une analyse ultérieure.

Il peut également servir de base de départ à d'autres tentatives de fuzzing dans l'optique de détecter encore plus d'anomalies dans ce programme.

Un exemple permet d'illustrer la technique employée : nous allons observer dans ce qui suit des programmes qui traitent les fichiers mp3. Ici nous modifions au hasard une longue séquence d'octets dans un fichier mp3 intact. Nous ne nous limitons donc pas au test de l'en-tête mp3, de chaque *frames* mp3 ou des *tags* ID3 à la fin du fichier, mais voulons éprouver dans son ensemble la structure du fichier.

Les résultats prouvent que certains lecteurs mp3 ont définitivement besoin de voir leur implémentation du parsing améliorée : lors du test, il n'a fallu que quelques minutes pour planter avec succès les programmes **mpg123**, **mpg321** et **mocp**. Le programme **mp3-decoder** s'est laissé également facilement « fuzzer » – quelques centaines d'essais ont ici suffi. Dans ce programme, une autre forme de crash a pu être provoquée :

```
$ mp3-decoder test2-2907.mp3
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layer 1, 2, and 3.
Version 0.59q (2002/03/23). Written and copyrights by Joe Drew.
Uses code from various people. See 'README' for more!
THIS SOFTWARE COMES WITH ABSOLUTELY NO WARRANTY! USE AT YOUR OWN RISK!

Playing MPEG stream from test2-2900.mp3 ...
MPEG 1.0 layer III, 128 kbit/s, 44100 Hz mono
[0:00] Decoding of test2-2900.mp3 finished.
Playing MPEG stream from test2-2901.mp3 ...
[0:00] Decoding of test2-2901.mp3 finished.
[...]
Playing MPEG stream from test2-2908.mp3 ...
[0:00] Decoding of test2-2908.mp3 finished.
Playing MPEG stream from test2-2909.mp3 ...
*** glibc detected *** double free or corruption (!prev): 0x08056dd8 ***
Aborted
```

Malheureusement, **Xmms** ne s'est pas laissé crasher – il a survécu sans difficulté plusieurs jours d'affilée aux tests comprenant des milliers d'entrées différentes. Par cette forme simple de fuzzing, il a été impossible d'identifier des vulnérabilités chez Xmms, mais la conception d'un fuzzer plus intelligent aurait peut-être conduit au succès.

Un meilleur fuzzer créerait des fichiers mp3 conformes au protocole, mais incluant des vulnérabilités précises, comme par exemple un fichier mp3 valide mais comprenant des erreurs lors de la compression sans perte ou du codage Huffman...

Parallèlement aux lecteurs logiciels, le fuzzing de l'iPod ou d'un autre lecteur matériel est possible. Nous ne l'aborderons cependant pas dans le cadre de cet article – l'un de nos auteurs a déjà assez de problèmes avec son iPod :-)

Une procédure similaire est envisageable également pour d'autres formats multimédias. Les techniques présentées antérieurement ont été utilisées pour modifier aléatoirement différents formats de films et ainsi tester les vulnérabilités de programmes comme **Mplayer**.

Ce dernier, en particulier, se prête bien à ce genre de tests de stress : grâce à l'option `-vo null` et `-ao null`, il se transforme simplement en un lecteur multimédia qui ne crée pas de sortie,

mais qui ne parse que les données d'entrée. Nous avons testé diverses sortes de formats d'entrée, par exemple des fichiers QuickTime et RealMedia. Dans ce programme, plusieurs fonctions ont été amenées à planter :

```
* MPlayer interrupted by signal 11 in module: demux_open
* MPlayer interrupted by signal 11 in module: decode_video
* MPlayer interrupted by signal 11 in module: video_read_properties
```

En dehors de cela, les tests démontrent que la gestion de mémoire de Mplayer subit des fuites – le besoin en ressources augmente foncièrement à chaque nouveau test. Le fuzzing de **Xine** n'est de loin pas aussi simple : le fait que Xine ouvre apparemment toujours une fenêtre ralentit naturellement le fuzzing.

Et dans le cas d'entrées que le programme déclare comme défectueuses, une boîte de dialogue prévient l'utilisateur. On peut oublier un fuzzing automatique de ce lecteur multimédia...

Par hasard, une simple erreur a été découverte dans un autre programme : **aaxine**. Celui-ci traite incorrectement les paramètres trop longs et plante si leur longueur dépasse les 1800 caractères (selon le système, bien que plus de 3000 les conduisent tous au crash) :

```
$ aaxine `perl -e "print 'A' x 3000"`
Segmentation fault
```

En dehors des programmes d'édition audio et vidéo, n'importe quelle sorte d'application peut être scrutée par une telle technique, entre autres : logiciels de retouche d'images, visionneuses PDF et suites Office, pour ne citer qu'eux.

Le fuzzing d'applications serveur

De même que pour les applications client, les vulnérabilités des applications serveur peuvent être recherchées par le biais de cette technique. Nous ne donnons ici qu'un court exemple de ce genre de test, puisque la méthode fondamentale ne diffère pas de celle exposée dans la section précédente.

Nous allons nous servir, à titre d'exemple de serveur à « fuzzer », de NTPd, le *Network Time Protocol daemon*. Et au lieu de le faire manuellement, nous allons employer cette fois un utilitaire qui possède les capacités nécessaires à cette méthode : **Scapy** (**http://secdev.org/projects/scapy/**).

Scapy est un utilitaire universel de manipulation de paquets réseaux. Ses possibilités dépassant de loin le cadre cet article, nous vous renvoyons à sa documentation **[scapy05]** et nous nous contentons de vous présenter brièvement ses atouts pour le fuzzing.

Depuis la version 1.0.0.42, scapy intègre les opérations de fuzzing, invoquées par la commande `fuzz()`. En guise d'exemple, on peut tester les vulnérabilités d'un serveur NTPd local avec ces quelques lignes :

```
>>> conf.L3socket=L3RawSocket
>>> send(IP(dst="127.0.0.1")/UDP()/fuzz(NTP()),loop=1)
```

Il nous faut d'abord configurer Scapy, de manière à ce que le kernel se charge par Raw-Sockets de la couche de liaison de données, c'est-à-dire que les paquets soient réellement envoyés par l'interface Loopback.

Puis nous envoyons à notre propre machine des paquets UDP valides, dont la charge utile consiste en paquets NTP aléatoires. Scapy assume ici le rôle d'un fuzzer intelligent, puisqu'il connaît la structure de NTP et remplit chaque champ de données semi-valides dans ce protocole. Il est également possible de réduire l'étendue de l'investigation, en limitant par exemple grâce à `fuzz(NTP(mode=3, version=3))` l'envoi de paquets client NTP à la version 3 du protocole.

Mais nos tests n'ont hélas pu réussir à crasher ni NTPd, ni OpenNTPd. Chez NTPd, on a pu cependant observer un comportement intéressant : si l'on supervise le daemon lors du fuzzing avec `gdb` ou `strace`, il peut rester stable pendant plusieurs jours. Dès qu'on y rattache `ltrace`, le daemon a toutes les chances de planter dans les 5000 prochains paquets – un fait qui est reproductible. La cause exacte de ce phénomène n'a pu encore être déterminée, mais nous persévérons dans ce but :-)

Avec d'autres applications serveur, la procédure est similaire au cas précédent. Nous transmettons des données semi-valides au serveur et étudions comment celui-ci réagit à ces entrées. Cela s'applique aux serveurs HTTP (fuzzing des requêtes HTTP), NFS (génération aléatoire de paramètres) et à toutes les diverses sortes de serveurs. Une étude poussée de SSH a été réalisée grâce à **SSHredder [SSHredder02]**, laquelle a décelé plusieurs bogues dans les implémentations SSH.

Les frameworks de fuzzing

Mis à part les utilitaires spécifiques présentés auparavant, il existe également des frameworks génériques qui allègent considérablement les tâches de fuzzing. L'exemple suivant va nous permettre de vous présenter brièvement l'un de ces frameworks et d'évoquer quelques implémentations supplémentaires.

SPIKE de Immunity **[SPIKE]** est un framework de fuzzing qui se prête particulièrement à l'évaluation d'applications serveur. Il est écrit en C et inclut un grand nombre d'outils, comme des scripts de test pour MSRPC, HTTP ou SMTP. SPIKE fonctionne d'après une méthode de blocs, ce qui procure certains avantages **[Block04]**. L'élément de base d'un tel projet est appelé un *SPIKE*. Il s'agit d'une description de haut niveau, complété de méta-informations de fuzzing. Observons maintenant ce que cela donne concrètement avec un serveur Web Sambar **[Sambar04]**.

```
s_string("POST /search/results.stm HTTP/1.1\r\n");
s_string("Host: MSUDGEDPU\r\n");
s_string("Content-Length: ");
s_blocksize_string("post",7);
s_string("\r\n\r\n");
s_block_start("post");
    s_string("spage=0&indexname=docs&query=");
    s_string_variable("MEEP");
    s_string("&style=page");
    s_string("\r\n\r\n");
s_block_end("post");
```

`s_string` permet d'invoquer des chaînes d'une taille fixe et définit grossièrement la structure des données à envoyer. Avec `s_block_start/s_block_end`, on démarre ou termine au choix un bloc. Dans ce bloc, nous définissons l'une des entrées comme chaîne d'une longueur variable (`s_string_variable`), notre fuzzing se concentrant par conséquent sur cet unique paramètre.

Pour nous permettre de toujours envoyer une requête HTTP-POST valide, nous nous devons à chaque fois d'adapter notre `Content-Length` si nous ne voulons pas nous faire rejeter immédiatement par le serveur Web.

Cela est rendu possible par la fonction `s_blocksize_string`, qui à la fin du bloc `"post"` introduit la longueur correcte. Un tel SPIKE suffit comme description, le framework pouvant se charger du fuzzing à proprement parler – dans notre cas c'est le paramètre `query` qui sera « fuzzé » par des chaînes de longueurs variables. L'utilisation de SPIKE permet donc de simplifier nos efforts: Après avoir réalisé une description de haut niveau des données, le framework se charge lui-même de conduire les tests.

Il existe d'autres frameworks, chacun adapté à un cas de figure particulier. Voila un rapide survol et un descriptif des plus célèbres frameworks en la matière :

- **Peach** : Framework de fuzzing en Python avec quelques exemples de mise en situation ;
- **Smudge** (*Software Mutilation Utility and Data Generation Engine*) : Implémentation proche de SPIKE en Python supportant un grand nombre de protocoles réseaux ;
- **SPIKEfile** : Adaptation de SPIKE, pour le fuzzing de formats de fichiers ;
- **Mangle** : Proche de la méthode de fuzzing simple de formats multimédias, en particulier les en-têtes de fichiers ;
- **COMbust** : Fuzzer binaire de test d'objects COM.

Conclusion

Dans cet article nous vous avons présenté les principes fondamentaux du fuzzing. Nous avons essayé d'illustrer comment l'employer pour découvrir assez rapidement des vulnérabilités dans des programmes. Quelques exemples ont pu prouver que cette méthode est efficace et que l'on pouvait effectivement débusquer des faiblesses chez de nombreux clients. Ce qui cependant n'implique naturellement pas que chacun de ces bogues soit également une vulnérabilité critique – une analyse plus poussée de chaque anomalie est incontournable.

Les démonstrations de fuzzing d'applications client ou serveur présentées ici peuvent également être employées pour les tests d'autres systèmes : l'envoi par exemple de données semi-valides à un NetworkStack, la création aléatoire d'arguments d'appels système ou de librairies, le fuzzing d'un descripteur de fichier, ou encore l'émission aléatoire de signaux à un processus en cours ou bien la manipulation des variables d'environnement qu'un programme emploie. L'imagination n'a ici pas de limites – soyez créatif et suivez de nouvelles voies :-) Le fuzzing est plutôt une méthode qu'une implémentation concrète et pour chaque problème, il faut concevoir un fuzzer adapté (*intelligent* ou *simple*). Bien sûr, des frameworks génériques comme SPIKE ou Peach peuvent soulager vos efforts...

On peut employer le fuzzing pour une recherche rapide de vulnérabilités dans un programme. Par rapport à l'analyse binaire ou de code source, ce procédé est terriblement plus rapide et livre souvent d'étonnants résultats. Néanmoins, le fuzzing n'est pas non plus une panacée. Il y a des bogues que l'on détecte rapidement par cette technique, mais aussi certains qui sont impossibles à « fuzzer ». Dans l'analyse de code source, là aussi, on peut rapidement arriver au but, ou n'avoir atteint aucun résultat après des heures d'essais infructueux. La meilleure recette est donc une combinaison des deux procédures. En premier lieu, tenter de récolter avec le fuzzing les « fruits pendants » puis faire subir aux parties critiques d'un programme une recherche d'erreurs approfondie par analyse du code source ou des fichiers binaires.

Références

[Block04] – David Aitel: « The Advantages of Block-Based Protocol Analysis for Security Testing », **http://immunitysec.com/downloads/advantages_of_block_based_analysis.html**

[htmler.py04] **http://felinemenace.org/~nd/htmler.py**

[ircfuzz05] **http://ilja.netric.org/files/fuzzers/ircfuzz.c**

[mangleme04] **http://freshmeat.net/projects/mangleme/** ou **http://freshmeat.net/redir/mangleme/54072/url_homepage/gallery**

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

[Sambar04] « Overflow in Sambar Webserver », **http://lists.virus.org/spike-0401/msg00009.html**

[scapy05] « Network packet manipulation with Scapy », **http://www.secdev.org/projects/scapy/**

[Snort05] « Snort Back Orifice Pre-processor Remote Buffer Overflow Exploit », **http://www.frsirt.com/exploits/20051025.THCsnortbo.c.php**

[SPIKE] **http://immunitysec.com/resources-freesoftware.shtml**

[SSHredder02] « Vulnerabilities in SSH2 Implementations from Multiple Vendors », **http://www.rapid7.com/advisories/R7-0009.html**

Fiabilisation d'exploits Windows

Quiconque aura un jour essayé d'utiliser des exploits Windows disponibles sur Internet sera sans doute du même avis : qualité et fiabilité ne semblent pas être au cœur des préoccupations des concepteurs. Passons outre les soucis de portabilité du code entre les différents compilateurs et plateformes, rarement pris en compte, pour nous intéresser à des exploits un tant soit peu fonctionnels. Quelles sont les chances qu'un tel exploit donne le résultat escompté dans un environnement différent de celui mis en place pour le développement du code ? Généralement, elles sont faibles.

Les aspects de régionalisation des applicatifs ou du système d'exploitation sont rarement pris en compte, de même que le niveau d'application des correctifs. Or ces éléments peuvent avoir une influence sur les différents aspects constitutifs de l'exploit, pas tant sur le *shellcode*, que sur les différentes adresses nécessaires à la bonne exécution de notre code. Voyons comment s'affranchir de ces limitations, et évidemment comment se prémunir contre des exploits de plus en plus fiables.

L'exemple de MS05-039

La vulnérabilité MS05-039 **[MS05-039]**, corrigée par Microsoft le 9 août 2005, impacte une opération de copie mémoire dans la bibliothèque dynamique `umpnpmgr.dll`. Cette portion de code peut être atteinte par 4 fonctions exportées par l'interface RPC `8d9f4e40-a03d-11ce-8f69-08003e30051b` **[PNP]**, dont 2 (`PNP_QueryResConfList` et `PNP_DetectResourceConflict`) présentent un chemin d'exécution menant à la faille particulièrement simple, accessible anonymement **[MSRPC]**. Ces informations en main, il ne reste plus qu'à construire une requête déclenchant le débordement et observer son impact sur le flux d'exécution, le contenu des registres et de la pile du processus hébergeant cette interface RPC. Le procédé ne sera pas détaillé ici, car dépassant largement le cadre de l'article.

Figure 1

```
Registers (FPU)
EAX 013EF58C
ECX 43434343
EDX 784751B6 ntdll.784751B6
EBX 013EF914 ASCII "BBBBCCCCDDDDDDDDDDDDDDDDDDDD
ESP 013EF4F4
EBP 013EF514
ESI 013EF5B4
EDI FF004444

EIP 43434343

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FF94000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty +UNORM 0178 00070178 00070178
ST2 empty +UNORM 0640 00000000 00000000
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
```

Etat des registres suite au débordement (services.exe)

Figure 2

```
013EF4F4  7846EB70  RETURN to ntdll.7846EB70
013EF4F8  013EF5B4
013EF4FC  013EF914  ASCII "BBBBCCCCDDDDDDDDDDDDDDDDDDDD"
013EF500  013EF5D0
013EF504  013EF58C
013EF508  013EF914  Pointer to next SEH record
013EF50C  784751B6  SE handler
013EF510  013EF914  ASCII "BBBBCCCCDDDDDDDDDDDDDDDDDDDD"
013EF514  013EF59C
013EF518  78475160  RETURN to ntdll.78475160 from ntdll.7846EB
013EF51C  013EF5B4
013EF520  013EF914  ASCII "BBBBCCCCDDDDDDDDDDDDDDDDDDDD"
013EF524  013EF5D0
013EF528  013EF58C
013EF52C  43434343
013EF530  FF004444
013EF534  00436508
013EF538  013EF5B4
013EF53C  013EF560
013EF540  7802711E  RETURN to MSVCRT.7802711E from MSVCRT.7801
```

Etat de la pile suite au débordement (services.exe)

La chaîne de caractère `"BBBBCCCCDD..."` apparaissant à plusieurs reprises a été soumise par nos soins et indique par conséquent une zone mémoire sous notre empire. A ce stade précis, nous avons le contrôle de EIP et donc du flux d'exécution. Il nous reste à déterminer quelle valeur affecter à ce registre afin de sauter dans notre zone mémoire, et donc potentiellement exécuter du code.

L'expérience vous apprendra qu'utiliser directement une adresse de la pile n'est généralement pas une bonne idée lorsque le processus est *threadé*, l'adresse de base de la pile ayant tendance à ne pas être bien stable. Ainsi, après un *reboot* et une deuxième tentative, notre *buffer* se verra déplacé de `0x13ef914` en `0x51f914`.

L'alternative réside dans la présence de l'adresse de notre zone mémoire à des emplacements référençables relativement au contenu des registres au moment de la prise de contrôle du flux d'exécution. La valeur à affecter au registre EIP devra être l'adresse d'une suite d'instructions permettant de rediriger le flux d'exécution vers le contenu d'un des emplacements trouvés précédemment. Dans notre exemple, EBX contient l'adresse de notre buffer. Ainsi, il nous faudra affecter à EIP l'adresse d'une instruction JMP EBX, ou équivalente, afin d'arriver dans notre buffer.

Vous remarquerez aussi plusieurs références à notre chaîne dans la pile, adressables relativement à ESP. Ainsi pour [ESP+8] (troisième entrée dans la pile en partant du haut), une instruction semblable à JMP [ESP+8] fera tout aussi bien l'affaire. Une variation couramment utilisée pour cette dernière instruction est une suite d'instructions POP REG1, POP REG2 suivie d'un RET (quels que soient REG1 et REG2, nous n'avons pas à nous préoccuper de leur contenu) : cela équivaut à rajouter 8 à ESP grâce aux deux POP, puis à sauter vers notre buffer grâce au RET, dans la mesure où notre adresse se trouve maintenant en haut de la pile.

Certaines des adresses possibles, utilisées dans la nature, sont présentées dans le tableau 1.

Kostya Kortchinsky
Ingénieur chercheur - EADS/CCR
kostya.kortchinsky@eads.net

TABLEAU 1

Quelques exploits publics pour la vulnérabilité PnP MS05-039

Date	Auteur	Adresse	Mnémoniques assembleur	Module	Commentaire
11 août 2005	H D Moore	0x767a38f6	pop edi, pop esi, retn	umpnpmgr.dll	Intégré à Metasploit
11 août 2005	m5l0nny	0x75021e3e	call ebx	ws2help.dll	
12 août 2005	houseofdabus	0x767a1567	pop eax, pop esi, retn	umpnpmgr.dll	
		0x767a366f	jmp ebx	umpnpmgr.dll	Adresse non utilisée
17 août 2005	Variante IRCBot	0x01013c79	pop esi, pop ebx, retn	services.exe	Inspiré du précédent

Dépendances à l'environnement

Usuellement, de telles adresses se trouvent au sein des différents modules chargés par le processus : exécutables, bibliothèques dynamiques. En prospectant directement les fichiers à la recherche de séquences hexadécimales correspondant aux instructions souhaitées, puis en traduisant les adresses physiques concordantes en adresses virtuelles, on obtient le résultat escompté. Cette méthode est utilisée notamment par l'outil msfpescan du *framework* Metasploit **[METASPLOIT]**.

Néanmoins, ces binaires sont amenés à évoluer avec les versions du système d'exploitation ou de l'applicatif concerné. Tout correctif pourra induire un déplacement (ou même la suppression) dans le code des séquences d'octets utilisables. Même entre deux versions identiques d'un binaire, l'adresse de base de chargement de ce dernier peut varier, entraînant une modification de l'adresse virtuelle des séquences en mémoire. Il en résulte une forte dépendance entre les adresses de retour et l'environnement dans lequel est exécuté le processus : version du système d'exploitation, version de l'application, régionalisation. Notez toutefois que les versions « poste de travail » ou « serveur » d'un même Windows (et autres si elles existent) sont construites sur un même *pool* de binaires, ce qui n'engendre pas de différence d'un produit à un autre pour un même numéro de mouture.

Les résultats exposés ci-dessus ont le mérite de mettre en évidence la volatilité des adresses des bases de certaines DLL en fonction du langage : parfois identiques pour deux versions dissemblables, parfois différentes pour deux versions similaires. Afin de contourner ce facteur contraignant, nous pouvons envisager plusieurs possibilités : trouver des adresses communes aux différentes versions, identifier finement la régionalisation et les applicatifs, provoquer des fuites d'information, attaquer par force brute.

Adresses de retour communes

Bien des exploits se targuent d'être universels, à savoir de cibler indifféremment tout système vulnérable à la faille impliquée (enfin du moins une sous-partie bien délimitée). La raison à cela : l'utilisation d'adresses de retour communes à un grand nombre de plateformes.

TABLEAU 2

Adresses de base de la bibliothèque kernel32.dll

Version de Windows 2000	Service Pack	Régionalisation	Version	Adresse de base
Professionnel	Aucun	Russe	5.0.2191.1	0x77e80000
Professionnel	3	Anglais	5.0.2195.5400	0x77e80000
Server	4	Japonais	5.0.2195.6688	0x77e50000
Advanced Server	4	Français	5.0.2195.6688	0x77e70000
Professionnel	4 avec URI	Espagnol	5.0.2195.7006	0x79450000

Ces adresses existent à des degrés variables, les unes cibleront par exemple la famille des Windows 2000 anglais, tandis que les autres adresseront les Windows 2000 avec Service Pack 4 quelle que soit leur régionalisation.

Les constats concernant les systèmes d'exploitation sont les suivants :

- Il existe un certain nombre de DLL qui, au sein d'une même version majeure de Windows (par exemple 2000), n'ont pas été modifiées au cours des versions mineures (*service packs*) tout en conservant la même adresse de base. Malheureusement, cette adresse de base varie d'une régionalisation à une autre. C'est dans les zones mémoires occupées par ces binaires que l'on trouvera des adresses « universelles » pour une régionalisation donnée de l'OS. Dans l'exemple de MS05-039, les adresses de `ws2help.dll` et `umpnpmgr.dll` ciblent les Windows 2000 anglophones quel que soit leur service pack (ainsi que quelques autres langues par effet de bord).
- Il existe des EXE, modifiés au fil des service packs, qui gardent une adresse de base identique pour toutes les régionalisations. C'est dans ceux-ci qu'il faudra chercher des adresses « universelles » pour une version mineure donnée. Dans l'exemple de MS05-039, l'adresse de `services.exe` cible tous les Windows 2000 SP4 – dont, bien sûr, les francophones.

Les binaires combinant les attributs énoncés précédemment sont **très** rares, et même s'ils existent, leur utilisation est souvent cantonnée à des programmes obscurs. Afin de les mettre en exergue, un utilitaire tel que DllVers **[DLLVERS]**, et un grand nombre de plateformes de test, vous seront indispensables. L'idée est de lister tous les binaires d'une installation de Windows (on peut se limiter aux DLL et EXE du répertoire `%WinDir%\system32` dans un premier temps), leur version et l'adresse de base de leur image en mémoire afin de détecter ceux communs. L'emploi d'outils de virtualisation tels VirtualPC ou VMware permet de faire cohabiter l'ensemble sur une même machine physique. Une alternative est la *Metasploit Opcode Database* **[OPCODEDB]**, qui vous permettra de rechercher, dans une base de données pré-remplie, des adresses qui répondent à des critères que vous définirez. Les plateformes disponibles sont nombreuses, avec pour l'instant trois régionalisations, et les réponses sont pertinentes. Il va sans dire que ces constats s'adaptent aussi aux appli-cations.

Concernant le tableau précédent

La bibliothèque `msvcp50.dll` est utilisée, entre autres, par le service *Message Queuing*, affecté par une vulnérabilité de type *stack overflow* [MS05-017]. Elle est par conséquent l'endroit idéal où trouver une adresse de retour universelle pour cette faille précise. Quant aux autres, leur usage est plus sporadique. Attention ! Les adresses de base peuvent changer dans l'éventualité où deux bibliothèques seraient chargées à cheval sur des zones mémoires déjà mappées (typiquement une collision entre deux DLL). C'est un aspect à prendre en compte. Notez aussi que ce tableau ne recense pas les exécutables, qui peuvent aussi constituer une source d'adresses de valeur.

TABLEAU 3

Bibliothèques dynamiques communes aux versions de Windows 2000 utilisées précédemment

Module	Version	Adresse de base
admparse.dll	5.0.2920.0	0x80000000
bootvid.dll	5.0.2172.1	0x80010000
dbmsadsn.dll	1999.10.20.0	0x42bd0000
dbmssocn.dll	1999.10.20.0	0x73330000
dbmsspxn.dll	1999.10.20.0	0x42be0000
gpkcsp.dll	5.0.2134.1	0x08000000
mcdsrv32.dll	5.0.2160.1	0x80010000
mspatcha.dll	5.1.2600.0	0x600a0000
msvcirt.dll	6.1.8637.0	0x780a0000
msvcp50.dll	5.0.0.7051	0x780c0000
slbcsp.dll	5.0.2134.1	0x08000000
slbkygen.dll	5.0.2144.1	0x08000000
sqlwid.dll	1999.10.20.0	0x412f0000
vcdex.dll	5.0.2134.1	0x0ffb0000
vdmredir.dll	5.0.2134.1	0x0ffa0000

Emulation de code

eEye a présenté une méthode plus évoluée de recherche d'adresses aux conférences BlackHat grâce à leur outil privé EEREAP (*eEye Emulating Return Address Purveyor*). En sauvegardant l'état du processus attaqué au moment de la prise de contrôle du flux d'exécution (mémoire, registres, drapeaux, etc.), l'outil tente de trouver les séquences d'octets qui, une fois exécutées, mènent à une zone mémoire sous leur contrôle, en les émulant. Ils peuvent alors découvrir des suites de plusieurs dizaines d'octets satisfaisant leurs critères et multiplier le nombre d'adresses de retour possibles par 4 ou 5. La plus grosse portion de code découverte dans leur cas d'étude contient 91 instructions.

