GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT SUR SYSTÈMES UNIX, OPEN SOURCE & EMBARQUÉ

N°206
JUILLET - AOÛT 2017

FRANCE
MÉTRO. : 7,90 €
DOM/TOM : 8,50 €
BEL/LUX/PORT.
CONT. : 8,90 €
CH : 13 CHF
CAN : 14 $CAD

L 19275 - 206 - F: 7,90 € - RD

Hack / Valgrind / Moteur de jeu

FAITES DU JEU DUKE NUKEM 3D

UN OUTIL SYSADMIN ! p.38

- Analysez le code du jeu libéré
- Personnalisez ses fonctionnalités
- Intégrez vos modifications

Python / Bot

NE TWEETEZ PLUS ! LAISSEZ VOTRE ROBOT LE FAIRE POUR VOUS ! p.16

NCIS / Hack

DÉCOUVREZ COMMENT BASCULER SUR UN NOUVEL OS ET UN NOUVEAU NOYAU AVEC UN SEUL REBOOT p.52

Sécurité / Signature électronique

MODIFICATION DE CONTRATS NUMÉRIQUES IMMUABLES ETHEREUM p.64

Nginx / Maven

METTEZ EN PLACE UN CACHE HTTP SUR UN SERVEUR JENKINS ET ACCÉLÉREZ LES CONSTRUCTIONS DE PROJETS JAVA p.28

Assembleur / Reverse Engineering

DÉSASSEMBLEZ ET ANALYSEZ DU CODE AVEC RADARE2 p.90

EN PLUS : COMPAREZ LES LANGAGES PYTHON ET GO - SURVEILLEZ DES RÉPERTOIRES AVEC INOTIFYWAIT...

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

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

GNU/Linux Magazine France
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 : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Résponsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 – v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

MIXTE
Papier issu de sources responsables
FSC® C015136

Membre April
www.april.org

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

p.43/44 DÉCOUVREZ TOUS NOS ABONNEMENTS MULTI-SUPPORTS !

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

EN VERSION PAPIER ET PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITO

Les mois d'été sont des mois propices à la détente, au jeu et permettent également de s'atteler enfin à telle ou telle tâche remisée de longs mois durant dans une liste (*todo list* pour les anglophones) qui, si elle n'avait pas été numérique, serait recouverte d'une épaisse couche de poussière. Dehors, le soleil risque de brûler votre délicate peau d'informaticien, blanchie par les nombreuses heures passées à recevoir un délicat rayonnement de la part de votre (vos ?) moniteur(s). Il vaut donc mieux rester enfermé, bien à l'abri ! Mais ce sont les vacances… vous n'allez tout de même exécuter mécaniquement et inlassablement les mêmes tâches...

Et si, exceptionnellement, pour une fois, vous joigniez l'utile à l'agréable en installant un jeu tout récent, sorti l'année dernière ? Ah non, attendez... On me dit que c'est un peu plus vieux… Plus de 20 ans ? **Duke Nukem 3D** a plus de 20 ans ? Pourtant je me souviens encore très bien de tous les niveaux, comme si c'était hier. Personne n'a pu oublier ce missile qui traverse l'écran alors que vous débutez le jeu sur la terrasse d'un bâtiment ni les différents passages secrets en passant au travers d'un écran de projection, d'un mur, etc. Eh bien soit, si Duke Nukem 3D a 20 ans, vous pourrez jouer à un vieux jeu, dépassé graphiquement, mais possédant un très bon *gameplay* ! Pour les nostalgiques, vous retrouverez des écrans que vous pensiez oubliés et pour les autres vous pourrez découvrir un très bon jeu « à l'ancienne » (ce qui est normal vu son âge vénérable…). Puis, lorsque vous serez lassés de *tataner* de l'alien, vous pourrez analyser son code, libéré en 2003 sous licence GPL, pour y intégrer toutes les modifications que vous souhaitez. Nous vous proposons de lister les appels systèmes d'une commande donnée à chaque fois qu'un ennemi est tué. Rien ne vous empêchera par la suite d'adapter le code pour lister et détruire des fichiers temporaires, des processus, ou toute autre opération fastidieuse à laquelle vous désirez ajouter un petit air de vacances :-)

Et puisque nous sommes en période estivale de tests divers, pourquoi ne pas construire un petit bot **Twitter ou** faire du *reverse engineering* avec **Radare2**. C'est l'été que diable, amusez-vous ! Et si vous ne pouvez vraiment pas faire autrement que de sortir, vous pourrez toujours lire votre *Linux Magazine*. Pour l'occasion nous lui avons même ajouté une fonctionnalité spéciale : le *ShadowMag 2.1* ! Bien orienté, vous pouvez utiliser votre magazine pour vous faire de l'ombre pendant la lecture ! Sachez également que, contrairement à certains de nos concurrents qui n'hésitent pas à déployer pléthore d'arguments racoleurs, nous avons choisi de ne pas intégrer de fonctionnalités fantaisistes de type *BarbeukStarter 5.2* ou encore *PaperPlane 0.2b*. Nous pensons que le contenu du magazine, la qualité des articles publiés chaque mois sont des arguments suffisant pour faire de *Linux Magazine* VOTRE magazine. N'hésitez donc pas à nous écrire à lecteurs@gnulinuxmag.com pour nous faire part de vos remarques ou de vos propositions d'articles. Et même si vous trouvez le magazine sans défaut, dites-le-nous ! :-)

Bonnes vacances, je vous retrouverai à la rentrée avec plaisir !

Tristan Colombo

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

CONNECT ÉVOLUE !

LISEZ CE NUMÉRO ET PLUS DE 230 AUTRES EN LIGNE !

ACTUELLEMENT SUR CONNECT :

CE NUMÉRO
et + de 160 autres numéros de GNU/Linux Magazine

+

75 numéros Hors-Séries de GNU/Linux Magazine

TOUT CELA À PARTIR DE 199 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

Appelez le 03 67 10 00 28 et donnez le code « GLMF206 » pour découvrir Connect gratuitement pendant 1 mois !

Pour tous renseignements complémentaires, contactez-nous via notre site internet : **www.ed-diamond.com**, par téléphone : **03 67 10 00 28** ou envoyez-nous un mail à **connect@ed-diamond.com** !

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE

N°206

ACTUS & HUMEUR

06 JE SUIS PASSÉ DE PYTHON À GO...

Après 15 ans à privilégier Python pour tous types de travaux, un contrat intéressant auprès d'un de mes clients m'a amené à découvrir Go...

IA, ROBOTIQUE & SCIENCE

16 CRÉEZ UN ROBOT QUI TWEETE POUR VOUS

Vous aimez tweeter, mais vous n'avez pas forcément le temps de répondre à des questions qui vous sont posées ?...

SYSTÈME & RÉSEAU

28 CACHE MAVEN PARTAGÉ AVEC NGINX

Le serveur HTTP Nginx est un outil souple et très puissant, utilisé par de nombreux administrateurs de systèmes, mais peu connu des développeurs Java...

KERNEL & BAS NIVEAU

38 DUKE NUKEM 3D : UN OUTIL VALGRIND ADAPTÉ À LA LECTURE D'APPELS SYSTÈMES

On ne présente plus ni Duke Nukem 3D, ni Valgrind. Les deux outils sont très complémentaires : d'un côté, nous avons un émulateur de processeur RISC qui permet de faire une abstraction des instructions d'origine, d'un autre vous avez un jeu d'arcade qui permet de supporter de longues heures passées sur le premier !

52 LE TEST DE PETER

On a tous un ou deux amis bizarres. C'est le cas par exemple de mon ami Peter. Dès qu'il trouve quelque chose de nouveau, il ne peut s'empêcher de faire un test. Il teste donc les spécialités culinaires, les paradigmes de programmation, les techniques de drague...

HACK & BIDOUILLE

62 SURVEILLEZ UN DOSSIER DE SOURCES POUR EXÉCUTER UNE COMMANDE AUTOMATIQUEMENT

Lorsque l'on développe avec certains outils, un site web avec Jekyll par exemple, on vous propose une option qui se révèle bien pratique, souvent appelée --watch. Lorsqu'elle est activée, l'outil surveille le dossier contenant votre code source pour le recompiler dès qu'une modification intervient...

LIBS & MODULES

64 BLOCKCHAIN : MODIFIER UN CONTRAT IMMUABLE

Dans un monde immuable, comment corriger un bug ?...

78 UTILISEZ DROPBOX DEPUIS PYTHON

DropBox est un service de stockage en ligne extrêmement populaire qui vous permet de sauvegarder des fichiers, ou des répertoires, d'en gérer des versions successives, de les visualiser en ligne et les partager...

MOBILE & WEB

84 PHP-RBAC : GÉREZ LES DROITS DE VOTRE APPLICATION À L'AIDE DE RÔLES

La gestion fine des droits des utilisateurs au sein des applications métier est un calvaire pour l'administrateur...

SÉCURITÉ & VULNÉRABILITÉ

90 L'ART DU REVERSE AVEC RADARE2

Dans un article de MISC de mai 2013, Nicolas RUFF faisait une introduction au reverse engineering, en concluant sur la nécessité de développer en France cette compétence décrite comme le « Graal » en sécurité informatique. Dans cet article, il réalisait un éloge du logiciel IDA et de son géniteur, et ce à juste titre...

ABONNEMENTS

43/44 : abonnements multi-supports

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

JE SUIS PASSÉ DE PYTHON À GO...

SÉBASTIEN MACCAGNONI
[DevOps depuis le XXe siècle]

MOTS-CLÉS : PYTHON, GO, GOFMT, COMPARAISON, LANGAGES

Après 15 ans à privilégier Python pour tous types de travaux, un contrat intéressant auprès d'un de mes clients m'a amené à découvrir Go. J'avais des préjugés plutôt négatifs, finalement je ne peux pas le nier : j'aime programmer en Go !

Dès ma première rencontre avec **Python** il y a un bon paquet d'années (ça date de Python 2.1), j'ai été séduit par ce langage : facile à apprendre, facile à lire, facile à écrire... C'est là que j'ai appris que le développement n'est pas nécessairement douloureux, que certains langages permettent d'avoir un résultat rapide et fonctionnel, motivant pour aller plus loin. Je pourrais faire une longue liste des choses qui me plaisent dans ce langage : la simplicité de la syntaxe, la myriade de modules par défaut, l'orientation générale (qui transparait dans le *Zen of Python*), la versatilité du langage...

Dans les nombreuses missions que j'ai réalisées toutes ces années, Python a toujours été mon langage de prédilection, quel que soit le besoin. Par la force des choses, un certain nombre d'entre elles ont porté sur des applications web, domaine pour lequel j'ai choisi de travailler avec le *framework* **Flask**, avec des bases de données **PostgreSQL** ou éventuellement **MySQL**, plus récemment (par choix personnel) avec **MongoDB**.

Fin 2016, un de mes clients m'a contacté concernant une mission de développement de quelques mois, pour laquelle j'ai eu le choix du langage : soit je travaillais en Python, soit je montais en compétences sur **Go**. Ce client a une

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

base de code Python assez importante, il a cependant choisi de passer sur Go pour différentes raisons, que j'ai découvertes au fil de cette mission. Ce fut l'occasion d'essayer ce langage : j'avais prévu de retourner sur Python si je n'arrivais pas à être productif avec Go rapidement. Comme vous pouvez le deviner, j'ai choisi d'y rester et la transition a été plutôt aisée !

Voici les aspects de Go qui me semblent les plus importants lorsque j'essaie de le comparer avec Python ; chacun des points abordés est accompagné de deux bouts de code équivalents, dans les deux langages... Attention, il ne s'agit pas là de vous apprendre à programmer en Go : il y a certains concepts que je passe sous silence et certains bouts de code qui peuvent être optimisés, à vous d'approfondir si ça vous intéresse.

Quoi qu'il en soit, s'il y a une chose dont je me rends compte en étant passé à Go, c'est ironiquement un élément du Zen de Python : *explicit is better than implicit !*

1. SYNTAXE

Ma première peur, quand je vois un autre langage que Python, c'est sa syntaxe. Python m'a tellement habitué à une syntaxe ultra-claire que j'ai une espèce de dégoût quand je vois des accolades, des parenthèses et autres points-virgules. En cela et à première vue, Go ne m'a pas laissé une bonne impression quand j'ai vu un premier bout de code : mon dieu, des accolades !

Je me suis quand même accroché, je n'allais pas abandonner pour si peu. J'ai alors remarqué qu'il n'y avait pas de point-virgule, pas de parenthèses inutiles... On reste tout de même assez proche de Python, en remplaçant les doubles-points et les indentations par des accolades (tableau n°1)...

Python	Go

`if 1 + 2 == 3:`
`  print("OUAIP")`
`else:`
`  # Impossible`
`  print("NOPE")`
	`if 1 + 2 == 3 {`
	`  fmt.Println("OUAIP")`
	`} else {`
	`  // Impossible`
	`  fmt.Println("NOPE")`
	`}`

Tableau n°1

De manière globale, Go est plus verbeux que Python, mais cela ne veut pas dire qu'on en écrit trop : il y a simplement moins de choses qui sont cachées.

De plus, cette syntaxe peut être imposée à n'importe quel code, avec la commande **gofmt** ; cela rend Go en même temps plus flexible et plus strict que Python. On peut écrire comme on veut, en respectant ou non les indentations et les espacements, le compilateur comprendra le code : on fait ce que l'on veut. En même temps, cette commande rétablit la mise en forme standard à n'importe quel code en Go dès qu'on le lui demande : si une autre personne a présenté son code n'importe comment, on rétablit un code plus compréhensible en une fraction de seconde.

Les environnements de développement et greffons adaptés à Go prennent cela en compte et proposent d'exécuter **gofmt** à chaque fois que l'on sauvegarde un fichier. De cette manière, on peut programmer sans faire attention à la présentation du code, celui-ci sera remis en forme dès que l'on sauvegarde. Depuis que j'y ai goûté, je ne peux plus m'en passer : mon code est propre alors que je ne fais pas attention ! Avec Python, on peut utiliser **autopep8**, en Go c'est standard.

Revenons sur un sujet de troll constant... Peut-être vous demandez-vous combien d'espaces composent l'indentation par défaut en Go... Et bien en Go, on utilise la tabulation ! Il n'y a pas de discussion à avoir, pas de temps perdu : quand on utilise **gofmt**, on se retrouve avec des tabulations dans le code et on peut passer à un autre sujet plus productif : RIP les trolls.

2. TYPAGE ET DÉCLARATION DE VARIABLES

Ah, le typage... On pourrait écrire des pages sur les avantages de l'une ou l'autre des méthodes... Pour ma part, j'ai été séduit par le typage dynamique de Python, car ça simplifie énormément les débuts d'un développement : on n'a pas besoin de réfléchir au type de la donnée à traiter. Mais on rencontre ensuite des problèmes un peu idiots, que l'on ne remarque parfois que tardivement, avec des exceptions qui nous font penser « ah bah oui, c'est évident »...

Go utilise un typage statique, structurel avec inférence de type. En clair, une variable a toujours un type donné, mais on n'est pas obligé de le déclarer, car le compilateur sait deviner quel doit être le type. On est par contre obligé de déclarer les types nécessaires aux arguments de fonctions ou le type retourné par la fonction.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
var a int
var b string
a = 1
b = "un"
```

```
var (
    a int
    b string
)
a = 1
b = "un"
```

```
var a = 1
var b = "un"
```

```
var (
    a = 1
    b = "un"
)
```

```
var (
    a int
    b = "un"
)
a = 1
```

```
var a = 1
b := "un"
```

```
a := 1
b := "un"
```

Tableau n°2

Go propose plusieurs moyens de déclarer des variables. Tout d'abord, on peut réunir des déclarations dans un bloc **var**. Dans ce bloc, on peut simplement déclarer les variables (auquel cas on précise leur type), on peut également les initialiser avec une valeur. Mais on peut se passer de ce bloc grâce à la syntaxe **:=**, qui initialise une variable non déclarée ; on ne peut par contre pas utiliser cette syntaxe plusieurs fois pour la même variable : la seconde occurrence résultera en une erreur parce que la variable est déjà initialisée. Ces méthodes peuvent être mélangées au sein d'une même fonction. Toutes les syntaxes Go suivantes sont alors équivalentes (tableau n°2).

Notons par ailleurs qu'une variable qui est déclarée, mais non initialisée contient tout de même une valeur par défaut (appelée *« zero value »*) : zéro pour le type **int**, une chaîne vide pour le type **string**, etc.

Afin de simplifier la déclaration des variables, on peut ne déclarer le type qu'une seule fois pour plusieurs variables, en séparant leurs noms par des virgules : **var a, b int**.

De plus, Go ne transforme jamais les types tout seul, contrairement à Python. Par exemple, pour additionner un **int** et un **float**, en Python cela retournera automatiquement un **float** alors qu'en Go, il faut instancier explicitement l'**int** en **float** avant l'addition (tableau n°3).

Python	Go
	Ici, la fonction **add** attend un **int** et un **float** et retourne un **float**. Dans la fonction **test**, on déclare **a** explicitement et **b** implicitement :

`def add(x, y):`
`  return x + y`
 	
`def test():`
 	
`  a = 2`
`  b = 3.5`
 	
`  if add(a, b) == 5.5:`
`  print("OUAIP")`
`  else:`
`  print("NOPE")`
	`func add(x int, y float64) float64 {`
	`  return float64(x) + y`
	`}`
	
	`func test() {`
	`  var (`
	`  a = 2`
	`  b = 3.5`
	`  )`
	`  if add(a, b) == 5.5 {`
	`  fmt.Println("OUAIP")`
	`  } else {`
	`  fmt.Println("NOPE")`
	`  }`
	`}`

Tableau n°3

Avec un typage statique, de nombreuses erreurs rencontrées en Python ne peuvent tout simplement pas exister en Go : tous les appels à des fonctions et toutes les opérations sont vérifiées et le programme ne compile pas s'il y a un conflit, avant même de le lancer, d'où un gain de temps certain pour le développeur !

3. GESTION DES ERREURS

Avec Python, la gestion des erreurs se fait par le biais d'exceptions, interceptées dans des blocs **try...except**. Ces blocs peuvent être imbriqués et les exceptions peuvent remonter d'un bloc **try** à son parent.

À l'inverse, en Go, l'erreur se gère généralement explicitement plutôt que par le biais d'exception. La plupart des fonctions qui peuvent être amenées à échouer retournent deux valeurs, dont la seconde est une erreur, une instance d'un objet de type **error** (ah oui : les fonctions peuvent retourner plusieurs valeurs, de la même manière qu'en Python).

Lorsque la fonction réussit, la valeur de cette erreur (que l'on place habituellement dans une variable nommée **err**) est **nil** (plus ou moins équivalent à **None** en Python). Lorsqu'elle échoue, l'erreur contient la raison de l'échec et la valeur de retour est la valeur par défaut du type concerné (la *« zero value »* mentionnée plus haut) ; dans tous les cas, cette valeur ne sera jamais aléatoire. On peut alors soit ignorer l'erreur (en se contentant de la valeur par défaut) soit la traiter (tableau n°4)...

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Python