Au delà des DLL...

Exécutables et bibliothèques dynamiques ne sont pas les seuls éléments à peupler l'espace mémoire d'un processus. On y trouve aussi quantité d'objets : des piles (*stack*), des tas (*heap*), des fichiers mappés, des sections partagées, des blocs d'environnement (TEB et PEB), etc. Même si les données de certains sont très volatiles, elles sont pour d'autres beaucoup plus stables et peuvent se révéler éminemment intéressantes. Afin d'élargir le panel des résultats, la recherche d'adresses utiles

devra s'étendre à la totalité de l'espace mémoire du processus à exploiter.

Grâce à un outil comme DumpOp **[DUMPOP]**, nous pouvons rapidement parcourir les zones de mémoire commises (affublées du drapeau MEM_COMMIT) pour un processus donné et lister les adresses qui satisfont nos critères. Il vous signalera également les adresses communes à différentes plateformes. Il fonctionne en allant lire l'espace mémoire d'un processus afin d'y repérer des séquences hexadécimales prédéfinies, et peut sauvegarder les résultats dans un fichier. Ce fichier pourra être ensuite chargé sur une autre plate-forme afin de ne conserver que les « collisions ».

TABLEAU 4

Adresses de retour potentielles communes aux versions de Windows 2000 utilisées précédemment

Adresse	Objet	Instructions
0x00187533	unicode.nls	jmp ebx
0x0018759f	unicode.nls	call ebx
0x001875e3	unicode.nls	call ebx
0x00187839	unicode.nls	jmp ebx
0x00187a1f	unicode.nls	jmp ebx
0x00189b5f	unicode.nls	jmp ebx
0x0018a485	unicode.nls	jmp ebx
0x001ecad0	sortkey.nls	push ebx, retn
0x001f0cd0	sortkey.nls	call ebx
0x001f4c30	sortkey.nls	jmp ebx

Les fichiers NLS (nécessaires au support natif de la langue), apparaissant dans le tableau ci-dessus, présentent trois avantages : ils sont toujours mappés au même endroit pour chaque version majeure de Windows (NT 4.0, 2000, XP et 2003), ne subissent pas de modification avec les service packs, et contiennent une pléthore d'adresses de retour utilisables. Deux inconvénients cependant : les pages mémoires ne sont pas marquées exécutables, l'octet de poids fort de ces adresses est nul.

Le premier ne devrait pas poser de problème dans le cadre de cette étude (lors d'exécution de code sur la pile, elle aussi marquée comme non exécutable, vous trouverez de plus amples informations sur cela dans l'article **[TINNES]** de ce numéro); quant au second, il peut nuire à l'exploitation de failles fondées sur des opérations de traitement de chaînes de caractères ANSI, sauf si ce caractère est le dernier de la chaîne provoquant le débordement. **Blaster**, pour exploiter la vulnérabilité MS03-036, utilisait comme adresse de retour `0x0018759f` lorsqu'il ciblait les Windows 2000, adresse que vous retrouverez dans notre tableau.

Dans le cas de MS05-039, il s'agit d'une copie mémoire, la présence d'un caractère nul n'influant pas sur le débordement. Dans le cas de débordement dû au traitement de chaîne de caractères UNICODE, tel que pour MS05-046 **[MS05-046]**, la présence de ce seul caractère nul ne gênera pas l'exploitation, puisque seul un couple d'octets nuls (qui plus est en position « paire ») marque la fin d'une chaîne.

Il est bien sûr possible de trouver d'autres adresses génériques, en fonction du processus étudié, des instructions recherchées. Plus on restreindra le nombre de cibles, plus la quantité d'adresses sera importante. On pourra alors se permettre davantage d'exigences telles que l'absence de certains octets dans les adresses, des adresses compatibles unicodes, etc.

Fuites d'information

Si la généricité « ultime » est complexe à atteindre, il peut être opportun d'obtenir un maximum d'informations sur le système à attaquer avant d'en exploiter une faille, afin de construire un code sur mesure. En environnement Windows, il est rare d'avoir la possibilité de lancer des attaques par force brute essayant exhaustivement l'ensemble des paramètres possibles, et la première attaque exécutée devra être la bonne.

Identifier finement la cible

La version majeure de l'OS est un minimum, son niveau de mise à jour est un plus, sa régionalisation se révèlera à l'usage essentielle. Il existe bien entendu des méthodes génériques de prise d'empreinte d'un système : pile TCP/IP, bannières, SNMP, qui peuvent parfois suffire. Windows offre cependant des possibilités supplémentaires d'identification avancée, ne nécessitant souvent qu'un minimum de privilèges, à savoir une session dite « nulle ». La plupart de ces fonctionnalités sont accessibles grâce aux API standards Microsoft, documentées dans la bibliothèques MSDN. Nous signalerons en particulier :

- `NetQueryDisplayInformation` : permet de récupérer des informations sur les comptes, les ordinateurs et les groupes sur la machine distante ;
- `NetRemoteTOD` : retourne les informations concernant la date et l'heure courantes sur la machine spécifiée ;
- `NetServerGetInfo` : retourne des informations sur la machine spécifiée grâce au service Serveur : plate-forme, nom, version, type ;
- `NetShareEnum` : énumère les partages disponibles sur le poste distant ;
- `NetSessionEnum` : détaille les sessions en cours sur le poste distant ;
- `NetUserEnum` : liste les utilisateurs existant sur le poste distant ;
- `NetWkstaGetInfo` : retourne des informations sur la machine spécifiée grâce au service « Poste de Travail » : plate-forme, nom, version, type.

Un exemple d'usage de ces fonctions est donné dans un article chinois **[XFOCUS]**, et les renseignements récupérables par ce biais sont complets : seul le niveau de mise à jour échappera à notre activité de reconnaissance. Pour pallier cela, nous pouvons

passer par l'énumération des interfaces RPC exposées **[SEKI]** car correctifs et service packs influent sur ces dernières, en retirant certaines, en exposant de nouvelles. Plusieurs indications figurent aussi à des endroits non accessibles directement à l'utilisateur, comme lors de l'établissement de sessions SMB. Les valeurs des champs `Native OS` et `Native LAN Manager` des paquets réponses `SessionSetupAndX` donnent la version majeure de Windows. Si vous êtes en mesure de capturer du trafic émanant de la machine ciblée, les données transmises par les applications clientes telles le navigateur préciseront avantageusement le contexte.

Fuites d'adresses

Si nous arrivons à obtenir des pointeurs issus de la mémoire de l'objectif, nous pourrons peut-être arriver à nos fins en passant outre toute tentative d'identification du système ou de l'applicatif. En général, c'est toute la problématique du *memory leak* ou fuite de mémoire qui nous intéressera. En particulier, ce sont certaines propriétés de l'implémentation DCERPC de Microsoft qui nous captiveront. Par exemple, l'attribut `[unique]` du langage MIDL, utilisé pour définir les interfaces RPC, spécifie un pointeur unique : si cet attribut est accolé à l'attribut `[out]` qui identifie un paramètre retourné par l'appel RPC, bingo ! Nous aurons dans la réponse à notre appel un pointeur vers une zone mémoire du processus. Dans le meilleur des cas, ce pointeur sera vers une zone mémoire sous notre contrôle. C'était le cas pour le service de Gestion de Licenses **[MS05-010]** : la fonction RPC `LlsrLocalServiceAddW` nous permettait de placer notre buffer en mémoire, tandis que la fonction `LlsrLocalServiceInfoGetW` nous retournait une copie de ce buffer en la précédant d'un pointeur vers cette copie. Il ne restait alors plus qu'à déclencher le débordement sans autre considération, grâce à cette adresse fournie gracieusement par le service attaqué, pointant dans une zone mémoire sous notre contrôle.

Conclusion ou comment s'en prémunir ?

Vous l'aurez sans doute remarqué, les outils d'exploitation publics de vulnérabilités ainsi que les vers en vogue sont loin d'intégrer tous ces éléments à leur code. Il est donc primordial de prendre les devants afin d'élever la difficulté d'exploitation de votre parc informatique et vous prémunir des évolutions prochaines à attendre dans ce domaine. Les premières mesures à adopter relèvent de considérations sécuritaires générales :

- Filtrez autant que faire se peut les ports associés à SMB et MSRPC : 135/TCP, 137/UDP, 138/UDP, 139/TCP, 445/TCP seront un bon début ;

Identification distante de la régionalisation de Windows

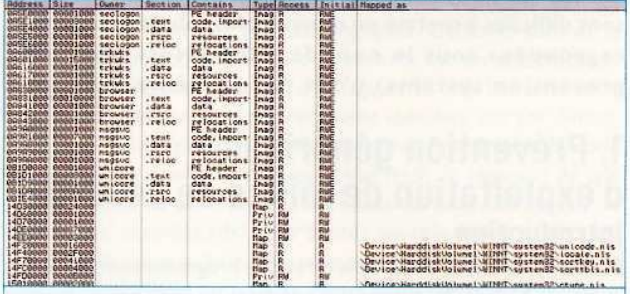
A ce jour, il ne semble pas exister publiquement de méthode fiable pour déterminer à distance la nature de la régionalisation d'une installation d'un système d'exploitation Windows. Si avec NT 4.0, le choix était relativement réduit, il a explosé avec 2000 : versions japonaise, chinoise, turque, danoise,...

Heureusement, certaines chaînes de caractères changent en fonction de la langue et transparaissent dans les réponses à certaines requêtes RPC. Des personnes ont suggéré l'utilisation de `NetQueryDisplayInformation` ou `NetUserEnum` afin de récupérer les noms des utilisateurs administrateur et invité et les comparer à ceux recensés en fonction de la régionalisation. Les résultats sont plutôt décevants, d'autant plus que Windows XP refuse ces requêtes par défaut. L'alternative réside dans le champ `Remark` des partages listés par la fonction `NetShareEnum`, hautement dépendant de la régionalisation de l'OS. Un exemple d'implémentation est donné dans l'utilitaire FpLang **[FPLANG]**. Seuls XP SP2 et 2003 SP1 rejettent par défaut ces appels distants.

- Désactivez les sessions nulles ainsi que tous les services dont vous n'avez pas l'utilité **[SERVICES]** ;
- Masquez ou modifiez toutes les bannières sur lesquelles vous avez un contrôle ;
- Bien évidemment, maintenez en permanence vos systèmes et applications à jour (Windows XP SP2 et Windows 2003 SP1 sont réellement plus sûrs que leurs prédécesseurs).

Cela réduira en partie les risques. Un autre élément de réponse réside dans l'altération des adresses de chargement en mémoire des bibliothèques dynamiques, des fichiers mappés, et autres objets. Pour les DLL, l'outil officiel `rebase.exe` de Microsoft permet de modifier leur adresse de base. Pour le reste, il faudra vous tourner vers *WehnTrust* **[WEHNUS]**, qui introduit une randomisation de la disposition de l'espace d'adressage d'un processus, neutralisant ainsi une bonne partie des méthodes usuellement employées lors d'attaques. Le procédé n'est pas parfait mais augmente tout de même le niveau de protection.

Figure 3



Extrait d'espace d'adressage avec WehnTrust (services.exe)

Notez dans l'image précédente que les fameux fichiers NLS ne sont plus chargés à leur emplacement usuel relevé dans les précédents paragraphes, de même pour toutes les bibliothèques dynamiques. Malheureusement, ce n'est pas le cas pour les exécutables...

Références

[MS05-039] *Vulnerability in Plug and Play Could Allow Remote Code Execution* : **http://www.microsoft.com/technet/security/Bulletin/MS05-039.mspx**

[PNP] *PnP interface* : **http://www.hsc.fr/ressources/articles/win_net_srv/ch04s07s12.html**

[MSRPC] *Implication of multiple RPC services in one process* : **http://www.hsc.fr/ressources/articles/win_net_srv/ch04s11.html**

[METASPLOIT] *The Metasploit Framework* : **http://www.metasploit.com/projects/Framework/**

[DLLVERS] DllVers.c : **http://recca.rstack.org/dllvers.c**

[OPCODEDB] *The Metasploit Opcode Database* : **http://www.metasploit.com/opcode_database.html**

[MS05-017] *Vulnerability in Message Queuing Could Allow Code Execution* : **http://www.microsoft.com/technet/security/Bulletin/MS05-017.mspx**

[DUMPOP] DumpOp.c : **http://recca.rstack.org/dumpop.c**

[TINNES] Protection de l'espace d'adressage : état de l'art sous Linux et OpenBSD, MISC n°23.

[MS05-046] *Vulnerability in the Client Service for NetWare Could Allow Remote Code Execution* : **http://www.microsoft.com/technet/security/bulletin/MS05-046.mspx**

[XFOCUS] Article XFocus : **http://xfocus.net/articles/200303/497.html**

[SEKI] *Fingerprinting through RPC* : **http://www.blackhat.com/presentations/win-usa-04/bh-win-04-seki-up2.pdf**

[FPLANG] FpLang.c : **http://recca.rstack.org/fplang.c**

[MS05-010] *Vulnerability in the License Logging Service Could Allow Code Execution* : **http://www.microsoft.com/technet/security/Bulletin/MS05-010.mspx**

[SERVICES] Minimisation des services réseaux sur les systèmes Windows : **http://www.hsc.fr/ressources/breves/min_srv_res_win.html**

[WEHNUS] **Wehnus :** When you need to trust Windows : **http://www.wehnus.com/**

Protection de l'espace d'adressage : état de l'art sous Linux et OpenBSD

La protection de l'espace d'adressage d'un processus permet de rendre plus difficile l'exploitation de failles de sécurité. Plusieurs techniques existent, notamment mises en œuvre au niveau du noyau Linux, qui se montrent très efficaces, pour un coût presque négligeable en performances et en temps d'administration. Sous Windows, par manque de contrôle sur les éléments de bas niveau du système, les techniques les plus avancées sont difficiles à mettre en œuvre et les solutions actuelles, regroupées sous le nom de HIPS (Host instrusion prevention systems) n'ont pas la même efficacité.

1. Prévention générique d'exploitation de failles de sécurité

Introduction

Beaucoup savent qu'une simple erreur de programmation peut être exploitée par un attaquant pour exécuter du code arbitraire avec les privilèges du processus vulnérable. La possibilité d'exploiter certaines failles est souvent due à des détails d'implémentation de bas niveau, rarement connus du programmeur : par exemple le fait que des données soient mélangées à des structures de contrôle, que ce soit sur la pile (variables locales et adresse de retour sur architecture Intel), sur le tas (données et métadonnées de gestion du tas) ou dans une chaîne de format. De nombreux domaines, *a priori* orthogonaux, se trouvent mélangés. Qui aurait prévu que les choix a priori indépendants des encodages ASCII et des *opcodes* d'Intel ou l'adresse virtuelle choisie par le *linker* pour une fonction jouent un rôle dans la facilité d'exploitation d'une faille ? De ce fait, déterminer l'exploitabilité ou non d'une faille de sécurité est délicat. Il est cependant possible de rendre plus difficile, voire impossible, l'exploitation de certaines erreurs de programmation de manière générique, sans connaître la faille en question a priori.

Classification

Dans cet article, nous nous limiterons aux failles de sécurité dont l'exploitation se fait en contrôlant le flot d'exécution du processus. Cela exclut par exemple les erreurs logiques et un cas particulier de dépassement de tampon où l'on pourrait écraser une variable vitale gérant nos privilèges.

Les étapes de l'exploitation d'une faille (prenons ici pour simplifier un *buffer overflow*) sont généralement les suivantes :

1. Recherche d'une faille dans un programme ;
2. Communication avec le processus vulnérable afin de déclencher le dépassement de tampon ;
3. Injection de code arbitraire dans l'espace d'adressage du processus (facultatif) ;
4. Contrôle du flot d'exécution du programme (éventuellement pour le rediriger vers le code arbitraire préalablement injecté) ;
5. Détournement du processus pour réaliser diverses actions (installation de *rootkit*, récupération de certaines données, lancement d'un nouvel *exploit*) ;

Le séquencement de ces étapes n'est pas strict, on peut par exemple imaginer utiliser le contrôle du flot d'exécution du programme pour injecter du code arbitraire ou, ce qui arrive plus fréquemment, pour rendre des données injectées exécutables (nous reviendrons là-dessus).

Des mécanismes de prévention générique existent pour prévenir chacune de ces étapes :

1. Détection d'erreurs de programmation par analyse statique (sparse, Coverity, Microsoft PREfix/PREfast, etc.) ;
2. Empêcher que les conditions de la faille (par exemple l'overflow) se produisent à l'exécution (libsafe, BCC (*bound checking* GCC) et CASH) ;
3. Empêcher l'injection de code arbitraire dans l'espace d'adressage d'un processus [0] (PaX, OpenBSD's W^X) ;
4. Empêcher le contrôle du flot d'exécution par l'attaquant (SSP/Propolice), obscurcissement de l'espace d'adressage (PaX, Ozone, Whentrust) ;
5. Empêcher un processus contrôlé par l'attaquant de faire certaines actions en limitant ses droits (Système de contrôle d'accès, SELinux, TrustedBSD, RSBAC, GrSecurity's RBAC, LIDS, McAfee Entercept, CISCO CSA).

Dans cet article, nous traiterons des étapes 3 et 4 sous Linux et OpenBSD. L'étape 5 sera toutefois abordée car les cinq étapes ci-dessus ne sont pas totalement disjointes, et la correspondance avec les étapes d'exploitation n'est pas stricte : par exemple, un système de contrôle d'accès peut participer à empêcher l'injection de code arbitraire dans l'espace d'adressage d'un processus (nous y reviendrons). Le cas de Windows sera à peine abordé, nous y reviendrons sans doute dans un prochain article.

2. Éléments de système

Nous allons décrire quelques éléments du fonctionnement d'un système d'exploitation sur architecture Intel.

Une grande partie de la sécurité d'un système d'exploitation moderne n'est possible que grâce à l'existence d'une MMU (*Memory Management Unit*) dans le processeur utilisé. Grâce à celle-ci :

- Il existe un niveau de privilège **matériel** qui sépare le noyau du code exécuté en mode utilisateur.

[0] Dans cet article, « injecter du code » signifiera injecter ce code dans une zone exécutable ou injecter ce code dans une zone non exécutable, puis rendre cette zone exécutable.

Julien Tinnès
France Télécom Division R&D MAPS/NSS
julien.tinnes@francetelecom.com

1. Le noyau contrôle quels périphériques sont accessibles pour un processus en mode utilisateur (à l'aide du champ `IOPL` du registre EFLAGS et du bitmap de permissions enregistré dans le TSS). En règle générale, seul le noyau peut accéder directement aux périphériques et, en mode utilisateur, il faut utiliser des services du noyau (appels système) pour que le processus puisse exécuter des instructions (contrôlées) en mode noyau et accéder au matériel.
2. La mémoire d'un processus est virtualisée. Il se voit seul dans son espace d'adressage.

Ces trois éléments sont essentiels pour la sécurité car ils rendent possible une séparation des privilèges entre processus : le noyau étant un passage obligé pour la communication entre les processus et le matériel, il est le superviseur de la sécurité.

En effet, grâce à l'espace d'adressage virtuel, un processus doit passer par le noyau pour pouvoir par exemple modifier l'espace d'adressage d'un autre processus (`ptrace()`, création de zones de mémoire partagée..). C'est ce qui permet des modèles de séparation de privilège simples (par uid/gid) ou complexes (*mandatory access control* par exemple).

Cette vision des choses est très importante lorsque l'on étudie les systèmes de contrôle d'accès (RSBAC, SELinux, TrustedBSD...), c'est-à-dire lorsque l'on suppose qu'un attaquant possède déjà le contrôle d'un processus : le fait qu'il soit ensuite obligatoire de passer par le noyau pour « sortir de ce processus » (en ouvrant des fichiers, en infectant l'espace d'adressage d'un autre processus, en ouvrant des connexions réseau...) est alors capital.

Segmentation et pagination

Dans cet article cependant, nous allons voir comment l'on peut réagir un niveau auparavant, afin d'empêcher le contrôle du processus par l'attaquant.

Ce qui nous intéresse surtout est le procédé de virtualisation de l'espace d'adressage et nous allons l'étudier rapidement dans le cadre de l'architecture Intel IA32 en mode protégé.

Un programme travaille avec des adresses dites « logiques » : dans l'instruction `mov eax, dword ptr ss:[esp+4]`, si `esp` vaut 0xBFFFFF80, le mot contenu à l'adresse logique pointée par ss :0xBFFFFF84 est mis dans le registre EAX.

Afin de savoir quelle adresse (physique) il doit utiliser sur le bus mémoire, le processeur effectue deux transformations :

1. La première, appelée « la segmentation », transforme l'adresse logique en adresse linéaire (aussi appelée adresse virtuelle) ;
2. La deuxième, appelée « pagination », transforme l'adresse linéaire en adresse physique.

Figure 1



De l'adresse logique à l'adresse physique: la segmentation et la pagination

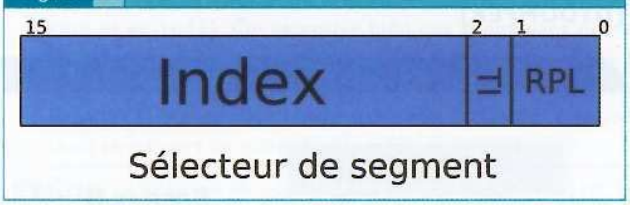
Unité de segmentation

Nous l'avons vu, les adresses mémoire utilisées par un programmeur sont des adresses logiques. Celles-ci sont de la forme « selector:offset » où « selector » est un sélecteur de segment de 16 bits et offset un décalage de 32 bits. Les sélecteurs de segment sont en pratique rarement spécifiés, car par défaut, (c'est-à-dire sans préfixe *segment override*), le segment de code (cs) est utilisé pour tous les *instruction fetch*, le segment de pile (ss) pour tous les `push`, `pop` et références utilisant ESP ou EBP, le segment de données (ds) pour toutes les références sauf celles liées à la pile ou aux chaînes, le sélecteur ES pour les instructions liées aux chaînes de caractères. Nous allons décrire, de manière peu détaillée (les lecteurs intéressés peuvent se reporter à la documentation **[Intel]**) le mécanisme de segmentation.

C'est dans la segmentation (et non pas dans la pagination) que l'on retrouve les fameux *rings*, c'est-à-dire les niveaux de privilèges Intel. Ils vont de 0 à 3, 0 étant le privilège le plus fort. En pratique les systèmes d'exploitation courants n'utilisent que les privilèges 0 (mode noyau) et 3 (mode utilisateur).

Considérons par exemple l'adresse logique précédente avec 7B pour valeur de SS (7B:0xBFFFF884). Le sélecteur de segment se décompose en trois champs (voir figure ci-dessous).

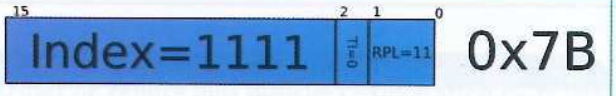
Figure 2



1. Le champ TI (table indicator) indique dans quelle table de descripteurs, GDT (global descriptor table) ou LDT (local descriptor table), le descripteur de segment correspondant doit être cherché.
2. Le champ Index indique l'index du descripteur de segment dans la GDT ou la LDT
3. Le champ RPL indique le requested privilege level, i. e. le plus haut privilège nécessaire pour avoir accès au segment. Dans le cas de CS, ce champ indique le CPL (current privilege level, c'est-à-dire le niveau de privilège courant (typiquement 0 en mode kernel et 3 en mode utilisateur).

Ainsi, notre sélecteur SS dont la valeur est 0x7B indique que le descripteur de segment à utiliser est le numéro 15 (0b1111) dans la GDT et que le RPL est 3.

Figure 3

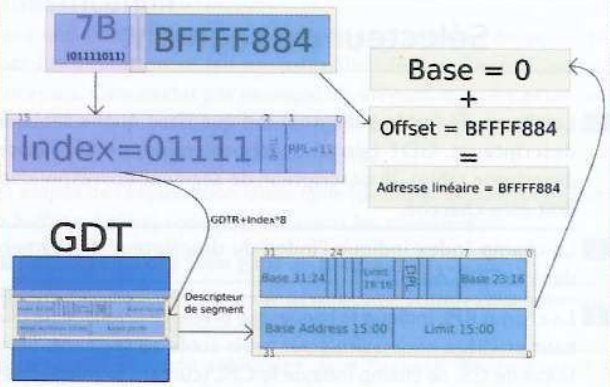


Exemple de sélecteur de segment

Grâce au sélecteur de segment, le processeur repère un descripteur de segment. Selon l'indicateur de table (TI), le descripteur de segment est recherché dans la LDT ou dans la GDT (les adresses linéaires de ces tables sont situées dans les registres LDTR et GDTR). À l'aide du champ `index`, le processeur repère le descripteur de segment désiré dans la table. Ce descripteur de 8 octets contient de nombreux champs, entre autres un champ `base` et un champ `limit` : l'addition de ce champ *base* et de l'offset donne l'adresse linéaire, le champ `limit` permet de limiter la taille du segment. Parmi les autres champs importants, citons le champ DPL (descriptor privilege level) qui, combiné au CPL (derniers bits de CS) et au RPL du descripteur utilisé permet de vérifier les privilèges.

Par défaut, la majorité des systèmes d'exploitation actuels utilisent le *Flat memory model*, c'est-à-dire qu'ils utilisent zéro comme base pour les principaux segments (ceux dont les sélecteurs sont chargés dans cs, ds, ss, es), au moins pour les segments utilisés en mode *user*, et 0xFFFFF comme limite (ce qui donne une limite de 4 Go au segment). L'offset d'une adresse logique peut ainsi dans la majorité des cas être identifiée à l'adresse linéaire. Ceci explique les nombreuses confusions entre adresse linéaire et adresse logique. La segmentation joue en général un rôle peu important dans les systèmes d'exploitation modernes, son utilisation est cependant obligatoire sur processeur Intel, pour des raisons historiques. Nous verrons cependant plusieurs méthodes pour tirer parti de la segmentation. Vous pouvez explorer vos descripteurs de segments à l'aide du programme **[DTDUMPER]**.

Figure 4



Segmentation: transformation d'une adresse logique en adresse linéaire

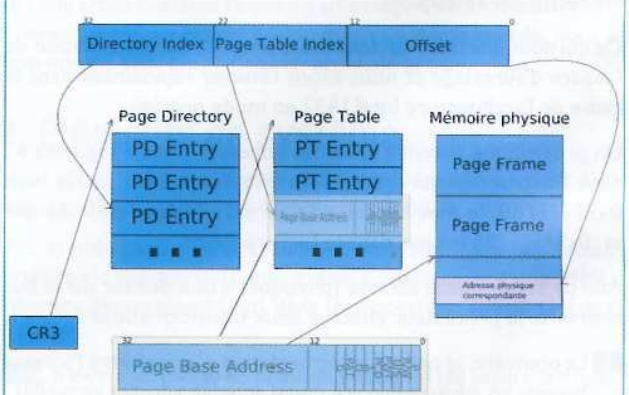
Unité de pagination

La pagination constitue la deuxième étape de la translation d'adresse : elle transforme les adresses linéaires en adresses physiques que le processeur envoie sur le bus mémoire. L'espace d'adressage linéaire est découpé en pages (en général de 4 Ko), la pagination fait la correspondance entre les pages et les cadres de pages (pages en mémoire physique). En pratique, sur la plupart des systèmes d'exploitation, c'est elle qui joue le rôle le plus important, elle permet de vérifier les droits d'accès aux pages (droit en écriture, privilège nécessaire pour accéder à la page)... et grâce à elle des mécanismes avancés sont possibles :

- Un cadre de page n'est pas forcément alloué en mémoire physique pour une page effectivement utilisable par le processus : l'utilisation de cette page provoquera une *page fault* et le *handler* correspondant du noyau allouera le cadre de page correspondant en mémoire physique, en y plaçant éventuellement les données nécessaires :
  - *Swaping* : des cadres de page peuvent être déchargés de la mémoire physique et placés sur le disque.
  - *On demand paging* : des fonctions telles que `mmap()` ou `MapViewOfFile` sont utilisées pour mapper un fichier en mémoire, les pages ne seront effectivement mappées qu'à l'utilisation.
- Le *copy on write* : deux processus sans mémoire partagée peuvent partager le même cadre de page ; lorsque l'un d'eux écrira sur sa page, le cadre de page sera dupliqué. Ce procédé est particulièrement avantageux pour mapper des bibliothèques.

Il existe plusieurs types de pagination sur les processeurs Intel. Nous allons pour simplifier décrire ici rapidement le mode de pagination simple avec pages de 4 Ko (sans PSE, sans PAE, sans IA32e). Dans ce mode, la pagination s'effectue à deux niveaux : une adresse linéaire contient un index de *Directory*, un index de *Table* (de 10 bits chacun, qui permettent de repérer une entrée dans une table de 1024 entrées) et un offset de 12 bits qui permet de se repérer à un octet près dans une page de 4 Ko.

Figure 5



Pagination: transformation d'une adresse linéaire en mémoire physique

Le registre cr3 contient l'adresse physique du *Page Directory*, cette adresse, combinée au *Directory Index* nous fournit l'adresse d'un PDE (*Page Directory Entry*), qui lui-même permet de repérer une *Page table* qui, combinée au *Table Index*, fournit l'adresse du PTE (*Page table entry*) de la page. Celui-ci contient un champ *Page base address* de 20 bits qui permet de repérer le cadre de page (*Page Frame*) en mémoire physique ainsi que divers *flags* qui indiquent notamment les droits de la page (*Read/Write*), ses privilèges (*User/Supervisor*; notons qu'il n'y a ici que deux niveaux et non pas 4 Rings), si elle est présente en mémoire physique, si elle a été accédée...

Nous ne détaillerons pas les PDE qui sont similaires aux PTE (voir Figure). Remarquons qu'un PTE permet de marquer une page comme étant disponible en écriture ou non (W), mais qu'il n'existe pas de flag pour marquer une page disponible en exécution.

L'absence de ce flag pose de gros problèmes de sécurité (il fut ensuite réintroduit en fanfare sous la dénomination de flag NX, nous y reviendrons). Notons que comme nous ne perdons pas d'information, les 32 bits d'une adresse linéaire permettent bien d'adresser 4 Go de mémoire.

Espace d'adressage d'un processus

Voyons à titre d'exemple comment se crée l'espace d'adressage d'un processus sous Linux 2.4. On peut voir la partie utilisateur de l'espace d'adressage linéaire d'un processus en lisant le fichier `/proc/pid/maps`. Sous Linux 2.4, tous les descripteurs de segment qui nous intéressent ont une base nulle; on peut donc ici directement confondre l'espace d'adressage logique (tel qu'il est vu par le processus) et l'espace d'adressage linéaire (après segmentation).

```
$ cat /proc/self/maps
08048000-0804c000 r-xp 00000000 03:03 32672      /bin/cat                [1]
0804c000-0804d000 rw-p 00003000 03:03 32672      /bin/cat                [2]
0804d000-0806e000 rwxp 00000000 00:00 0                                  [3]
40000000-40016000 r-xp 00000000 03:03 179102     /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:03 179102     /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
4001d000-40145000 r-xp 00000000 03:03 179607     /lib/libc-2.3.2.so
40145000-4014d000 rw-p 00127000 03:03 179607     /lib/libc-2.3.2.so
4014d000-40150000 rw-p 00000000 00:00 0
40150000-4019f000 r--p 00000000 03:03 942989     /usr/lib/locale/locale-archive
bfffe000-c0000000 rwxp fffff000 00:00 0                                  [pile]
```

L'espace d'adressage d'un processus est changé lors de l'appel système `sys_execve()` qui remplace le contexte d'exécution d'un processus (la création d'un nouveau processus est due à l'appel système `sys_fork()`).

L'espace d'adressage linéaire au-dessus de 3 Go (0xC0000000) est réservé au noyau et est mappé dans chaque processus (c'est-à-dire que les pages situées au-dessus de 0xC0000000 dans chaque processus pointent vers les mêmes cadres de page). Décrivons rapidement quelques étapes du remplacement du contexte d'exécution d'un processus réalisées par la fonction `sys_execve()`.

Les exécutables sous Linux utilisent généralement le format ELF. Un fichier ELF peut être vu de deux manières, selon le *header* considéré : le *Program Header* offre une vision adaptée à un chargeur de programme, alors que la *section header table* offre une vision adaptée à un linker statique.

Le Program Header décrit le fichier exécutable comme une suite de segments. Les segments d'un programme sont des informations pour le chargeur de programme, ils n'ont absolument **rien** à voir avec la segmentation Intel. Voici le program header de « `cat` ».

```
$ readelf -l cat

Elf file type is EXEC (Executable file)
Entry point 0x8048b70
There are 7 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x03a2d 0x03a2d R E 0x1000
  LOAD           0x003a3  0x0804ca30 0x0804ca30 0x001d0 0x0033c RW  0x1000
  DYNAMIC        0x003a7c 0x0804ca7c 0x0804ca7c 0x000c8 0x000c8 RW  0x4
  NOTE           0x000128 0x08048128 0x08048128 0x00020 0x00020 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r
          .rel.dyn .rel.plt .init .plt .text .fini .rodata
   03     .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
   04     .dynamic
   05     .note.ABI-tag
   06
```

Voici quelques étapes du chargement d'un programme :

1. La fonction `flush_old_exec()` du noyau libère les ressources utilisées par le précédent programme, en particulier toutes les régions mémoire *user*.
2. La pile est allouée, juste en dessous de la limite de l'espace utilisateur. Nous pouvons la voir ici de 0xbfffe000 à 0xc0000000.
3. Les segments de type `PT_LOAD` sont chargés en mémoire. On peut voir ici qu'il y a une zone mémoire avec les permissions `read` et `execute` [1] et une zone mémoire avec des permissions `read` et `write` [2]. On retrouve bien ces zones dans notre fichier maps. Notons que le deuxième segment a une `MemSiz` supérieure à sa `FileSiz`. Cela est à l'origine de la création de la zone [3] qui contient les données non initialisées (section `.bss`) et qui sert de point de départ au tas (*heap*).
4. Grâce au Program Header (repéré par son type `PT_INTERP`), le noyau localise le chargeur dynamique (`ld-linux.so`), le charge en mémoire et lui transfère l'exécution.
5. Le chargeur dynamique utilise l'appel système `sys_mmap()` pour charger les bibliothèques dynamiques (entrées `DT_NEEDED` dans la table dynamique) en mémoire. Ces bibliothèques dynamiques sont elles-mêmes au format ELF, de type `ET_DYN`.
6. Le chargeur dynamique réalise des relocations pour le programme principal et les bibliothèques chargées. C'est lui qui met éventuellement en place le mécanisme GOT/PLT.
7. Le chargeur dynamique transfère l'exécution au point d'entrée du programme

Le format ELF est très complexe (il est par exemple beaucoup plus complexe que le format PE de Microsoft). Les lecteurs intéressés peuvent se reporter à la spécification TIS **[ELF]**.

Le fichier maps de la page précédente correspond à un noyau 2.4. Avec un noyau 2.6 nous verrions quelques différences :

- Presque tout en haut de l'espace d'adressage linéaire (0xfffe000), une page est réservée par le noyau. Elle contient une bibliothèque ELF appelée `vDSO` qui est utilisée pour réaliser des appels systèmes. Celle-ci est chargée automatiquement en mémoire par le kernel dans tous les processus. Selon le type de processeur, cette bibliothèque réalise des appels système en utilisant `int 0x80/iret` ou `sysenter/sysexit`.
- Les adresses des bibliothèques dynamiques sont changées car l'allocation se fait maintenant du haut vers le bas.
- À partir du noyau 2.6.12 on peut constater une faible *randomisation* des adresses des bibliothèques dynamiques et de la pile (nous reviendrons dessus dans la partie consacrée à Exec-Shield).

3. Prévention d'exécution de code arbitraire

Nous décrivons ici plusieurs techniques visant à empêcher la prise de contrôle du processus par un attaquant. Comme nous l'avons expliqué, nous nous limiterons aux prises de contrôles fortes qui permettent de détourner le flot d'exécution du programme. Cela est réalisé en :

1. Empêchant l'attaquant de détourner le flot d'exécution du programme ;
2. Empêchant l'attaquant d'injecter du code arbitraire.

Un attaquant peut simplement détourner le flot d'exécution du processus sans injecter de code arbitraire. En revanche le contrôle est souvent beaucoup plus simple et plus fort en injectant du code arbitraire puis en détournant le flot d'exécution du processus vers celui-ci. Les choses se compliquent si l'on considère que l'on peut détourner le flot d'exécution du programme vers du code existant dans l'espace d'adressage, de sorte à injecter du code arbitraire et à l'exécuter.

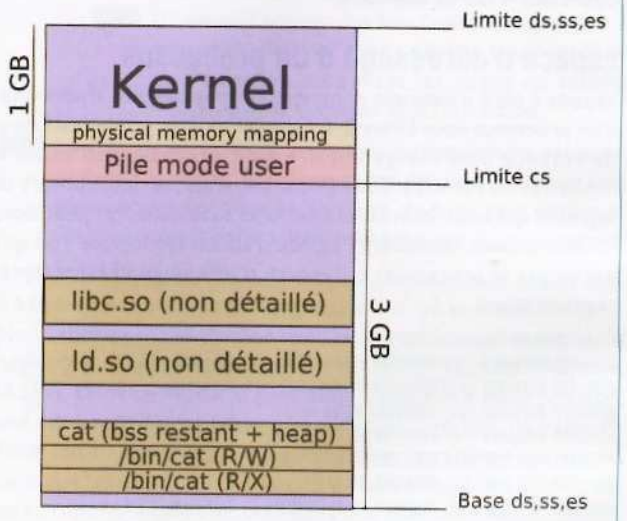
OpenWall

Le premier patch pour le noyau Linux destiné à rendre plus difficile l'exploitation d'erreurs de programmation fut OpenWall (sorti en 1998) [Openwall]. Bien avant l'ère des exploits de *format strings* (~2000) et avant la démocratisation des *heap overflows*, la plupart de ces erreurs de programmation avaient pour conséquence un dépassement de tampon sur la pile. La technique classique d'exploitation étant d'écraser une adresse retour afin d'exécuter du code injecté sur la pile, il semblait souhaitable de rendre la pile non exécutable.

On peut certes marquer la pile comme étant non exécutable (c'est-à-dire changer les données internes au noyau, les *virtual memory areas*), mais à cause de l'absence de flag marquant le droit d'exécution dans les PTE, cela n'impliquera pas de changement dans les structures utilisées par le processeur et n'aura donc aucun effet utile. Pour pallier ce défaut de l'architecture Intel, Solar Designer utilisa une technique relativement simple afin de rendre la pile non exécutable : en mode utilisateur, le descripteur de segment référencé par le registre cs possède une limite inférieure à l'adresse linéaire de la pile. Les autres descripteurs de segment sont inchangés.

Ainsi, lorsqu'un accès à la pile est réalisé relativement aux registres ds, ss et es, aucune différence n'apparaît. La base du descripteur de segment (0) est ajoutée à l'offset demandé, la somme ne dépassant pas la limite du segment (4 Go), aucune protection matérielle due à la segmentation ne vient perturber l'accès à la mémoire. En revanche, lorsque le processeur tente d'exécuter du code sur la pile, le descripteur de segment référencé par le sélecteur contenu dans le registre cs est utilisé. La somme de la base de ce descripteur (0) et de l'offset demandé dépasse la limite de ce segment (modifiée dans ce but), le processeur lève une exception de protection générale et le noyau Linux tue le processus.

Figure 6



Espace d'adressage linéaire sous OpenWall (sans ASCII armor)

Les limites de cette protection sont nombreuses. Il reste en effet possible d'injecter du code exécutable ailleurs que sur la pile et d'utiliser du code existant. Solar Designer fut également le premier à proposer d'utiliser la technique du *return-to-libc* afin de passer outre sa protection. Mais on peut également imaginer utiliser de nombreuses techniques plus avancées, par exemple des return-to-libc chaînés (voir l'article de Rafal Wojtczuk dans Phrack **[Wojtczuk]**), afin de recopier son code exécutable dans une autre zone mémoire puis de l'exécuter.

Afin de déjouer ces techniques visant à exploiter du code existant dans l'espace d'adressage, Solar Designer a proposé l'une des premières techniques d'obfuscation de l'espace d'adressage, l'armure ASCII. Le principe est que les bibliothèques sont chargées à une adresse inférieure à 0x01010000 afin que leurs adresses contiennent toujours un zéro. Cela peut en effet

empêcher l'injection d'une adresse de retour cible (par exemple celle de `system()`) dans l'espace d'adressage d'un processus via des chaînes de caractères. Cette méthode n'est bien entendue utile que s'il est difficile pour un attaquant d'injecter des zéros et il est toujours possible d'utiliser du code existant dans les zones mémoires correspondants au fichier exécutable principal, qui lui ne se trouvera pas dans l'*ASCII armor*.

Malgré ses limites très importantes, ce patch extrêmement simple déjoue un grand nombre des exploits publics visant des *stack overflows* (ce qui n'a certes pas grand chose à voir avec la sécurité réelle, n'en déplaise aux éditeurs d'antivirus).

PaX

PaX [PaX] est né en 2000, il était alors plus une preuve de concept exploitant une des idées du projet PleX86 (utiliser les TLB séparés pour pouvoir instrumenter séparément la lecture/écriture et l'exécution d'une page sur architecture Intel) qu'un véritable patch de sécurité. Il a évolué en introduisant successivement l'ASLR (*Address Space Layout Randomization*) en juillet 2001 puis le *VMA mirroring*, (base de SEGMEXEC et RANDEXEC) en juillet 2002, ainsi qu'en démocratisant des méthodes de compilation permettant d'obtenir des exécutables ET_DYN. Il supporte plus de dix architectures (Alpha, i386, ia64, Mips, Mips64, Parisc, PPC, PPC64, Sparc, Sparc64, x86_64) et sert de base aux distributions sécurisées Hardened Gentoo et Adamantix ainsi qu'au patch GrSecurity.

Nous l'avons vu, la pagination Intel fournit nativement un flag permettant de rendre ou non une page disponible en écriture dans les PTE. Idéalement, les protections d'une page devraient pouvoir être une combinaison arbitraire de PROT_READ, PROT_WRITE, PROT_EXEC (cette terminologie est celle utilisée notamment dans l'appel système `mprotect()` qui est utilisé pour changer les protections d'une zone mémoire), mais nous avons vu que seul PROT_WRITE avait un sens au niveau matériel.

Le premier rôle de PaX est d'émuler une sémantique de pages non exécutables sur les processeurs Intel qui ne le supportent pas, c'est-à-dire de donner une signification à PROT_EXEC (cette fonctionnalité s'appelle NOEXEC). Il existe deux techniques : la première, PAGEEXEC, induit une surcharge, la deuxième, SEGMEXEC, n'introduit pas de surcharge réelle mais ne permet d'utiliser que 1,5 Go de l'espace d'adressage virtuel utilisateur. Depuis l'introduction des Pentium 4 5xxj (puis 6xx supportant l'EM64T ou x86_64) et des Athlons 64, les processeurs supportent nativement cette fonctionnalité (appelée souvent flag NX). Dans ce cas, PaX (PAGEEXEC) utilise la fonctionnalité native du processeur (ce qui malheureusement nécessite l'utilisation du mode PAE du processeur, fastidieux et plus lent).

PAGEEXEC

Nous décrivons rapidement le fonctionnement de PAGEEXEC, pour plus d'informations, le lecteur pourra se reporter à la documentation de PaX **[PaX]**. Lorsque le processeur ne supporte pas physiquement la possibilité de marquer une page comme étant non exécutable, PaX utilise la séparation des TLB introduite avec les processeurs Pentium pour émuler cette fonctionnalité. Les TLB (*Translation Lookaside Buffers*) sont des caches situés dans le processeur qui contiennent les entrées récemment utilisées des Page Directories et Page Tables afin d'accélérer la traduction d'adresses linéaires en adresses physiques. Dans les processeurs récents, les TLB sont séparés selon que l'adresse translatée est utilisée pour récupérer du code (instruction `fetch`) (ITLB) ou pour écrire/lire en mémoire (DTLB). De plus, c'est au noyau d'invalider explicitement les TLB lorsqu'il modifie une Page Table ou un Page Directory. Cela signifie qu'il est possible pour le système d'exploitation de contrôler de manière fine les PTE contenus dans les TLB et en mémoire. Le fonctionnement de PAGEEXEC est le suivant :

- Les pages à rendre non exécutables sont marquées « superviseur » (flag U/S dans le PTE).
- Toute utilisation d'une telle page par le processus en mode utilisateur génère une faute. Le *Page fault handler* du noyau vérifie si cette faute est due à une tentative d'exécution de code (en comparant EIP du mode utilisateur et l'adresse de la page). Dans ce cas le processus est terminé (en réalité, les fonctionnalités EMUTRAMP et EMUSIGRT qui servent à émuler deux cas communs de génération de code à la volée, trampolines GCC et signal return du noyau sur la pile (vieilles libcs), compliquent ce processus).
- Dans le cas où l'accès demandé n'est pas en exécution, le PTE est modifié afin de permettre un accès à la page depuis le mode utilisateur, puis un accès est réalisé afin de charger le PTE correspondant dans le DTLB.
- Le drapeau `supervisor` est alors repositionné dans le PTE afin que tout accès en exécution soit à nouveau détecté.
- Lorsque l'exécution du processus en mode utilisateur reprend, tout accès en lecture/écriture utilisera le DTLB et ne provoquera pas de nouvelle faute alors qu'un accès en exécution passera par le PTE avec le drapeau superviseur.

Ce procédé est relativement coûteux en temps processeur. C'est pourquoi une autre méthode, SEGMEXEC, a été introduite en 2002. Cependant, fin 2004, PAGEEXEC a été amélioré dans le noyau 2.6. Il combine maintenant pagination et segmentation : une limite au segment de code est établie, au-dessus de la page exécutable la plus basse. De ce fait, toutes les pages situées au-dessus de cette limite ne sont pas exécutables grâce à la segmentation, ce qui a pour conséquence de ne pas imposer une surcharge lors de l'utilisation de ces pages. Les pages non exécutables situées en dessous de cette limite sont traitées avec les TLB séparés, comme expliqué ci-dessus. D'un point de vue pratique, cette nouvelle méthode est relativement efficace car les « grosses » zones de données (le tas par exemple), sont situées au-dessus de la limite de cs. La surcharge n'existe que pour les zones de données qui n'ont pas pu être logées au-dessus de cette limite (par exemple les sections `.data` des bibliothèques). Des mesures de performances sont disponibles sur [paxperf].

SEGMEXEC

L'introduction de SEGMEXEC dans la deuxième moitié de 2002 a été un pas majeur pour PaX. Grâce à cette nouvelle méthode, les surcharges induites par PaX sont devenues négligeables, au prix cependant d'une complexité d'implémentation accrue. SEGMEXEC est une méthode ingénieuse permettant d'obtenir des pages non exécutables en utilisant la segmentation, grâce à une technique appelée le *VMA mirroring*. Le prix à payer est que la

taille de l'espace des adresses logiques réellement utilisables par un processus en mode utilisateur passe de 3 Go à 1,5 Go.

L'idée est de séparer complètement le segment de code utilisateur du segment de données utilisateur. Nous l'avons vu, en temps normal les segments utilisateurs ont une base de 0 et une limite de 4 Go et se chevauchent donc parfaitement dans l'espace d'adressage linéaire : une adresse 0xAABBCCDD utilisée pour lire/écrire des données ou pour exécuter du code se traduira invariablement par une adresse linéaire 0xAABBCCDD. Imaginons maintenant que le segment de code et le segment de données soient disjoints dans l'espace d'adressage linéaire de la manière suivante :

- Un descripteur de segment de données « User » (USER_DS) a pour adresse de base 0 et pour limite 1,5 Go, celui-ci est référencé par les sélecteurs chargés dans ds, es et ss
- Un descripteur de segment de code « User » (USER_CS) a pour adresse de base 0x60000000 (1,5 Go) et pour limite 1,5 Go, celui-ci est référencé par le sélecteur chargé dans le registre cs

Grâce à cette technique, une page située dans le segment de code (c'est-à-dire entre 1,5 et 3 Go dans l'espace d'adressage linéaire) sera exécutable (accessible depuis cs) mais pas lisible/inscriptible (rappelez-vous que les accès aux données se font par rapport aux sélecteurs ds, ss ou es), alors qu'une page située dans le segment de données (c'est-à-dire entre 0 et 1,5 Go dans l'espace d'adressage linéaire) ne sera pas exécutable.

Afin d'être certain que des pages situées dans le segment de données ne puissent jamais être exécutées (le but étant par la suite de pouvoir **prouver** que l'injection de nouveau code exécutable dans l'espace d'adressage est impossible, cf. la partie dédiée à MPROTECT), il faut s'assurer qu'un autre sélecteur de segment moins restrictif ne puisse être chargé dans cs. C'est pourquoi PaX s'assure que la GDT utilisée par les processus utilisant SEGMEXEC possède un seul descripteur de segment de code en DPL3. En effet, il ne faut pas qu'un processus contrôlé par l'attaquant (par exemple à l'aide de return-to-libc chaînés) puisse être amené à charger dans cs un descripteur de segment lui permettant d'exécuter des pages contenues dans le segment de données (on pourrait imaginer par exemple que l'attaquant exécute un *far return* dont l'opcode était déjà présent dans la partie exécutable de l'espace d'adressage du processus).

Avons-nous à ce stade réellement une sémantique de pages non exécutables ? Pas vraiment, car notre nouvelle protection PROT_EXEC n'est pas ici combinable avec les protections PROT_WRITE et PROT_READ : il n'est pas possible de lire des données qui seraient situées dans une page exécutable. Cela est bloquant; si vous faites attention à la sortie de `readelf -l /bin/cat` ci dessus, vous verrez que la section `.rodata` (qui contient les données en lecture seule) par exemple se retrouve dans le segment (au sens ELF) de code. En pratique, il est, à de nombreuses occasions, nécessaire de pouvoir lire une page exécutable.

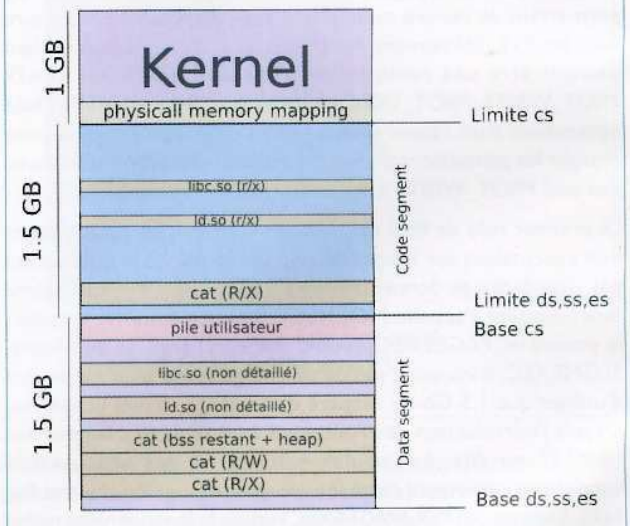
La solution à ce problème, le VMA mirroring, est ce qui rend SEGMEXEC complexe. Toute page disponible dans le segment de code doit être disponible également dans le segment de données. Pour cela, pour toute page présente dans le segment de code (adresses linéaires de 1,5 à 3 Go), un PTE « miroir » est ajouté dans le segment de données, référençant le même cadre de page que le PTE présent dans le segment de code (notez bien qu'il n'y a toujours qu'un cadre de page correspondant en mémoire physique !).

Notre espace d'adressage linéaire présente l'aspect suivant où l'on peut remarquer trois zones exécutables *mirorées* (voir également figure 7) :

```
08048000-0804c000 r-xp 00000000 03:03 32672     /bin/cat                [1]
0804c000-0804d000 rw-p 00003000 03:03 32672     /bin/cat
0804d000-0806e000 rw-p 00000000 00:00 0
20000000-20016000 r-xp 00000000 03:03 179102    /lib/ld-2.3.2.so        [2]
20016000-20017000 rw-p 00015000 03:03 179102    /lib/ld-2.3.2.so
20017000-20018000 rw-p 00000000 00:00 0
2001d000-20145000 r-xp 00000000 03:03 179607    /lib/libc-2.3.2.so      [3]
20145000-2014d000 rw-p 00127000 03:03 179607    /lib/libc-2.3.2.so
2014d000-20150000 rw-p 00000000 00:00 0
20150000-2019f000 r--p 00000000 03:03 942989    /usr/lib/locale/locale-archive
5fffe000-60000000 rw-p 00000000 00:00 0                                 [pile]
68048000-6804c000 r-xp 00000000 03:03 32672     /bin/cat                [1]
80000000-80016000 r-xp 00000000 03:03 179102    /lib/ld-2.3.2.so        [2]
8001d000-80145000 r-xp 00000000 03:03 179607    /lib/libc-2.3.2.so      [3]
```

De nombreuses modifications sont nécessaires afin que les deux pages « miroir » restent synchronisées en cas de changement d'état (lorsque le kernel gère un page fault, un `mremap()`, un `mprotect()`..), ce qui rend le VMA mirroring complexe.