```python
def test():
    a = "Salut"
    try:
        b = int(a)
    except:
        b = 0
        print("{} n'est pas un entier".format(a))

    print(b)
```

Go

```go
func test() {
    a := "Salut"

    b, err := strconv.Atoi("Salut")
    if err != nil {

        fmt.Printf("%s n'est pas un entier", a)
    }
    fmt.Println(b)
}
```

Tableau n°4

Avec ces deux valeurs en retour de la fonction, on sait qu'il y a une erreur à traiter (ou à ignorer si on le souhaite). Si on ne la traite pas immédiatement, elle est perdue et le programme continue avec une valeur par défaut, sans planter.

4. FOR : LA BOUCLE UNIVERSELLE

Voici une petite particularité amusante : il n'y a pas de mot-clé **while** en Go. En effet, c'est **for** qui remplit ce rôle. Il n'y a en Go qu'une seule instruction de boucle.

De plus, l'instruction **for** de Go fonctionne nativement comme celle de C (boucle sur une série de valeurs) et non comme celle de Python (boucle sur des éléments). Elle prend trois instructions :

- une instruction d'initialisation (par exemple **i=0**) ;
- une instruction de condition (par exemple **i<10**) ;
- une instruction de fin d'itération (par exemple **i++**).

Les instructions d'initialisation et de fin d'itération sont optionnelles : c'est en n'ayant qu'une instruction de condition que **for** « devient » un **while**.

On peut également utiliser l'instruction **range** dans l'instruction de condition ; cette instruction parcourt un itérable : et voilà, on peut retrouver le même fonctionnement de **for** qu'en Python. Dans ce cas, chaque occurrence de la boucle initialise deux variables, le compte et la valeur (comme **enumerate** en Python).

Enfin, on peut exécuter **for** sans même aucune instruction de condition : dans ce cas, la boucle est infinie (tableau n°5)...

Python

```python
def boucles():



    for i in range(0, 10):
        print(i)

    i = 0
    while i < 10:
        print(i)
        i = i + 1

    a = [43, 24, 66, 12]
    for i in a:
        print(i)

    for i, j in enumerate(a):
        print("{}: {}".format(i, j))

    while True:
        print("Bloqué dans une boucle !")
        time.sleep(1)
```

Go

```go
func boucles() {
    var (
        i, j int
    )
    for i = 0; i < 10; i++ {
        fmt.Println(i)
    }
    i = 0
    for i < 10 {
        fmt.Println(i)
        i++
    }
    a := []int{43, 24, 66, 12}
    for _, i = range a {
        fmt.Println(i)
    }
    for i, j = range a {
        fmt.Printf("%d: %d\n", i, j)
    }
    for {
        fmt.Println("Bloqué dans une boucle !")
        time.Sleep(1*time.Second)
    }
}
```

Tableau n°5

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

5. CLASSES OU STRUCTURES

En Go, l'élément de base de la programmation orientée objet, c'est la structure. On retrouve ici l'inspiration du C et du C++, où les classes et les structures sont des éléments très proches. Lorsque l'on définit une fonction, on peut choisir de l'appliquer à une structure : elle devient alors une méthode de la structure. On retrouve assez facilement une logique assez similaire à ce que l'on connaît en Python (tableau n°6)...

Python

```
class Add:


    def __init__(self, x, y):
        self.x = x
        self.y = y


        def run(self):
        return self.x + self.y


def petite_addition():
    addition = Add(3, 4.2)
    print(addition.run())
```

Go

```
type add struct {
    x int
    y float64
}

func NewAdd(x int, y float64) *add {
    theAdd := &add{
        x : x,
        y : x
    }
    return  theAdd
}

func (a add) run() float64 {
    return float64(a.x) + a.y
}

func petiteAddition() {
    addition := NewAdd(3, 4.2)
    fmt.Println(addition.run())
}
```

Tableau n°6

6. TRAITEMENT DES DONNÉES : LES TAGS

Les structures sont très utilisées pour le stockage de données : c'est l'élément de base dès que l'on interagit avec une base de données, quel que soit son format.

On utilise alors ce qu'on appelle les tags, pour faire une correspondance entre la donnée « interne » en Go et la donnée lue ou écrite. Grâce à cela, on lit et écrit très facilement (et de manière similaire) du **JSON**, du **YAML**, du **XML**, du **SQL**... Cela simplifie énormément le traitement de données, comparé aux deux méthodes utilisées en Python : soit une extraction dans des dictionnaires, soit des classes spécifiques à chacune des bibliothèques utilisées. Par conséquent, si on change la source de données, en Go il suffit d'adapter les tags dans les structures ; en Python, il peut être nécessaire de réécrire toute une fonction.

Ici, l'exemple en Python est plus court, mais le code Go est plus explicite et plus facile à traiter par la suite (tableau n°7)...

Python

```
def display_people():


    ppl = ET.parse('people.xml').getroot()
    for p in ppl.getchildren() :
        id_ = int(p.attrib['id'])
        name = p.find("name").text
        age = int(p.find("age").text)
        print(
            "L'ID {}, {}, a {} ans".format(
                id_, name, age
                )
            )
```

Go

```
type person struct {
    ID   int    'xml:"id,attr"'
    Name string 'xml:"name"'
    Age  int    'xml:"age"'
}
type people struct {
    People []person 'xml:"person"'
}

func displayPeople() {
    var ppl people
    c, _ := ioutil.ReadFile("people.xml")
    xml.Unmarshal(c, &ppl)
    for _, p := range ppl.People {


        fmt.Printf(
            "L'ID %d, %s, a %d ans\n",
            p.ID, p.Name, p.Age,
        )

    }
}
```

Tableau n°7

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ces exemples lisent un fichier XML de ce type :

```
<people>
  <person id="54"><name>Pierre Dupont</name><age>32</age></person>
  <person id="75"><name>Pascal Dupuis</name><age>54</age></person>
</people>
```

7. LISTES ET DICTIONNAIRES

Pour les équivalents des listes en Go, on parlera d'*arrays* et de *slices*. Pour les dictionnaires, on parle de *maps*. Pour faire simple, un *array* est une liste immuable et un *slice* est un pointeur vers un *array*. On travaille rarement avec des *arrays*, c'est plutôt les *slices* qui sont utilisés : les *arrays* sous-jacents sont gérés de manière transparente. L'équivalent du dictionnaire est la *map*, qui fonctionne à peu près de la même manière. Notons également qu'une *map* doit être explicitement instanciée avec la fonction **make**. Ajoutons à cela que la fonction **append** n'est pas une méthode de l'objet **slice**, mais une fonction à part : on voit alors aisément dans le code que c'est une fonction plus lourde qu'il n'y paraît, chose qui est implicite en Python (tableau n°8).

Python	Go

`def test():`
`  lettres = []`
`  chiffres = {}`
`  lettres.append("un")`
`  lettres.append("deux")`
`  chiffres["un"] = 1`
`  chiffres["deux"] = 2`
`  print(lettres)`
`  print(chiffres)`
	`func test() {`
	`  var lettres []string`
	`  chiffres := make(map[string]int)`
	`  lettres = append(lettres, "un")`
	`  lettres = append(lettres, "deux")`
	`  chiffres["un"] = 1`
	`  chiffres["deux"] = 2`
	`  fmt.Println(lettres)`
	`  fmt.Println(chiffres)`
	`}`

Tableau n°8

La grosse différence avec Python, c'est qu'un *slice* ou une *map* ne peut contenir qu'un seul type de données, d'ailleurs on les instancie en précisant le type qu'ils vont contenir. Cela enlève beaucoup de flexibilité, mais en échange il devient très facile de travailler avec ces données. Si un habitué de Python trouve cela limitant, en réalité ce n'est pas un problème, vu que les données sont stockées dans des structures, comme on l'a vu précédemment !

8. POINTEURS

En Go, on peut gérer les variables par valeur ou par pointeur. Mais cela est bien plus simple qu'en C ! L'objectif est uniquement de pouvoir choisir comment on transmet des données à une autre fonction : soit on fait une copie de l'objet transmis (valeur), soit on demande à la fonction de travailler directement sur le même objet (pointeur). Pour cela, il suffit de préfixer une valeur par un « et commercial » pour pointer l'adresse de l'objet ou par un astérisque pour obtenir la valeur d'un pointeur. Par ailleurs, Go ne permet pas de faire des opérations sur les pointeurs eux-mêmes : il n'y a pas de risques de rencontrer les mêmes erreurs qu'avec C, où on déplace par exemple un pointeur alors que l'on veut incrémenter sa valeur.

La gestion des pointeurs est totalement transparente en Python, cela signifie aussi que c'est obligatoire : Python ne permet pas de transmettre une variable par valeur (sauf pour les types de base) (tableau n°9)...

Python	Go

`def trois(lettres):`
`  lettres.append("trois")`
 	
`def test():`
`  lettres = []`
`  lettres.append("un")`
`  lettres.append("deux")`
`  trois(lettres)`
`  print(lettres)`
	`func trois(lettres *[]string) {`
	`  *lettres = append(*lettres, "trois")`
	`}`
	
	`func test() {`
	`  var lettres []string`
	`  lettres = append(lettres, "un")`
	`  lettres = append(lettres, "deux")`
	`  trois(&lettres)`
	`  fmt.Println(lettres)`
	`}`

Tableau n°9

9. FIN DE FONCTION

Go permet d'ajouter l'exécution d'une instruction dans la pile des appels exécutés en fin de fonction. Cela permet de s'assurer que les fichiers sont fermés, par exemple, quelle que soit la manière dont la fonction se termine. On peut comparer ce principe avec l'instruction **with** de Python, mais dans une version bien plus flexible, car cela peut s'appliquer à n'importe quelle instruction, avec l'inconvénient que cette exécution ne se fera qu'en fin de fonction. Cela étant dit, Go permet très facilement de créer des fonctions anonymes, cela permet de découper le code en de nombreuses fonctions, aussi petites soient-elles (tableau n°10, page suivante).

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Python	Go

class CouldNotWrite(Exception):
    pass
 	
def write_it():
    with open("/tmp/something", "w") as f:
 	
        try:
            f.write("Hello world !")
        except:
            raise CouldNotWrite
	func writeIt() error {
	    f, _ := os.OpenFile(
	        "/tmp/something",
	        os.O_RDWR|os.O_CREATE,
	        0755,
	    )
	    defer f.Close()
	
	    _, err := f.Write([]byte("Hello world !"))
	    if err != nil {
	        return errors.New("Could not write")
	    }
	    return nil
	}

Tableau n°10

10. PROGRAMMATION MULTITÂCHE

Avec Python, le programmeur qui souhaite faire exécuter plusieurs tâches en parallèle choisit ce qu'il veut lancer (un processus ou un *thread*) et le système instancie ce qui est demandé. Go, quant à lui, gère cela de manière transparente : on instancie des goroutines, qui seront placées dans des *threads* séparés quand c'est nécessaire (pour les goroutines bloquantes, en particulier), mais cela n'est pas obligatoire, plusieurs routines peuvent se partager un seul *thread*. C'est cela qui permet de lancer bien plus de goroutines en Go que de *threads* avec n'importe quel autre langage.

Notons que la goroutine principale n'attend pas que les autres goroutines se soient terminées avant d'arrêter l'exécution du programme : il faut explicitement les attendre ; cela se fait avec un **WaitGroup** : on annonce le nombre de goroutines qui seront lancées et chaque goroutine termine par la décrémentation de ce nombre (tableau n°11)...

Par ailleurs, les goroutines communiquent habituellement entre elles par le biais des *channels* : ce sont des canaux de communication qui simplifient l'interaction, les goroutines évitant ainsi de partager un même espace mémoire. En Python, l'équivalent de ces *channels* serait **Queue.Queue** ou **multiprocessing.Pipe**... Une différence importante, par contre, est que la réception de données de plusieurs goroutines

Python	Go

def do_it(pos):
 	
    print("Bonjour {} !".format(pos))
 	
def spawn_two():
 	
    threading.Thread(
        target=do_it, args=("premier",)
).start()
    threading.Thread(
        target=do_it, args=("second",)
).start()
	var wg sync.WaitGroup
	
	func doIt(pos string) {
	    defer wg.Done()
	    fmt.Printf("Bonjour %s !\n", pos)
	}
	
	func spawnTwo() {
	    wg.Add(2)
	    go doIt("premier")
	
	    go doIt("second")
	
	    wg.Wait()
	}

Tableau n°11

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

se fait en un seul bloc **select** ; il n'y a pas besoin de faire du *polling* sur différents canaux... Cependant, toujours dans un souci de clarté, de simplicité et de performance, un *channel* ne peut faire transiter qu'un seul type de données ; on peut toutefois créer autant de *channels* que nécessaire...

11. NOMMAGE DES VARIABLES

Il y a plusieurs choses à dire à propos du nommage de variables... Pour commencer, concernant la convention de nommage, en Go on utilise le *CamelCase* (**nomDeVariable**) plutôt que l'*underscore* (**nom_de_variable**)... C'est une convention qui a été décidée par les créateurs du langage, on peut choisir de ne pas s'y contraindre, mais certains outils risquent de râler.

Ensuite, la façon d'identifier les variables (et fonctions) privées des variables (et fonctions) publiques est directement liée à leur nommage. Cela est extrêmement simple : si un nom commence par une majuscule, alors l'élément est public. Par conséquent, si vous développez une bibliothèque, vous utiliserez des majuscules pour les fonctions ou variables publiques (par exemple **ReadData**) et les minuscules pour les fonctions privées (par exemple **doGetDataFromRemote**).

Enfin, évoquons une variable un peu spéciale (la seule, en réalité) : **_** (oui, oui, l'underscore, le caractère de soulignement). C'est un peu le **/dev/null** de Go, on l'utilise généralement dans deux cas de figure : soit pour ignorer une valeur de retour (par exemple pour ne tester qu'une erreur ou, au contraire, pour ignorer l'erreur), soit pour « consommer » une variable qui n'est pas utilisée par ailleurs (car Go refuse de compiler si une variable est inutilisée, ce qui peut être le cas quand on est en train de développer par exemple) (tableau n°12).

12. TESTS UNITAIRES

Que serait un langage moderne sans tests unitaires ? Tout comme Python, Go propose une bibliothèque standard pour écrire des tests unitaires... Avec Go, il n'y a pas besoin d'importer le paquet que l'on veut tester : un paquet étant représenté par un répertoire, il suffit d'avoir un fichier de tests dans le répertoire pour savoir que l'on veut tester ce paquet (tableau n°13, page suivante).

Sur le même modèle, on peut créer des fonctions de *benchmark* et des fonctions d'exemples (simplifiant la documentation)...

Notons que différents *packages* supplémentaires existent pour améliorer les tests, l'un d'entre eux permettant d'ajouter la fonction **assert** pour effectuer des tests comme avec Python.

13. COMPILATION ET PERFORMANCES

Parmi les avantages d'un langage compilé, on peut commencer par citer les performances : les programmes développés en Go sont légers et rapides, souvent bien plus rapides que leurs équivalents en Python. Cela prend tout son sens

Python

```python
def test():

    un = 1
    deux = 2
    troisStr = "3"

    chiffres = [un]

    try:
        trois = int(troisStr)
    except:
        trois = 0

    # TODO Terminer la liste de chiffres



    print(chiffres)
```

Go

```go
func test() {
    var (
        un = 1
        deux = 2
        troisStr = "3"
        // un est consommé ici
        chiffres = []int{un}
    )
    // troisStr est consommé ici
    trois, _ := strconv.Atoi(troisStr)



    // TODO Terminer la liste de chiffres
    // deux et trois sont consommés ici
    _ = deux
    _ = trois
    fmt.Println(chiffres)
}
```

Tableau n°12

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Python	Go

`import unittest`
`import malib`
 	
`class TestMaLib(unittest.TestCase):`
`  def test_add(self):`
`  self.assertEqual(malib.add(2, 3), 5)`
	`package main`
	
	`import "testing"`
	
	`func TestAdd(t *testing.T) {`
	`  v := Add(2, 3)`
	`  if v != 5 {`
	`  t.Error("Expected 5, got ", v)`
	`  }`
	`}`

`$ python -m unittest truc_test`	`$ go test`

Tableau n°13

sur des plateformes légères (**Raspberry Pi** par exemple). De plus, le typage statique et la compilation permettent de détecter un bon paquet d'erreurs avant même d'exécuter un programme : ça simplifie grandement le travail de développement !

Ajoutons que le *garbage collector* de Go est extrêmement rapide et tourne dans sa propre goroutine : l'impact en terme de performances est négligeable, dans le pire des cas il ne bloque pas l'exécutable plus de 100 µs. Grâce à ces performances, on peut aisément développer des jeux vidéos en Go. Ce langage pourrait presque être utilisé pour des systèmes temps réel...

Par ailleurs, Go supporte nativement la compilation croisée : inutile d'installer une collection de bibliothèques pour compiler un programme pour une autre plateforme, inutile de développer du code spécifique à telle ou telle plateforme... Il suffit de définir les bonnes variables d'environnement lors de l'appel à **go build**. Par exemple, pour compiler pour Windows en 32 bits :