Figure 7



Espace d'adressage linéaire sous PaX avec SEGMEXEC

Tous ces changements influent sur la segmentation et changent l'espace d'adressage linéaire.

Pour un processus, qui lui ne « voit » que l'espace d'adressage logique, rien n'a changé, si ce n'est que son espace d'adressage logique utilisable en mode User a maintenant une taille de 1,5 Go (en effet, les segments de code et de données ont tous les deux une taille de 1,5 Go et l'utilisation d'un offset supérieur à 1,5 Go produirait une exception de protection générale).

Le programme **[DTDUMPER]** affiche précisément les descripteurs de segments :

```
DT Dumper
julien@cr0.org

GDT size: 0x7F (16 entries), GDT LA: 0xC03028D0
LDT's selector is 0x68 (entry #13 in GDT)
TSS's selector is 0x60 (entry #12 in GDT)
IDT size: 0x7FF (256 entries), IDT LA: 0xC0302000
CS: 0x23        DS: 0x2B        SS: 0x2B        ES: 0x2B
Dumping GDT (0xC03028D0)
[...]
(#004) 0x23 60C5FB000000FFFF  Code  D/B=32bts  AVL=0  Present  DPL=3  TYPE= _RA
           BASE=0x60000000  LIMIT=0x5FFFFFFF
(#005) 0x2B 00C5F3000000FFFF  Data  D/B=32bts  AVL=0  Present  DPL=3  TYPE= _WA
           BASE=0x00000000  LIMIT=0x5FFFFFFF
[...]
```

MPROTECT

Nous avons vu deux méthodes, PAGEEXEC et SEGMEXEC, permettant d'obtenir des pages non exécutables, même sur une architecture Intel ne les supportant pas (sur les architectures le supportant nativement, PAGEEXEC utilise bien sûr le support natif du processeur). Les restrictions MPROTECT de PaX permettent d'utiliser ces pages non exécutables afin de contribuer à rendre impossible l'injection de code arbitraire dans l'espace d'adressage d'un processus. Le noyau Linux maintient pour chaque zone de mémoire linéaire (VMA pour Virtual Memory Area) un ensemble de drapeaux qui déterminent si la zone est exécutable, inscriptible, potentiellement exécutable ou potentiellement inscriptible (c'est-à-dire après un appel à `mprotect()`): VM_EXEC, VM_WRITE, VM_MAYEXEC, VM_MAYWRITE.

Les restrictions MPROTECT font en sorte qu'un VMA ne possède jamais une combinaison d'un drapeau WRITE et d'un drapeau EXEC. Pour cela :

- Les mappings anonymes (pile et tas en particulier) ainsi que les mappings de mémoire partagée sont automatiquement marqués VM_WRITE|VM_MAYWRITE.
- Les mappings de fichiers sont marqués avec VM_WRITE|VM_MAYWRITE si la protection PROT_WRITE a été demandée lors de l'appel de `mmap()` et avec VM_EXEC|VM_MAYEXEC dans le cas contraire.

Une exception existe afin de gérer le cas des fichiers ELF (le plus souvent des bibliothèques) ayant des relocations dans le segment de code (entrée DT_TEXTREL dans la table dynamique) : le linker dynamique doit pouvoir changer les droits d'une page, afin d'effectuer une relocation, puis rechanger les droits. Cette exception peut être supprimée si l'on est certain d'avoir un système ne contenant aucun fichier ELF avec des relocations dans le segment (au sens ELF) exécutable. Nous reviendrons sur ce sujet dans la partie suivante, consacrée à la mise aléatoire de l'espace d'adressage.

Les restrictions MPROTECT ne laissent qu'une possibilité à un attaquant pour introduire du nouveau code exécutable dans un processus dont il aurait pris le contrôle (en utilisant du code existant, à l'aide de return-to-libc enchaînés par exemple) : charger en mémoire à l'aide de `mmap()` un fichier dans lequel il a injecté du code, en demandant la protection PROT_EXEC. C'est avec cette même méthode que les bibliothèques utilisées par un programme sont chargées en mémoire. C'est le rôle d'un système de contrôle d'accès (RSBAC, SELinux, partie RBAC de GrSecurity) de prévenir cette attaque.

Si un tel contrôle d'accès est correctement mis en place, et si l'exception liée aux relocations TEXTREL est supprimée, on peut avec PaX prouver qu'il est impossible d'injecter du code arbitraire dans l'espace d'adressage du processus et cela sans faire aucune hypothèse sur le degré de contrôle d'un attaquant sur le processus !

Même sans la restriction du mappage de fichier avec la protection PROT_EXEC, notons que l'attaquant doit pouvoir bénéficier d'un grand contrôle du processus vulnérable (en ne s'appuyant que sur du code déjà présent dans l'espace d'adressage !) pour pouvoir lui faire créer un fichier, lui faire mapper ce fichier avec la protection PROT_EXEC, puis lui faire exécuter son code (évidemment le fichier peut être créé indépendamment, si l'attaquant a déjà un compte local par exemple).

Les restrictions MPROTECT sont évidemment incompatibles avec les quelques programmes générant du code à la volée (par exemple la machine virtuelle JAVA). Pour pallier ce problème, PaX permet de marquer tout fichier exécutable afin de désactiver certaines fonctions, dont MPROTECT.

Address space layout randomization (ASLR)

Comme nous l'avons mentionné, nous nous limitons dans cet article aux techniques permettant à l'attaquant d'avoir un contrôle fort du flot d'exécution du processus. Le contrôle le plus fort est l'exécution de code arbitraire, nous avons vu dans la partie précédente que PaX, s'il est bien utilisé, le prévient totalement.

Cependant il est également possible d'obtenir un contrôle relativement puissant du processus en réutilisant du code existant et en détournant le flot d'exécution. Le degré de contrôle est toutefois sans commune mesure avec celui obtenu avec l'exécution de code arbitraire.

C'est pourquoi des exploits utilisent souvent ces méthodes pour ensuite réussir à exécuter du code arbitraire, par exemple en utilisant `mprotect()` pour rendre un buffer exécutable puis en retournant sur celui-ci.

Dans le cas de PaX, ceci n'est pas possible, mais il reste quand même souhaitable d'empêcher le contrôle du processus à l'aide de code existant (sans compter que PaX n'est pas toujours utilisé de manière optimale et que les restrictions MPROTECT sont parfois retirées).

L'idée derrière la randomisation de l'espace d'adressage est d'empêcher toute adresse écrite en dur d'avoir un sens. Un exploit fiable doit au maximum éviter les adresses écrites en dur et utiliser des techniques plus avancées ou avoir recours à l'*information leak*, mais c'est loin d'être toujours possible.

Pour une version compilée donnée d'un programme les adresses dans le fichier ELF (ET_EXEC) sont par exemple considérées comme stables. La randomisation de l'espace d'adressage utilisateur se décompose en trois parties :

1. Randomisation de la pile utilisateur (RANDUSTACK)
2. Randomisation des zones mmap()ées (sans adresse fixée) (RANDMMAP)
3. Randomisation de l'exécutable principal (RANDEXEC)

Les deux premières randomisations sont simples (voir **[ASLR26]** pour une implémentation d'ASLR pur). Pour RANDUSTACK, les bits 4 à 27 sont randomisés (alignement sur 16 octets). Cela est réalisé au niveau de deux fonctions de `fs/exec.c` responsables de la création de la pile lors de l'appel système `execve()`. Pour RANDMMAP, la fonction `arch_get_unmapped_area()` (`mm/mmap.c`) est modifiée.

Celle-ci est responsable de la recherche d'une zone de mémoire libre lorsque le drapeau MAP_FIXED n'a pas été utilisé (c'est-à-dire lorsque l'appelant ne souhaite pas spécifier d'adresse précise où sera créé son mapping, qu'il soit anonyme où d'un fichier). Les bits 12 à 27 sont ici randomisés (alignement sur une page). Grâce à RANDMMAP, les adresses des bibliothèques, du tas lorsque la libc utilise `mmap()` pour le gérer (rare en pratique), et des requêtes manuelles de mappage de fichier ou anonyme qui n'utilisent pas le drapeau MAP_FIXED (c'est presque toujours le cas) sont automatiquement aléatoires.

Cependant, les adresses de l'exécutable principal (ET_EXEC) ne sont pas randomisées par RANDMMAP. Comme le tas, lorsqu'il est géré avec `brk()` (ce qui est courant) débute dans la section `.bss`, mappée avec l'exécutable principal, celui-ci n'est pas non plus randomisé (en fait une faible randomisation est tout de même ajoutée par PaX en « gâchant » volontairement de l'espace).

Cela est évidemment très gênant, d'autant plus qu'un attaquant peut ainsi retrouver les adresses de fonctions dans les bibliothèques (comme `system()`), en utilisant `dl-resolve()` et le mécanisme GOT/PLT **[Wojtczuk]** et passer ainsi outre la randomisation des bibliothèques.

Le problème avec l'exécutable principal est que celui-ci est placé en mémoire à partir d'un fichier ELF, au format ET_EXEC qui contient énormément d'adresses « en dur ». De plus ce fichier ne contient pas les informations de relocations nécessaires pour le reloger !

La solution consiste à générer des fichiers exécutables de type ET_DYN (type normalement réservé pour les bibliothèques). La PaX Team a introduit en 2003 une méthode pour générer de tels exécutables **[ET_DYN]**: en utilisant un crtl.o relogeable (utilisant la GOT), l'option `-shared` de GCC, et en spécifiant un lieur dynamique à utiliser, il est possible de générer un fichier exécutable ET_DYN. Notons que ceci n'était pas révolutionnaire, la libc est déjà un tel exécutable (vous pouvez essayer de l'exécuter !).

La génération de fichiers exécutables ET_DYN est également l'occasion d'éviter les relocations dans le code (cf. TEXTREL mentionnée dans la partie MPROTECT) en utilisant le *switch* `-fPIC` permettant d'obtenir du code indépendant de la position. Avec ces modifications, il est maintenant possible de randomiser l'exécutable principal : cela est fait dans la fonction `load_elf_binary()` du noyau et cela fait partie de l'option RANDMMAP de PaX.

Redhat s'est appuyé sur cette idée pour introduire en 2003 un switch `-pie` (*position independant executable*) à `ld` (et `-fPIE`, similaire à `-fPIC`, dans GCC), réalisant exactement les opérations décrites plus haut. Redhat a ensuite introduit le switch `-z relro` **[RELRO]** permettant de rassembler toutes les sections qui ne sont disponibles en écriture que pour des raisons de relocation et de créer un program header spécifique (PT_GNU_RELRO) afin que le lieur dynamique puisse s'il le souhaite réaliser un `mprotect()` sur cette zone après que les relocations aient été faites pour la mettre en lecture seule. Grâce à ce changement, on peut rendre la GOT disponible en lecture seule après le chargement du programme.

Lorsque notre système n'utilise que des exécutables de type ET_DYN, nous pouvons être dans une situation de *full ASLR*, où tout l'espace d'adressage du processus est aléatoire. C'est le cas dans les distributions **[Adamantix]**, Hardened **[Gentoo]**, et en partie dans les dernières Redhat. Il faut cependant garder à l'esprit plusieurs choses :

- Certaines erreurs de programmation permettent de faire du leak d'information, et de donner à l'attaquant des informations précieuses sur l'espace d'adressage, parfois des adresses complètes.
- Il est possible d'affaiblir la randomisation à l'aide d'un bourrage avec des instructions de type `"nop"`. Cela est particulièrement réaliste sur la pile où des bits de poids faible (à partir du 5ième) sont aléatoires.
- Tout n'est ici qu'une question de probabilités : si l'attaquant a de la chance ou peut utiliser la force brute aussi longtemps qu'il le souhaite, il finira par réussir. Il faut absolument utiliser des techniques anti-bruteforce, telles que SegvGuard ou celle utilisée dans **[GRSecurity]**.
- L'espérance mathématique lors d'une tentative de brute force est très différente selon que l'on crashe tout le démon (*one-shot*) ou seulement un fils. En effet, si les processus que l'on attaque sont tous issus d'un `fork()` du même démon, l'espace d'adressage est toujours semblable et nous en apprenons un peu plus après chaque tentative
- Lors d'une attaque locale il est facile d'obtenir des informations sur la mémoire d'un processus, même sans privilèges (par exemple `/proc/pid/maps`). Il faut utiliser un patch tel que **[PAX+OBS]** ou un système de contrôle d'accès pour l'éviter.

Notons pour être complet qu'il existe dans PaX une méthode, appelée RANDEXEC, ingénieuse, mais très coûteuse en performances, permettant de randomiser les exécutables de type ET_EXEC. L'idée repose sur le fait que beaucoup de branchements sont réalisés de manière relative et que, pour ceux-ci, il est possible de reloger l'exécutable principal « tel quel ».

L'exécutable est relogé, mais un miroir (utilisant le VMA-mirroring décrit dans la section SEGMEXEC) existe à l'adresse originale.

Ce miroir à l'adresse originale n'est pas exécutable (cette partie repose sur PAGEEXEC ou SEGMEXEC [notons que dans le cas de SEGMEXEC il n'y aura pas 3 miroirs comme on pourrait s'y attendre car on n'a pas besoin d'accéder aux données de manière relative, et on ne veut pas qu'il soit possible d'accéder au code de manière absolue]).

Les adresses absolues peuvent donc être utilisées pour écrire ou lire des données. Si une adresse absolue est utilisée pour exécuter du code, une exception se produira et PaX va utiliser une heuristique pour déterminer s'il s'agit d'une exécution légitime ou d'une attaque. Cette méthode est simplement une preuve de concept et ne doit pas être réellement utilisée à cause

de la faiblesse de l'heuristique et des coûts en performances. Elle a été retirée dans les dernières version de PaX pour noyaux 2.6.

Le programme **[PAXTEST]** est capable de vérifier la randomisation par des tests statistiques. On retrouve des valeurs proches des valeurs théoriques (16 bits pour RANDMMAP et 24 pour RANDUSTACK), sauf pour le heap qui bénéficie d'un traitement spécial (randomisation par « perte de place ») :

```
Anonymous mapping randomisation test     : 16 bits (guessed)
Heap randomisation test (ET_EXEC)        : 13 bits (guessed)
Heap randomisation test (ET_DYN)         : 25 bits (guessed)
Main executable randomisation (ET_EXEC)  : No randomisation
Main executable randomisation (ET_DYN)   : 17 bits (guessed)
Shared library randomisation test        : 16 bits (guessed)
Stack randomisation test (SEGMEXEC)      : 23 bits (guessed)
Stack randomisation test (PAGEEXEC)      : 24 bits (guessed)
```

Exec-Shield

Exec-Shield a été annoncé en 2003 par Ingo Molnar, un « *kernel hacker* » réputé de chez Redhat. À son annonce, exec-shield était une évolution d'OpenWall, dont l'idée était d'avoir une limite dynamique (au lieu de statique) pour le segment de code, correspondant pour tout processus à l'adresse mémoire exécutable la plus haute de son espace d'adressage, ce qui permet, contrairement à OpenWall, de rendre non exécutables d'autres zones que la pile.

De plus, Exec-Shield utilise également l'armure ASCII et y place les mappings PROT_EXEC (donc les bibliothèques), afin de rendre potentiellement plus difficiles certains return-to-libc et de concentrer le maximum de code exécutable le plus bas possible afin d'avoir le segment de code le plus petit possible. Voici les descripteurs de segment d'un processus **[DTDUMPER]** et son fichier maps (noyau 2.4), on peut voir la limite d'exécutabilité à 0x804A000:

```
DT Dumper
julien@cr0.org

(#004) 0x23 00C0FB0000000049  Code  D/B=32bts  AVL=0  Present  DPL=3 TYPE= cRA
BASE=0x00000000  LIMIT=0x00049FFF
(#005) 0x2B 00CFF3000000FFFF  Data  D/B=32bts  AVL=0  Present  DPL=3  TYPE= eWA
BASE=0x00000000  LIMIT=0xFFFFFFFF

00c15000-00d3d000 r-xp 00000000 03:02 765565     /lib/libc-2.3.2.so
00d3d000-00d45000 rw-p 00127000 03:02 765565     /lib/libc-2.3.2.so
00d45000-00d48000 rw-p 00000000 00:00 0
00d5a000-00d70000 r-xp 00000000 03:02 765562     /lib/ld-2.3.2.so
00d70000-00d71000 rw-p 00015000 03:02 765562     /lib/ld-2.3.2.so
08048000-0804a000 r-xp 00000000 03:02 162889     /root/es/dtdumper
0804a000-0804b000 rw-p 00001000 03:02 162889     /root/es/dtdumper
09da3000-09dc4000 rw-p 00000000 00:00 0
200b3000-200b5000 rw-p 00000000 00:00 0
bffad000-c0000000 rw-p fffb8000 00:00 0
```

Depuis 2003, Exec-Shield a légèrement évolué. Outre quelques corrections de bugs exploitables, il a repris l'idée de randomisation utilisée dans PaX et tire en particulier profit des exécutables ET_DYN générés à l'aide des switches `-fPIE` et `-pie` que Redhat utilise de plus en plus pour générer les exécutables « sensibles ».

Il est intéressant de noter qu'Exec-Shield a été voulu comme un patch simple et léger, la sécurité n'étant pas la préoccupation fondamentale de son auteur. L'approche d'Exec-Shield a de nombreuses limites :

- On n'a pas de réelle sémantique des pages non exécutables, cela se traduit par un problème important : certaines zones de données seront situées sous la limite du segment de code et seront donc exécutables. En particulier, toutes les zones correspondant aux sections `.data` ou `.bss` des bibliothèques seront donc exécutables. Pour certains programmes, cela peut être bien pire si une zone exécutable est chargée à des adresses hautes pour une raison ou pour une autre.
- Exec-Shield a longtemps été incompatible avec le vDSO (`linux-gate.so.1`, le système introduit dans Linux 2.6 permettant de tirer parti des appels système rapides de Intel à l'aide de sysenter/sysexit) : en effet, il s'agit d'une zone exécutable située tout en haut de l'espace d'adressage, ce qui est incompatible avec une limite basse pour le segment de code. Cependant, en juin 2005, Roland McGrath a proposé un patch grâce auquel il est possible de reloger le vDSO, qui est donc maintenant relogé et même randomisé dans l'ASCII armor comme une bibliothèque normale.
- Il n'y a pas d'équivalent aux restrictions MPROTECT de PaX : un attaquant ayant acquis un certain degré de contrôle du processus (par exemple à l'aide de return-to-libc enchaînés) peut faire un `mprotect()` afin de rendre la pile (et donc tout l'espace d'adressage car la pile est située en haut !) exécutable ! Contrairement à PaX qui distingue complètement l'ASLR de l'anti-injection de code exécutable, Exec-Shield repose indirectement sur l'ASLR pour rendre potentiellement plus difficile (et non pas impossible) l'injection de code exécutable.
- La randomisation dans Exec-Shield est plus faible que celle de PaX, notamment à cause du fait que les bibliothèques et l'exécutable principal doivent êtres « groupés » sous la limite du segment de code :

```
Anonymous mapping randomisation test     : 8 bits (guessed)
Heap randomisation test (ET_EXEC)        : 13 bits (guessed)
Heap randomisation test (ET_DYN)         : 13 bits (guessed)
Main executable randomisation (ET_EXEC)  : No randomisation
Main executable randomisation (ET_DYN)   : 12 bits (guessed)
Shared library randomisation test        : 12 bits (guessed)
Stack randomisation test                 : 19 bits (guessed)
```

La « fonctionnalité » la plus controversée introduite par Exec-Shield est sans nul doute PT_GNU_STACK. Les autres fonctionnalités s'inscrivent bien dans le modèle « faire le plus possible dans un patch simple et léger sans trop se soucier de la sécurité », mais PT_GNU_STACK introduit de réels problèmes et est un non-sens du point de vue de la sécurité. PT_GNU_STACK est un drapeau du program header permettant de marquer un fichier ELF (programme ou bibliothèque) comme « ayant besoin d'une pile exécutable ».

Ce marquage est réalisé de manière automatique par la *toolchain* (GCC, ld,...) en détectant des constructions ayant besoin d'une pile exécutable (en pratique, PT_GNU_STACK se limite aux trampolines GCC). Le lieur dynamique tire parti de cette information lors du chargement du programme principal ou d'une bibliothèque pour, si l'un au moins de ces composants réclame une pile exécutable, appeler la fonction `make_stack_executable()`.

Le problème de cette approche est qu'il y a inévitablement des faux positifs et des faux négatifs. De plus, un lieur dynamique

supportant cette fonctionnalité est incompatible avec PaX, puisque PaX verrouille à l'`execve()` les « privilèges » (droit de mprotect PROT_EXEC par exemple) du processus, alors que PT_GNU_STACK est par essence dynamique.

Avec un lieur dynamique supportant PT_GNU_STACK, si une bibliothèque marquée comme utilisant une pile exécutable est chargée, la pile sera alors marquée exécutable (le processus et donc l'attaquant, a le droit de rendre la pile exécutable dans le modèle PT_GNU_STACK), PaX l'interdira, ce qui conduira le lieur dynamique à abandonner le chargement.

Notons aussi que l'existence même d'une fonction `"make_stack_executable()"` est hallucinante **[!]** puisque cette fonction devient évidemment la cible privilégiée pour un attaquant capable d'appeler du code de son choix existant dans le processus : en effet, comme nous l'avons mentionné, rendre la pile exécutable sous exec-shield le désactive complètement. Nous voyons là encore qu'exec-shield repose complètement sur sa randomisation (dont la limitation à 12 bits pour les bibliothèques est ici particulièrement ennuyeuse).

Malgré ses nombreuses limitations, grâce à la notoriété d'Ingo Molnar, Exec-Shield fait peu à peu son entrée dans le noyau Linux. La version 2.6.12 du noyau a été la première à inclure une version allégée de la randomisation de l'espace d'adressage d'Exec-Shield (désactivable avec `/proc/sys/kernel/randomize_va_space`). Voici le résultat de [paxtest] :

```
Anonymous mapping randomisation test     : 8 bits (guessed)
Heap randomisation test (ET_EXEC)        : No randomisation
Heap randomisation test (ET_DYN)         : No randomisation
Main executable randomisation (ET_EXEC)  : No randomisation
Main executable randomisation (ET_DYN)   : No randomisation
Shared library randomisation test        : 10 bits (guessed)
Stack randomisation test                 : 19 bits (guessed)
```

OpenBSD's W^X

W^X (W xor X) est apparu dans OpenBSD 3.3 en mai 2003. Il a pour but d'empêcher l'existence de pages à la fois disponibles en écriture et en exécution. Dans la version 3.3 d'OpenBSD, W^X ne fonctionnait que sur les processeurs supportant le marquage non exécutable d'une page. W^X pour i386 est apparu avec la version 3.4 d'OpenBSD en novembre 2003.

L'approche d'OpenBSD est de ne réaliser que des modifications simples dans le noyau et de réaliser le gros du travail en *userland*. Là encore, sur i386, la segmentation est utilisée : le segment de code a une limite à 512 Mo, ce qui signifie que toute adresse (dans l'espace logique) supérieure à 0x20000000 ne sera pas exécutable.

La première étape pour la réalisation de W^X (utile même sur les processeurs ayant un marquage NX) a été de s'arranger pour ne plus avoir besoin de zones à la fois exécutables et disponibles en écriture. Pour cela le trampoline `'sigreturn'` a été sorti de la pile (contrairement à ce qui a été fait sous Linux, il n'est cependant pas intégré dans la libc sous OpenBSD) et la GOT et la PLT ne sont plus disponibles en écriture (le linker met/retire les protection à l'exécution selon les besoins, cette approche est à mettre en parallèle avec celle de **[RELRO]**).

D'autres assainissements ont également été réalisés, par exemple les sections `.ctors` et `.dtors` qui contiennent des pointeurs ne sont plus disponibles en écriture et la section `.rodata` n'est plus exécutable : vous pouvez consulter à ce sujet les *slides* de Theo **[W^X]**.

La deuxième étape dans la réalisation de W^X sous i386 fut de s'arranger pour mapper toutes les zones de données au-dessus de la limite des 512 Mo et toutes les zones exécutables en dessous. De nombreux changements ont été réalisés au niveau du chargeur dynamique pour rendre possible ce mappage séparé, où les bibliothèques et l'ELF principal ne sont plus chargés d'un bloc.

OpenBSD utilise également un ASLR similaire à celui de PaX, mais plus limité. Voici le résultat de **[PAXTEST]** :

```
Anonymous mapping randomisation test     : 20 bits (guessed)
Heap randomisation test (ET_EXEC)        : No randomisation
Main executable randomisation (ET_EXEC)  : No randomisation
Shared library randomisation test        : 16 bits (guessed)
Stack randomisation test                 : 15 bits (guessed)
```

Comme Exec-Shield, OpenBSD n'offre pas d'équivalent aux restrictions MPROTECT de PaX. De ce fait, le processus (et donc potentiellement l'attaquant !) est autorisé à rendre une page exécutable. Nous avons également pu remarquer à l'aide de **[DTDUMPER]** que plusieurs entrées dans la LDT et la GDT sont des segments de code parfaitement valides pour exécuter du code n'importe où dans la partie utilisateur de l'espace d'adressage (les limites 0x1FFFFFFF correspondent à la limite de 512 Mo alors que les limites 0xCFBFDFFF couvrent tout l'espace utilisateur).

Cela constitue un énorme trou de sécurité puisqu'il suffit de réaliser un branchement inter-segment (*far call, far jump, far ret,...*) afin d'exécuter du code dans une zone censée être non exécutable ! Voici les segments de code utilisables :

```
DT Dumper
julien@cr0.org

GDT size: 0xFFFF (8192 entries), GDT LA: 0xE7B25000
LDT's selector is 0x18 (entry #3 in GDT)
TSS's selector is 0x168 (entry #45 in GDT)
IDT size: 0x7FF (256 entries), IDT LA: 0xD05CEF60
CS: 0x1F    DS: 0x27    SS: 0x27    ES: 0x27

Dumping GDT (0xE7B25000)
[...]
(#004) 0x23 00CCFB000000FBFD  Code  D/B=32bts  AVL=0  Present  DPL=3  TYPE= cRA
           BASE=0x00000000  LIMIT=0xCFBFDFFF
(#005) 0x2B 00C1FB000000FFFF  Code  D/B=32bts  AVL=0  Present  DPL=3  TYPE= cRA
           BASE=0x00000000  LIMIT=0x1FFFFFFF
[...]
Dumping LDT (0xD05CEEC0)
(#002) 0x17 00CCFB000000FBFD  Code  D/B=32bts  AVL=0  Present  DPL=3  TYPE= cRA
           BASE=0x00000000  LIMIT=0xCFBFDFFF
(#003) 0x1F 00C1FB000000FFFF  Code  D/B=32bts  AVL=0  Present  DPL=3  TYPE= cRA
           BASE=0x00000000  LIMIT=0x1FFFFFFF
[...]
```

Par exemple, dans le cas d'un débordement de tampon sur la pile, W^X ne sert absolument à rien car il suffit d'effectuer un far ret vers le segment de code illimité. Par chance (pour les pirates), l'opcode de far ret (0xCB ou 0xCA) ne prend qu'un octet, il est

[1] Comme dit mon ami Yoann Guillot, il manque give_me_a_root_shell().

donc très simple de le trouver dans des zones exécutables à adresse fixe (afin de ne pas avoir à déjouer l'ASLR) : par exemple la section `.text` de l'exécutable principal (OpenBSD ne propose pas de système d'exécutables relogeables ET_DYN).

Il suffit donc en cas de stack overflow de préparer la pile : le descripteur 0x23 (ou 0x17), l'adresse de retour classique (qui peut être l'adresse du shellcode si on la connaît ou quelque chose de plus compliqué..), l'adresse d'un octet 0xCB. Le code ci-dessous **[RETF_DEMO]** simule cette situation (en construisant la pile à l'aide de `push` au lieu d'un overflow) avec un shellcode sur la pile et un retour sur un `jmp esp` :

```
; This would be in our target program
fret    db 0xCB
runstack:
jmp esp         ; obviously we execute code on the stack

; Entry point
global _start
_start:

; This is our payload
push 0xFEEB     ; shellcode (this is jmp -2)
push 0x17       ; our segment selector
push runstack   ; this is the classic return address
                ; don't expect to have a jmp esp in real-life though ;)
push fret       ; finding a static offset with 0xCB in standard ELF's .text
                ; is very easy

; This is the standard ret after we control the stack
ret
```

Quand la fonction vulnérable retourne, l'exécution reprend sur un far ret, ce qui a pour conséquence de retourner sur l'adresse `runstack` tout en chargeant le sélecteur de segment sur la pile (0x17) dans CS. `runstack` peut être l'adresse de tout code menant à l'exécution de notre shellcode (par exemple directement l'adresse du shellcode, si on la connaît !).

Notons toutefois que pour le cas précis des débordements sur la pile, un autre dispositif de prévention est utilisé dans OpenBSD : Propolice **[SSP]**.

Comparaison

Nous avons vu plusieurs techniques qui essaient d'empêcher l'injection de code arbitraire dans un processus. Nous avons vu que parmi celles-ci, l'approche de PaX est la seule qui permette de réellement prouver qu'il n'est pas possible d'injecter du code (en dehors du `mmap()` PROT_EXEC d'un fichier, à gérer via un système de contrôle d'accès).

L'approche de PaX consiste à ne pas faire d'hypothèse sur le degré de contrôle de l'attaquant et d'empêcher le processus lui-même d'injecter du nouveau code à l'exécution, c'est une application du principe de moindre privilège. Dans PaX empêcher l'attaquant de détourner le flot d'exécution du programme se fait à l'aide de la randomisation qui est une fonctionnalité à différencier complètement de la partie anti-injection de code.

PaX est particulièrement efficace lorsqu'il est utilisé au sein du *framework* **[GRSECURITY]** qui apporte certaines des fonctionnalités manquantes : anti *info-leak*, anti bruteforce et système de TPE (*Trusted Path Executable*) ou RBAC (*Role Based Access Control*) pour lutter contre les `mmap()` PROT_EXEC de fichiers contrôlés par l'attaquant.

À l'opposé, OpenBSD et Exec-Shield laissent à l'application le privilège de pouvoir injecter du code exécutable et s'appuient sur la randomisation pour éviter le détournement du flot d'exécution du programme (qui une fois acquis par l'attaquant peut donc être utilisé pour injecter du code exécutable).

Cette approche est surtout efficace lorsque l'attaquant a un faible contrôle du flot d'exécution (par exemple en cas de heap overflow), mais est très limitée avec un contrôle plus fort permettant d'enchaîner les appels de fonctions (par exemple en cas de débordement sur la pile).

Cette approche est également limitée par l'existence de potentiels bugs permettant l'*information leaking* (ou *features*, cf. `/proc/pid/maps` en local sous Linux) et oblige à se défendre contre le bruteforce pour rester efficace. En revanche, OpenBSD comme Exec-Shield y gagnent beaucoup en simplicité du code du noyau.

Autres considérations

Protection d'un noyau

Ce thème avait déjà été évoqué à une *Rump Session* de SSTIC **[SECULINUX]**. À l'heure actuelle, lorsqu'un pirate a déjà un accès local à une machine sous GNU/Linux et désire élever ses privilèges, les failles du noyau sont souvent la meilleure option. La plupart des binaires privilégiés sont largement audités et au fil du temps de mieux en mieux sécurisés.

Au contraire, le noyau Linux est en développement constant, le code est complexe et rarement audité et de nombreuses vulnérabilités sont présentes dans ses millions de lignes de code. D'autant plus qu'en mode noyau l'erreur ne pardonne vraiment pas : plus que jamais le moindre bug peut se traduire en faille de sécurité.

Pour ajouter encore à cela, le paradigme est complètement différent lorsque l'on essaie d'exploiter un bug du noyau : on possède déjà l'exécution de code arbitraire en mode utilisateur et l'on veut simplement augmenter ses privilèges. Cela signifie que l'on peut créer un processus dont on contrôle parfaitement l'espace adressage en mode utilisateur et que l'on peut exécuter le code de notre choix pour créer la situation où la faille pourra être exploitée.

Par exemple, les déréférences de pointeurs NULL qui sont, lorsqu'on exploite un programme en mode user, difficilement exploitables pour le commun des mortels **[DELALLEAU]** deviennent exploitables très facilement puisqu'on contrôle la partie basse de l'espace d'adressage.

Protéger le noyau avec des méthodes classiques telles que rendre la pile noyau non exécutable, rendre son adresse aléatoire (RANDKSTACK dans PaX) ou recompiler le noyau avec **[SSP]** (OpenBSD) ne sert en général à rien : presque toute erreur de programmation dans le noyau est une vulnérabilité et il est difficile de les classifier.

On voit finalement très peu d'erreurs classiques telles que les débordements de tampon. De plus comme le code du noyau est le plus privilégié, il n'est pas possible d'appliquer une politique de « moindre privilège » comme le fait PaX pour les processus en mode utilisateur : on ne peut pas empêcher un noyau compromis d'effectuer certaines opérations.

Il est donc actuellement très difficile d'empêcher un attaquant d'exploiter des failles dans son noyau. La seule méthode est sans doute de recourir à un système de contrôle d'accès ou un TPE (*Trusted Path Execution*) pour empêcher les démons et les utilisateurs locaux d'exécuter du code qui n'a pas été préalablement validé par l'administrateur.

Là encore le changement de paradigme est lourd de conséquences : on doit sans doute veiller à interdire l'utilisation d'interpréteurs (Perl, Python, Ruby...) qui permettent plus ou moins à leur utilisateur d'exécuter du code arbitraire (en tout cas d'avoir un grand degré de contrôle du processus de l'interpréteur).

Il faut également voir que dans cette situation, tout exécutable autorisé par l'administrateur devient privilégié (puisqu'il est autorisé à exécuter du code), et devient donc une cible pour l'attaquant : un bug exploitable dans `/bin/ls` donne à l'attaquant l'occasion d'exécuter du code arbitraire et d'exploiter une faille noyau. Cette situation est délicate, car tous ces programmes n'ont pas été conçus pour être considérés comme privilégiés et sont très rarement audités.

Le cas de Windows

Il est difficile pour un éditeur différent de Microsoft de protéger l'espace d'adressage car cela nécessite des modifications au cœur du système. Microsoft a cependant, avec Windows XP SP2 et Windows server 2003, commencé à introduire quelques éléments de prévention générique. Outre les modifications au niveau du compilateur, proches de SSP/Propolice, on peut notamment citer l'utilisation du flag NX lorsqu'il est présent afin de rendre certaines zones de données non exécutables. Il n'y a pas cependant sous Windows de mécanisme qui correspondrait aux restrictions `mprotect()` de PaX.

Cependant, on voit apparaître depuis environ un an des solutions HIPS. Si certaines comme Ozone ou Whentrust font de l'ASLR, la plupart des logiciels de « gros » éditeurs (CISCO CSA, McAfee Entercept) sont des systèmes de contrôle d'accès.

La mise aléatoire de l'espace d'adressage souffre de plusieurs limitations importantes sous Windows : outre la possibilité pour un attaquant, commune à tous les OS, de réaliser de l'information leaking, il est très difficile de reloger certaines DLL, notamment `ntdll.dll` et `kernel32.dll` à cause du processus d'amorçage de Windows. De plus, les exécutables PE n'ont pas les informations de relocation nécessaires et ne peuvent donc pas être relogés.

Dans ces conditions, l'attaquant possède au moins une source fiable d'opcodes à adresses fixes (`kernel32` et `ntdll` étant susceptibles d'évoluer avec les services packs), et les exploits fiables sous Windows n'utilisent depuis longtemps pas directement l'adresse présumée de la pile : une technique courante en cas de stack overflow est d'écraser une structure `EXCEPTION_REGISTRATION` d'un *Frame-Based exception Handler* et d'écraser le pointeur vers le handler avec une adresse contenant un `jmp ebx` (depuis Windows XP SP1, EBX ne pointe plus vers la structure `EXCEPTION_REGISTRATION`, cependant un pointeur est toujours présent sur la pile à esp+8, et on peut donc utiliser des instructions telles que `pop xxx; pop yyy; ret` présentes dans notre PE).

Windows XP SP2 vérifiant maintenant que le handler appelé est enregistré (ou qu'il n'est pas dans un module PE avec un *Load Configuration directory*) ces techniques évoluent de plus en plus.

Les systèmes de contrôle d'accès (appelés souvent systèmes de détection comportementale sous Windows), sont utilisés de manière classique pour confiner un processus contrôlé par l'attaquant, mais aussi pour empêcher l'appel d'un NSS (*Native System Service*, les appels système sous Windows) ou d'une fonction d'une bibliothèque depuis des zones jugées anormales comme la pile.

Le paradigme dans lequel travaillent ces logiciels est que l'attaquant a déjà l'exécution de code arbitraire et qu'il faut essayer de le détecter le plus tôt possible. Pour cela, des *hooks* sont réalisés au niveau de certaines bibliothèques et des NSS et une analyse sont réalisés afin de déterminer si la situation est saine : par exemple ces produits peuvent vérifier que l'adresse de retour n'est pas située dans la pile.

L'exécution de code arbitraire, même sans pouvoir appeler de fonction de bibliothèques ou de NSS permet déjà de faire énormément de choses.

On peut obtenir beaucoup d'informations sur l'espace d'adressage du processus exploité en utilisant le TEB et le PEB que l'on peut retrouver via le sélecteur de segment fs et un offset fixe (ce qui rend la randomisation du TEB/PEB (par rapport au segment ds) réalisée par Windows XP SP2 inutile une fois l'exécution de code arbitraire acquise par l'attaquant). Ces informations peuvent ensuite être exploitées :

- On peut recopier son shellcode dans une zone mémoire jugée « saine » (en général, cela nécessite un peu de travail car une zone mémoire disponible en écriture ne devrait pas être jugée saine).
- On peut mettre à profit du code existant dans l'espace d'adressage afin qu'il réalise les appels de fonctions pour nous, tout en réalisant un « lifting » approprié de la pile pour simuler une situation normale.
- On peut, si l'on sait d'avance quel HIPS on veut contourner « suivre les hooks » pour retrouver la fonction originale.

Ces techniques sont toujours utilisables, mais sont plus ou moins complexes à mettre en œuvre selon la qualité de l'heuristique du produit à contourner.

Il faut également noter que les hooks de bibliothèques en mode user peuvent être contournés en appelant directement les NSS (pour rester générique, on peut utiliser NTDLL.DLL pour en retrouver les numéros).

Remerciements

Pipacs et Bradley Spengler pour leur sympathie et pour contribuer grandement à la sécurité sous Linux. Laurent Butti, Yoann Guillot, Raphaël Rigo et Philippe Biondi pour leur relecture.

Références

[Intel] *Intel Pentium Processor manuals* : **http://developer.intel.com/design/pentium/manuals/**

[ELF] *Executable and Linking format specification* : **http://www.x86.org/ftp/manuals/tools/elf.pdf**

[Openwall] OpenWall : **http://www.openwall.com/linux/**

[Wojtczuk] *The advanced return-into-lib(c) exploits* : **http://www.phrack.org/show.php?p=58&a=4**

[PaX] *The PaX project* : **http://pax.grsecurity.net**

[DTDUMPER] Julien Tinnès, « dtdumper » : **http://cr0.org/progs/dtdumper**

[paxperf] PjVenda, « *Pax performance impact* » : **http://www.pjvenda.org/linux/doc/pax-performance**

[ET_DYN] PaX Team, « *Generating ET_DYN executables* » : **http://pax.grsecurity.net/et_dyn.tar.gz**

[RELRO] Jakub Jelinek, « *Little hardening DSOs/executables* » : **http://sources.redhat.com/ml/binutils/2004-01/msg00070.html**

[ASLR26] Julien Tinnès, « *Preliminary ASLR port on 2.6.0-test9* » : **http://cr0.org/aslr26/**

[GRSecurity] Bradley Spengler, « *GRSecurity kernel patch* » : **http://www.grsecurity.net**

[PAX+OBS] Julien Tinnès, « *PaX obscurity patch* » : **http://cr0.org/pax-obscure/**

[Adamantix] Adamantix : **http://www.adamantix.org/**

[Gentoo] *Hardened Gentoo* : **http://www.gentoo.org/proj/en/hardened/**

[PAXTEST] Peter Busser, « Pax test » : **http://www.adamantix.org/paxtest/**

[W^X] Theo de Raadt, « *Exploit mitigation techniques* » : **http://www.openbsd.org/papers/auug04/index.html**

[RETF_DEMO] Julien Tinnès, « *Far ret demo* » : **http://cr0.org/divers/obsdretf.asm**

[SSP] Stack smashing protector : **http://www.research.ibm.com/trl/projects/security/ssp/**

[DELALLEAU] Gaël Delalleau, « Vulnérabilité et gestion des limites mémoire » : **http://actes.sstic.org/SSTIC05/Vulnerabilites_et_gestion_des_limites_memoire/**

[SECULINUX] Sécurité du noyau Linux : **http://actes.sstic.org/SSTIC05/Rump_sessions/SSTIC05-rump-Tinnes-SecuLinux.pdf**

Évaluation de la sécurité des terminaux mobiles sous Windows CE

Cet article présente une étude concernant le système Windows CE sur terminal mobile. Il présentera les nouveaux risques associés à l'utilisation d'un système ouvert sur un téléphone mobile. Le modèle de sécurité de Windows CE sera ici évalué et des méthodes permettant d'exploiter ses faiblesses seront présentées. Il est à noter que les autres systèmes que Windows CE ont également leurs faiblesses (Symbian OS par exemple) et qu'actuellement aucune solution satisfaisante sur le plan de la sécurité n'est disponible.

Introduction du contexte

Deux systèmes dominent actuellement le marché des terminaux mobiles de dernière génération : Symbian et Windows Mobile, ce dernier faisant l'objet de cet article. Bien évidemment, à l'idée d'évolution technologique des téléphones portables, il faut penser aussi à celle des technologies associées. En effet, l'introduction de systèmes ouverts autorisant l'installation et le développement d'applications a fondamentalement changé la donne de la sécurité sur les terminaux mobiles. Les premières menaces, relativement anodines ([0], [1]), sont apparues très précisément avec l'introduction de ces nouveaux systèmes offrant des possibilités importantes et en parallèle introduisant de nouveaux risques.

L'étude présentée ici se concentre plus particulièrement sur les SmartPhone et PocketPC qui utilisent Windows Mobile, un dérivé de Windows CE. Deux versions majeures de ce système existent : Windows Mobile 2002 (SPV E100, M1000) qui est construit sur Windows CE 3.0 et Windows Mobile 2003 (SPV E200, M2000) reposant lui sur Windows CE 4.2 (.NET). Un très récent article [2] détaille les différences essentielles de la gestion de la mémoire sur ces environnements. Il est par ailleurs à noter qu'il existe depuis peu Windows Mobile 2005 (Magneto) que nous ne traiterons pas, faute de tests appropriés. Dans la suite de l'article, nous nous intéresserons aux différentes techniques pouvant être employées sur ces environnements pour la réalisation de *backdoors*. Après un rapide rappel de l'architecture mémoire de Windows CE, nous parlerons de la création de backdoors *userland* et expliquerons leurs limites. Nous introduirons alors les backdoors noyau et verrons dans quelle mesure elles sont réalisables. Enfin, nous conclurons notre étude par l'inventaire des moyens mis à disposition des utilisateurs pour se prémunir de tels dangers.

Protection de la mémoire virtuelle et notion de certification de module

Windows CE possède une organisation de la mémoire virtuelle relativement atypique. La bonne documentation ne manquant pas ([3], [4] et plus récemment [5]), dans le cadre de cet article, nous nous contenterons d'en rappeler les principes majeurs.

Ce qu'il est important de retenir se résume en quelques points :

- La mémoire virtuelle (VM) est adressée sur 32 bits, 2 Go de cette mémoire étant réservée au noyau (`0x80000000 - 0xFFFFFFFF`).
- L'espace d'adressage virtuel réservé aux applications est divisé en un nombre limité de *slots* de 32 Mo (64 Mo pour Windows CE .NET), chacun d'eux étant occupé par un processus bien précis. Ceci implique que le nombre de processus soit limité sous Windows CE contrairement au nombre de *threads* (en théorie du moins). Le slot 0 est quant à lui réservé au processus courant, c'est-à-dire au processus auquel l'ordonnanceur a donné la main.
- Une application ne peut pas accéder à l'espace d'adressage, donc au slot, d'un autre processus. Toute tentative d'accès à cette mémoire déclenche en effet une exception.
- La plupart des API système ainsi que quelques routines clefs du kernel fonctionnent en accédant directement aux champs de la structure `KDataStruct` dont on peut bien évidemment trouver la définition dans les sources du noyau de Window CE mais également sur Internet [5]. Un point intéressant est que sur architecture ARM, cet objet se situe à l'adresse statique `0xFFFFC800`.

Alors que les autres versions de Windows fondent la restriction d'utilisation d'API sensibles sur les droits des utilisateurs, Windows CE, lui, fonctionne complètement différemment. À chaque module (exécutable, bibliothèque, driver), est en effet associé un niveau de privilège qui peut être `OEM_CERTIFY_TRUST` pour un module certifié possédant tous les privilèges (le module est signé par l'intégrateur), `OEM_CERTIFY_RUN` pour un module à accès restreint (l'utilisation des Trusted API, qui est une gamme d'API sensible lui est interdite) ou `OEM_CERTIFY_FALSE` pour un module interdit de chargement.

Figure 1 : *Exécution d'un programme non trusté*



Deux fonctions sont utilisées par le noyau pour gérer la certification des applications : `OEMCertifyModuleInit()` qui est appelée lors de l'initialisation de tout module non issu de la ROM et `OEMCertifyModule()` qui s'occupe de la vérification proprement dite à l'aide de la clef publique de l'intégrateur stockée dans la mémoire même du noyau.

Ce mécanisme est intéressant dans la mesure où il permet d'interdire l'utilisation des Trusted API, dont on trouvera une liste exhaustive sur [6],

Roderick Asselineau
France Télécom R&D MAPS/NSS – roderick.asselineau@rd.francetelecom.com

Jean-Marc Hospital
France Télécom R&D MAPS/NSS – jeanmarc.hospital@rd.francetelecom.com

à toute application non référencée par l'intégrateur donc potentiellement dangereuse. En effet, lorsqu'une application non signée par l'intégrateur est exécutée via `ShellExecuteEx()` ou `(Ce)CreateProcess()` sur un Smartphone, alors l'utilisateur reçoit un avertissement lui demandant s'il autorise l'exécution du code. Si oui, le code en question passe en niveau `OEM_CERTIFY_RUN`. Dans le cas contraire le code est rejeté. Raisonnablement, on peut considérer que le niveau de sécurité global du mobile est alors très correct, à l'exploitation de bugs de sécurité près, tant au niveau du mécanisme de vérification de la signature que du noyau **[7]**, si les niveaux de privilèges sont bien définis.

Il est cependant essentiel de rappeler que le marché visé par Microsoft avec Windows CE est celui de l'embarqué. De ce fait, Microsoft fournit, moyennant finance, un kit complet de développement appelé « Platform Builder » **[8]** qui permet de recompiler Windows CE pour une plate-forme donnée. Ce kit comprend un IDE (*Integrated Development Environment*), une abondante documentation ainsi qu'une très grosse partie des sources de Windows CE ce qui a permis aux intégrateurs de fournir des produits à base de Windows CE plus ou moins différents d'un équipement à un autre.

Ce qu'il est important de noter est que le mécanisme noyau de certification des modules est activé en initialisant les variables globales `pOEMCertifyModule` et `pOEMLoadModule` de la fonction `OEMInit()` de l'OAL (*OEM Adaptation Layer*) respectivement avec les adresses de `OEMCertifyModuleInit()` et `OEMCertifyModule()`. Les intégrateurs, devant la pression des utilisateurs, ont alors fait le choix de désactiver à la compilation cette protection sur la gamme des Pockets PC, ainsi que de fournir un utilitaire (SecurityOff) pour la désactiver sur les SmartPhones, ceci afin de favoriser le développement et l'installation d'applications tierces telles que des jeux.

Malheureusement, ce choix contestable donne l'accès aux Trusted API à tout code non signé présent sur le mobile. Un code malicieux pourra alors, par exemple, appeler quelques API au nom relativement évocateur :

- `DebugActiveProcess()` qui permet de se déclarer parent debugger d'un processus;
- `LoadDriver()` qui permet le chargement d'un driver dans l'instance du programme;
- `LoadKernelLibrary()` qui permet le chargement d'une DLL dans l'espace du noyau;
- `{Read,Write}ProcessMemory()` qui permettent respectivement de lire et écrire dans la mémoire d'un autre processus;
- `SetKMode()` qui permet de passer un processus en kernel mode;

Avec de telles API, on peut envisager d'injecter du code dans certains processus et de détourner alors certaines de leurs fonctions clefs. Ce type d'infection est relativement commun dans le monde Windows et les techniques employées sont similaires, aussi verrons-nous comment adapter ces techniques au monde des plates-formes mobiles. On peut également se poser la question de la création de backdoors kernel, ce qui sera l'objet d'une réflexion à part. On pourrait croire à tort qu'un code malicieux nécessite obligatoirement le niveau d'exécution `OEM_CERTIFY_TRUST`, mais il n'en est rien car les contraintes liées à un réseau de télécommunication mobile sont bien différentes de celles d'un réseau informatique classique. Il est en effet essentiel pour l'opérateur :

- De garantir la sécurité des informations de l'usager, que ce soit celles contenues dans sa carte SIM ou celles contenues dans la Flash du mobile ;
- De protéger l'usager de toute utilisation abusive du mobile qui entraînerait une surfacturation ;
- Que l'intégrateur fournisse un mobile résistant aux attaques. Le mobile ne doit en effet pas être détérioré et son fonctionnement global ne doit pas être altéré (impossibilité de passer des appels ou d'envoyer des SMS, carte SIM bloquée, etc.).

Or le problème posé avec les API fournies par Microsoft pour un niveau de privilège au moins égal à `OEM_CERTIFY_RUN` est qu'il permet l'utilisation :

- De technologies de type GPRS ou Bluetooth qui sont susceptibles de devenir des vecteurs de propagation virale;
- D'API liées à la téléphonie et aux SMS, ce qui peut permettre d'engendrer une surfacturation de la consommation de l'usager;
- D'API de manipulation de la SIM ainsi que d'accès aux fichiers ce qui implique une mauvaise protection des données de l'usager;

Infection de processus, mode d'emploi

Dans la mesure où les dangers liés au niveau d'exécution `OEM_CERTIFY_RUN` ne nécessitent pas vraiment d'autre connaissance que celle de l'utilisation d'API en langage C, il n'est pas utile de les détailler plus. En ce qui concerne les infections de processus sous Windows (2k,XP), il existe principalement trois techniques que nous ne détaillerons pas, l'excellent article de Kdm **[8]** le couvrant de façon bien plus détaillée que nous ne pourrions le faire en quelques lignes :

- L'utilisation de `CreateRemoteThread()` permet de démarrer un nouveau thread dans un processus, ce qui permet naturellement l'exécution de code malicieux.
- En définissant un *hook* par l'utilisation de `SetWindowsHookEx()` qui permet l'exécution de code arbitraire pour un type de message donné, pour une application donnée (hook

local) comme pour l'ensemble des applications de l'espace utilisateur aux problèmes de droits près (hook global).

- On peut également se déclarer parent debugger du processus cible, manipuler le contexte d'un thread particulier (tout processus a toujours au minimum un thread principal) et ainsi rediriger le flux d'exécution.

Alors que la première technique est irréalisable sous Windows Mobile puisque l'API `CreateRemoteThread()` est indisponible sur ce type de plate-forme, la seconde devrait a priori, elle, l'être. En effet, bien que non documentée par Microsoft, la fonction `SetWindowsHookEx()` existe bel et bien dans la bibliothèque `coredll.dll` partagée par tous les processus.

En revanche, elle est clairement incomplète et ne permet de hooker que les messages de type `WH_KEYBOARD_LL` ce qui est néanmoins sensible dans la mesure où cela permet de réaliser potentiellement un *keylogger* très simplement.