```
GOOS=windows GOARCH=386 go build -o super_programme.exe
```

Cela fonctionne avec n'importe quelle architecture et n'importe quel environnement, y compris **Darwin** (**OS X**), **Linux**, **Android** (et d'autres), sur plateformes Intel 32 et 64 bits, ARM, MIPS, PPC... Vous voulez de la cross-compilation pour Linux sur Rasbperry Pi à partir d'un PC sous Windows ? C'est natif !

Enfin, on peut noter que pour immédiatement exécuter et tester le code que l'on vient d'écrire, Go propose la commande **go run**, à utiliser à la place de **go build**.

14. GÉNÉRATION DE CODE

Depuis 2014 et la version 1.4 de Go, celui-ci inclut la fonction **generate**, qui a pour but d'exécuter des commandes avant compilation, à la manière de ce qu'on ferait avec **make**. En réalité, tout ce que fait **go generate** est faisable avec un **Makefile**, il n'y a rien de fondamentalement révolutionnaire. Mais c'est maintenant directement intégré au langage...

Dans ce cadre, on peut par exemple exploiter une bibliothèque d'inclusion de fichier (*assets*) dans le binaire, comme **go-bindata**. Cet outil (qui n'est pas une bibliothèque Go, mais bien un outil complémentaire) transforme des fichiers classiques en données dans un code source en Go. On peut ensuite utiliser ces données plutôt que de desservir des fichiers plats. Bien sûr, cela alourdit le binaire et son empreinte mémoire. Par contre, cela permet de distribuer un programme entier, illustrations comprises par exemple, en un seul binaire.

15. PAQUETS PAR DÉFAUT ET À INSTALLER

Avec Go, comme avec Python, on a accès par défaut à une grande quantité de *packages* (paquets, modules) standards, permettant de faire énormément de choses sans télécharger des bibliothèques supplémentaires. Ces *packages* sont clairement documentés, avec exemples à l'appui, sur le site officiel de Go : https://golang.org/pkg/ ; c'est l'équivalent du *Python Global Module Index*.

J'ai toutefois noté une différence avec Python, qui est certainement liée au fait que ce langage est plus jeune : ces bibliothèques sont plus faciles à utiliser que les équivalents en Python et des fonctions plus « haut niveau » y sont présentes (par exemple Go inclut

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

en standard un *framework* web basique, que l'on peut comparer à **Bottle**, ou encore une bibliothèque de *templates*).

L'installation d'une bibliothèque, quant à elle, est très simple : on exécute la commande `go get` avec un identifiant vers la bibliothèque en question (qui est bien souvent l'URL de sa page GitHub, sans le préfixe de protocole). Go va alors récupérer la bibliothèque et la placer dans les bibliothèques à sa disposition, qui se situe dans le *gopath*. Par exemple :

```
$ go get github.com/stretchr/testify
```

CONCLUSION

De manière générale, tout en permettant une grande flexibilité dans l'écriture de code au quotidien avec une syntaxe relativement simple, Go nécessite de la discipline grâce à son compilateur qui ne laisse pas passer les petits oublis que l'on peut parfois avoir. Cela rend les programmes écrits en Go plutôt robustes...

Si on ajoute à cela son efficacité redoutable pour la programmation parallèle, on obtient un langage très intéressant pour de nombreux usages, même si son domaine de prédilection est le Web (notamment pour la création d'API).

De plus, tout un écosystème d'outils existe pour aider les développeurs en Go. Nous avons déjà abordé gofmt, et on peut également évoquer **Govet** et **Golint**, qui permettent de s'assurer que le code est propre, tout comme **Pylint** permet de le faire en Python : ceci peut se faire à chaque sauvegarde d'un fichier, grâce aux extensions de certains environnements de développement. Dans le même ordre d'idées, on peut citer **Goimport** (qui génère tout seul les imports en début de fichier) ou encore **GoMetaLinter** (qui exécute plusieurs « linters » en une seule passe).

Par ailleurs, pour la distribution d'outils écrits en Go, tout est très simplifié : on compile rapidement des binaires autonomes pour différentes plateformes à partir d'une seule machine, il n'y a plus qu'à distribuer ce binaire.

Enfin, n'oublions pas de parler de la documentation : recherchez n'importe quel terme associé à « golang » sur le Web, vous trouverez de nombreuses documentations et de nombreux articles de blog... ■

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Enseignants, Lycées, Écoles, Universités...

LINUX MAGAZINE / FRANCE

Besoin de ressources pédagogiques ?

...Permettre à mes élèves de consulter la plateforme de lecture en ligne ?

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail : abopro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

CRÉEZ UN ROBOT QUI TWEETE POUR VOUS

TRISTAN COLOMBO

MOTS-CLÉS : TWITTER, ROBOT, AUTOMATISATION, TWEET

Vous aimez tweeter, mais vous n'avez pas forcément le temps de répondre à des questions qui vous sont posées ? Ou alors justement vous détestez tweeter et vous souhaiteriez que quelqu'un le fasse pour vous ? Et si vous créiez un robot qui tweete pour vous ?

Que l'on aime ou pas **Twitter**, nous avons à peu près tous un compte que nous alimentons de façon plus ou moins régulière. Grâce à l'API Twitter il est possible d'automatiser certaines actions assez simplement et il serait dommage de ne pas en profiter. Dans cet article, nous allons voir comment utiliser cette API et quelles précautions prendre avant de se lancer.

1. AVERTISSEMENTS PRÉLIMINAIRES

Pour pouvoir créer un robot agissant sur votre compte Twitter, il faut bien entendu posséder un compte Twitter... mais également être conscient des risques encourus en cas de non-respect des règles du service. Ces règles, que je ne saurais trop vous recommander de lire, se trouvent sur le site officiel de Twitter **[1] [2]**. Pour les plus pressés d'entre vous, voici les points sur lesquels il faudra être particulièrement vigilant :

- *« Vous ne devez pas utiliser le service Twitter dans le but de spammer quelqu'un. Ce qui est considéré comme spam (...) :*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- *Vous vous êtes abonné à et/ou désabonné de nombreux comptes sur une courte période, particulièrement si vous l'avez fait par des moyens automatisés (abonnements et désabonnements agressifs).*
- *Vos mises à jour consistent principalement en des liens, et non en des messages personnels.*
- *Vous suivez des comptes, ou vous aimez ou retweetez des Tweets de façon aléatoire ou agressive. »*

Donc si votre compte est fortement automatisé, il faudra veiller aux abonnements automatiques, au type de tweets et à la façon de retweeter ou aimer des tweets. Cela devra être fait de manière intelligente sous peine de voir son compte suspendu.

- *« Les fonctionnalités de réponse et de mention facilitent la communication entre les utilisateurs. Leur automatisation en vue de toucher un grand nombre de personnes est considérée comme inappropriée. Si votre application crée ou facilite des réponses ou mentions automatisées touchant de nombreux utilisateurs, les destinataires doivent au préalable avoir demandé à être contactés ou avoir indiqué leur volonté de l'être. Par exemple, l'envoi de réponses automatisées basé sur les recherches de mots-clés n'est pas autorisé. Les utilisateurs doivent également disposer d'un moyen clair et facile d'indiquer qu'ils ne souhaitent pas recevoir des réponses et mentions automatisées depuis votre application. »*

Cette règle est admirable dans la mesure où dans la page des « Règles et bonnes pratiques d'automatisation » **[2]**, on nous présente et on nous vante l'exemple de **@klmfares** (**KLM airlines**) qui détecte des mots-clés dans les tweets **[3]** pour répondre automatiquement avec des tarifs de vols. Pourtant aucune approbation ne vous sera ici demandée... N'oubliez pas que Twitter est une société, elle ne fait pas dans le mécénat et il est indiqué dans les règles de contacter le chargé de clientèle ou de partenariat.

- *« L'automatisation des Retweets conduit souvent à du spam et à d'autres expériences utilisateur négatives. En conséquence, il est interdit de retweeter de manière groupée ou automatisée. Les Retweets automatisés sont autorisés pour les applications ou comptes que Twitter considère comme bénéfiques à la communauté. »*
- *« Il est interdit de développer toute application qui permet d'aimer des Tweets de manière groupée ou automatisée. L'usage agressif de l'option J'aime constitue une violation des Règles de Twitter* **[1]**. *»*

Voici donc les principales règles auxquelles il faudra faire attention. Mais... qu'allons-nous pouvoir programmer en suivant des règles aussi restrictives ? Le discours de Twitter est double puisque d'un côté il met à disposition une API permettant l'automatisation de tout ce qu'il interdit d'un autre côté. De plus l'utilisation de cette API est encouragée dans leur blog. On peut donc en déduire que c'est un moyen de défense qui leur permet d'être tout puissant quant à la décision du blocage d'un compte (qui peut de toute façon arriver même sans robot).

Pour minimiser les risques, je vous recommande donc de ne surtout pas utiliser votre compte de tous les jours, celui où 12K *followers* vous suivent journellement pour savoir si vous avez bien acheté votre baguette et ce que vous allez faire de votre soirée. Par exemple pour les besoins de l'article j'ai créé un nouveau compte **@TwitbotEssai** sur lequel je ne cache pas du tout qu'il s'agit d'un robot. Et même en faisant très attention, j'ai eu droit à un blocage temporaire (voir figure 1).

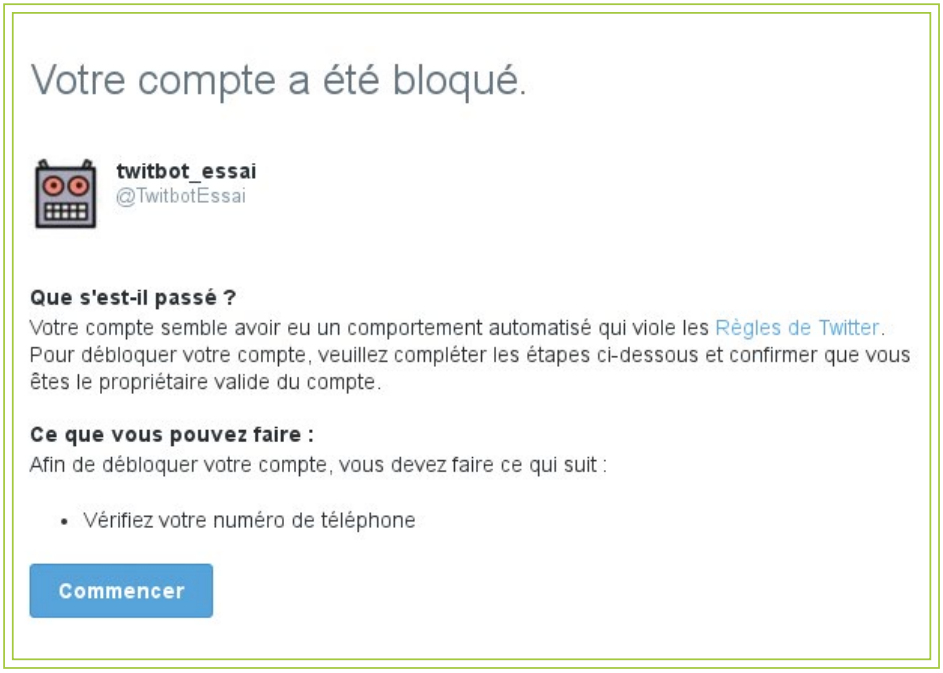


Fig. 1 : Ce qui risque de vous arriver, même en faisant très attention...

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2. CONFIGURER LE COMPTE TWITTER POUR POUVOIR INTERAGIR AVEC LUI

Après avoir ouvert votre compte Twitter, rendez-vous sur https://apps.twitter.com/ (ou alors identifiez-vous directement sur cette page). Vous pourrez alors créer une nouvelle application en cliquant sur le bouton ***Create New App*** (ce n'est pas compliqué, il n'y a qu'un gros bête bouton au milieu de la page). Vous devrez ensuite remplir un petit formulaire :

- ***Name*** : le nom de votre application ;
- ***Description*** : la description de ce que fait l'application ;
- ***Website*** : l'URL permettant d'accéder aux informations sur votre application ;
- ***Callback URL*** : l'URL à appeler après une authentification réussie (paramètre facultatif).

N'oubliez pas de cocher la case du bas indiquant que vous avez lu et que vous acceptez le contrat de développeur Twitter puis cliquez sur le bouton ***Create your Twitter Application***.

Par défaut, le niveau d'accès est *Read and Write* donc nous n'aurons rien à modifier à ce niveau. Cliquez sur l'onglet ***Keys and Access Tokens*** et cliquez ensuite sur le bouton ***Create my access token*** en bas de page. Notez (ou plutôt copiez dans un fichier) vos *Access Token*, *Access Token Secret*, *Consumer Key* et *Consumer Key Secret*. Pour ne pas avoir à re-manipuler ces éléments illisibles, je vous conseille de créer dès maintenant un fichier de configuration **twitter.ini** :

```
[config]
consumer_key = xxxxxxxxxxxxxx
consumer_secret = xxxxxxxxxxxxxxx
access_token = xxxxxxxxxxxxx
access_token_secret = xxxxxxxxxxxxx
```

La phase de configuration est terminée.

3. ENVOYER UN TWEET

Pour accéder à Twitter, nous allons utiliser le module **tweepy** :

```
$ sudo pip3 install tweepy
```

Les étapes permettant d'envoyer un tweet seront :

1. Lecture des données d'authentification ;
2. Authentification ;
3. Envoi du tweet.

Le code permettant de réaliser ces opérations est le suivant :

```
01: import tweepy
02: import configparser
03:
04:
05: def readconfig(filename='twitter.ini'):
06:     config = configparser.ConfigParser()
07:     config.read(filename)
08:     data = {}
09:     for key in config['config']:
10:         data[key] = config['config'][key]
11:     return data
12:
13: def get_api(cfg):
14:     auth = tweepy.OAuthHandler(cfg['consumer_key'],
cfg['consumer_secret'])
15:     auth.set_access_token(cfg['access_token'],
cfg['access_token_secret'])
16:     return tweepy.API(auth)
17:
18: def tweet(api, text):
19:     status = api.update_status(status=text)
20:
21:
22: if __name__ == '__main__':
23:     cfg = readconfig()
24:     api = get_api(cfg)
25:     tweet(api, 'Un petit tweet, mais un grand pas
pour @TwitbotEssai')
```

On commence nécessairement par l'import de **tweepy** en ligne 1 et de **configparser** (en ligne 2) pour lire le fichier ini. La fonction **readconfig()** des lignes 5 à 11 a pour objectif de récupérer les informations du fichier ini de nom **filename** (par défaut **twitter.ini**). On lit toutes les clés de la section **'config'** **[4]** et on les stocke dans le dictionnaire **data** qui sera retourné par la fonction.

La fonction **get_api()** (lignes 13 à 16) utilise les informations lues dans le fichier ini (et transmises sous forme de dictionnaire dans le paramètre **cfg**) pour réaliser l'authentification auprès de Twitter et fournir une instance de l'objet de communication **API**.

La fonction **tweet()** (lignes 18 et 19) permet de... tweeter le message passé en paramètre dans **text** (il faut également fournir l'instance de communication avec Twitter (**api**).

ATTENTION

Dans tweepy, un tweet est appelé « status ». Il s'agit du statut du détenteur du compte qui peut être modifié à l'aide d'un texte et qui correspond plus communément à un tweet.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Fig. 2 : Et voilà notre premier tweet !

Dans les lignes 22 à 25 du programme principal, nous utilisons simplement les fonctions précédentes pour envoyer un tweet qui sera visible après exécution (voir figure 2).

4. ENVOYER ENCORE UN TWEET… MAIS AVEC UN CONTENU MULTIMÉDIA

Nous avons envoyé un tweet « simple », mais comment faire si l'on souhaite ajouter une image ou une vidéo à notre texte ? Tweepy propose la méthode **update_with_media()** **[5]** qui permettra de réaliser cette opération :

```
...
21: def tweet_media(api, text, media):
22:     status = api.update_with_media(status=text, filename=media)
23:
24: if __name__ == '__main__':
25:     cfg = readconfig()
26:     api = get_api(cfg)
27:     tweet_media(api, 'Voici une photo qui ne passe pas',
'TwitbotEssai.xxx')
...
```

Comme vous l'avez constaté précédemment, nous n'avons pas géré les erreurs… j'ai donc voulu voir ce qui se passait en donnant en lien une image inexistante :

```
$ python3 twitbot.py
Traceback (most recent call last):
...
FileNotFoundError: [Errno 2] No such file or directory:
'TwitbotEssai.xxx'

During handling of the above exception, another exception
occurred:
...
tweepy.error.TweepError: Unable to access file: No such
file or directory
```

Comme prévu, nous obtenons bien une exception et même deux : l'exception classique du **FileNotFoundError** et une exception de tweepy **TweepError** **[6]**. Corrigeons donc notre code pour le simplifier et y intégrer le traitement des erreurs :

```
...
18: def tweet(api, text, media=None):
19:     try:
20:         if media is None:
21:             status = api.update_
status(status=text)
22:         else:
23:             status = api.update_with_
media(status=text, filename=media)
24:     except tweepy.RateLimitError as e:
25:         print('API Rate Limits
atteintes ! Veuillez patienter 15mn')
26:         exit(1)
27:     except tweepy.TweepError as e:
28:         try:
29:             print(e.args[0][0]
['code'], ':', e.args[0][0]['message'])
30:             print('Plus d\'infos sur :
https://dev.twitter.com/overview/api/
response-codes')
31:         except Exception:
32:             print(e)
33:         exit(2)
...
38:     tweet(api, 'Voici cette fois
une véritable photo de @TwitbotEssai',
'TwitbotEssai.png')
```

J'ai regroupé les fonctions **tweet()** et **tweet_media()** dans une seule fonction **tweet()** qui enverra un tweet simple si le paramètre media est à **None** et un tweet accompagné d'une image/vidéo sinon (lignes 20 à 23). Au niveau du traitement des erreurs, on commence par intercepter les exceptions **RateLimitError**, car elles héritent de **TweepError** (**RateLimitError** est une **TweepError** et donc une interception de **TweepError** va « attraper » les deux types d'exceptions).

Un **TweepError** dispose de plusieurs attributs : **api_code**, **args**, **reason**, **response** et **with_traceback**. Leur utilisation n'est pas forcément très aisée

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

de par les choix de conception effectués : **reason** est une chaîne de caractères au format JSON, **args** est un tuple de listes de dictionnaires, etc. Voilà pourquoi pour afficher le code de l'erreur et le message associé on doit passer par **e.args[0][0]** en ligne 29. De plus, avec les cascades d'exceptions, la variable **e** ne contient pas forcément le message d'erreur renvoyé par tweepy (par exemple si un fichier n'est pas trouvé). Il a donc fallu récupérer une autre exception pour afficher simplement le contenu de **e** (lignes 31 et 32).

Cette fois notre tweet sera bien envoyé avec une image (voir figure 3) et les cas d'erreurs seront gérés (essayez d'envoyer un tweet avec un message vide **''**).

Fig. 3 : Un tweet contenant une image.

5. UNE SÉRIE DE TWEETS

Un tweet étant limité à 140 caractères, il n'est pas rare de voir des suites de tweets. Puisque nous tweetons automatiquement, pourquoi ne pas intégrer à notre programme la possibilité de découper automatiquement les tweets trop longs ?

```
...
04: MAX_LEN_TEXT = 140
...
20: def segmentizeText(text):
21:     max_line = MAX_LEN_TEXT - 6
22:     lines = []
23:     current_line = ''
24:     for word in text.split(' '):
25:         if len(current_line) + len(word) +
1 > max_line:
26:             lines.append(current_line)
27:             current_line = word
28:         else:
29:             if current_line != '':
30:                 current_line += ' '
31:             current_line += word
32:     if current_line != '':
33:         lines.append(current_line)
34:     return lines
35:
36: def tweet(api, text, media=None):
37:     try:
38:         if media is None:
39:             if len(text) > MAX_LEN_TEXT:
40:                 lines = segmentizeText(text)
41:                 nb_lines = len(lines)
42:                 for counter in range(nb_lines):
43:                     txt = '{} {}/{}'.
format(lines[counter], counter + 1, nb_lines)
44:                     status = api.update_
status(status=text)
45:             else:
46:                 status = api.update_
status(status=text)
47:         else:
48:             status = api.update_with_
media(status=text, filename=media)
49:     except tweepy.RateLimitError as e:
50:         print('API Rate Limits atteintes !
Veuillez patienter 15mn')
51:         exit(1)
52:     except tweepy.TweepError as e:
53:         try:
54:             print(e.args[0][0]['code'], ':',
e.args[0][0]['message'])
55:             print('Plus d\'infos sur : https://
dev.twitter.com/overview/api/response-codes')
56:         except Exception:
57:             print(e)
58:         exit(2)
59:
60: if __name__ == '__main__':
...
63:     tweet(api, 'Ceci est un exemple de tweet
contenant un texte relativement long et dépassant
donc allègrement la limite des 140 caractères. La
fonction segmentizeText() va débiter automatiquement
ce texte de manière à envoyer plusieurs tweets
numérotés ayant une taille autorisée. Du coup plus de
problème!')
```

La longueur maximale d'un tweet a été définie dans une constante **MAX_LEN_TEXT** en ligne 4 (comme ça si Twitter modifie la taille des tweets nous pourrons facilement modifier le paramètre). Pour savoir si un tweet est trop long, nous testons sa taille dans la fonction **tweet()** en ligne 39. S'il est effectivement trop long, alors on appelle la fonction **segmentizeText()** des lignes 20 à 34 qui va créer une liste de chaînes de caractères de taille inférieure à **MAX_LEN_TEXT - 6** (on considère avoir au maximum à ajouter 6 caractères : un espace, deux caractères pour le numéro du tweet, un caractère *slash*, et deux caractères pour le nombre total de tweets). On parcourt ensuite cette liste dans les lignes 42 à 44 et on tweete ligne à ligne en ajoutant la position du tweet au format **numéro/total**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Le résultat obtenu est visible en figure 4 et oui, il serait plus intéressant de publier le premier tweet puis de répondre à chaque fois au précédent... une chose à la fois.

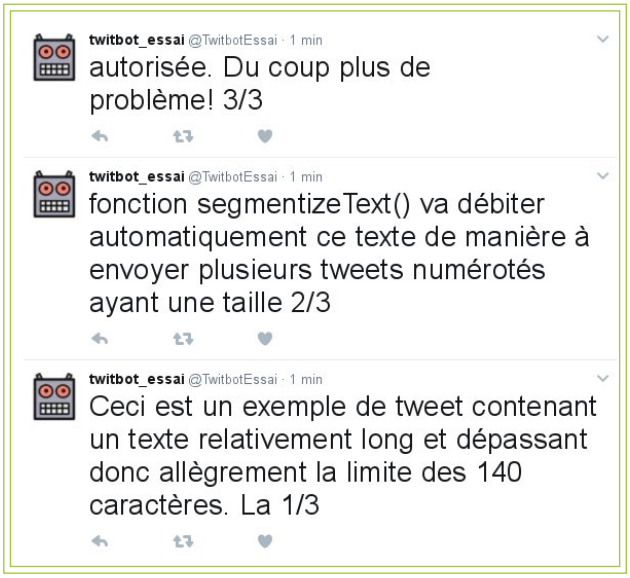


Fig. 4 : Série de tweets découpée automatiquement.

ATTENTION

Si le texte est vraiment très long (et donc normalement inadapté à Twitter), pensez à ajouter une temporisation avec **`time.sleep()`** pour ne pas déclencher une **`RateLimitError`**.

6. RÉPONDRE À UN TWEET

Pour répondre à un tweet, nous utiliserons toujours les mêmes méthodes **`update_status()`** et **`update_with_media()`** qui acceptent un paramètre **`in_reply_to_status_id`** contenant l'identifiant du tweet auquel nous souhaitons répondre. Reste alors à trouver l'identifiant... ce qui ne sera pas très compliqué sur nos propres tweets. Au cas vous n'auriez pas déjà succombé à la curiosité, regardons le contenu de la variable **`status`**, retour des fonctions **`update_status()`** et **`update_with_media()`** (j'ai ajouté après la ligne 46 un affichage de **`dir(status)`** et de **`status`**) :

```
$ python3 twitbot.py
['__class__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__eq__', '__
format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__
module__', '__ne__', '__new__', '__
reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__
subclasshook__', '__weakref__', '_api', '_
json', 'author', 'contributors', 'coordinates',
'created_at', 'destroy', 'entities', 'favorite',
'favorite_count', 'favorited', 'geo', 'id',
'id_str', 'in_reply_to_screen_name', 'in_reply_
to_status_id', 'in_reply_to_status_id_str', 'in_
reply_to_user_id', 'in_reply_to_user_id_str',
'is_quote_status', 'lang', 'parse', 'parse_
list', 'place', 'retweet', 'retweet_count',
'retweeted', 'retweets', 'source', 'source_url',
'text', 'truncated', 'user']
Status(geo=None, retweet_count=0, in_reply_to_
user_id_str=None, entities={'user_mentions':
[], 'urls': [], 'hashtags': [], 'symbols':
[]}, favorite_count=0, in_reply_to_screen_
name=None, in_reply_to_status_id_str=None,
coordinates=None, _api=<tweepy.api.API
object at 0x7f69c99e1ba8>, in_reply_to_
status_id=None, user=User(followers_count=0,
following=False, has_extended_profile=True,
name='twitbot_essai', profile_sidebar_fill_
color='DDEEF6', screen_name='TwitbotEssai', is_
translation_enabled=False, profile_background_
color='F5F8FA', url=None, profile_sidebar_
border_color='C0DEED', notifications=False,
created_at=datetime.datetime(2017, 3, 29, 14,
14, 11), profile_background_image_url=None,
listed_count=0, translator_type='none',
friends_count=0, _api=<tweepy.api.API object
at 0x7f69c99e1ba8>, verified=False, profile_
background_tile=False, id=847089508646633472,
profile_text_color='333333', contributors_
enabled=False,
...
source_url='https://github.com/GLMF',
source='TwitbotEssai_bot', favorited=False,
lang='fr', place=None)
```

La sortie, très longue, a été tronquée. On voit que l'on est en présence d'un objet **`Status`** (ce qui tombe plutôt bien puisque c'est ce qu'indiquait la documentation) et que celui-ci comporte de nombreux attributs (là par contre la documentation n'en disait rien). Nous devrions donc normalement pouvoir utiliser **`status.id`** pour obtenir l'identifiant de l'un de nos tweets et ainsi pouvoir tweeter notre série de tweets de manière plus lisible :

```
...
36: def tweet(api, text, media=None):
37:     try:
38:         if media is None:
39:             if len(text) > MAX_LEN_TEXT:
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
40:                 lines =
segmentizeText(text)
41:                 nb_lines =
len(lines)
42:                 for counter in
range(nb_lines):
43:                     txt = '{} {}/
{}'.format(lines[counter], counter + 1,
nb_lines)
44:                     if counter == 0:
45:                         status =
api.update_status(status=text)
46:                     else:
47:                         status
= api.update_status(status=text, in_
reply_to_status_id=reply_to)
48:                     reply_to =
status.id
49:             else:
50:                 status = api.update_
status(status=text)
...
```

Les modifications se trouvent dans les lignes 44 à 48 : lors du premier tweet (**counter** vaut **0**), bien entendu on ne répond à aucun tweet (ligne 45), sinon on ajoute une valeur **reply_to** à **in_reply_to_status_id** (ligne 47). La variable **reply_to** contient le **status.id** du tweet qui vient d'être envoyé en ligne 45 ou 47.

Comme vous pouvez le voir en figure 5, nous obtenons bien des tweets qui se suivent, mais cette fois sous forme de réponses.

Nous avons réglé un premier problème concernant la réponse à un tweet. Mais comment faire si l'on souhaite répondre à un tweet provenant d'une autre personne ? Nous allons rechercher les tweets dans lesquels nous sommes mentionnés et, pour ne pas traiter plusieurs fois les mêmes tweets nous allons conserver le dernier identifiant de tweet traité (il faudra donc un numéro d'identifiant supérieur pour être en présence d'un tweet qui n'a pas encore été vu). De plus, nous prendrons garde de ne répondre qu'à nos *followers* pour ne pas trop éveiller la curiosité de Twitter :

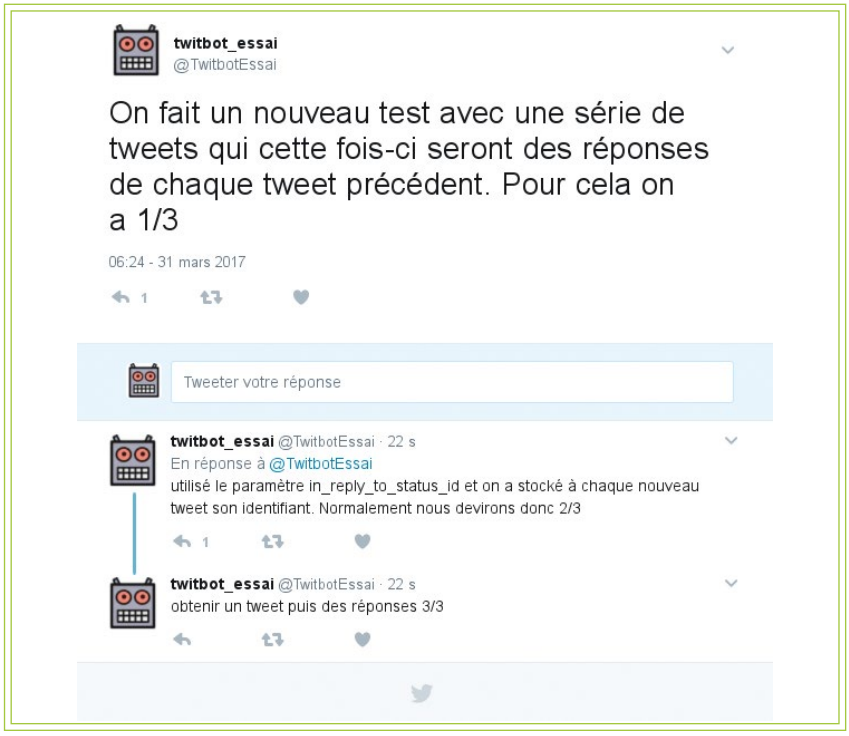


Fig. 5: Série de tweets sous forme de réponse au tweet précédent (et oui, il y a une faute de frappe à « devirons »... mais c'est aussi ça la magie de l'instantanéité de Twitter :-)).

```
...
036: def tweet(api, text, reply_to=None, media=None):
037:     try:
038:         if media is None:
039:             if len(text) > MAX_LEN_TEXT:
040:                 lines = segmentizeText(text)
041:                 nb_lines = len(lines)
042:                 for counter in range(nb_lines):
043:                     txt = '{} {}/{}'.
format(lines[counter], counter + 1, nb_lines)
044:                     if counter == 0:
045:                         if reply_to is None:
046:                             status = api.update_
status(status=text)
047:                         else:
048:                             status = api.update_
status(status=text, in_reply_to_status_id=reply_to)
049:                     else:
050:                         status = api.update_
status(status=text, in_reply_to_status_id=reply_to)
051:                     reply_to = status.id
052:             else:
053:                 if reply_to is None:
054:                     status = api.update_
status(status=text)
055:                 else:
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
056:                         status = api.update_
status(status=text, in_reply_to_status_id=reply_to)
057:             else:
058:                 if reply_to is None:
059:                     status = api.update_with_
media(status=text, filename=media)
060:                 else:
061:                     status = api.update_with_
media(status=text, filename=media, in_reply_to_status_
id=reply_to)
062:         except tweepy.RateLimitError as e:
063:             print('API Rate Limits atteintes ! Veuillez
patienter 15mn')
064:             exit(1)
065:         except tweepy.TweepError as e:
066:             try:
067:                 print(e.args[0][0]['code'], ':',
e.args[0][0]['message'])
068:                 print('Plus d\'infos sur : https://dev.
twitter.com/overview/api/response-codes')
069:             except Exception:
070:                 print(e)
071:             exit(2)
072:
073: def getLastId(filename='.lastTweetId'):
074:     try:
075:         with open(filename, 'r') as fic:
076:             id = int(fic.read())
077:     except IOError:
078:         print('IOError sur fichier {} : on utilise
l\'identifiant 0'.format(filename))
079:         return 0
080:     return id
081:
082: def saveLastId(id, filename='.lastTweetId'):
083:     last_id = getLastId(filename)
084:
085:     if last_id < id:
086:         print('Sauvegarde de l\'id {}'.format(id))
087:         try:
088:             with open(filename, 'w') as fic:
089:                 fic.write(str(id))
090:         except IOError:
091:             print('IOError lors de l\'écriture dans
{}'.format(filename))
092:             exit(3)
093:     else:
094:         print('Identifiant inférieur au dernier
identifiant sauvegardé : pas de sauvegarde')
095:
096: def response(api):
097:     try:
098:         id = getLastId()
099:         followers = api.followers_ids()
100:         # print(followers)
```

```
101:         to_me = api.mentions_
timeline()
102:         for message in to_me:
103:             if message.id > id and
message.user.id in followers:
104:                 tweet(api, 'Salut
@{} !'.
format(message.user.screen_name),
message.user.id)
105:
saveLastId(message.id)
106:     except tweepy.RateLimitError
as e:
107:         print('API Rate Limits
atteintes ! Veuillez patienter 15mn')
108:         exit(1)
109:     except tweepy.TweepError as e:
110:         try:
111:             print(e.args[0][0]
['code'], ':', e.args[0][0]['message'])
112:             print('Plus d\'infos
sur : https://dev.twitter.com/overview/
api/response-codes')
113:         except Exception:
114:             print(e)
115:         exit(2)
116:
117: if __name__ == '__main__':
118:     cfg = readconfig()
119:     api = get_api(cfg)
120:     response(api)
```

Nous avons tout d'abord modifié la fonction tweet de manière à ajouter un paramètre **reply_to** permettant de spécifier que nous répondons à un tweet. Par défaut, ce paramètre est initialisé à **None** (ligne 36). Des tests sont alors ajoutés pour envoyer un tweet simple ou une réponse (lignes 45 à 48, 53 à 56 et 58 à 61).

La fonction **getLastId()** permet de récupérer l'identifiant du dernier tweet auquel nous avons répondu et qui est stocké dans un fichier (par défaut **.lastTweetId**). Si le fichier n'existe pas, on retourne l'identifiant **0** de manière à traiter tous les tweets (ligne 79).

La fonction **saveLastId()** sauvegarde l'identifiant passé en paramètre **id** dans le fichier **filename** (toujours **.lastTweetId** par défaut). Si **id** est

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

inférieur à l'identifiant présent dans le fichier, alors la sauvegarde n'a pas lieu (ligne 94).

Enfin, la fonction **response()** répond automatiquement aux tweets mentionnant **@TwitbotEssai** :

- On récupère l'identifiant du dernier tweet auquel on a répondu (ligne 98) ;
- On récupère la liste des *followers* de **@TwitbotEssai** (ligne 99) et on peut même les afficher pour vérification (ligne 100 en commentaire) ;
- On récupère la liste des tweets mentionnant **@TwitbotEssai** dans **to_me** (ligne 101) ;
- Il n'y a plus qu'à parcourir la liste to_me et si l'identifiant des tweets et supérieur à l'identifiant du dernier tweet traité et que l'utilisateur fait parti de nos *followers* (ligne 103), alors on lui répond (ligne 104) et on enregistre l'identifiant du tweet (ligne 105).

Bien entendu dans cet exemple nous répondrons de la même manière à tous les messages... Nous verrons par la suite comment améliorer cela.

7. RETWEETER ET SUIVRE

Pour le retweet, on va dire que **@TwitbotEssai** est un grand admirateur de tout ce que je peux faire et il va donc retweeter **@TristanColombo** toujours et sans contrôle (s'il existe c'est aussi un peu grâce à moi quand même !). Pour ne pas que le compte soit bloqué, on va introduire un peu de variabilité quand même et faire en sorte que le retweet puisse être crédible :

- **@TwitbotEssai** et **@TristanColombo** vont se suivre l'un l'autre ;
- **@TwitbotEssai** retweetera aléatoirement les tweets avec une probabilité de **1/5**.

```
...
003: import random
...
118: def retweet(api, authorizedUsers):
119:   try:
120:       id = getLastId()
121:       for user in authorizedUsers:
122:           tweets = tweepy.Cursor(api.user_timeline, screen_
name=user).items()
123:           for tweet in tweets:
124:               if tweet.id > id:
125:                   if random.randint(1, 5) == 1:
126:                       api.retweet(tweet.id)
127:                       print('On retweete {} !'.format(tweet.id))
128:                   saveLastId(tweet.id)
129:   except tweepy.RateLimitError as e:
130:       print('API Rate Limits atteintes ! Veuillez patienter 15mn')
131:       exit(1)
132:   except tweepy.TweepError as e:
133:       try:
134:           print(e.args[0][0]['code'], ':', e.args[0][0]['message'])
135:           print('Plus d\'infos sur : https://dev.twitter.com/
overview/api/response-codes')
136:       except Exception:
137:           print(e)
138:       exit(2)
139:
140: if __name__ == '__main__':
141:     cfg = readconfig()
142:     api = get_api(cfg)
143:     retweet(api, ('TristanColombo',))
```

La structure de la fonction **retweet()** reste la même que celle de **response()**, mais cette fois-ci, en ligne 121, nous allons parcourir la liste des utilisateurs que nous pouvons potentiellement retweeter (**authorizedUsers**) et rechercher leurs tweets à l'aide de **tweepy.Cursor()** (ligne 122) où nous précisons que nous désirons les tweets de l'utilisateur **name** (**screen_name**). Ensuite, nous vérifions si l'identifiant de chaque tweet est supérieur au dernier tweet traité (lignes 123 et 124) et nous retweetons aléatoirement un tweet sur cinq (lignes 125 à 127). La figure 6 montre le résultat de quelques tests.

*Fig. 6 : Les retweets de **@TwitbotEssai**.*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DISPONIBLE DÈS LE 14 JUILLET

LINUX PRATIQUE HORS-SÉRIE N°39 !

MÉMO
LIGNE DE
COMMANDES

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ATTENTION

Nous recherchons les tweets des utilisateurs d'après le nom... or ce nom peut être modifié. Bien que moins lisible, une solution plus pérenne serait d'utiliser les identifiants des utilisateurs que l'on souhaite retweeter.

8. ANALYSER UN TWEET POUR Y RÉPONDRE

Avec tout ce que nous avons vu précédemment, il n'est guère compliqué d'analyser un tweet pour y répondre. Nous définirons une simple règle à laquelle **@TwitbotEssai** pourra répondre, le principe sera le même pour toutes les règles que vous voudrez ajouter. Cette règle sera d'envoyer un tweet de la forme **@TwitbotEssai Magazine @Magazine** où **@Magazine** est le nom de compte d'un magazine. Ici les seuls magazines valides seront ceux des Éditions Diamond (comme c'est étrange...) : GNU/Linux Magazine (**@gnulinuxmag**), Hackable Magazine (**@hackablemag**), MISC (**@MISCRedac**) et Linux Pratique (**@linuxpratique**).