En ce qui concerne la troisième méthode, il faut savoir qu'elle possède un certain nombre de contraintes :

- Elle repose sur un très grand nombre de Trusted API, ce qui implique un niveau d'exécution `OEM_CERTIFY_TRUST`. Elle n'est donc pas applicable sur un SmartPhone pour lequel le mécanisme de signature n'a pas été désactivé via SecurityOff, mais l'est nativement sur un PocketPC.
- Certains processus ne peuvent pas être débogés sous Windows Mobile. Si par exemple un code malicieux tente de déboger NK.exe, qui est le processus correspondant au noyau, ou GWES.exe, qui est celui responsable de la gestion des drivers, alors le système plante et un *reset* est nécessaire.
- La portabilité ne peut pas toujours être garantie d'un mobile à un autre, puisque les intégrateurs sont dotés des sources et les recompilent à leur guise. En particulier, certains champs dans certaines structures clefs sont susceptibles d'être supprimés. *A contrario*, certains pourraient tout à fait être ajoutés même si cela reste moins vraisemblable puisque la tendance est à l'économie de la mémoire dans l'embarqué.

La portion de code suivante illustre le fonctionnement de l'attaque :

```
[...]
hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE|
TH32CS_SNAPPROCESS|TH32CS_SNAPTHREAD, 0);
if(!hSnapshot)
{
        GetLastError();
        return -1;
}
if(Process32First(hSnapshot, &lppe))
{
        do{
                [...]
                if(!strcmp(TARGET_PROCESS, szExeFileAnsi))
                {
                        MyId = lppe.th32ProcessID;
                        break;
                }
        } while(Process32Next(hSnapshot, &lppe));
}
CloseToolhelp32Snapshot(hSnapshot);
pAddr = VirtualAlloc((LPVOID)0x4FC00000, 0x2000, MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);
memcpy(pAddr, ShellCode, ShellCodeLen);
/* Let's debug the target process */
if(!DebugActiveProcess(MyId))
        goto end;
while(1)
{
        [...]
        if(WaitForDebugEvent(&DebugEvent, 100))
        {
                [...]
                if((suspend = SuspendThread(hThread) >= 0)
                        fprintf(logit, "Thread OFF\n");
                memset(&context, 0, sizeof(CONTEXT));
                context.ContextFlags = CONTEXT_FULL;
                if(GetThreadContext(hThread, &context))
                {
                    if(!shellcode_patched)
                    {
                        memcpy(pAddr, &context.Pc, 4);
                        shellcode_patched++;
                    }
                    [...]
                Context.Pc = (DWORD)((DWORD)pAddr + 4);
                SetThreadContext(hThread, &context);
            }
            [...]
            if(ResumeThread(hThread) >= 0)
            fprintf(logit, "Thread ON\n");
        }
        [...]
}
VirtualFree(pAddr, 0x2000, MEM_DECOMMIT);
[...]
```

Le processus malicieux alloue une zone de mémoire via `VirtualAlloc()` de quelques pages avec des droits de lecture, écriture et exécution dans laquelle il copie un *shellcode*. Il se déclare alors parent debugger du processus cible via `DebugActiveProcess()` puis redirige le flux d'exécution de l'un des threads du processus en question vers la zone mémoire allouée via `SetThreadContext()`. Cette façon de procéder est possible, car le premier paramètre passé à la fonction `VirtualAlloc()` est une adresse mémoire, ici `0x4FC00000`, accessible à n'importe quel processus.

Infecter un processus peut être réalisé dans deux buts distincts : exécuter du code malicieux dans l'instance d'un programme dont on sait qu'il ne sera pas contrôlé par un anti-virus ou détourner certaines fonctionnalités de ce programme. L'exemple le plus classique du premier cas est l'infection d'Internet Explorer sous Windows qui permet d'envoyer des informations via les sockets en toute furtivité puisque l'utilisation des API correspondantes est légitime de la part de Internet Explorer.

Le second cas correspond à la modification de l'IAT (*Import Address Table*) afin de réaliser des détournements (hook) de fonctions. Nos tests indiquent que sous Windows Mobile un processus même en tant que parent debugger ne semble pas pouvoir modifier la mémoire d'un autre processus par `WriteProcessMemory()` et ce, contrairement à la définition de la MSDN.

En revanche, il peut s'automodifier, l'astuce présentée précédemment est donc la seule solution pour modifier l'IAT d'un processus donné. Nous ne détaillerons pas par la suite pas cette technique supposée bien connue du lecteur tant elle est classique. Intéressons-nous cependant un peu à la création de shellcodes.

Quoi qu'on en dise, la création de shellcodes, qui intervient à de multiples niveaux en sécurité, n'est pas toujours un exercice évident suivant l'architecture considérée. Toutefois, nous nous concentrons ici sur l'infection de processus, évitant de ce fait la très contraignante problématique des octets nuls.

Les articles **[5]**, **[10]**, **[11]** et très récemment **[2]**, constituent d'excellentes références en la matière. Il en ressort un certain nombre d'idées clefs :

- Pour construire du code *position indépendant*, la pseudo instruction `adr` permet d'adresser des chaînes ou des blocs de code sans avoir recours à une astuce particulière pour déterminer la position de l'objet en question.
- Pour appeler une API, on peut le faire de deux façons : en déterminant l'adresse de ces API dans la bibliothèque concernée par l'utilisation des informations fournies dans `KDataStruct` ou en appelant directement les appels système concernés. Chacune des méthodes a ses avantages et ses inconvénients. Le principal avantage de la première est sa plus grande portabilité alors que la seconde permettra d'obtenir des shellcodes de taille réduite.

Quelle fonctionnalité intelligente pouvons-nous donner à notre shellcode ? Puisqu'un bon shellcode est un petit shellcode, alors il est souhaitable de n'appeler que le plus petit nombre d'API possible. Dans le cas d'une infection de processus, appeler la fonction `LoadLibrary()`, qui chargera une bibliothèque malicieuse est judicieux. C'est en effet extrêmement pratique et ce pour deux raisons :

- Le code dans cette bibliothèque sera écrit en C donc portable et pouvant être relativement complexe sans que cela gêne l'attaquant outre mesure.
- Les bibliothèques de Windows ont la possibilité d'exporter un `entry point`, et du code sera donc automatiquement exécuté au chargement de la bibliothèque ce qui évite d'avoir à appeler ses fonctions à partir du shellcode.

Les limites du détournement d'API

La méthode précédente repose sur le détournement d'API, en conséquence une application peut détecter son emploi en opérant à un plus bas niveau, celui du noyau. On a vu précédemment que l'objet `KDataStruct` était employé par certaines API système et autres routines du noyau.

Or cet objet est accessible à une adresse fixe en lecture/écriture à tout processus en kernel mode, c'est-à-dire dans le mode processeur pour lequel les 5 premiers bits du registre `Psr` sont mis à `SYSTEM_MODE`. Un processus malicieux dans un tel mode peut alors s'affranchir complètement de l'utilisation de certaines API en manipulant judicieusement certains champs de cet objet.

Aussi surprenant que cela puisse paraître, un processus démarre en kernel mode sur la gamme des Pocket PC !

La raison qui justifie ce choix est celle de l'abandon de l'encombrant contexte utilisateur, ce qui permet d'accroître les performances. En contrepartie, outre une réduction de la stabilité de l'OS, tout processus peut accéder librement à la mémoire de donnée du noyau, ce qui constitue une faille de sécurité énorme. Ratter a vraisemblablement été, avec DUST, le premier à exploiter pleinement cet état de fait.

Sur la gamme des SmartPhones, le passage provoqué d'un processus en kernel mode ne peut se faire qu'en utilisant la Trusted API `SetKMode()`, ce qui limite considérablement les risques puisque seul un module certifié aura alors accès à `KDataStruct`.

À titre d'exemple, la portion de code suivante permet de lister les processus tournant sur le système. Puisqu'il ne repose pas sur l'API `CreateToolhelp32Snapshot()`, il n'est pas possible de le hooker avec la technique précédente. De ce fait, un processus caché de l'Explorer par exemple serait alors révélé.

```
#include "stdafx.h"
// contient la definition de certaines structures non publiques
#include "ps_without_api.h"

FILE *logit =NULL;

// Trusted APIs
extern "C" DWORD SetProcPermissions(DWORD dwPerms);
extern "C" DWORD SetKMode(BOOL toto);

/*
 * On suppose que les tailles des structures sont alignées sur 4 octets.
 * On se donne 10% de marge sur la taille de la structure.
 */
DWORD GimmeOffset(struct KDataStruct *k)
{
        DWORD BigOffset = 0, Offset = 0;
        DWORD Min = 0, Max = 0;
        DWORD div = 0;
        DWORD i = 0;

        /* Les deux champs de K utilisés sont critiques
        /* et les champs d'avant aussi.
         * On peut donc postuler que ces pointeurs sont valides. */
        BigOffset = ((DWORD)k->pCurPrc - k->aInfo[KINX_PROCARRAY]);

        Min = (DWORD)(0.9 * (sizeof(PROCESS) - sizeof(PROCESS) % 4));
        Min -= Min%4;
        Max = (DWORD)(1.1 * (sizeof(PROCESS) - sizeof(PROCESS) % 4));
        Max -= Max%4;

        for(i = Min; i <= Max; i+=4)
        {
                div = (DWORD)(BigOffset/i);
                if( BigOffset == (div * i))
                        return i;
        }
        return 0;
}
/*
 * Fonction d'affichage de la liste des processus
 */
void DumpKernelInfo()
{
        struct KDataStruct *k= (struct KDataStruct *)0xffffc800;
        PPROCESS proc = NULL;
        DWORD offset = 0;
        int j = 0;

        fprintf(logit, "sizeof(PROCESS) = %d bytes\n", GimmeOffset(k));
        for(j=0; jaInfo[KINX_PROCARRAY] + offset*j);
                else
                        return;
                if(j==0 || proc->procnum)
                {
```

```
                        WideCharToMultiByte(CP_ACP, 0, proc->lpszProcName,
                        -1, buffer, sizeof(buffer), NULL, NULL);
                fprintf(logit,"[%d] proc %p [%s]\n",j++, proc, buffer);
                        }
                }
                return;
}
int WINAPI WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPTSTR lpCmdLine,
                int nCmdShow)
{
                BOOL bMode = SetKMode(TRUE);
                DWORD dwPerm = SetProcPermissions(0xFFFFFFFF);
                logit = fopen("Kernel-Map.txt", "w");
                DumpKernelInfo();
                /* Restauration des privilèges */
                SetProcPermissions(dwPerm);
                SetKMode(bMode);
                /* Bye */
                fclose(logit);
                return 0;
}
```

Le champ `aInfo` de `KDataStruct` est un tableau de pointeurs dont l'un est l'adresse du tableau d'objets `PROCESS`, chacun d'entre eux pouvant décrire un processus donné. Le code précédent accède donc à ce tableau et écrit dans le fichier `Kernel-Map.txt` le nom des différents processus du système.

Toutefois, ainsi que nous l'avons évoqué précédemment, un problème susceptible de survenir est la modification de champs dans certains objets clefs du noyau (ici dans la structure `PROCESS`) par les intégrateurs.

Dans ces conditions, lors du parcours du tableau de `PROCESS`, il est risqué d'employer des raccourcis usuels du langage C de type `proc++`. Cette manipulation repose en effet sur le calcul implicite par le compilateur de la taille de l'objet et, puisque les sources de celui-ci ont peu de chance de correspondre exactement à la réalité, le calcul sera erroné. Le code précédent emploie donc une petite astuce qui permet d'accroître sensiblement la portabilité.

L'idée de départ est que `KDataStruct` fournit deux adresses d'objets `PROCESS` différents : le pointeur sur l'objet `PROCESS` du début de tableau et celui correspondant au processus courant. De ce fait, on en déduit un multiple de `sizeof(PROCESS)`.

La détermination de la bonne taille se fait en supposant que celle-ci sera proche de la taille de la structure `PROCESS` dont on possède les sources à une erreur de 10% près. Ce code a été testé avec succès sur Pocket PC M1000 et M2000, ainsi que sur HP IPAQ RX 3715.

Aller plus loin, s'attaquer au noyau

Plus un code malicieux intervient à un niveau bas et plus il sera difficile pour le programme chargé de le détecter de mener à bien sa tâche. Cependant, bien qu'il soit souvent possible d'aller encore plus bas, le noyau a souvent été et vraisemblablement restera encore longtemps l'un des points clefs les plus étudiés par les attaquants.

Sur les Windows modernes classiques tels que 2000 ou XP, la corruption du kernel peut se faire de plusieurs façons, que ce soit par le chargement d'un driver comme par l'accès en lecture/écriture à certaines interfaces type `\Device\PhysicalMemory` **[14]**. Quelle que soit la méthode employée, l'idée est toujours de détourner certaines fonctions clefs du kernel.

Dans le cas des SmartPhones et des Pocket PC, créer une backdoor kernel est plus compliqué et cela s'explique d'au moins deux façons :

- Le code du kernel est situé en ROM et exécuté depuis cet emplacement, ce qui signifie que toute technique liée à une modification directe de ce code, telle que celle décrite dans **[15]**, est *a priori* impraticable.
- L'ajout d'un driver sur le système correspond au chargement d'une dll via `LoadDriver()` dans l'instance du processus adéquat (`FileSys.exe`, `GWES.exe` ou `Device.exe`) et ce, suivant le type de driver. Il se s'agit donc pas de code kernel à proprement parler, mais cela ne signifie pas pour autant qu'il ne soit pas possible d'en tirer parti.

En dépit de ces limitations, force est de constater que l'architecture de Windows CE est néanmoins permissive puisque l'utilisation de la Trusted API `LoadKernelLibrary()` permet le chargement d'une DLL ayant les mêmes droits d'exécution que le noyau. Le seul inconvénient est que celle-ci ne peut pas importer de symboles, la DLL ne sera en effet pas chargée si elle possède une *import table*.

De ce fait, appeler une fonction du kernel revient à être capable de la localiser dans la mémoire, un problème bien connu... ;-). Pour réaliser un hook au niveau noyau, la solution est donc de remplacer certains pointeurs présents dans la mémoire de donnée du noyau par les adresses de fonctions présentes dans la DLL.

On peut envisager d'autres scénarios, mais celui-ci reste l'un des plus simples. À noter que cette méthode possède quelques inconvénients.

D'une part, la DLL ainsi chargée est visible comme DLL chargée par le processus NK.exe et, d'autre part, il est impossible de la décharger sans un reset, ce qui n'a pas que des inconvénients mais peut, dans certains cas précis, compromettre la furtivité.

Protéger son Pocket PC

On peut envisager deux voies de sécurisation des terminaux :

- Lors de l'intégration de l'OS sur le terminal ;
- Après l'achat du terminal par l'utilisateur.

En ce qui concerne la première étape, elle peut être réalisée par l'intégrateur comme par le concepteur de l'OS embarqué (ici Microsoft), puisqu'il s'agit d'une étape précédant la compilation. Bien évidemment, sécuriser un OS alors que ce n'était pas une contrainte initiale de développement coûte énormément d'argent en R&D.

Ce n'est donc a priori pas le concepteur qui effectuera une telle démarche de son propre chef sauf si cela lui semble absolument nécessaire. Ce travail de sécurisation passe essentiellement par un audit approfondi de l'API, du noyau ainsi que des services exécutés sur le mobile. Il consiste également en une éventuelle redéfinition de l'architecture de sécurité.

Dans le cas de Windows Mobile, il est clair que l'architecture n'est pas initialement problématique si l'on suppose le mécanisme de signature actif (donc compilé). Il pourrait être cependant judicieux de définir un certain nombre de niveaux d'exécution intermédiaires pour restreindre l'utilisation des API de communication GSM/GPRS ou les accès SIM. Un mécanisme alternatif tel que celui des *capabilities* proposé par Symbian OS pourrait en outre être tout à fait profitable. Bien que différentes solutions à base de VM (*Virtual Machine*) puissent être envisagées, les seules disponibles à ce jour sur le marché pour l'utilisateur sont le firewall personnel et l'anti-virus.

Malheureusement, ceux-ci ne constituent absolument pas un rempart suffisant et ce pour plusieurs raisons :

- Le processus défensif ne dispose pas d'un niveau de privilège supérieur à celui des autres processus ce qui signifie qu'il peut être tué par une application malicieuse (donc non signée) via un simple appel à l'API `TerminateProcess()`.
- Le moteur de signature des antivirus est absolument insuffisant et apparaît comme relativement trivial à contourner, ce qui implique l'impossibilité de détection de variantes de codes malicieux connus.

Le lecteur l'aura compris, les logiciels de protection actuels pour mobiles sont relativement rudimentaires. Le problème ne vient pas forcément d'un manque d'implication des sociétés concernées, mais aussi de l'environnement cible. Dans le domaine de la téléphonie mobile, les équipements sont très peu puissants. En particulier, les OS type Windows Mobile ou Symbian occupent une grande partie des ressources. Il est donc impossible de réaliser des moteurs de détection comportementale, de lutter contre le polymorphisme, etc.

Plus critique, il est difficile de gérer un mécanisme de signature complexe faute de moyen processeur suffisant. N'oublions par ailleurs pas que toute exécution d'un code un peu lourd modifie sensiblement le temps de réaction du système, ce qui est perceptible par l'utilisateur, voire éventuellement altère certaines fonctionnalités essentielles du portable.

Conclusion

Plus d'un an après la publication du code source de DUST par Ratter, il est clair que le nombre de virus ou de *worms* apparus sur les terminaux Windows Mobile est plus que réduit, pour ne pas dire ridicule. Il serait cependant dangereux de s'en féliciter... Nous avons en effet vu que les moyens de défense des utilisateurs étaient encore extrêmement réduits or l'absence d'activité virale n'incite pas véritablement à un développement de la sécurité. Pourtant, les menaces sur ce type de plates-formes sont bel et bien réelles et si pour le moment la grande hétérogénéité du parc nous protège, il n'en sera pas toujours ainsi.

Remerciements

Un grand merci à Nicolas Ruff pour nos sympathiques discussions sur le sujet, à Corentin, à la relectrice mystère et à Franck Veysset pour leur relecture de l'article ainsi qu'à l'équipe du NSS pour sa bonne humeur au quotidien (dédicace spéciale à Anne-So ;-)).

Références

[0] E. Filiol, « Analyses de codes malveillants pour mobiles : le ver CABIR et le virus DUST », MISC 16.
[1] N. Ruff, « Sécurité des téléphones portables », MISC 16.
[2] T. Hurman, *Exploring Windows CE Shellcode*, Pentest Limited.
[3] J. Y. Wilson and A. Havewala, *Building Powerful Platforms with Windows CE*, Addison Wesley, 2001
[4] MSDN, **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/_wcesdk_Windows_CE_Memory_Architecture.asp**
[5] San, « Hacking Windows CE », Phrack 63.
[6] MSDN, **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcedsn40/html/cgcontrustedapis.asp**
[7] Bugtraq, **http://www.securityfocus.com/bid/10914**
[8] **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcepbguide5/html/wce50connPlatformBuilderUsersGuide.asp**
[9] Kdm, NTIllusion : « *A portable Win32 userland rootkit* », Phrack 62.
[10] C. Mulliner, « *Exploiting PocketPC, What The Hack!* », Juillet 2005.
[11] Seth Fogie, « *Embedded Reverse Engineering: Cracking Mobile Binaries* », Defcon 11, 2003.
[12] Seth fogie, « *Pocket PC Abuse* », Defcon 12, 2004.
[13] C. Peikari, S. Fogie, Ratter/29A, « *Details Emerge on the First Windows Mobile Virus (Part {1,2,3} of 3)* », septembre 2004.
[14] Crazylord, « *Playing with Windows /dev/(k)mem* », Phrack 59
[15] Mayhem, « *Linux x86 kernel function hooking emulation* », Phrack 58

Comment Leurré.com observe l'Internet

Le projet « Leurré.com », lancé en janvier 2003 par l'Institut Eurécom, a pour but de collecter du trafic anormal sur Internet et d'en étudier les propriétés et les causes. À cette fin, Eurécom et les organismes associés au projet déploient des « honeypots » dans divers endroits du monde.

Présentation de Leurré.com

Alors que la sécurité sur Internet est devenue un enjeu économique et politique primordial, le développement de méthodes et d'outils dans ce domaine se heurte à une difficulté majeure : le manque de données fiables, récentes et représentatives du trafic sur le réseau. Encore aujourd'hui, la plupart des travaux sont fondés sur le trafic consigné dans les célèbres fichiers de la DARPA, dont le moindre inconvénient n'est pas le fait qu'ils datent de 1999. Autant dire que des résultats obtenus par des techniques ou des outils sur un tel jeu de données ne présument en rien de leurs performances sur le terrain, et force est de constater que celles-ci sont généralement décevantes. Le projet Leurré.com [1] a été fondé précisément dans ce but : construire une infrastructure afin de collecter et d'analyser des données réelles et les rendre disponibles à des fins de recherche et de développement.

Les honeypots

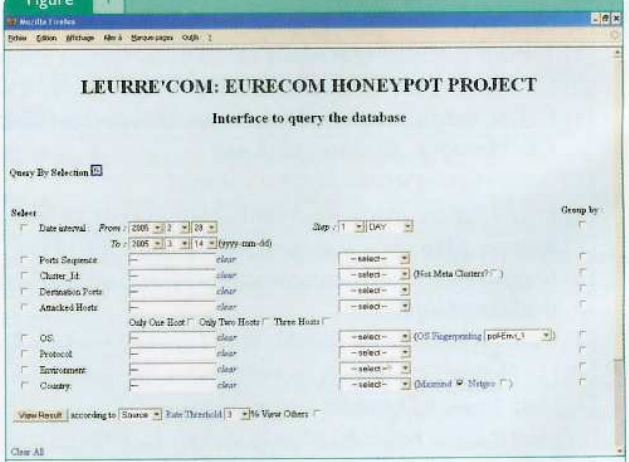
Un *honeypot*, appelé aussi « pot de miel » en français, désigne une ressource informatique dont le rôle est de servir de cible à des usages anormaux ou malveillants. Un honeypot ne fournit aucun service destiné à être utilisé : théoriquement, aucun trafic sur le réseau ne devrait par conséquent être destiné à un honeypot. La présence d'un tel trafic est généralement considérée comme le symptôme d'une anomalie ou d'une tentative d'intrusion (usage illégitime des ressources du réseau).

Il existe deux types de honeypots. Les honeypots dits à « haute interaction » implémentent des services correspondant à une utilisation réelle dans le but d'en enregistrer l'usage, par hypothèse, considéré comme anormal. Cette technique permet donc en principe d'enregistrer des attaques effectives contre les services exposés et les actions délictueuses ainsi perpétrées, fournissant des données utiles aux enquêteurs dans le cadre d'une analyse légiste. Un tel honeypot consiste donc essentiellement en une implémentation opérationnelle des services et en tant que tel, il est lui-même exposé à des attaques. Pour cette raison, les honeypots à haute interaction sont généralement installés sur des machines virtuelles du type *VMware* [2], dont le fonctionnement est facile à contrôler et à rétablir dans le cas où le système serait compromis par une attaque. Un honeypot du second type, dit à « faible interaction », est un système factice qui n'offre pas de services réellement exploitables. L'implémentation de ces derniers reste embryonnaire et n'est conçue que pour fournir la réponse attendue aux tentatives d'identification du système (*scanning* et *fingerprinting*). Ces honeypots sont typiquement réalisés à l'aide du Logiciel libre Honeyd [3]. L'avantage principal de cette approche est sa plus grande sécurité : un honeypot à basse interaction offre a priori des possibilités d'attaque ou de détournement très limitées et les attaques dirigées contre le système émulé n'aboutissent pas. En revanche, pour cette même raison, un honeypot à basse interaction ne permet pas d'enregistrer des intrusions effectives mais seulement les tentatives d'intrusion. D'autre part, il demeure aisé de détecter à distance un honeypot à basse interaction. Un attaquant peut prendre cette information en compte et modifier son comportement en conséquence. Les honeypots à basse interaction induisent donc intrinsèquement un biais, notamment dans le cas où l'attaquant est un opérateur humain et non un programme autonome simple tel qu'un ver. Ce biais et, plus généralement, ces limitations par rapport à des honeypots à haute interaction, font partie des problèmes étudiés dans le cadre du projet Leurré.com.

Infrastructure

À l'heure actuelle, le projet Leurré.com déploie des honeypots à basse interaction construits sur Honeyd. Chaque site participant au projet expose une ou plusieurs plateformes composées de trois hôtes virtuels sous Honeyd, qui émulent des systèmes Linux, Windows 2000 et Windows NT. Un kit d'installation comprenant une distribution de Linux, Honeyd et sa configuration est proposé aux sites désirant se joindre au projet. Fin 2005, plus de 40 sites de ce type ont été déployés sur tous les continents à l'exception de l'Antarctique. Pour des raisons évidentes, la localisation précise des sites est tenue confidentielle : en effet, s'il était connu qu'une adresse IP particulière correspond à un honeypot, le trafic enregistré par ce site pourrait être fortement biaisé.

Figure



Interface Web de la base de données de Leurré.com

L'ensemble du trafic enregistré par *TCPdump* sur chaque site est transmis et stocké dans la base de données centrale de Leurré.com. Il s'agit d'informations telles que l'adresse d'émission et

Jacob Zimmermann
j.zimmerm@isrc.aut.edu.au

le contenu (*payload*) de chaque paquet IP reçu par l'un des honeypots déployés, sa date de réception, le numéro de port visé sur le honeypot, etc. Cette masse de données (voir Figure 1) est accessible à tous les participants au projet et est utilisée actuellement par plusieurs travaux de recherche, portant sur la reconnaissance et la classification des outils d'attaque ou le développement de techniques de détection. Au-delà de ces travaux concrets menés dans le cadre du projet, Leurré.com a la vocation de fédérer diverses recherches relatives à l'analyse du trafic sur Internet et à encourager la coopération et le partage des informations. Cette fédération se déroule sur le plan technique, par exemple en mettant les bases de données à la disposition de tous les participants et partenaires du projet, mais également sur le plan de la recherche, en développant un cadre théorique général pour intégrer diverses méthodes d'analyse et synthétiser les résultats. Nous présentons certains de ces travaux dans la suite de cet article.

Caractérisation des outils d'attaque

Les premières analyses effectuées sur les données collectées, menées par l'équipe d'Eurécom, portaient sur la détection et la classification des outils d'attaque à l'origine du trafic enregistré par les honeypots [4].