```
...
140: def analyze(text):
141:     data = text.split(' ')
142:     if data.pop(0).lower() == '@twitbotessai':
143:         action = data.pop(0).lower()
144:         if action == 'magazine':
145:             mag = data.pop(0).lower()
146:             # if len(data) != 0:
147:             #    return ''
148:             msg = 'Rédacteur en chef de {} ({}) : {} ({})'
149:             if mag == '@gnulinuxmag':
150:                 return msg.format('GNU/Linux Magazine', '@gnulinuxmag',
'Tristan Colombo', '@TristanColombo')
151:             elif mag == '@hackablemag':
152:                 return msg.format('Hackable Magazine', '@hackablemag',
'Denis Bodor', '@Lefinnois')
153:             elif mag == '@miscredac':
154:                 return msg.format('MISC', '@MISCRedac', 'Cédric Foll',
'@follc')
155:             elif mag == '@linuxpratique':
156:                 return msg.format('Linux Pratique', '@linuxpratique', 'Aline
Hof', '404 not found')
157:     return ''
158:
159: def robot(api):
160:     try:
161:         id = getLastId()
162:         followers = api.followers_ids()
163:         to_me = api.mentions_timeline()
164:         for message in to_me:
165:             if message.id > id and message.user.id in followers:
166:                 response = analyze(message.text)
167:                 if response != '':
168:                     tweet(api, '@{} {}'.format(message.user.screen_name,
response), message.id_str)
169:                 saveLastId(message.id)
170:     except tweepy.RateLimitError as e:
171:         print('API Rate
Limits atteintes ! Veuillez
patienter 15mn')
172:         exit(1)
173:     except tweepy.
TweepError as e:
174:         try:
175:             print(e.args[0]
[0]['code'], ':', e.args[0][0]
['message'])
176:             print('Plus
d\'infos sur : https://dev.
twitter.com/overview/api/
response-codes')
177:         except Exception:
178:             print(e)
179:         exit(2)
180:
181: if __name__ == '__main__':
182:     cfg = readconfig()
183:     api = get_api(cfg)
184:     robot(api)
```

C'est la fonction **robot()** des lignes 159 à 179 qui va être appelée pour répondre automatiquement aux tweets. On retrouve la structure de récupération des tweets mentionnant notre compte déjà vue dans la fonction **response()** (lignes 160 à 165). Par contre ici on transmet le texte du tweet (**message.text**) à la fonction **analyze()** (ligne 166) qui va nous renvoyer le texte du tweet de réponse ou une chaîne vide si aucune réponse n'a pu être déterminée.

Le fonctionnement de **analyze()** (lignes 140 à 157) est très simple :

- on découpe le texte du tweet passé en paramètre suivant les caractères espace (ligne 141) ;
- on vérifie que le premier terme du tweet est bien **'@twitbotessai'** (ligne 142) en retirant le premier élément de la liste **data** à l'aide de **pop(0)** et en passant tout son contenu en minuscules (pour simplifier les comparaisons) ;
- on récupère le deuxième terme du texte en ligne 143 et on le place dans la variable **action** (on peut définir plusieurs règles et donc tester des actions différentes) ;

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

• on récupère le « paramètre » associé à l'action en ligne 145 puis en fonction de celui-ci on construit le tweet de réponse (lignes 148 à 156).

NOTE

Les lignes 146 et 147 ont été mises en commentaire pour autoriser des tweets contenant des mots superflus. En effet, vous ne pouvez pas tweeter deux fois exactement le même texte donc en conservant ces lignes le robot ne répondra qu'une seule fois à un tweet respectant précisément la règle puis ne pourra plus répondre si vous ajoutez des mots.

Comme on peut le voir en figure 7, à l'exécution, une fois que le robot reçoit des tweets respectant ces règles, il y répond.

Fig. 7 : Réponse automatique et adaptée à un tweet.

CONCLUSION

Nous en avons fini avec notre création d'un robot Twitter, il ne reste maintenant plus qu'à lui associer d'autres règles. La difficulté sera surtout de maintenir un aspect « humain » pour éviter tout blocage du compte et donc de définir les règles de manière réfléchie, éventuellement d'ajouter des temps de latence aléatoires entre les tweets ou les retweets (`time.sleep(n)` pour `n` secondes d'attente), etc. Vérifiez que vous n'effectuez pas de retweets « sauvages » et tout ce genre de choses qui mettraient fin prématurément à votre robot car, vous l'aurez compris, dans ce genre de développement le point bloquant ne viendra pas forcément de la technique... ■

NOTE

Pour les fans de Bourvil vous pouvez également faire Pouet-Pouet avec **Mastodon [8]**, mais c'est une autre histoire/API... (oui, sur Twitter on tweete et sur Mastodon on *pouete*, c'est plus *pouétique* tant que l'on ne *pouete* pas plus haut que... enfin, je m'égare...).

RÉFÉRENCES

[1] Règles de Twitter : https://support.twitter.com/articles/75576

[2] Règles et bonnes pratiques d'automatisation : https://support.twitter.com/articles/96178

[3] Présentation des commandes acceptées par le Twitterbot **@klmflares** : https://www.klm.com/travel/gb_en/about/news_press/KLM_on_social_media/klmfares_on_twitter.htm

[4] COLOMBO T., *« Mémo Python », « Recette 15 : Lire un fichier ini », GNU/Linux Magazine HS n°86*, 2016.

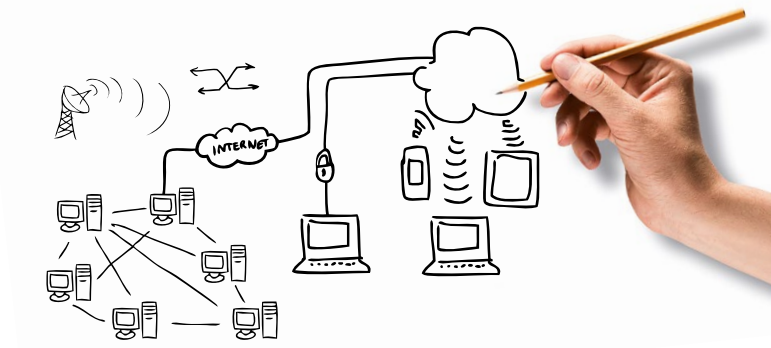
[5] Documentation de Tweepy : http://docs.tweepy.org/en/latest/api.html

[6] Les exceptions de Tweepy : http://docs.tweepy.org/en/latest/api.html#tweepy-error-exceptions

[7] Tweeter API Rate Limits : https://dev.twitter.com/rest/public/rate-limiting

[8] Mastodon : https://mastodon.social

LICENCE PRO
RÉSEAUX ET TÉLÉCOMS



UN BAC+3 QUI
AIME
L'OPEN-SOURCE
À L'IUT DE BÉZIERS

IUT Béziers
iutbeziers.fr/merit

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

CACHE MAVEN
PARTAGÉ AVEC NGINX

ROMAIN PELISSE
[Sustain Developer @Red Hat]

Relecture par Pauline Durand-Mabire <paulinedurandmabire@gmail.com>

MOTS-CLÉS : CONTINOUS INTEGRATION & DELIVERY, JAVA/JEE, BUILD, MAVEN, JENKINS, NGINX

Le serveur HTTP Nginx est un outil souple et très puissant, utilisé par de nombreux administrateurs de systèmes, mais peu connu des développeurs Java. Le serveur d'intégration Jenkins leur est par contre souvent familier, et encore plus Maven qui est probablement l'outil de construction de logiciel le plus utilisé dans le monde Java. Qu'est-ce que ces deux mondes bien distincts ont à voir ensemble? Eh bien, justement, c'est ce que nous allons voir dans cet article ! En étudiant comment le premier peut venir au secours du second...

Un serveur d'intégration continue comme **Jenkins [1]** construit de nombreux projets en parallèle, mais aussi bien souvent plusieurs versions du même logiciel, pour des raisons de maintenance évidentes. En effet, une équipe de développeurs en charge d'un logiciel met souvent en place différentes tâches de construction pour s'assurer que les modifications effectuées sur les différentes branches du projet n'entraînent aucune régression.

Dans le cas d'un projet **Java**, cette construction est souvent exécutée à l'aide de l'outil **Maven [2]**. Ce dernier prend en charge le téléchargement de nombreuses dépendances depuis le dépôt central fourni par la communauté Maven, mais pas seulement.

Le problème qui nous concerne ici est justement le téléchargement de ces dépendances. En effet, les constructions étant isolées les unes des autres, en l'occurrence utilisant des conteneurs légers (gérés, par exemple, par **Docker [3]**). Chaque projet télécharge ainsi ses propres dépendances, dans son propre environnement, même si elles, dans les faits, peuvent être les exactes mêmes dépendances que celles d'autres projets.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ceci aboutit à des constructions très lentes, qui doivent attendre le téléchargement de larges fichiers – les archives Java « JAR » faisant facilement plusieurs mégabytes, depuis un serveur distant, le plus souvent situé sur la toile (comme **Maven Central**). Ceci n'est pas très efficace surtout quand on sait que la plupart de ces constructions téléchargent sans cesse les mêmes fichiers !!!

C'est pour adresser cette problématique que nous allons discuter en détail de la mise en place d'un simple serveur mandataire inverse (« reverse proxy ») HTTP. Ce dernier fera en effet office de cache HTTP, pour éviter de télécharger à nouveau les fichiers JAR. Ainsi, à chaque construction, les archives Java ne seront plus téléchargées depuis un serveur distant, mais depuis un serveur local au serveur d'intégration – améliorant grandement les performances de ce dernier !

Nous verrons également par la suite comment la présence de ce simple serveur HTTP peut se révéler utile à notre serveur Jenkins de bien des façons...

1. MISE EN PLACE DU SERVEUR MANDATAIRE INVERSE

La première étape consiste, une fois **Nginx** installé, à mettre en place un « reverse proxy » entre le système hébergeant le serveur d'intégration continue et le miroir principal de Maven. C'est heureusement très simple, il suffit juste de définir une nouvelle **location** associée à un chemin, dans notre cas **/maven**, dans la configuration du serveur :

```
location /maven/ {
  proxy_pass http://central.
maven.org/maven2/;
}
```

Et oui, en seulement quelques lignes, le tour est joué. Si l'on accède maintenant au chemin **/maven** sur ce serveur, les requêtes sont transférées directement vers le site de Maven Central. Vérifions ceci à l'aide d'une requête construite avec **curl** :

```
$ curl -I http://localhost/maven/
HTTP/1.1 200 OK
Server: nginx/1.10.2
Date: Wed, 05 Apr 2017 14:27:31 GMT
Content-Type: text/html
Content-Length: 110101
Connection: keep-alive
Last-Modified: Wed, 05 Apr 2017 09:17:41 GMT
ETag: "abb9fc42ed23db4c590dd6a8d41a4703"
Via: 1.1 varnish
Fastly-Debug-Digest:
9648f889e542c791069a3712497fbde4749f85854bc0f9a7409e5b5c0130957f
Accept-Ranges: bytes
Via: 1.1 varnish
Age: 18469
X-Served-By: cache-iad2133-IAD, cache-ams4130-AMS
X-Cache: HIT, HIT
X-Cache-Hits: 1, 149
X-Timer: S1491402451.479684,VS0,VE0
# qui est bien identique à
$ curl -I http://central.maven.org/maven2/
HTTP/1.1 200 OK
Last-Modified: Wed, 05 Apr 2017 09:17:41 GMT
ETag: "abb9fc42ed23db4c590dd6a8d41a4703"
Content-Type: text/html
Via: 1.1 varnish
Fastly-Debug-Digest:
9648f889e542c791069a3712497fbde4749f85854bc0f9a7409e5b5c0130957f
Content-Length: 110101
Accept-Ranges: bytes
Date: Wed, 05 Apr 2017 14:28:08 GMT
Via: 1.1 varnish
Age: 18506
Connection: keep-alive
X-Served-By: cache-iad2133-IAD, cache-ams4143-AMS
X-Cache: HIT, HIT
X-Cache-Hits: 1, 93
X-Timer: S1491402489.800976,VS0,VE0
```

Bien évidemment, ceci n'est qu'une première étape. Il nous faut maintenant mettre en place un cache pour s'assurer qu'une fois les fichiers téléchargés, ils seront désormais servis en local. Néanmoins, il est important de s'assurer que le serveur mandataire fonctionne comme attendu avant d'étudier la mise en place du cache.

2. MISE EN PLACE DU CACHE HTTP

2.1 Rappel sur le fonctionnement d'un cache : éviction et expiration

La configuration du cache HTTP est plus complexe que celle du serveur mandataire. En effet, c'est une fonctionnalité beaucoup plus complexe, qui doit être en mesure de prendre certaines décisions telles que décider de l'expiration ou de l'éviction d'une de ses entrées.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

En effet, si une requête correspond à une entrée dans le cache, mais que cette entrée n'est plus à jour, le cache va décider de la supprimer et de télécharger à nouveau les données depuis le serveur source (« upstream »). Dans notre cas, ce serveur source sera le serveur de Maven Central.

Si ceci arrive, on parle alors d'expiration de l'entrée du cache. Cette entrée a simplement atteint la fin de son cycle de vie naturel au sein du cache et a été supprimée ou mise à jour.

Avant de se lancer dans la configuration du cache, il est important de bien distinguer l'expiration de l'éviction. En effet, l'éviction des données dans le cache ne repose pas sur sa validité en tant que telle, mais simplement sur le besoin de libérer de la place dans le cache pour ne pas sauvegarder plus de données que prévu par exemple. Cette suppression de données n'est donc pas dirigée par un besoin fonctionnel, mais bien technique.

L'éviction d'une entrée est généralement basée sur sa fréquence d'accès. Ainsi, les données les moins utilisées sont retirées en premier, de manière à assurer que le contenu du cache est le plus pertinent possible. On notera néanmoins qu'il existe d'autres stratégies d'éviction (et même d'expiration), mais que ces dernières dépassent de loin le cadre de cet article.

2.2 Définition de l'espace de stockage associé au cache

L'instruction **proxy_cache_path** placée en amont du fichier de configuration de Nginx, fournit au serveur les informations de stockage nécessaires à ce dernier :

```
proxy_cache_path /home/maven_central_
cache keys_zone=maven_central:20m
max_size=20g inactive=90d;
```

Sans surprise, le premier élément de configuration est l'emplacement des données du cache sur le système. Dans notre cas, il s'agit du répertoire **/home/maven_central_cache**. On notera tout de suite qu'il est important de vérifier que l'utilisateur associé à Nginx a bien accès à ce répertoire, en lecture comme en écriture. Dans le même ordre d'idée, il faudra aussi s'assurer que les politiques de sécurité **SELinux** ne bloquent pas l'accès à ce répertoire.

La seconde instruction indique l'identifiant du cache, dans notre cas **maven_central**, ainsi que la taille des entrées. Cette taille peut jouer un rôle très important dans le réglage fin du cache, mais ceci dépasse là aussi le cadre de cet article.

La troisième instruction **max_size** indique la taille maximum du cache. Compte tenu de la taille des archives Java, et de la taille d'un dépôt Maven local, nous avons opté ici pour une taille limite assez importante, **20** Gio. À noter que si jamais le cache atteint cette taille limite, le serveur pratiquera l'éviction de certaines entrées pour y placer de nouvelles entrées. Nginx ne pourra donc, en aucun cas, mettre le système en péril, en consommant plus de place que prévu.

Enfin, la dernière instruction importante dans notre cas d'utilisation est la définition de l'expiration des données. Dans notre cas, les données placées dans le cache ne devront que très rarement être mises à jour, l'expiration a donc été fixée à trois mois (**90** jours).

Attention, cette valeur d'expiration n'est pas absolue ! Elle peut être surchargée lors de l'utilisation du cache, comme nous le verrons plus loin.

2.3 Utilisation du cache au sein du serveur mandataire

Notre cache maintenant défini, reste à indiquer au serveur quand l'utiliser. Pour ceci, nous allons mettre à jour la configuration de notre **location** et l'associer au cache nommé **maven_central**.

```
location /maven/ {
  proxy_cache maven_central;
  proxy_pass http://central.maven.org/maven2/;
}
```

C'est la seule instruction nécessaire pour mettre en place le cache. Vérifions ceci rapidement :

```
$ curl -I http://localhost/maven/io/netty/netty-all/4.0.2.Final/
netty-all-4.0.2.Final.jar
HTTP/1.1 200 OK
Server: nginx/1.10.2
Date: Fri, 31 Mar 2017 16:48:29 GMT
Content-Type: application/java-archive
Content-Length: 1424935
Connection: keep-alive
ETag: "f0fc8db352e51e7949d09b00c14e0cb7"
Last-Modified: Wed, 17 Jul 2013 14:27:34 GMT
X-Checksum-MD5: f0fc8db352e51e7949d09b00c14e0cb7
X-Checksum-SHA1: 6ddc7ce505409ba8829a050d64dd8a26c9f5aed0
Via: 1.1 varnish
Fastly-Debug-Digest:
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
9a1ae14c3cbc26a85e1854dcd3241b14fefb9f06e3dc19d561dc4040ce54e733
Via: 1.1 varnish
Age: 92753
X-Served-By: cache-iad2135-IAD, cache-hhn1540-HHN
X-Cache: HIT, HIT
X-Cache-Hits: 1, 1
X-Timer: S1490978908.969885,VS0,VE16
Accept-Ranges: bytes
```

À titre de vérification, exécutons la même requête sur le serveur source, soit Maven Central, directement :

```
$ curl -I http://central.maven.org/maven2/io/netty/netty-
all/4.0.2.Final/netty-all-4.0.2.Final.jar
HTTP/1.1 200 OK
ETag: "f0fc8db352e51e7949d09b00c14e0cb7"
Content-Type: application/java-archive
Last-Modified: Wed, 17 Jul 2013 14:27:34 GMT
X-Checksum-MD5: f0fc8db352e51e7949d09b00c14e0cb7
X-Checksum-SHA1: 6ddc7ce505409ba8829a050d64dd8a26c9f5aed0
Via: 1.1 varnish
Fastly-Debug-Digest:
9a1ae14c3cbc26a85e1854dcd3241b14fefb9f06e3dc19d561dc4040ce54e733
Content-Length: 1424935
Accept-Ranges: bytes
Date: Fri, 31 Mar 2017 16:50:19 GMT
Via: 1.1 varnish
Age: 92863
Connection: keep-alive
X-Served-By: cache-iad2135-IAD, cache-hhn1521-HHN
X-Cache: HIT, HIT
X-Cache-Hits: 1, 1
X-Timer: S1490979019.111781,VS0,VE4
```

Néanmoins, ne nous arrêtons pas à cette configuration primaire. Certains paramètres peuvent en effet être adaptés à notre cas d'utilisation.

Le premier est le paramètre **proxy_cache_min_uses** qui définit combien de requêtes doivent être exécutées vers une entrée potentielle pour justifier sa mise en cache. Dans notre cas, si une dépendance est téléchargée par une construction, il est pratiquement sûr qu'elle le sera à nouveau, lors de la prochaine construction, ou même par une autre construction. Il semble donc pertinent d'indiquer au serveur que la mise en cache doit avoir lieu dès la première requête vers la ressource.

Un autre paramètre important, surtout dans notre contexte d'utilisation, est le **proxy_cache_use_stale** qui indique de ne pas sauvegarder les entrées associées aux pages d'erreurs. En effet, inutile de conserver les pages d'erreurs d'éventuels téléchargements qui auraient échoué de manière temporaire.

Mettons à jour la configuration de notre cache en conséquence :

```
location /maven/ {
  proxy_cache maven_central;
  proxy_cache_min_uses 1;
  proxy_cache_use_stale error timeout http_500 http_502
http_503 http_504;

  proxy_pass http://central.maven.org/maven2/;
}
```

Comme mentionné plus haut, il est possible de surcharger la date de validité d'une entrée du cache, pour ignorer celle indiquée dans la configuration globale (à l'aide de l'instruction de **proxy_cache_valid**). Cette instruction permet aussi d'indiquer le code HTTP associé aux données à placer dans le cache, suivi d'une durée de validité.

Attention néanmoins, si la durée de validité est omise, la valeur par défaut, de 3 minutes, sera utilisée et non celle définie dans la configuration globale du cache !

```
location /maven/ {
  proxy_cache maven_central;
  proxy_cache_min_uses 1;
  proxy_cache_use_stale error timeout
http_500 http_502 http_503 http_504;
  proxy_cache_valid 200 60d;

  proxy_pass http://central.maven.org/
maven2/;
}
```

Une dernière modification est ici recommandée : l'ajout d'en-tête HTTP (« HTTP Headers ») spécifique pour indiquer au client qu'il utilise un serveur mandataire et non la source :

```
location /maven/ {
  proxy_cache maven_central;
  proxy_cache_min_uses 1;
  proxy_cache_use_stale error
timeout http_500 http_502 http_503
http_504;
  proxy_cache_valid 200 60d;

  add_header Cache-Status $upstream_
cache_status;
  add_header Cache True;
  add_header Source-IP $remote_addr;

  proxy_pass http://central.maven.
org/maven2/;
}
```

On notera l'utilisation de variables spécifiques à Nginx :

- **$upstream_cache_status** : indique l'état du cache ;
- **$remote_addr** : l'adresse IP de la source.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Enfin, il est de bon ton d'ajouter un autre en-tête qui rend les règles de gestion publiques :

```
location /maven/ {
  proxy_cache maven_central;
  proxy_cache_min_uses 1;
  proxy_cache_use_stale error timeout http_500 http_502 http_503 http_504;
  proxy_cache_valid 200 60d;

  add_header Cache-Status $upstream_cache_status;
  add_header Cache True;
  add_header Source-IP $remote_addr;

  add_header Cache-Control public;

  proxy_pass http://central.maven.org/maven2/;
}
```

On peut constater dans la requête ci-dessous que nos entêtes supplémentaires sont désormais bien ajoutés à la réponse HTTP :

```
$ curl -I http://localhost/maven/io/netty/netty-all/4.0.2.Final/netty-
all-4.0.2.Final.jar
HTTP/1.1 200 OK
Server: nginx/1.10.2
Date: Fri, 31 Mar 2017 16:54:59 GMT
Content-Type: application/java-archive
Content-Length: 1424935
Connection: keep-alive
ETag: "f0fc8db352e51e7949d09b00c14e0cb7"
Last-Modified: Wed, 17 Jul 2013 14:27:34 GMT
X-Checksum-MD5: f0fc8db352e51e7949d09b00c14e0cb7
X-Checksum-SHA1: 6ddc7ce505409ba8829a050d64dd8a26c9f5aed0
Via: 1.1 varnish
Fastly-Debug-Digest:
9a1ae14c3cbc26a85e1854dcd3241b14fefb9f06e3dc19d561dc4040ce54e733
Via: 1.1 varnish
Age: 93143
X-Served-By: cache-iad2135-IAD, cache-hhn1524-HHN
X-Cache: HIT, HIT
X-Cache-Hits: 1, 1
X-Timer: S1490979299.307801,VS0,VE2
Cache-Status: MISS
Cache: True
Source-IP: 127.0.0.1
Cache-Control: public
Accept-Ranges: bytes
```

Avec l'ajout de tous ces en-têtes, on dispose désormais d'un outil précieux pour diagnostiquer d'éventuels problèmes avec le cache, mais aussi pour mettre en place une supervision adaptée.

3. MODIFICATION SUR LE SERVEUR D'INTÉGRATION CONTINUE

Nous avons désormais un cache local à notre système, s'exécutant « côte à côte », en quelque sorte, avec notre serveur Jenkins. Il nous reste encore à nous assurer que ce dernier l'utilise lors de la construction des projets. Il y a plusieurs approches possibles pour mettre ceci en place.

Une approche « globale », mais peu recommandée, consiste à redéfinir l'adresse de **central.maven.org**, dans le fichier **/etc/hosts** du système. Néanmoins, ceci implique aussi d'utiliser une adresse IP « fixe » dans la configuration de Nginx (sinon les requêtes tourneront en « boucle infinie »), ce qui est loin d'être élégant et peut surtout entraîner des problèmes de production plus tard si par exemple, Maven Central change d'adresse IP. Bref, ce n'est pas la meilleure solution ici, ni celle que l'on retiendra.

Une seconde option consiste à indiquer à Maven l'existence d'un serveur HTTP mandataire (« HTTP Proxy ») à utiliser pour ses connexions. Ceci fonctionne assez bien, mais force l'ensemble des requêtes HTTP à être dirigées vers notre serveur Nginx, ce qui n'est pas optimal, car toutes ne concernent pas Maven Central.

Il faudra donc, a minima, ajouter des règles de routage génériques pour rediriger le reste du trafic vers Internet. Mais surtout cette étape supplémentaire, dans l'établissement des connexions autres que le téléchargement des dépendances, ne fera que ralentir le serveur et ses constructions, sans réelle plus-value ajoutée. Enfin, ceci chargera, inutilement aussi, notre instance Nginx. Là encore, ce n'est pas la solution que nous allons retenir.

Au final, la solution la plus élégante, et surtout la plus flexible, consiste à modifier la configuration de Maven. Ceci pour lui indiquer l'existence d'un nouveau « miroir » associé au dépôt Maven Central.

Ceci peut se faire pour chaque projet ou de manière « globale » (en modifiant la configuration par défaut, située dans le répertoire d'installation de Maven ou **MAVEN_HOME**). Nous recommandons néanmoins de simplement modifier la configuration de chaque projet.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ACTUELLEMENT DISPONIBLE

HACKABLE N°19 !

CRÉEZ DES LIAISONS RADIO DE PLUSIEURS KILOMÈTRES !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Pour ce faire, il suffit de modifier (ou d'ajouter) un fichier **settings.xml** et de lui ajouter la définition du miroir :

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
          xsi:schemaLocation="http://maven.apache.org/
SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-
1.0.0.xsd">
  <pluginGroups/>
  <proxies/>
  <servers/>
  <mirrors>
    <mirror>
      <id>maven.central.cache</id>
      <mirrorOf>central</mirrorOf>
      <name>Maven Central Local Cache</name>
      <url>http://localhost/maven</url>
    </mirror>
  </mirrors>
  <profiles/>
</settings>
```

Bien qu'il puisse sembler plus simple de modifier la configuration globale de Maven, pour s'assurer que l'ensemble des constructions utilise le cache, nous recommandons la seconde approche : un fichier **settings.xml** par projet.

En effet, celle-ci est légèrement plus laborieuse, mais permet de mieux documenter l'utilisation (ou non) du cache. Elle offre aussi l'option appréciable, si besoin est, de pouvoir désactiver ce cache, sans intervention d'un administrateur du serveur d'intégration continue.

4. SÉCURISER L'ACCÈS À JENKINS ET AMÉLIORER SES PERFORMANCES À L'AIDE D'UN CACHE

Nous disposons désormais, à moindre coût, d'un cache HTTP, mis en place à l'aide de Nginx sur le même système exécutant notre serveur d'intégration Jenkins. Ceci nous ouvre en fait de nombreuses portes d'optimisation pour ce dernier.

Nous allons, dans cette section, discuter de deux options de réglages fins, qui peuvent se révéler très pratiques et bien améliorer l'expérience utilisateur, la sécurité et les performances de notre serveur d'intégration continue.

4.1 Déporter le chiffrage SSL sur Nginx

Avec les capacités d'un serveur d'intégration tel que Jenkins, qu'il s'agisse du déploiement de scripts personnalisés à l'aide de l'interface « web », ou simplement de son accès à un serveur IMAP, il est plus que recommandé de sécuriser l'accès à ce serveur. Idéalement, on aura a minima mis en place une authentification, pour limiter l'accès au service, et surtout assurer une nécessaire traçabilité en cas d'audit. Sans quoi, n'importe quel utilisateur malintentionné pourrait utiliser les capacités du serveur à des fins néfastes (par exemple, l'utiliser pour une attaque de type déni de service distribué).

Dans la même logique, il est important de chiffrer la communication entre le serveur et le monde extérieur, si ce n'est pour s'assurer que les identifiants utilisés pour l'authentification ne sont pas transmis « en clair » sur le réseau.

Ce chiffrement (et déchiffrement) peut bien évidemment être réalisé sur le serveur Jenkins en lui-même. Néanmoins, ceci ne fera qu'ajouter à la charge de la machine virtuelle Java l'exécutant, sachant qu'elle est déjà bien occupée avec la gestion des nombreuses constructions qu'on lui confie. À l'inverse, notre instance Nginx, même avec son cache HTTP, est loin d'être aussi plébiscitée. On va donc en profiter pour lui confier ces tâches de cryptage et décryptage de données.

Ceci permettra aussi d'exécuter Jenkins sur la boucle locale (localhost/127.0.0.1), avec un simple accès HTTP et ne laisser que Nginx écouter sur l'interface externe du système. Ceci permet d'envisager, entre autres, la mise en place d'un serveur mandataire inverse (« reverse proxy ») à des fins de sécurité, sur Nginx, qui analysera et vérifiera que les requêtes envoyées vers Jenkins sont valides, et non une tentative, par exemple, de pirater le serveur.

Néanmoins, dans le cadre de cet article nous nous limiterons, en terme de sécurité, à la mise en place du chiffrement et déchiffrement des requêtes. En effet, la mise en place d'un tel contrôle sur une application « web » aussi complexe que Jenkins est quelque peu laborieuse et l'objectif de cet article est de rester didactique.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

La première étape de la mise en place du chiffrement est la génération d'un certificat, ou simplement son obtention auprès des services gérant l'infrastructure informatique de votre compagnie. Une fois cela fait, il faut modifier la configuration de Nginx pour lui indiquer l'emplacement du certificat et lui indiquer de rediriger les requêtes vers Jenkins.

À cette fin, nous allons définir une nouvelle **location** au sein de notre configuration Nginx :

```
location /jenkins {
 proxy_pass http://localhost:8480;
 proxy_redirect     http://localhost:8480 https://localhost;

 proxy_set_header   Host        $host;
 proxy_set_header   X-Real-IP   $remote_addr;
 proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
 proxy_set_header   X-Forwarded-Proto $scheme;
 proxy_set_header   X-Forwarded-Port 443;
}
```

En fait, il suffit d'une paire de lignes pour définir la redirection. L'ajout des entêtes HTTP est essentiel pour faciliter le diagnostic d'éventuels problèmes et informer le client des requêtes que celles-ci ont été redirigées par un serveur mandataire.

Si nous avons redirigé les requêtes entrantes vers le serveur Jenkins, il reste encore à mettre en place le chiffrement. Ceci se fait au niveau de la section dédiée au **server** et non par **location** :

```
server {
      listen      <nom du système>:443 default_server;
      server_name <nom du système>;

      ssl on;
      ssl_certificate /etc/nginx/ssl/<nom du système>.crt;
      ssl_certificate_key /etc/nginx/ssl/<nom du système>.key;
...
}
```

La configuration ci-dessus est suffisante pour mettre en place le déchiffrement SSL par Nginx. Néanmoins, elle n'est pas très optimale, il est donc important d'utiliser quelques réglages fins supplémentaires.

Tout d'abord, il est de bon ton de placer dans un cache les identifiants de connexion, surtout que l'on peut sauvegarder ainsi ces informations associées jusqu'à 4000 sessions pour seulement 1Mb de mémoire !

On a ajouté au passage une durée limite de 3 heures (**180** mn) pour chaque connexion, ce qui semble, en première approximation largement suffisant :

```
      ssl_session_cache shared:SSL:1m; # 1mb holds 4000 sessions...
      ssl_session_timeout 180m;
```

Une seconde optimisation consiste à préférer systématiquement l'utilisation de TLS à SSL **[4]** et à choisir la liste d'algorithmes de chiffrage à privilégier (ou à ne pas utiliser) :

```
      ssl_protocols TLSv1 TLSv1.1
TLSv1.2;
      ssl_prefer_server_ciphers on;
      ssl_ciphers ECDH+AESGCM:ECDH
+AES256:ECDH+AES128:DH+3DES:!ADH:!
AECDH:!MD5;
```

4.2 Améliorer les performances « web » de Jenkins avec Nginx

Maintenant que nous avons placé, en frontal de notre serveur Jenkins, une instance de Nginx, il est possible aussi d'améliorer les performances de l'interface graphique du serveur d'intégration continue à l'aide de Nginx. En effet, de la même manière dont nous avons placé les dépendances téléchargées par les constructions de logiciels au sein d'un cache, on peut maintenant tout autant placer les ressources statiques du serveur (image, fichier de style CSS...) dans un cache géré par Nginx, pour servir ce contenu plus rapidement et sans solliciter Jenkins.

Sans surprise, la configuration est très similaire à celle que nous avons étudiée lors de la première partie, à l'exception, bien évidemment, de la redirection effectuée vers Maven Central. On commence donc, en en-tête du fichier de configuration principal de Nginx, par déclarer un nouveau cache :

```
# Caching static files for
Jenkins
proxy_cache_path /home/jboss/
jenkins_workspace/.http_cache
levels=1:2 keys_zone=jenkins_
cache:10m max_size=500m;
```

On adapte les paramètres utilisés précédemment à la mise en cache des ressources statiques de Jenkins, beaucoup moins conséquentes en taille que les archives Java téléchargées par Maven, et on définit un répertoire différent pour les

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

conserver. Ici, nous avons opté pour placer ces fichiers dans le répertoire de travail de Jenkins, pour rester en cohérence avec la logique de déploiement de ce dernier. On s'assure ainsi que si Jenkins est supprimé ou déplacé, le cache le sera aussi.

On met maintenant à jour la configuration proposée plus haut pour la redirection vers le chemin **/jenkins** avec les paramètres nécessaires à la mise en cache :

```
location /jenkins {
  add_header X-Cache-Status $upstream_cache_status;
  proxy_cache jenkins_cache;

  proxy_pass http://localhost:8480 ;
  proxy_redirect     http://localhost:8480 https://localhost;
  proxy_set_header   Host        $host;
  proxy_set_header   X-Real-IP   $remote_addr;
  proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
  proxy_set_header   X-Forwarded-Proto $scheme;
  proxy_set_header   X-Forwarded-Port 443;
}
```

4.3 Simplifier la configuration des entêtes HTTP

Par souci de cohérence, on utilise, là encore, les mêmes en-têtes spécifiques que nous avons définis auparavant. Comme ces derniers deviennent systématiques et surtout redondants, on peut les placer dans un fichier séparé, qui sera inclus à chaque utilisation d'un cache. Dans notre cas, nous avons opté pour nommer ce fichier **/etc/nginx/proxy.conf** :

```
  proxy_set_header   Host             $host;
  proxy_set_header   X-Real-IP        $remote_addr;
  proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
  proxy_set_header   X-Forwarded-Proto $scheme;
  proxy_set_header   X-Forwarded-Port 443;
```

Ce fichier défini, on peut donc remplacer son contenu par une simple inclusion, ce qui allège beaucoup la configuration de notre serveur :

```
server {
  ...
  location /jenkins {
    proxy_cache jenkins_cache;

    proxy_pass http://{{ ansible_nodename }}:8480 ;
    proxy_redirect     http://{{ ansible_nodename
}}:8480 https://{{ ansible_nodename }};
    include /etc/nginx/proxy.conf;
  }
    ...
  location /maven/ {
    proxy_cache maven_central;
    proxy_cache_min_uses 1;
    proxy_cache_use_stale error timeout http_500 http_502
http_503 http_504;
    proxy_cache_valid 200 60d;

    proxy_pass http://central.
maven.org/maven2/;
    include /etc/nginx/proxy.conf;
}
```

CONCLUSION

D'un point de vue microscopique, on peut noter qu'installer Nginx, sur le même système que Jenkins, n'a pratiquement aucun coût, tout en apportant beaucoup d'assistance au serveur d'intégration continue. De manière plus macroscopique, séparer les fonctionnalités et construire, même au sein d'un simple système, offre beaucoup de souplesse. Trop souvent, on tente de régler des problèmes, ou d'ajouter des fonctionnalités, en augmentant une base de code, et en construisant une solution « complète ».