Modélisation du trafic

Pour les besoins de cette analyse, le trafic est structuré en « sessions » de 24 heures. Plus précisément, chaque « session » comprend l'ensemble des paquets transmis entre une source et une destination en l'espace de 24h. Elle peut donc correspondre à une ou plusieurs sessions TCP, mais aussi des séries de paquets utilisant d'autres protocoles comme UDP ou ICMP. Cette durée de 24 heures a été choisie arbitrairement, sachant que 24 heures correspondent, entre autres, au délai de validité par défaut des adresses IP allouées dynamiquement par DHCP. On considère donc qu'il est impossible de garantir qu'une adresse IP donnée corresponde effectivement à la même source de trafic pendant plus de 24 heures. Chaque session de 24 heures associée à un hôte virtuel donné est désignée par le terme de « *tiny session* ». La modélisation tient compte également de l'ensemble des paquets reçus par une plate-forme donnée en l'espace de 24 heures : on désigne cet ensemble par le terme de « *large session* ». Une *large session* correspond donc à l'union d'une à trois *tiny sessions*, en fonction de l'activité des trois hôtes virtuels durant 24h.

Classification des empreintes d'activités

Chaque occurrence de trafic anormal ou malveillant pendant 24h peut ainsi être caractérisée par son empreinte sur la plate-forme cible. En groupant les « sessions » observées en fonction de la similarité de leurs empreintes, on obtient une classification des comportements anormaux et, par conséquent, des outils à l'origine de ces comportements. Cette classification est obtenue par l'algorithme à base de règles d'association « Apriori » [5], appliqué dans le but de grouper les occurrences de trafic selon des caractéristiques telles que :

- Le nombre d'hôtes virtuels sur la plate-forme qui reçoivent un trafic en provenance d'une même source ;
- Le nombre de paquets dans chacune des tiny sessions enregistrées correspondantes ;
- Le nombre de paquets dans la large session correspondante (c'est-à-dire la somme des nombres de paquets par tiny-session) ;
- La suite des ports TCP ou UDP visés par ces paquets ;
- L'adresse IP de la source du trafic et sa localisation géographique ;
- etc.

L'algorithme « Apriori » consiste à construire un ensemble de règles d'association de la forme « si A et B, alors C » où A, B et C correspondent à des caractéristiques telles que celles énumérées ci-dessus. L'algorithme consiste à itérer sur la longueur de telles règles, où chaque itération utilise l'ensemble de règles de longueur k (c'est-à-dire des règles de la forme « si E(1) et E(2) ... et E(k-1) alors E(k) ») pour construire des règles de longueur k+1 (de la forme « si E(1) et E(2) ... et E(k) alors E(k+1) »).

Les règles sont construites selon le principe suivant :

- Pour chaque règle de longueur k connue, considérer les règles candidates de longueur k+1 telles que N_{k+1} soit supérieur à un pourcentage prédéfini du nombre total d'éléments à classifier, où N_{k+1} désigne le nombre d'éléments qui possèdent les caractéristiques E(1)... E(k+1) : ce paramètre s'appelle le « support » de la règle candidate ;
- Retenir les règles candidates dont le rapport N_k/N_{k+1} (appelé la « confiance » de la règle) est supérieur à une valeur prédéfinie.

L'algorithme débute en sélectionnant les caractéristiques qui présentent chacune un « support » supérieur au minimum prédéfini ; celles-ci constituent alors un ensemble de départ de règles unitaires. Le processus s'arrête lorsque l'ensemble de règles k+1 retenues est vide, c'est-à-dire lorsqu'il n'est plus possible de construire de nouvelles associations. L'ensemble de règles d'association ainsi construites est utilisé pour grouper les occurrences de trafic enregistré en *clusters* correspondant à des occurrences similaires selon ces critères. Chaque cluster est par la suite divisé en sous-clusters en fonction du contenu des paquets (payload). La règle utilisée pour évaluer la similarité des contenus est la Distance de Levenshtein [6]. À ce stade, on considère que chaque sous-cluster correspond à une cause unique de trafic anormal, c'est-à-dire à une occurrence d'un outil

d'attaque, car il regroupe des empreintes dont les caractéristiques sont similaires.

Applications et directions actuelles

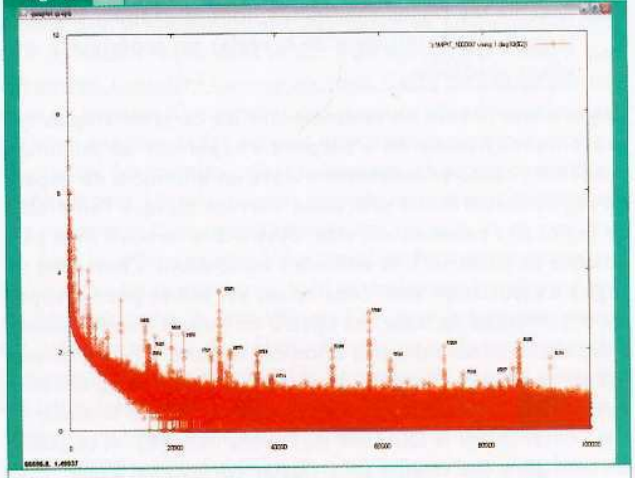
Chaque sous-cluster obtenu par cette technique constitue une signature d'une anomalie particulière. Dans les cas où cette anomalie peut être identifiée (par exemple parce que ses caractéristiques sont connues ou en comparant ses caractéristiques à celles observées lors d'une exécution d'un outil donné dans un environnement de test), il est ainsi possible de surveiller sa propagation sur Internet, en particulier lorsqu'il s'agit de vers. Des résultats publiés identifient ainsi certains pays ou régions comme des sources ou des cibles majeures de certains types d'attaques [4]. De même, l'apparition de nouveaux clusters aux caractéristiques nouvelles est le symptôme probable de l'activité d'un nouvel outil d'attaque ou ver, qu'il est nécessaire d'identifier et dont la propagation doit être surveillée.

Analyse des délais entre réceptions de paquets

L'équipe australienne de l'*Information Security Institute* (ISI) de la *Queensland University of Technology* travaille en collaboration avec Eurécom et étudie la possibilité de détecter des anomalies du réseau en analysant les délais entre les réceptions par un honeypot de deux paquets consécutifs provenant de la même source. Dans les publications en anglais, ces délais sont désignés par les termes de « *Inter-Arrival Times* » ou simplement IAT [7], sigle que nous utilisons également dans cet article. L'objectif consiste ici à développer une technique de détection extrêmement légère. En effet, les méthodes de détection d'anomalies non rudimentaires disponibles actuellement restent lourdes à mettre en œuvre et reposent en grande partie sur l'analyse du contenu des paquets.

La puissance de calcul et la capacité de stockage requises pour appliquer ces traitements à l'ensemble du trafic reçu rendent leur implémentation impossible sur des serveurs de production soumis à une forte charge. En revanche, l'observation des IAT est très économique : la séquence d'IAT correspondant à un trafic donné consiste en un flux de valeurs numériques, auxquelles il est possible d'appliquer divers algorithmes directement à la volée.

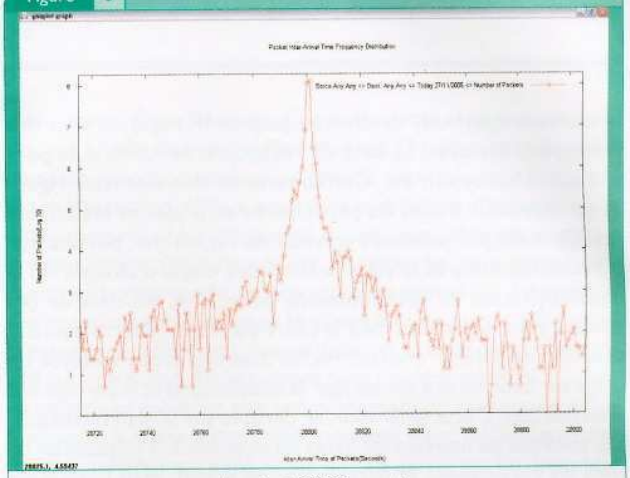
Figure 2



Distribution statistique globale des IATs dans le trafic enregistré

Il est possible d'identifier ainsi rapidement des phénomènes suspects qu'il convient d'examiner en détail par la suite.

Figure 3



Le «pic» à 28800 secondes

Étude de cas

Certaines anomalies du réseau se manifestent par une proportion anormalement élevée d'IAT d'une durée particulière. La Figure 2 montre la distribution statistique globale des IAT dans l'ensemble du trafic enregistré depuis janvier 2003. Elle indique que des IAT de certaines durées apparaissent avec une fréquence inattendue. Un exemple concret est le *pic* correspondant aux IAT d'une durée de 28800 secondes, qui comptait 5077 occurrences en mai 2005 quand cette étude a été menée (valeur à comparer avec le nombre d'occurrences d'IAT de durée similaire, comme l'illustre la Figure 3). On peut remarquer que cette durée très longue (exactement 8 heures) et le fait qu'elle se reproduit essentiellement avec une précision de +/- 1 seconde excluent l'hypothèse qu'une telle anomalie puisse résulter de latences ou de dysfonctionnements du réseau tels que des pertes de paquets.

Une analyse plus poussée des informations contenues dans la base de données de Leurré.com révèle d'autres traits réguliers liés à cette durée particulière. Ainsi, 99,6% des sessions dans lesquelles apparaît un IAT de 28800 secondes comportent des paquets adressés au port UDP 38293. Plus de 83% de ces sessions sont par ailleurs composées de 18 ou 36 paquets précisément : dans ce dernier cas, il s'agit pour la plupart d'une série de 18 paquets répétée deux fois. Enfin, il ressort que deux contenus (payloads) distincts seulement sont observés pour l'ensemble de ces sessions. De plus, ces deux payloads présentent le même nombre d'occurrences dans l'ensemble des sessions examinées (en réalité, leur différence est de 1, probablement à cause d'une perte de paquet). Une brève recherche sur Google indique que le port UDP 38293 est utilisé par certaines versions du serveur de mises à jour de Norton Antivirus. En tenant compte de l'ensemble des caractéristiques observées de ce trafic, il est manifeste qu'il correspond à des installations de Norton Antivirus mal configurées qui disposent d'une adresse erronée du serveur de mises à jour. Cet exemple illustre les possibilités offertes par une infrastructure telle que Leurré.com. En l'occurrence il s'agit ici non pas d'attaques mais de dysfonctionnements, qui peuvent être aisément détectés et corrigés. Des observations rapides

d'anomalies, faciles à automatiser, indiquent un problème dont quelques recherches simples dans la base de données permettent d'établir un diagnostic complet, incluant la liste des sites à l'origine de ce trafic, leurs emplacements géographiques et leurs adresses IP.

Généralisation de la détection des pics

Des travaux actuellement en cours à l'ISI visent à systématiser l'analyse d'IAT et obtenir ainsi une technique économique pour déceler effectivement des problèmes potentiels, qui justifieraient alors des analyses plus complexes en utilisant des techniques lourdes. Ces recherches ne sont pas terminées à l'heure actuelle, il est cependant prévu de soumettre les résultats de ces travaux à la publication au cours du 1er trimestre 2006. Nous résumons ici les observations et analyses effectuées à l'heure où cet article est écrit. Les premières analyses utilisaient le modèle des « sessions » de 24 heures utilisé par les travaux de l'équipe Eurécom. Il apparaît cependant qu'en écartant cette limite de 24 heures et en considérant comme « session » l'ensemble de tous les paquets envoyés par une même source à une même destination au cours de l'existence du projet, de nouvelles anomalies apparaissent.

Des pics ont ainsi été observés pour des IAT d'une durée aussi longue que 300000 secondes, soit plus de 3 jours et 12 heures. D'autre part, il existe un phénomène de « pics harmoniques ». Le premier exemple concernait les IAT de 28800 secondes ; il existe cependant un pic similaire pour une durée d'IAT exactement double, soit 57600 secondes (16 heures). Des « harmoniques » de ce type ont pu être observées pour un grand nombre de pics, un exemple marquant en est l'ensemble des pics correspondant aux durées d'IAT respectives de 300, 600, 900, 1800, 2400 et 3600 secondes exactement. (L'analyse plus poussée des IAT d'une durée inférieure à 1 heure, c'est-à-dire 3600 secondes, a toutefois été provisoirement écartée, car les IAT de cet ordre de grandeur peuvent être fortement biaisés par des latences ou dysfonctionnements du réseau). Ce phénomène n'est pas totalement expliqué à l'heure actuelle. Il est certain toutefois qu'il ne s'agit pas d'un simple artefact dû à la perte de paquets. Si des paquets sont émis régulièrement vers une destination donnée avec une durée d'IAT fixe, la perte d'un paquet donne naturellement lieu à un IAT de durée double. Les observations indiquent cependant que le taux de perte est de l'ordre de 10%. Or, les harmoniques observés comptent des nombres d'occurrences d'IAT très supérieurs à 10% du trafic concerné par le pic original, la perte de paquets n'est donc pas seule à leur origine. De plus, par exemple dans le cas de la mauvaise configuration de Norton Antivirus, le trafic correspondant à l'harmonique de 57600 secondes présente certaines caractéristiques différentes de celui correspondant au pic de 28800 secondes, notamment en termes de nombre de paquets par session de 24h et de contenus. L'une des directions des études actuelles concerne donc les corrélations entre pics. Les observations reposent pour l'heure essentiellement sur une interprétation qualitative des résultats numériques. La Table 1 montre le nombre de sessions de 24h qui donnent lieu à au moins deux valeurs distinctes d'IAT parmi celles correspondantes aux 10 pics les plus importants. Il apparaît ainsi que de tels cas sont relativement rares, en comparaison avec le nombre de sessions où n'apparaît qu'une seule valeur de « long » IAT, éventuellement se produisant plus d'une fois (ces nombres sont indiqués sur la diagonale de la Table 1).

Tableau 1

	57600	28800	9754	3600	2400	1800	1200
300	6	5	3	32	15	9	53
600	0	8	1	33	9	12	16
900	0	0	0	17	14	3	22
1200	3	3	1	183	215	3	1048
1800	0	0	1	13	7	707	
2400	3	2	0	199	610		
3600	3	9	0	5649			
9754	0	2	490				
28800	13	2686					
57600	767						

Inter-dépendences entre IATs

La corrélation entre un pic et un port TCP ou UDP (ou un ensemble de ports) est également intéressante : il apparaît statistiquement qu'en dehors des ports utilisés par le trafic usuel (essentiellement les ports 80, 443 et 135), les paquets correspondant à deux pics distincts non harmoniques visent des ports différents. Pour des IAT longs, un pic et ses harmoniques fournissent par conséquent une signature statistique d'un type particulier de trafic.

Recherche de motifs

Le trafic enregistré par chaque honeypot consiste en des rafales de paquets reçus, c'est-à-dire des séries de paquets avec des IAT de courte durée (de l'ordre de quelques secondes au plus), séparées par des IAT plus ou moins longs. Les observations indiquent que ces périodes de rafales tendent à former des motifs répétitifs qu'il convient d'analyser. Le modèle utilisé actuellement pour cette analyse distingue donc entre des IAT « courts » et des IAT « longs », ces derniers ayant des durées différentes. La limite entre les IAT « courts » et « longs » reste relativement arbitraire. Elle peut être fixée manuellement afin d'étudier certains cas particuliers ; par défaut elle est définie comme étant l'IAT correspondant au premier pic supérieur à 1 minute. On peut donc représenter le trafic entre une source et une destination données par une suite de couples <longueur_rafale;long_IAT>. Par exemple, supposons que le trafic enregistré se traduise par la suite des IAT suivante :

0, 0, 1, 7982, 3, 2, 0 7960, 0, 2, 0, 8004, 1, 3, 8023, 0, 1, 0, 7989, 1, 0, 2, 7978, 0, 0, 0

Notons que la date de réception des paquets (*timestamp*) est stockée avec une résolution de 1 seconde ; aussi, un IAT de 0 seconde correspond à deux paquets reçus avec moins de 1 seconde d'intervalle. Si, dans l'exemple présent, la limite entre « court » et « long » est fixée à 7960 secondes (ce qui sera le cas par défaut en l'absence d'autres données), la séquence ci-dessous pourra être représentée par :

<3,7982> - <3,7960> - <3,8004> - <2,8023> - <3,7989>- <3,7978> - <3,_>

L'ensemble des séquences représentées de cette façon est soumis à un algorithme simple de recherche de motifs afin de déterminer des enchaînements de rafales qui correspondent à des modèles de comportement répétés. Cette recherche concerne uniquement les rafales et ignore la durée des IAT longs qui les sépare.

Elle repose sur deux règles simples :

1. Deux rafales sont considérées identiques si leurs longueurs sont égales avec une tolérance de +/- 10% (ceci afin de prendre en compte le taux de perte de paquets).
2. Pour une séquence donnée, chaque motif est retenu comme représentatif si l'ensemble de ses occurrences dans la séquence « couvre » au minimum 90% des paquets de cette séquence par défaut.

Dans cet exemple, la séquence contient ainsi trois motifs obéissant à ces critères :

- Le motif « 3 » (une rafale de quatre paquets, c'est-à-dire 3 IAT courts), apparaissant 6 fois dans la séquence au total ;
- Le motif « 3-3-3 » (trois rafales consécutives de quatre paquets chacune), apparaissant 2 fois dans cette et séquence ;
- Le motif « 3-3-3-2-3-3-3 », qui apparaît une fois et correspond à la séquence entière.

Les séquences sont alors groupées en fonction des motifs qu'elles contiennent. Les résultats font apparaître un ensemble restreint de motifs apparaissant très fréquemment dans le trafic enregistré. Il s'agit principalement des motifs « 2 », « 17 » et « 4-2 ». Ce dernier est remarquable : en effet, il désigne une alternance régulière de rafales comprenant 5 paquets (4 IAT) et 3 paquets (2 IAT). Deux autres observations méritent d'être relevées. D'une part, l'occurrence de tels motifs semble être indépendante du type de trafic, elle serait davantage liée à son origine (i. e. la source). D'autre part, il existe un phénomène de « convergence » : dans certains cas, le trafic débute de manière erratique (en termes de séquences d'IAT), les occurrences répétées d'un motif ne commençant qu'après un certain nombre de paquets émis. L'observation de ces motifs répétés et leur détection sont actuellement utilisées afin d'élaborer une nouvelle définition de « sessions d'attaque », basée sur l'identification d'activités élémentaires au sein du trafic enregistré. Une autre application possible des IAT a toutefois été suggérée par des travaux menées indépendamment aux États-Unis. Zhou et Lang ont appliqué la transformée de Fourier discrète aux IAT dans les enregistrements DARPA [8], utilisés traditionnellement pour évaluer des méthodes d'analyse de trafic. Leurs résultats indiquent que des motifs d'IAT courts, identifiés par des pics dans le spectre des fréquences, peuvent être considérés en tant que signatures de certaines attaques ou anomalies.

Conclusion

Nous avons présenté le projet Leurré.com et certains travaux menés dans le cadre de ce projet par les équipes d'Eurécom (Nice) et de QUT-ISI (Brisbane). Ce projet représente aujourd'hui un effort mondial dans le domaine de l'analyse de trafic sur Internet et a la vocation d'offrir une base solide de coopération internationale. Aussi, tout institut ou organisme désirant participer à la recherche dans ce domaine est cordialement invité à rejoindre le projet. Chaque nouveau site participant s'engage par écrit à respecter les règles de confidentialité propres au projet (en particulier, à ne pas divulguer d'informations concernant l'emplacement des honeypots) et, bien sûr, à déployer et à maintenir au moins une plate-forme Honeyd et transmettre le trafic enregistré vers la base de données centrale. En contrepartie, le site se voit offrir un identifiant et un mot de passe permettant d'accéder à cette base, et dispose ainsi d'un accès complet aux données brutes en provenance des honeypots, aussi bien qu'aux résultats des analyses telles que celles présentées dans cet article. Ceux qui le désirent peuvent par ailleurs enrichir la base de données de leurs propres résultats, les mettant ainsi à la disposition de l'ensemble de la communauté du projet. Le projet Leurré.com et les travaux de recherche mentionnés dans cet article ont été financés en partie dans le cadre de l'accord de coopération franco-australien DEST-FAST et du projet HoFa, et également par les programmes ACI-SI du ministère de la Recherche français et ARC-Linkage International de son homologue australien. Nous souhaitons par ailleurs remercier Marc Dacier, George Mohay et Andrew Clark, responsables des projets menés par Eurécom et QUT-ISI, ainsi que Fabien Pouget et Saleh Al-Motairi pour leur aide lors de la rédaction de cet article. Pour plus d'informations, nous vous invitons à visiter le site du projet : **www.leurrecom.org**.

Références

[1] *The Leurré.com Honeypot Project* : **http://www.leurrecom.org**

[2] VMware : **http://www.vmware.com**

[3] *Developments of the Honeyd Virtual Honeypot* : **http://www.honeyd.org**

[4] *Honeypot-Based Forensics* : F. Pouget and M. Dacier, *Proceedings of the Asia Pacific Information Technology Security Conference* (AusCERT), 2004.

[5] *Fast Algorithms for Mining Association Rules* : R. Agrawal and R. Srikant, *Proceedings of the 20th International Conference on Very Large Databases* (VLDB), 1994.

[6] *Levenshtein Distance in Three Flavors* : **http://www.merriampark.com/ld.htm**

[7] J. Zimmermann, A. Clark, G. Mohay, F. Pouget et M. Dacier, « *The Use of Packet Inter-Arrival Times for Investigating Unsolicited Internet Traffic* », SADFE'05 – *First International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2005.

[8] M. Zhou and S. D. Lang, « *Mining Frequency Content of Network Traffic for Intrusion Detection* », *Proceedings of IASTED International Conference on Communication, Network and Information Security* (CNIS), 2003.

De la sécurité d'une architecture DNS d'entreprise

Ces dix dernières années, les entreprises ont vu une interconnexion toujours croissante de leurs systèmes d'information (SI) entre eux ainsi qu'avec l'internet. Cette interconnexion s'est accompagnée d'une adoption massive de protocoles standards. Parmi ces protocoles, le DNS (Domain Name Service) affiche la double caractéristique d'être le plus systématiquement utilisé et sûrement le moins correctement surveillé.

1. Introduction

Le DNS est souvent vu comme un protocole utilitaire au fonctionnement opaque. Et à la vue de son caractère primordial pour le bon fonctionnement des protocoles applicatifs tels SMTP ou HTTP, les portes de ce protocole sont souvent laissées grandes ouvertes tout au long de la chaîne de liaison du SI (postes, serveurs, routeurs, etc.).

Nous verrons dans cet article comment sécuriser ce protocole et en particulier son implémentation dans un réseau d'entreprise. Les exemples de configuration reposeront sur le logiciel Bind, version 9.1.x et suivantes [BIND].

1.1 Terminologie

Avant de plonger dans le DNS, accordons-nous sur les termes utilisés dans ce document :

- *Resolver* : logiciel client DNS sollicité par les applications sur les postes clients afin d'obtenir une résolution ;
- *Serveur cache* : serveur sollicité par les resolvers ou par d'autres serveurs cache. Ce type de serveur assure la fonction de récupération d'informations auprès de serveurs de noms ou d'autres serveurs cache. Il stocke les informations collectées dans son cache. Ce serveur ne gérant aucune zone DNS, le type de recherche supportée est récursif ;
- *Serveur SOA* : serveur de noms faisant autorité répondant aux requêtes pour une (ou plusieurs) zone(s). Ce serveur n'ayant pour mission que de répondre à des requêtes sur un ou plusieurs domaines, dans notre configuration, nous n'autorisons aucune récursion.

1.2 Les vulnérabilités DNS

À présent, regardons les différents types de menaces pouvant toucher le DNS :

- La fuite d'informations via des canaux cachés ;
- La corruption de résolution DNS permettant d'abuser les utilisateurs du système, d'informations et de leur dérober des informations sensibles ;
- Le déni de services par l'acceptation de requêtes illégales ou par corruption de données dans les zones hébergées par les serveurs DNS internes.

La fuite d'informations

La fuite d'informations par tunnel DNS a été couverte de manière complète dans les pages de MISC [TUN]. Les contre-mesures pour lutter contre ce type d'évasion se concentreront autour de la politique d'accès au service DNS ainsi qu'aux requêtes autorisées en fonction du type de client. La mise en œuvre de cette politique sera décrite dans le paragraphe « Configuration du serveur DNS cache interne ».

Corruption de résolution

Ce type d'attaque a pour but de corrompre les données DNS fournies au client afin de, par exemple, le diriger à son insu vers des sites différents de ceux visés. Et ainsi, de lui dérober tout type d'informations sensibles.

Ce type d'attaque peut être réalisé par :

- Compromission du processus de résolution ;
- Modification du trafic DNS ;
- Ou compromission des données dans le cache d'un serveur de noms.

Les réponses que cet article amènera viseront à :

- Sécuriser le processus de résolution et en maîtriser les possibilités de modification ;
- Limiter au strict minimum l'usage des requêtes récursives ;
- S'assurer de l'intégrité des transactions.

Le déni de services

Le déni de services distribué est une menace commune à tous les protocoles TCP/IP implémentés sur les serveurs accessibles de l'internet.

En dehors des mesures de protection réseau (IPS) ou de répartition de charges, peu de réponses spécifiques à la configuration DNS peuvent être appliquées. Cependant, les mesures suivantes permettent de limiter l'impact et l'implication des serveurs DNS dans ces attaques :

- Restriction des transferts de zone ;
- Interdiction de toute récursion ;
- Contrôle des requêtes simples ;
- Restriction des notifications.

Christophe Brocas,
christophe.brocas@free.fr

Jean-Michel Farin,
jeanmichel.farin@free.fr

2. Architecture

Les règles du jeu

Pour maîtriser au mieux le DNS et son utilisation, nous fixons les règles suivantes :

- Aucune machine du réseau interne ne peut accéder à l'internet. Elles utilisent toutes un proxy HTTP pour le web ou un relais SMTP pour la messagerie ;
- Aucune machine ne peut donc résoudre autre chose que le domaine `mondomaine.fr`.

Cela permet d'être sûr de pouvoir déployer une politique de sécurité maîtrisable.

2.1 Les serveurs DNS et leur rôle

La base pour fixer une politique de résolution est donc de définir un périmètre et des règles d'accès :

- Un serveur DNS cache interne assure la résolution de tout type de requête issue d'une machine du réseau interne.
- Un serveur DNS SOA assure la gestion du domaine de l'entreprise pour le réseau interne de l'entreprise. Ce serveur ne peut être sollicité que par le serveur de résolution.
- Un serveur DNS cache en DMZ assure la résolution des requêtes sur les noms internet issues du serveur DNS cache interne.
- Un serveur DNS SOA externe assure la gestion du domaine de l'entreprise pour les machines de l'internet. Ce serveur sera sollicitable par toute machine sur l'internet.