Néanmoins, et particulièrement avec des solutions *open source*, il est souvent beaucoup plus pertinent de « diviser pour mieux régner », et d'utiliser différentes solutions pour leurs fonctionnalités primaires, et de les imbriquer pour implémenter l'ensemble de son cahier des charges. Comme nous espérons l'avoir démontré dans cet article... ■

RÉFÉRENCES

[1] Jenkins CI : https://jenkins.io/

[2] Apache Maven : https://maven.apache.org/

[3] Docker : http://docker.io/

[4] TLS vs SSL : https://security.stackexchange.com/questions/5126/whats-the-difference-between-ssl-tls-and-https/w

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DISPONIBLE DÈS LE 14 JUILLET

GNU/LINUX MAGAZINE HORS-SÉRIE N°91 !

CRÉEZ, PUBLIEZ ET MONÉTISEZ VOTRE APPLICATION AVANCÉE ANDROID

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DUKE NUKEM 3D : UN OUTIL VALGRIND ADAPTÉ À LA LECTURE D'APPELS SYSTÈMES

SÉBASTIEN TRICAUD
[Je bidouille des trucs, j'aime courir dans mon jardin, faire rire les enfants, gonfler des ballons et faire mes propres pâtes]

MOTS-CLÉS : VALGRIND, DUKE NUKEM 3D, LANGAGE C, MACHINE VIRTUELLE, VEX

On ne présente plus ni Duke Nukem 3D, ni Valgrind. Les deux outils sont très complémentaires : d'un côté, nous avons un émulateur de processeur RISC qui permet de faire une abstraction des instructions d'origine, d'un autre vous avez un jeu d'arcade qui permet de supporter de longues heures passées sur le premier ! Alors, pourquoi ne pas être dans l'honnêteté intellectuelle la plus complète et fusionner les deux outils afin de rendre le debugging great again ?

1. À LA DÉCOUVERTE DE VALGRIND !

Valgrind est un outil fantastique, qui a déjà plusieurs années au compteur et permet aux développeurs bas niveau d'éviter principalement les problèmes de fuites mémoires... mais pas seulement : Valgrind permet aussi de débugger les conditions d'exécution (*race conditions*) ou, ce que je préfère personnellement, le profilage avec des fichiers créés que l'on ouvre ensuite dans l'interface **KCachegrind** afin de déterminer les fonctions qui prennent le plus de temps.

Valgrind permet une abstraction du processeur et permet de traduire les instructions en code RISC via la machine virtuelle **VEX** les différents processeurs suivants : arm64, amd64, ppc32, ppc64, arm v8, mips32, mips64, x86 et s390x.

La machine virtuelle compile en temps réel (JIT) cette abstraction et permet de décoder le code CPU invité pour assembler le code hôte. **Coregrind** permet d'émuler les appels systèmes du système d'exploitation.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Les instructions sont envoyées par blocs IRSB pour *Intermediate Representation Super Block* (Super Bloc de représentation intermédiaire... dans l'ordre c'est plus clair ! Il ne faut pas s'étonner après que les Anglais roulent à gauche!). Chaque IRSB contient un certain nombre d'instructions qui démarre par le typage des variables temporaires, comme :

```
t0:I64 t1:I64 t2:I64 t3:I64 t4:I64 t5:I64
t6:I64 t7:I64
t8:I64 t9:I64 t10:I64 t11:I64 t12:I64
```

Ici toutes les variables stockent une valeur de type entier sur 64 bits.

Nous allons revenir un peu plus loin sur la représentation intermédiaire de la machine VEX, mais en attendant, essayons d'utiliser Valgrind le plus simplement du monde.

Supposons que nous souhaitions trouver des fuites mémoires. Prenons l'exemple le plus trivial, celui d'une fuite mémoire et regardons ce que Valgrind nous dit. On crée un fichier **simpleleak.c** :

```
#include <stdlib.h>

int main(void)
{
  void *p = malloc(42);

  return 42;
}
```

On compile, en n'oubliant pas de mettre les symboles de débug :

```
$ gcc simpleleak.c -o simpleleak -ggdb
```

Ensuite, nous pouvons lancer dans Valgrind et admirer le résultat :

```
$ valgrind ./simpleleak
==13792== Memcheck, a memory error detector
==13792== Copyright (C) 2002-2015, and GNU
GPL'd, by Julian Seward et al.
==13792== Using Valgrind-3.12.0 and LibVEX;
rerun with -h for copyright info
==13792== Command: ./simpleleak
==13792==
==13792==
==13792== HEAP SUMMARY:
==13792==     in use at exit: 42 bytes in 1
blocks
==13792==   total heap usage: 1 allocs, 0
frees, 42 bytes allocated
==13792==
==13792== LEAK SUMMARY:
==13792==    definitely lost: 42 bytes in 1 blocks
==13792==    indirectly lost: 0 bytes in 0 blocks
==13792==      possibly lost: 0 bytes in 0 blocks
==13792==    still reachable: 0 bytes in 0 blocks
==13792==         suppressed: 0 bytes in 0 blocks
==13792== Rerun with --leak-check=full to see
details of leaked memory
==13792==
==13792== For counts of detected and suppressed
errors, rerun with: -v
==13792== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
```

Valgrind nous indique la fuite de mémoire et nous propose de le relancer avec l'option **–leak-check=full**.

Ce qui nous donne :

```
$ valgrind --leak-check=full ./simpleleak
==14456== Memcheck, a memory error detector
==14456== Copyright (C) 2002-2015, and GNU GPL'd,
by Julian Seward et al.
==14456== Using Valgrind-3.12.0 and LibVEX; rerun
with -h for copyright info
==14456== Command: ./simpleleak
==14456==
==14456==
==14456== HEAP SUMMARY:
==14456==     in use at exit: 42 bytes in 1 blocks
==14456==   total heap usage: 1 allocs, 0 frees,
42 bytes allocated
==14456==
==14456== 42 bytes in 1 blocks are definitely lost
in loss record 1 of 1
==14456==    at 0x4C2DB9D: malloc (vg_replace_
malloc.c:299)
==14456==    by 0x400507: main (simpleleak.c:5)
==14456==
==14456== LEAK SUMMARY:
==14456==    definitely lost: 42 bytes in 1 blocks
==14456==    indirectly lost: 0 bytes in 0 blocks
==14456==      possibly lost: 0 bytes in 0 blocks
==14456==    still reachable: 0 bytes in 0 blocks
==14456==         suppressed: 0 bytes in 0 blocks
==14456==
==14456== For counts of detected and suppressed
errors, rerun with: -v
==14456== ERROR SUMMARY: 1 errors from 1 contexts
(suppressed: 0 from 0)
```

Nous voyons ici très clairement que la fuite mémoire provient d'une allocation de **42** octets faite à la ligne 5, soit :

```
void *p = malloc(42);
```

On remédiera à cela en utilisant la fonction **free()**, mais il est intéressant de noter la précision de Valgrind afin d'aider le programmeur ou la programmeuse à mieux compiler le **C**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

2. LES OUTILS DE VALGRIND

En disant que Valgrind permettait de traquer les fuites mémoires, j'ai fait un raccourci. Plus précisément nous utilisions l'outil **Memcheck** de Valgrind, qui est l'outil utilisé par défaut. Il n'y a pas besoin de le nommer sur la ligne de commandes, mais supposons que nous souhaitions faire du profilage de fonction, nous aurions dû utiliser l'outil **Cachegrind**. Rajoutons une fonction **attendre()** qui contient une pause d'une seconde comme ceci :

```
void attendre(void)
{
  sleep(1);
}
```

Nous pouvons ajouter le paramètre **--tool=cachegrind** à Valgrind afin de changer d'outil :

```
$ valgrind --tool=cachegrind ./profilage
==10021== Cachegrind, a cache and branch-prediction profiler
==10021== Copyright (C) 2002-2015, and GNU GPL'd, by
Nicholas Nethercote et al.
==10021== Using Valgrind-3.12.0 and LibVEX; rerun with -h
for copyright info
==10021== Command: ./profilage
==10021==
--10021-- warning: L3 cache found, using its data for the LL
simulation.
==10021==
==10021== I   refs:      158,274
==10021== I1  misses:        817
==10021== LLi misses:        808
==10021== I1  miss rate:    0.52%
==10021== LLi miss rate:    0.51%
==10021==
==10021== D   refs:       51,615  (40,086 rd   + 11,529 wr)
==10021== D1  misses:      2,930  ( 2,382 rd   +    548 wr)
==10021== LLd misses:      2,431  ( 1,926 rd   +    505 wr)
==10021== D1  miss rate:     5.7% (   5.9%     +    4.8%  )
==10021== LLd miss rate:     4.7% (   4.8%     +    4.4%  )
==10021==
==10021== LL refs:         3,747  ( 3,199 rd   +    548 wr)
==10021== LL misses:       3,239  ( 2,734 rd   +    505 wr)
==10021== LL miss rate:      1.5% (   1.4%     +    4.4%  )
```

En regardant les fichiers du répertoire courant, on admirera la création d'un fichier **cachegrind.out.<pid>** ; il est possible d'utiliser ce fichier avec l'outil KCachegrind pour voir le suivi des appels fonctions et le temps mesuré à l'intérieur de chacune d'entre elles.

Valgrind fournit encore d'autres outils, que je vous invite à regarder : **Helgrind**, **Massif**, **Callgrind** ou encore **Lackey**.

Ce que nous allons d'abord faire c'est d'écrire un outil pour Valgrind qui ira lire le registre d'instruction du processeur afin de récupérer le numéro de l'appel système et de l'écrire dans un fichier que l'on utilisera ensuite avec **duke3d** pour lire ces appels systèmes et les afficher à chaque fois que l'on tuera un ennemi \o/

3. LA REPRÉSENTATION INTERMÉDIAIRE DE VEX

L'intérêt de la représentation intermédiaire proposée par VEX est un découpage granulaire de chaque étape. Ce qui serait une simple opération d'ajout de la valeur d'un registre vers un autre et ne serait qu'une seule instruction pour une architecture x86 se transforme en 4 instructions pour VEX :

1. Récupérer la valeur du premier registre dans une variable temporaire ;
2. Récupérer la valeur du deuxième registre dans une variable temporaire ;
3. Ajouter dans une variable temporaire l'opération ;
4. Mettre dans le registre voulu le résultat de cette dernière variable temporaire.

Ce qui peut dans un premier temps paraître un peu lourd, mais permet une super instrumentation du code exécuté!

Afin de comprendre les différentes étapes que nous aurons à traiter dans notre code, voici quelques exemples d'instructions pour certaines opérations de base (voir tableau en haut page suivante).

4. L'ACCÈS AUX REGISTRES

Lorsque l'on utilise l'instruction **GET** ou **PUT**, le paramètre nécessaire est le numéro correspondant au registre de l'architecture du CPU hôte. Voici un tableau de correspondance entre le numéro et le registre avec le pointeur d'instruction en cadeau bonus (OK, nous en aurons besoin !) pour processeurs x86_64 :

Numéro	Registre
16	RAX
24	RCX
32	RDX
40	RBX
72	RDI
184	RIP

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Description	Instruction
Constante	`PUT(144) = 0x4:I64`
Récupérer la valeur contenue dans un registre	`t2 = GET:I64(16)`
Mettre une valeur dans un registre	`PUT(72) = t40` ou bien `PUT(184) = 0x4001193:I64`
Récupérer une valeur depuis la mémoire	`t14 = LDle:I64(t2)`
Mettre une valeur en mémoire	`STle(t44) = t2`
Conditions, où ici `t40` stocke la condition, `t29` si la condition est vraie et `t10` si elle est fausse	`t67 = ITE(t40,t29,t10)`
Quitter	`if (t79) { PUT(184) = 0x40047F2:I64; exit-Boring }`
Retour de fonction	`PUT(184) = t23; exit-Return`
Appel de fonction, où `I64` représente le type de retour	`t34 = amd64g_calculate_rflags_c[mcx=0x9] {0x3813af80}(t12,t13,t14,t15):I64`

Ainsi que la liste de lecture des arguments donnés à un appel système (la liste complète se trouve à cette adresse : https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-r252.pdf):

Registre	Ordre d'argument
RDI	1
RSI	2
RDX	3
RCX	4
R8	5
R9	6

Notez bien ces numéros, car ils seront fort utiles dans la partie suivante où nous allons les lire afin de comprendre quand il y a un appel système d'exécuté.

5. DÉFINITION DES APPELS SYSTÈMES DU SYSTÈME D'EXPLOITATION

Le but de Valgrind est de suivre absolument tout ce que le programme fait, et dans le cas où un appel système se produit, c'est le noyau qui prend le relai, exécute son schmilblick, et rend la main au programme. Le noyau va régulièrement modifier certains endroits de la mémoire du programme. Du coup les outils doivent être au courant de ce qui est effectué.

Si l'on cherche à avoir le numéro des appels systèmes sur une architecture x86 pour Linux, on peut aller fouiller dans le code du noyau directement. Tout est dans le fichier `arch/x86/entry/syscalls/syscall_64.tbl`.

Les appels systèmes ont leur prototype identifié dans `coregrind/m_syswrap/syswrap-*.c`, en remplaçant * par le système d'exploitation visé.

Voici quelques exemples de correspondance entre le numéro et l'appel système :

Valeur dans RAX	Valeur en décimal de RAX	Appel système
0xC	12	sys_brk
0x3F	63	sys_newuname
0x15	21	sys_access
0x9	9	sys_mmap
0x2	2	sys_open
0x0	0	sys_read
0x5	5	sys_newfstat
0xA	10	sys_mprotect
0x3	3	sys_close
0x9E	158	sys_arch_prctl
0xB	11	sys_munmap
0xE7	231	sys_exit_group

Il ne nous reste plus maintenant qu'à orchestrer tout ceci, et écrire le code qui permettra à Valgrind d'enregistrer le fichier qui listera les appels systèmes que nous utiliserons ensuite dans **Duke Nukem 3D**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

6. RÉCUPÉRATION DU CODE DE VALGRIND, CRÉATION DE NOTRE OUTIL « VSTRACE », POUR VALGRIND STRACE !

Tout d'abord, il faut récupérer le code de Valgrind :

```
$ svn co svn://svn.valgrind.org/valgrind/trunk valgrind
```

Ensuite, nous allons appliquer le patch suivant, qui permet de rajouter **vstrace** dans le système de compilation de Valgrind :

```
Index: Makefile.am
===================================================================
--- Makefile.am (revision 15750)
+++ Makefile.am (working copy)
@@ -10,10 +10,12 @@
   lackey \
   none \
   helgrind \
+  vstrace \
   drd

 EXP_TOOLS = exp-sgcheck \
   exp-bbv \
+  vstrace \
   exp-dhat

 # Put docs last because building the HTML is slow and we
want to get
Index: configure.ac
===================================================================
--- configure.ac (revision 15750)
+++ configure.ac (working copy)
@@ -4148,6 +4148,10 @@
    exp-dhat/tests/Makefile
    shared/Makefile
    solaris/Makefile
+
+   vstrace/Makefile
+   vstrace/tests/Makefile
+
 ])
 AC_CONFIG_FILES([coregrind/link_tool_exe_linux],
                 [chmod +x coregrind/link_tool_exe_linux])
```

Nous allons créer un répertoire **vstrace** dans les sources de Valgrind, et nous allons créer un fichier qui correspond à l'exécution du code dans la machine virtuelle VEX, et un autre qui contient une simple table de correspondance entre le numéro de l'appel système et son nom sous forme d'une chaîne lisible. Nous créons donc le fichier **vstrace_syscall_names.h**, qui contient cette table :

```
#ifndef _VSTRACE_SYSCALL_NAMES_H_
#define _VSTRACE_SYSCALL_NAMES_H_

char *syscall_names[] = {"sys_read","sys_
write","sys_open",...,"sys_memfd_
create","sys_ni_syscall","sys_ni_syscall"};

#endif // _VSTRACE_SYSCALL_NAMES_H_
```

ATTENTION

Je me permets de raccourcir, car tout le code est disponible sur http://www.github.com/stricaud/articles/, ce qui permet dans cet article de mettre en avant les points importants seulement.

Maintenant nous allons aller au cœur de l'émulation en créant un fichier **vstrace_main.c** qui suit un prototype particulier afin d'être intégré comme outil de Valgrind.

Tout commence par la définition des *callbacks* :

```
static void vstrace_pre_clo_init(void)
{
  VG_(details_name)            ("vstrace");
  VG_(details_version)         (NULL);
  VG_(details_description)     ("Creating a file with
syscalls");
  VG_(details_copyright_author)(
     "Copyright (c) 2017 Sebastien Tricaud");
  VG_(details_bug_reports_to)  (VG_BUGS_TO);

  VG_(details_avg_translation_sizeB) ( 275 );

  VG_(basic_tool_funcs)        (vstrace_post_clo_init,
                                vstrace_instrument,
                                vstrace_fini);


}

VG_DETERMINE_INTERFACE_VERSION(vstrace_pre_clo_init)
```

Lisez ce code de bas en haut. Nous indiquons à Valgrind la fonction qui contient le code d'enregistrement des informations sur l'auteur et des *callbacks* à appeler lorsque l'on initialise, instrumente et termine.

À noter que les fonctions sont un peu bizarres, car elles commencent toutes par un **VG_**, ce qui est tout à fait normal, car nous sommes dans l'abstraction la plus totale et n'avons plus accès à la **libc**. Il faut donc utiliser les fonctions de Valgrind qui réimplémentent toutes ces fonctions.

Nous allons créer un fichier **vstrace.out**, ce qui nous impose d'avoir une variable statique au début du fichier que l'on utilisera dans les différentes fonctions :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Abonnez-vous !

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO 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.

LINUX MAGAZINE / FRANCE

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 :

.....

.....

Ce document est la propriété exclusive de Johann Locatelli (jacques.thimonier@businessdecision.com)

Bon d'abonnement

CHOISISSEZ VOTRE OFFRE !

Prix TTC en Euros / France Métropolitaine*

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC
LM	11n° GNU/Linux Magazine France	LM1	69,-	LM13	245,-
LM+	11n° GNU/Linux Magazine France + 6n° Hors-Série	LM+1	125,-	LM+13	299,-
LES COUPLAGES AVEC NOS AUTRES MAGAZINES					
A	11n° GNU/Linux Magazine France + 6n° Linux Pratique	A1	105,-	A13	399,-
A+	11n° GNU/Linux Magazine France + 6n° Hors-Série + 6n° Linux Pratique + 3n° Hors-Série	A+1	189,-	A+13	489,-
B	11n° GNU/Linux Magazine France + 6n° MISC	B1	109,-	B13	499,-
B+	11n° GNU/Linux Magazine France + 6n° Hors-Série + 6n° MISC + 2n° Hors-Série	B+1	185,-	B+13	629,-
C	11n° GNU/Linux Magazine France + 6n° Linux Pratique + 6n° MISC	C1	149,-	C13	669,-
C+	11n° GNU/Linux Magazine France + 6n° Hors-Série + 6n° Linux Pratique + 3n° Hors-Série + 6n° MISC + 2n° Hors-Série	C+1	249,-	C+13	769,-
J	11n° GNU/Linux Magazine France + 6n° Hackable	J1	105,-	J13	399,-
J+	11n° GNU/Linux Magazine France + 6n° Hors-Série + 6n° Hackable	J+1	159,-	J+13	459,-
LA TOTALE DIAMOND !					
L	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC	L1	189,-	L13	839,-
L+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	L+1	289,-	L+13	939,-

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

Particuliers = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com

pour consulter toutes les offres !