Figure 1

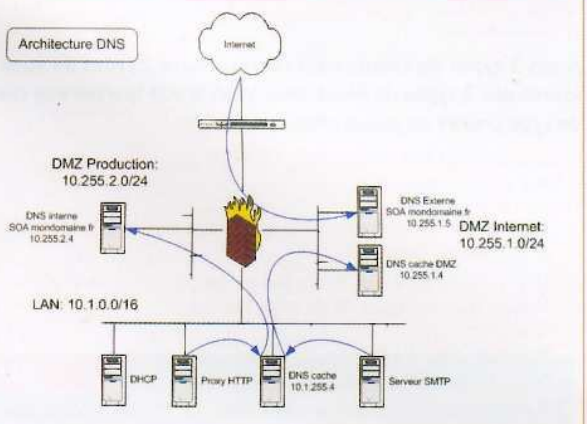


Fig. 1 : Architecture générale

2.2 Résolution d'une requête interne du domaine ou d'un sous-domaine de l'entreprise

Tout d'abord, le resolver envoie sa requête au serveur DNS référencé dans sa configuration. Ce serveur est un DNS cache qui vérifie que le resolver est habilité à demander des résolutions pour le domaine ou sous-domaine demandé via la mise en place d'ACL sur adresses IP (directive `acl`) et de vues (directive `view`). Le DNS cache ne contient aucune zone, mais sait trouver le serveur maître de la zone interrogée (directive `forwarders`). Nous utilisons les fonctionnalités de retransmission sélective pour ce type de résolution (définition de zones de type `forward` et non maître ou esclave). Afin de préserver les serveurs maîtres au mieux des attaques, ceux-ci sont placés dans des DMZ et seuls les serveurs cache sont autorisés à y accéder.

2.3 Résolution d'une requête interne d'un domaine Internet

Le resolver interroge le serveur cache référencé. Le serveur cache vérifie que le resolver est habilité à résoudre les domaines inconnus (donc Internet). Le domaine interrogé n'étant pas connu par le serveur cache comme zone faisant l'objet d'une retransmission sélective, le serveur cache retransmet la requête au serveur cache délégué aux résolutions externes que nous avons pris soin de positionner dans une DMZ. Ici aussi, seuls les serveurs cache sont autorisés à y accéder. En cas d'attaque (via une hypothétique mauvaise implémentation du contrôle d'état UDP sur notre pare-feu par exemple !), l'attaquant n'accédera qu'à un serveur en DMZ uniquement capable de résoudre des requêtes vers Internet.

2.4 Résolution d'une requête externe du domaine de l'entreprise

Dans ce cas, les requêtes proviennent d'Internet vers un serveur que nous hébergeons. Bien entendu ce serveur est en DMZ. Étant donné le peu de sûreté que constitue Internet, il ne nous semble pas utile de mettre en place des habilitations pour les résolutions.

Nous restreignons les données de notre domaine au strict minimum pour Internet. Le resolver trouve alors sa réponse dans le DNS que nous lui proposons, la requête ne doit jamais entrer dans l'entreprise.

3. Serveur DNS cache interne

3.1 Attention au poste de travail !

Comme nous le disions en introduction, la corruption de données de résolution est le problème majeur du DNS. Et bien, la première attaque DNS est d'éviter toute requête DNS !

En effet, la résolution de noms dans les systèmes d'exploitation modernes implique des composants autres que les serveurs DNS.

Par exemple, sous un système Microsoft Windows, le processus de résolution suit les étapes suivantes :

- Vérification si on demande la résolution du propre nom de la machine ;
- Présence du nom dans le fichier `hosts (%Systemroot%\System32\Drives\Etc\hosts)` ;
- DNS ;
- WINS ;
- *broadcast* réseaux ;
- LMHOSTS.

Ainsi, si vous modifiez le contenu du fichier `hosts`, la chaîne de résolution DNS est corrompue. Toute la politique de sécurité DNS est déjà réduite à néant !

Cette méthode est utilisée par exemple par les vers de la lignée MYTOB **[VER]** qui font ainsi pointer des sites de type **symantec.com** ou **update.symantec.com** vers `127.0.0.1` et empêcher ainsi toute mise à jour de l'antivirus.

La solution est donc de s'assurer que les utilisateurs n'ont pas les droits administrateur et donc ne peuvent exécuter de processus pouvant modifier le fichier `hosts`. Idem pour ce qui est de l'accès aux paramètres DNS.

3.2 Configuration du serveur DNS cache interne

Une fois que le poste client nous semble correctement configuré, voyons la configuration du serveur de cache interne. Dans notre exemple, les machines du réseau interne sont dans le réseau `10.1.0.0/16` et le serveur cache interne à l'adresse `10.1.255.4`.

Les vues sous Bind

Nos besoins nous conduiront dans le paragraphe suivant à vouloir faire varier les résultats des requêtes en fonction du type de client (ici PC normaux et machines proxies/pivots). Or, Bind nous donne le moyen de réaliser cela : les vues. Une vue est une configuration de Bind pour répondre à certains clients certains résultats grâce aux instructions suivantes :

- `acl` : permet de définir sous un nom un jeu de clients (adresses IP) ;
- `view` : définit une configuration complète pour un type de client ;
- `match-clients` : instruction indiquant les clients à qui s'adresse une vue.

Les vues sous Bind – suite

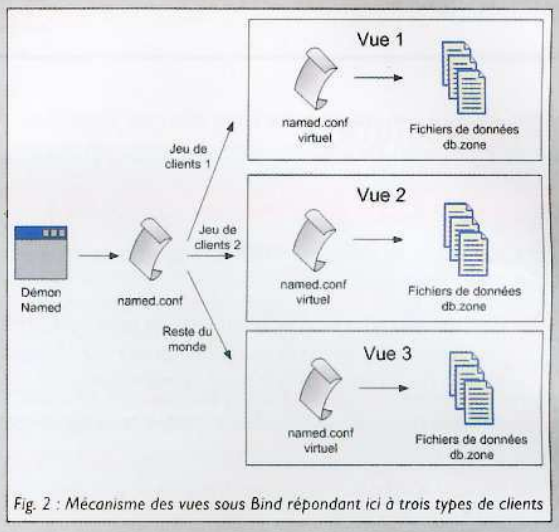


Fig. 2 : Mécanisme des vues sous Bind répondant ici à trois types de clients

Au chapitre 2, nous avons défini différents types de clients au sein du réseau interne (machines standards et machines passerelles). Ces différents types de clients vont se traduire par 2 ACL différentes :

NOTE

Tous les fichiers de configuration sont disponibles sur le site de MISC **[MISC23]**.

```
// ENTETE DU FICHIER NAMED.CONF du DNS CACHE INTERNE
// Definition des différents types de clients
// les proxies
acl "passerelles" { 10.1.255.3/32; };
// les machines du reseau interne de l'entreprise
acl "reseaulocal" { 10.1.0.0/16; };
// le DNS cache en DMZ
acl "dns-cache-internet" { 10.255.1.4/32; };
// le DNS soa interne
acl "dns-soa" { 10.255.2.4/32; };
// localisation des fichiers de configuration
options { directory "/var/named";
version "unavailable"; };
```

À ces 2 types de clients vont correspondre 2 types de vues qui fourniront 2 types de résolution. Voici la vue fournie aux clients de type *proxies* ou passerelles.

```
// Suite du FICHIER NAMED.CONF du DNS CACHE INTERNE
// vue s'adressant aux proxies
view "passerelles" {
    // machines à qui s'adressent cette vue. Ici les proxies de l'entreprise.
    match-clients { "passerelles"; };
    // en cas de requête faite sur un domaine non géré par ce serveur,
    // envoi des requêtes sur le DNS cache internet
    forward only;
    forwarders { 10.255.1.4; };
    // requête sur le mondomaine.fr :
    // redirection de la requête vers le DNS SOA interne
    zone "mondomaine.fr" {
            type forward;
```

```
        forwarders { 10.255.2.4;};
      forward only;
        };
  // redirection identique pour les requetes sur la zone reverse
  zone "10.in-addr.arpa" {
        type forward;
        forwarders { 10.255.2.4; };
   forward only;
        };
};
```

Figure 3

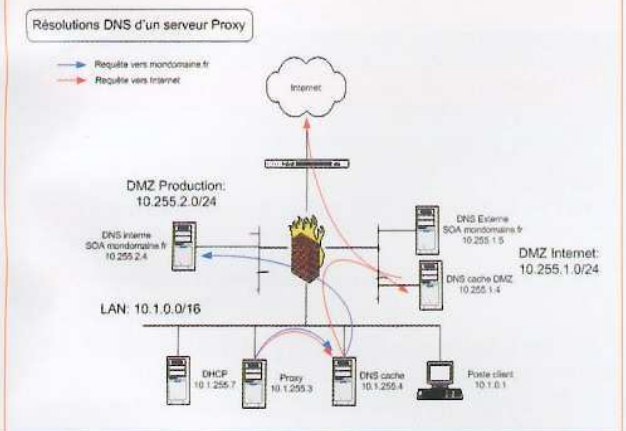


Fig. 3 : Chemins des résolutions issues d'une machine proxy

Les passerelles peuvent effectuer tout type de requête. Ces requêtes seront satisfaites de la manière suivante :

- Requête sur le domaine `mondomaine.fr` : envoi d'une requête auprès du serveur maître interne (IP : `10.255.2.4`) ;
- Requête sur tout autre domaine : envoi vers le serveur cache en DMZ (adresse IP `10.255.1.4`).

Voici ensuite la vue pour les postes lambda du réseau local :

```
// Fin du FICHIER NAMED.CONF du DNS CACHE INTERNE
// vue s'adressant aux PC du réseau interne de l'entreprise
view "reseaulocal" {
      // Machines à qui s'adressent cette vue.
      // Ici, les machines lambda (PC) du réseau interne.
         match-clients { "reseaulocal"; };
      // requête sur le mondomaine.fr :
      // redirection de la requête vers le DNS SOA interne
         zone "mondomaine.fr" {
                  type forward;
                  forwarders { 10.255.2.4; };
            forward only;
         };
      // redirection identique pour les requêtes sur la zone reverse
         zone "10.in-addr.arpa" {
                  type forward;
                  forwarders { 10.255.2.4; };
            forward only;
         };
};
```

NOTE

Les zones de type `forward` ont été introduites en version 9.1.x de Bind.

Figure 4

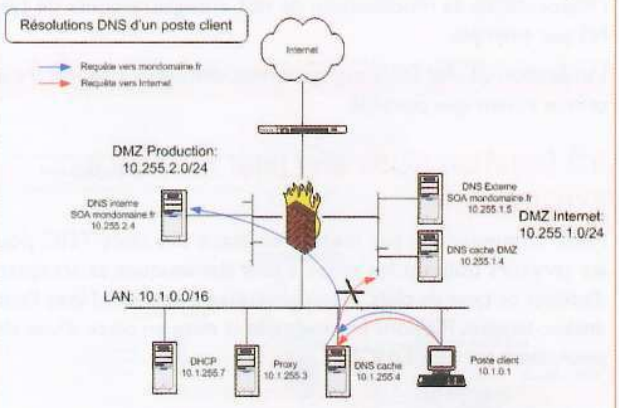


Fig. 4 : Chemins des résolutions issues d'un PC standard du réseau interne

Cette configuration ne permet de résoudre que des noms dans le domaine `mondomaine.fr`. Les autres noms de domaines ne sont résolus que pour les machines passerelles.

Cette mesure combinée à la fermeture de tous les ports réseau (comme HTTP/S ou SMTP) pour les machines du LAN au niveau du *firewall* empêche toute capacité d'évasion par tunnel DNS autonome. Pour que l'évasion soit désormais possible, il faut que l'attaquant dispose du code d'authentification de l'utilisateur sur le proxy applicatif ce qui limite grandement la menace des attaques automatisées. Et nous sortons là du périmètre de la sécurisation du DNS.

4. DNS SOA interne

4.1 Mise en sûreté des informations

Les informations contenues dans le DNS SOA interne font de ce DNS le cœur de notre système. En effet, il est le seul à contenir les données de résolution et doit à ce titre être le plus protégé. Ce DNS est placé derrière un pare-feu voire un IPS implémentant un contrôle d'état sur UDP performant. Des règles de sécurité sont mises en place pour n'autoriser que les DNS cache interne à interroger notre DNS SOA interne. Ces règles sont mises en place sur l'équipement de sécurité réseau et dans la configuration du DNS.

Ces restrictions limitent les risques de déni de services sur notre DNS SOA interne, le nombre de clients simultanés étant limité par le nombre de DNS cache.

4.2 Les mises à jour dynamiques

Certaines applications de l'entreprise peuvent nécessiter la mise en place des mises à jour dynamiques (serveur DHCP, contrôleur de domaine, etc.) sur notre DNS SOA. Dans ce cas, il est intéressant de créer des sous-domaines particuliers pour ces applications, celles-ci ne peuvent alors pas tout modifier dans notre DNS.

Par exemple, on peut créer le sous-domaine `dhcp.mondomaine.fr` sur notre DNS et autoriser le serveur DHCP à y effectuer ses mises à jour sans lui donner de droits sur `mondomaine.fr`. L'option `update-policy` permet de plus de limiter les opérations possibles

sur la zone. Ainsi, nous pouvons autoriser le serveur DHCP à ne modifier que les enregistrements de type A, garantissant ainsi l'impossibilité de modification de nos enregistrements de type NS par exemple.

L'utilisation de clef TSIG est également une sécurité à mettre en œuvre autant que possible.

4.3 Création d'une clef pour mises à jour dynamiques

Nous commençons par mettre en place des clefs TSIG pour les serveurs utilisant les mises à jour dynamiques et acceptant d'utiliser ce type de clefs. Nous générerons notre clef avec l'outil dnssec-keygen. Prenons par exemple la mise en place d'une clef pour notre serveur DHCP :

```
dnssec-keygen -a HMAC-MD5 -b 512 -n HOST dhcp.mondomaine.fr
```

Explication des options :

- L'option `-a` permet de choisir l'algorithme utilisé par la clef (ici `HMAC-MD5`).
- L'option `-b` est la longueur de la clef (ici 512 bits).
- L'option `-n` spécifie le type de clef créé (dans notre cas, il s'agit d'une clef de type `HOST`).
- Le dernier paramètre est le nom de la clef.

Nous obtenons deux fichiers du type `Kdhcp.mondomaine.fr.+157+55315.key` et `Kdhcp.mondomaine.fr.+157+55315.private` contenant chacun la clef. En effet, dans notre cas, ces deux fichiers sont identiques. Les noms de ces fichiers sont toujours formés par la lettre `K` majuscule, suivie du nom de la clef, puis de l'identifiant de l'algorithme utilisé et un identifiant.

Le fichier en `.key` contient notre clef sous la forme :

```
dhcp.mondomaine.fr. IN KEY 512 3 157 S5m+161iLtk6Y7+nK+8jpV1f1l2AHLxN8SqVeZqzxQy
g+L1187PbT5w0 pteHcN4NdwO59t2TYZkpsbUocMvl+g==
```

et le fichier `.private` contient cette même clef mais sous la forme :

```
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: S5m+161iLtk6Y7+nK+8jpV1f1l2AHLxN8SqVeZqzxQyg+L1187PbT5wOpteHcN4NdwO59t2TY
ZkpsbUocMvl+g==
```

Pour nos besoins, nous créons un nouveau fichier ne contenant que la clef. Ce fichier s'appelle `clefdhcp.key` et contient :

```
secret "S5m+161iLtk6Y7+nK+8jpV1f1l2AHLxN8SqVeZqzxQyg+L1187PbT5wOpteHcN4NdwO59t2
TYZkpsbUocMvl+g==";
```

4.4 Configuration

Nous filtrons les adresses IP source via l'option `allow-query`.

```
acl "dns-cache-interne" { 10.1.255.4/32; };
acl "updateservers" { 10.1.255.7/32; };

options {
        // Répertoire où se trouvent les fichiers de zone
        directory "/var/named";

        // Modification du nom de version renvoyé
        version "unavailable";

        // Interdiction des transferts de zone
        allow-transfer{"none";};

        // Interdiction des notifications de mise à jour
        allow-notify {none;};

        // Autorisations de requêtes internes et pour le DNS cache
        allow-query {127.0.0.1; "dns-cache-interne";"updateservers";};

        // Pas de récursion
        recursion no;
};

// Définition de la clef du serveur DHCP
key "key.dhcp.mondomaine.fr" {
        algorithm hmac-md5;
        include "clefdhcp.key";
};

// Définition de la sous-zone maitre dhcp
zone "dhcp.mondomaine.fr" {
        type master;
        file "db.dhcp.mondomaine.fr";

        // Définition des droits de mise à jour (type A pour le serveur utilisant
la clef dhcp)
            update-policy {
        grant key.dhcp.mondomaine.fr name dhcp.mondomaine.fr A ;
          };
            forwarders {};
};

// Définition de la zone maitre
zone "mondomaine.fr" {
        type master;
        file "db.mondomaine.fr";
        forwarders {};
};
```

5. Serveur DNS cache en DMZ

Ce serveur a pour vocation d'effectuer pour le compte du serveur DNS cache interne les résolutions de noms de domaines internet. Cette particularité nous permet de restreindre les clients pouvant le solliciter à ce seul serveur.

NOTE

Si vous utilisez une version de Bind inférieure à 9.x, il faut prendre soin de se prémunir de 2 risques majeurs de corruption de caches :

- Le *glue fetching* qui consiste pour un serveur, recevant un enregistrement NS sans enregistrement de type A, à chercher à obtenir cet enregistrement A entraînant des risques de corruption du cache.
- La prédiction du numéro de l'ID du message DNS **[HEADER]** qui rend le serveur sensible aux attaques de force brute.

Voici la configuration du serveur :

```
// définition d'une ACL pointant le DNS cache interne
acl "dns-cache-interne" { 10.255.1.4/32; };

options {
  // répertoire contenant les fichiers de configuration
  directory "/var/named";

  // aucun transfert de zone autorisé
  allow-transfer{"none";};
  // seul le dns cache interne est autorisé à
  allow-query { "dns-cache-interne"; };
```

```
 // pour BIND 8 : randomisation de l'ID de message DNS
 //use-id-pool yes;
 // pour Bind 8 : pas de glue fetching
 //fetch-glue no;
};

// Définition de la zone racine :
// Elle contient la liste des serveurs racines à qui le serveur va s'adresser
// pour commencer à résoudre les requêtes.
zone "." {
  type hint;
  file "/var/named/named.root";
}
```

Ports DNS et Firewall

Voici un petit mémento sur les ports utilisés pour les requêtes DNS :

- Envoi de la requête port > 1023 vers port 53 en TCP ou UDP ;
- Réponse à la requête : port 53 vers port > 1023 en TCP ou UDP.

On dit généralement que l'UDP est utilisé pour les requêtes simples et TCP pour les transferts de zones. Mais, en fait, en fonction de la taille de la réponse, TCP peut être obligatoire même pour les requêtes simples.

6. Serveur DNS SOA externe

Le serveur DNS Internet est celui le plus exposé en considérant le danger extérieur (nous ne reviendrons pas sur ce vaste sujet...). Il est donc important de bien lui appliquer les règles de sécurité appliquées à tout serveur disponible sur Internet.

Au niveau de la configuration DNS, elle est très simple. Il suffit de créer une zone de type maître sur ce serveur et d'interdire toute récursion.

Il faut ensuite remplir la zone avec les données devant être disponibles sur Internet.

```
options {
        // Répertoire où se trouvent les fichiers de zone
        directory "/var/named";

        // Modification du nom de version renvoyé
        version "unavailable";

        // Interdiction des transferts de zone
        allow-transfer{"none";};

        // Interdiction des notifications de mise à jour
        allow-notify {none;};

        // Autorisations de requêtes
        allow-query {any;};

        // Pas de récursion
        recursion no;
};

// Définition de la zone maître
zone "mondomaine.fr" {
        type master;
        file "db.mondomaine.fr";
        forwarders {};
};
```

Figure 5

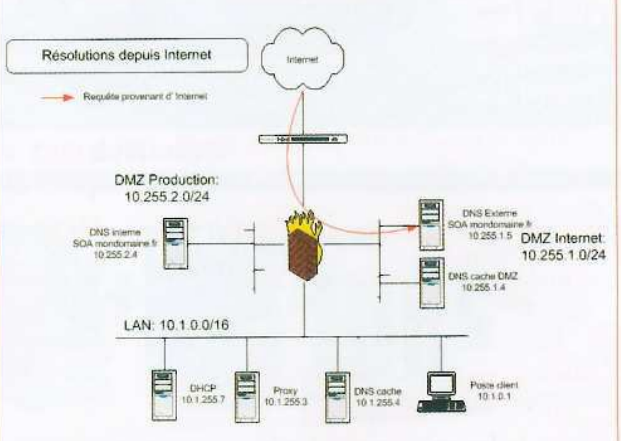


Fig. 5 : Requête sur mondomaine.fr émise par un poste sur l'internet

NOTE

Nous pouvons fusionner physiquement les 2 serveurs DNS Internet (le cache et le serveur DNS internet). La configuration utiliserait alors le principe des vues en fonction de l'adresse appelante comme nous l'avons vu au chapitre 3 pour la configuration du DNS cache interne.

7. Les logs

Par défaut, le démon `named` envoie tous ses messages au `syslog` de la machine qui peut aussi recevoir les messages des autres démons du système. Le manque de lisibilité d'un tel `syslog` nuit directement à la sécurité du DNS en gênant sa supervision.

Heureusement, il est possible de configurer les logs de `named` via des canaux de logs afin de, dans notre cas, rediriger certains types de messages dans des fichiers spécifiques. `named` peut utiliser plusieurs canaux de logs en parallèle traitant les mêmes informations ou non. Dans notre exemple, nous utilisons deux canaux de logs, un premier pour les messages concernant les problèmes de sécurité et les transferts de zone et un second pour les autres messages et les transferts de zone (messages par défaut).

NOTE

Ces lignes de configuration sont à placer dans les fichiers `/etc/named.conf` de tous les DNS.

```
logging {
  // Paramétrage du canal des messages de sécurité
  channel securitylogs {
  // Envoi vers le fichier dns_security.log avec roulement de 10 archives de 5Mo
        file "/var/log/named-sec.log" versions 10 size 5m;
  // Affichage de tous les messages (+messages debug si activation du mode
debug)
```

```
        severity dynamic;
    // Affiche le nom de la catégorie du message
        print-category yes;
    // Affiche la sévérité du message dans les logs
        print-severity yes;
    // Affiche la date du message dans les logs
        print-time yes;
    };

    // Paramétrage du canal par défaut
    channel defaultlogs {
    // Envoi vers le fichier dns_default.log avec roulement de 10 archives de 5Mo
        file "/var/log/named-default.log" versions 10 size 5m;
    // Affiche les messages de sévérité "info" et supérieur
        severity info;
    // Affiche le nom de la catégorie du message
        print-category yes;
    // Affiche la sévérité du message dans les logs
        print-severity yes;
    // Affiche la date du message dans les logs
        print-time yes;
      };

    // Envoi des messages par défaut à notre canal defaultlogs
      category default { defaultlogs; };

    // Envoi des messages de transferts à nos canaux defaultlogs et securitylogs
      category xfer-in {  securitylogs; defaultlogs; };
      category xfer-out {  securitylogs; defaultlogs; };

    // Envoi des messages de sécurité à notre canal securitylogs
      category security { securitylogs; };
};
```

Il est également possible de créer un canal spécial pour les mises à jour dynamiques sur les serveurs utilisant ce service :

```
channel updatelogs {
        file "/var/log/named-update.log" versions 10 size 5m;
        severity dynamic;
        print-category yes;
        print-severity yes;
        print-time yes;
    };
category update { updatelogs; };
```

Voici deux exemples de messages lus dans le fichier de logs de sécurité :

```
Nov 25 10:34:57.910 security: info: client 192.168.9.9#1349: query (cache) denied
Nov 25 15:22:53.390 security: error: client 192.168.9.9#32776: zone transfer
'mondomaine.fr/IN' denied
```

Ces deux messages ont les significations suivantes :

- Le premier est une interdiction d'interrogation.
- Le second est une interdiction de transfert de zone.

8. Conclusion

Comme nous avons pu le voir, la sécurisation du DNS passe avant tout par le fait de se poser les bonnes questions sur son utilisation. Qui l'utilise ? Pour quels types de résolution ? Est-ce légitime ?

La réponse à ces questions sous l'angle de la sécurité nous a permis d'arriver aux solutions suivantes :

- Une séparation des différents services DNS (résolution internet vs résolution du domaine interne) par spécialisation des serveurs ;
- Une politique centralisée et maîtrisée d'accès à l'internet limitant le spectre de clients pouvant solliciter et donc potentiellement exploiter le DNS.

Une amélioration supplémentaire pourrait être l'utilisation de DNSSEC [DNSSEC] pour le contrôle d'intégrité et l'authentification des demandes de résolutions de noms et les réponses associées. Cette utilisation serait à implémenter au niveau du DNS SOA internet.

L'ensemble de cette architecture a bien entendu un coût par rapport à un DNS non maîtrisé (postes ouverts sur l'internet, etc.). Ce coût doit être identifié et prévu tant sur le plan financier, organisationnel que humain. En effet, la sécurisation du composant DNS au sein du SI ne pourra être menée à bien sans cette étude d'impact. Et nous espérons que cet article vous aidera à calibrer vos besoins par rapport à votre environnement.

Références

[TUN] VUILLARD, V. « Tunnel DNS : fuite d'information universelle », in *MISC 18*, Mars-avril 2005.

[VER] TREND Micro System, Description du virus MYTOB.KG sur **http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM%5FMYTOB%2EKG&VSect=T**

[MISC23] BROCAS C., FARIN J.M., Fichiers de configuration des différents serveurs DNS décrits sur **http://www.miscmag.com/fr/articles/23-MISC/secdns/**

[BIND] Page d'accueil de BIND, **http://www.isc.org/sw/bind/**

[HEADER] RFC 1035, « *Domain names – implementation and specification* », **http://www.ietf.org/rfc/rfc1035.txt?number=1035, chapitre 4.1.1**

[DNSSEC] RFC 4033, 4034 et 4035, Ressources DNSSEC, **http://www.ietf.org/rfc/rfc4033.txt**, **rfc4034.txt** et **rfc4035.txt**.

MISC
est édité par Diamond Editions
B.P. 121 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@miscmag.com
Abonnement : abo@miscmag.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Bodor
Conception graphique & mise en page : Franck TOUSSAINT

Impression : Presses de Bretagne
Secrétaire de rédaction : Dominique Grosse
Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08

Distribution :
(uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes : Distri-médias :
Tél. : 05 61 72 76 24
Service abonnement :
Tél. : 03 88 58 02 08

Dépôt légal : 2e Trimestre 2001
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K 81 190
Périodicité : Bimestrielle
Prix de vente : 7,45 euros

Imprimé en France

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

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le sytème, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent.

MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.