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

Professionnels = CONNECTEZ-VOUS SUR :

http://proboutique.ed-diamond.com

pour consulter toutes les offres !

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

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
static const HChar *vstrace_out_file =
"vstrace.out";
static VgFile *fp;
```

Notez que les types sont aussi particuliers.

Nous pouvons maintenant passer à ce que nous faisons dans notre fonction d'initialisation :

```
static void vstrace_post_clo_init(void)
{
  fp = VG_(fopen)(vstrace_out_file,
                  VKI_O_CREAT|VKI_O_TRUNC|
VKI_O_WRONLY,
                  VKI_S_IRUSR|VKI_S_IWUSR);
  if (!fp) {
    VG_(umsg)("Error: cannot open the file to
write syscalls into: %s\n", vstrace_out_file);
    return;
  }
}
```

Nous ouvrons en écriture le fichier **vstrace.out** et faisons un test sur la variable pointant sur le type **VgFile**. Si nous ne pouvons pas écrire dans ce fichier, nous renvoyons une erreur à l'utilisateur.

N'oublions pas de fermer ce fichier dans la fonction de terminaison :

```
static void vstrace_fini(Int exitcode)
{
  VG_(fclose)(fp);
}
```

Maintenant, regardons le cœur de l'instrumentation. Comme expliqué plus haut, nous recevons des blocs et nous devons dérouler chaque instruction de ces différents blocs :

```
 static
IRSB* vstrace_instrument ( VgCallbackClosure* closure,
                           IRSB* sbIn,
                           VexGuestLayout* layout,
                           VexGuestExtents* vge,
                           VexArchInfo* archinfo_host,
                           IRType gWordTy, IRType hWordTy )
{
  Int i;
  IRStmt *st;
  int syscall_id;

  i = 0;

  for (; i < sbIn->stmts_used; i++) {
    st = sbIn->stmts[i];

    switch(st->tag) {
    case Ist_Put:
      if (st->Ist.Put.offset == 16) {
        /* RAX */
        const IRExpr* e;
        const IRConst* con;
        e = st->Ist.Put.data;
        if (e->tag == Iex_Const) {
          con = e->Iex.Const.con;
          if (con->tag == Ico_U64) {
            syscall_id = (int *)con->Ico.U64;
          }
        }
      }
      break;
    }
  } // for (; i < sbIn->stmts_used; i++) {



  switch(sbIn->jumpkind) {
  case Ijk_Sys_syscall:
    VG_(fprintf)(fp, "%s\n", syscall_names[syscall_id]);
  break;
  }


  return sbIn;
}
```

C'est lorsque nous recevons une instruction **PUT** sur **RAX** que nous stockons la valeur émise pour la réutiliser après lors d'un saut et récupérer le nom de l'appel système depuis notre table. Nous l'enregistrons simplement dans notre fichier. Le pointeur d'instruction **RIP** est appelé au moment du saut, que VEX nous classifie comme **Ijk_Sys_syscall**, et nous n'avons plus qu'à reprendre la valeur qui était stockée dans le registre **RAX** pour connaître depuis notre table le nom du registre et l'enregistrer dans le fichier.

Maintenant, nous pouvons lancer un **make install**, et c'est parti ! On lance la commande **ls** depuis notre outil :

```
$ ~/opt/bin/valgrind --tool=vstrace ls
==3925== vstrace, Creating a file with syscalls
==3925== Copyright (c) 2017 Sebastien Tricaud
==3925== Using Valgrind-3.12.0.SVN and LibVEX;
rerun with -h for copyright info
==3925== Command: ls
==3925==
bar vstrace.out
==3925==
```

Nous avons deux fichiers, **bar** et **vstrace.out** (car dès le premier appel système de **ls** on enregistre et du coup **ls** le voit. J'adore cet esprit de récursivité !), qui apparaissent sur la console, mais le plus intéressant reste le fichier **vstrace.out** qui vient d'être créé :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
$ cat vstrace.out
sys_brk
sys_newuname
sys_open
sys_read
sys_newfstat
sys_mmap
sys_mmap
sys_mprotect
sys_close
sys_access
sys_open
sys_close
sys_close
sys_arch_prctl
sys_mprotect
sys_munmap
sys_set_tid_address
sys_set_robust_list
sys_rt_sigaction
sys_rt_sigprocmask
sys_getrlimit
sys_statfs
sys_statfs
sys_brk
sys_brk
sys_access
sys_open
sys_newfstat
sys_mmap
sys_close
sys_ioctl
sys_ioctl
sys_open
sys_getdents
sys_close
sys_write
sys_close
sys_exit_group
```

Vous pouvez admirer l'ensemble des appels système qui ont été exécutés par la commande **ls**.

Si vous souhaitez approfondir encore plus Valgrind, je vous recommande la lecture des commentaires laissés dans les entêtes des différents fichiers sources, et toute littérature écrite par Julian Seward, tel que https://www.cs.columbia.edu/~junfeng/09fa-e6998/papers/valgrind.pdf.

Maintenant que nous avons obtenu ce que nous voulions de Valgrind, nous voulons utiliser ce fichier en entrée pour Duke Nukem 3D, afin de lister ces appels système un par un dès lors que nous tuons un ennemi !

7. DÉMARRER AVEC LE CODE DE DUKE NUKEM 3D

Tout d'abord, nous allons récupérer le code du port nommé **chocolate-duke3D** :

```
$ git clone https://github.com/fabiensanglard/chocolate_duke3D
```

Que nous compilons avec les autotools :

```
$ ./autogen.sh && ./configure && make
```

Il vous faudra installer la bibliothèque **sdl** et **sdl-mixer** si jamais vous ne les avez pas. Cela crée un binaire **chocolate-duke3d**.

Il vous faut ensuite le fichier **duke3d.grp**, qui correspond aux niveaux et éléments graphiques du jeu.

Nous allons sur le site de l'éditeur pour récupérer les niveaux disponibles en *shareware* : http://legacy.3drealms.com/duke3d/.

Nous récupérons le bon fichier, depuis ftp://ftp.3drealms.com/share/3dduke13.zip.

```
$ unzip 3dduke13.zip
Archive:  3dduke13.zip
  inflating: LICENSE.TXT
  inflating: INSTALL.EXE
  inflating: DN3DSW13.SHR
  inflating: FILE_ID.DIZ
```

Et nous décompressons le fichier **.SHR** :

```
$ unzip DN3*
Archive:  DN3DSW13.SHR
replace LICENSE.TXT? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: LICENSE.TXT
  inflating: COMMIT.EXE
  inflating: DEFS.CON
  inflating: DEMO1.DMO
  inflating: DEMO2.DMO
  inflating: DEMO3.DMO
  inflating: DN3DHELP.EXE
  inflating: DUKE.RTS
  inflating: DUKE3D.EXE
  inflating: DUKE3D.GRP
  inflating: GAME.CON
  inflating: MODEM.PCK
  inflating: README.DOC
  inflating: SETMAIN.EXE
  inflating: SETUP.EXE
  inflating: ULTRAMID.INI
  inflating: USER.CON
```

Nous pouvons maintenant copier le fichier **DUKE3D.GRP** où se trouve le binaire **chocolate-duke3d**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

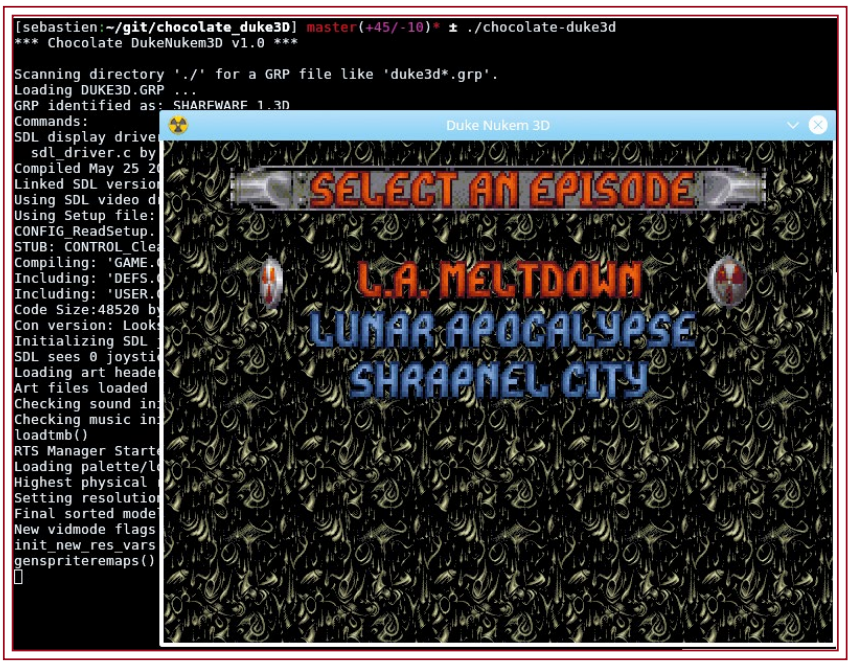


Fig. 1 : Premier lancement.

Nous sommes obligés de tester (voir figure 1).

Maintenant nous allons démarrer notre investigation afin de déterminer comme nous pouvons placer du texte sur l'écran.

Cherchons où existe la chaîne de caractères « Large Medkit », qui apparaît dès le début, lorsque l'on saute via le conduit et que l'on atterrit sur un kit médical :

```
$ grep -ri "large medkit" *
Binary file DUKE3D.GRP matches
```

Bon, il ne nous reste plus qu'à aller chercher cette chaîne dans le fichier binaire, car elle n'est présente nulle part dans le code source !

```
$ hexdump -C DUKE3D.GRP |less
00022550  21 0d 0a 64 65 66 69 6e  65 71 75 6f 74 65 20 36  |!..definequote 6|
00022560  31 20 20 20 20 20 20 20  20 53 4d 41 4c 4c 20 4d  |1        SMALL M|
00022570  45 44 4b 49 54 3a 20 2b  31 30 0d 0a 64 65 66 69  |EDKIT: +10..defi|
00022580  6e 65 71 75 6f 74 65 20  36 32 20 20 20 20 20 20  |nequote 62      |
00022590  20 20 4c 41 52 47 45 20  4d 45 44 4b 49 54 3a 20  |  LARGE MEDKIT: |
000225a0  2b 33 30 0d 0a 64 65 66  69 6e 65 71 75 6f 74 65  |+30..definequote|
000225b0  20 36 33 20 20 20 20 20  20 20 20 41 4d 4d 4f 20  | 63        AMMO |
```

Trouvé ! Nous voyons que « LARGE MEDKIT » apparaît juste après le mot-clé « definequote », cherchons alors pour ce mot-clé dans les sources :

```
$ grep -r definequote *
Binary file chocolate-duke3d matches
Binary file DUKE3D.GRP matches
Binary file Game/src/libGame_a-gamedef.o matches
Game/src/gamedef.c:    "definequote",      // 79
Binary file Game/src/libGame.a matches
```

Et dans le fichier **gamedef.c**, il y a donc bien un tableau avec **definequote** :

```
char  *keyw[NUMKEYWORDS] =
{
    "definelevelname",  // 0
    "actor",            // 1    [#]
    "addammo",          // 2    [#]
    "ifrnd",            // 3    [C] [:]
    "ifcansee",         // 5    [C]
...
```

On voit un gros **switch** un peu plus bas, où le tableau **keyw** est utilisé. On cherche la partie contenant le **case 79** :

```
        case 79:
            scriptptr--;
            transnum();
            k = *(scriptptr-1);
            if(k >= NUMOFFIRSTTIMEACTIVE)
            {
                printf("  *
ERROR!(L%hd) Quote amount exceeds
limit of %d characters.\n",line_
number,NUMOFFIRSTTIMEACTIVE);
                error++;
            }
            scriptptr--;
            i = 0;
            while( *textptr == ' ' )
                textptr++;

            while( *textptr != 0x0a )
            {
                fta_quotes[k][i] =
*textptr;
                textptr++,i++;
                if(i >= 64)
                {
                    printf("  *
ERROR!(L%hd) Quote exceeds character
size limit of 64.\n",line_number);
                    error++;
                    while( *textptr !=
0x0a ) textptr++;
                    break;
                }
            }
            fta_quotes[k][i] = '\0';
            return 0;
```

Et là je me dit que ce ne sera pas aussi facile que pour faire **Wolfotrack**, le code est *inteubable* ! J'essaie donc ma chance avec **game.c**, et là, dès le début, je vois :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
#define MINITEXT_BLUE   0
#define MINITEXT_RED    2
#define MINITEXT_YELLOW 23
#define MINITEXT_GRAY   17

#define COLOR_ON  MINITEXT_YELLOW
#define COLOR_OFF MINITEXT_BLUE
```

Génial, on dirait que l'on avance ! En cherchant ligne 1169, on voit :

```
minitext(offx+(stepx*(i&3)),offy+7+((i&4)>>2)*stepy,
text, COLOR_ON,2+8+16);
```

Les bouts de texte écrits s'appellent des « quotes », nous cherchons dans le fichier **game.c** celles qui sont définies afin de pouvoir faire les nôtres. Ligne 394, nous voyons ceci :

```
short user_quote_time[MAXUSERQUOTES];
char  user_quote[MAXUSERQUOTES][128];
```

En recherchant le **user_quote_time**, nous tombons sur la fonction :

```
void operatefta(void)
```

Puis plus loin, ligne 9571 :

```
for(i=0;i<MAXUSERQUOTES;i++)
     if (user_quote_time[i])
     {
         user_quote_time[i]--;
         if (!user_quote_time[i]) pub = NUMPAGES;
     }
```

Nous pouvons en déduire que décrémenter la valeur de **quote_time** permet de réduire le temps d'affichage, ce qui veut dire que la boucle d'évènement passe par un *timer*.

Du coup, nous allons créer une variable globale (oui je sais, je m'adapte au code de duke3d très vite !), que l'on appellera **tick_display_time**, ligne 96 :

```
extern int tick_display_time = 180;
```

180 correspond au temps utilisé pour les autres textes utilisateur à afficher par défaut, parfait !

Du coup dans la fonction **domovethings()** ligne 9517, nous décrémentons cette valeur :

```
if (tick_display_time>0) tick_display_time--;
```

On teste en créant une fonction **myowntext()** qui contient ceci :

```
void myowntext(void)
{
  if (tick_display_time>0) {
    gametext(10,20,"Coucou!", 0,2+8+16);
  }
}
```

Et dans la fonction **displayrest()**, ligne 2841, nous appelons cette fonction :

```
    myowntext();
    coolgaugetext(screenpeek);
    operatefta();
```

Nous recompilons et lançons le jeu, bingo comme le montre la figure 2 !

Fig. 2 : Premier affichage dans le jeu.

Maintenant nous souhaitons passer à l'étape au-dessus, et afficher un texte de notre choix lorsque l'on tue un ennemi. Après une séance de **grep** et **printf**, il se trouve que le code exécuté se trouve dans **gamedef.c** ligne 2472. Du coup, nous allons créer une deuxième variable globale afin de pouvoir changer le texte. Duke3d n'étant pas *multithreadé*, pas besoin de créer des verrous ! Dans **game.c**, au début, en dessous de notre **tick_display_time**, nous ajoutons :

```
extern char *displaytext = "";
```

Et nous supprimons notre « coucou » pour mettre **displaytext** à la place. Nous en profitons aussi pour initialiser **tick_display_time** à **0** au lieu de **180** :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
void myowntext(void)
{
  if (tick_display_time>0) {
    gametext(10,20,displaytext, 0,2+8+16);
  }
}
```

Et dans **gamedef.c**, ligne 44 nous pouvons mettre nos variables afin de récupérer leur adresse :

```
int tick_display_time;
char *displaytext;
```

Il ne nous reste maintenant plus qu'à utiliser nos deux variables :

```
        case 88:
            insptr++;
            tick_display_time = 180;
            displaytext = "Actor killed! w00t\n";
            ps[g_p].actors_killed += *insptr;
            hittype[g_i].actorstayput = -1;
            insptr++;
            break;
```

Et à tester (voir figure 3) !

Fig.3 : Affichage d'un texte au choix.

Oh yeah !

8. INTÉGRATION AVEC LE FICHIER CRÉÉ PAR VALGRIND

Voilà, nous avons les morceaux, il suffit de les coller entre eux afin de lire le fichier et d'afficher le nom de l'appel système dès qu'un ennemi sera tué !

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ACTUELLEMENT DISPONIBLE !
MISC n°92

EXPLORATION DES TECHNIQUES DE
REVERSE ENGINEERING

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

Fig. 4 : Appels systèmes effectués par vstrace sur la commande ls.

Nous allons d'abord rajouter le paramètre **-vstrace <fichier_vstrace>** qui contient la liste des appels système crées par Valgrind :

```
if (stricmp(c, "-vstrace") == 0)
{
  i++;
  c = argv[i];
  fp = fopen(c, "r");
  if (!fp) {
    fprintf(stderr, "Cannot read vstrace file!\n");
    return -1;
  }
  fseek(fp, 0, SEEK_END);
  vstrace_size = ftell(fp);
  fseek(fp, 0, SEEK_SET);
  vstrace_buf = malloc(vstrace_size + 1);
  fread(vstrace_buf, vstrace_size, 1, fp);
  vstrace_buf[vstrace_size] = '\0';
  fclose(fp);

  i++;

    continue;
}
```

Vous noterez que nous ne faisons qu'enregistrer le contenu de ce fichier dans une chaîne de caractères, ce qui nous permet ensuite de créer des *tokens* à chaque fois que l'on rencontrera le caractère de retour de ligne afin d'aller à l'appel système suivant.

On ajoute dans **game.c** une variable globale qui contient la totalité du fichier crée par Valgrind :

```
extern char *vstrace_buf =
NULL;
```

On crée dans **gamebuf.c** quelques variables utiles :

```
char *vstrace_buf;
int begin_parse_buf = 1;
int done_reading = 0;
```

Et il ne reste plus qu'à finaliser le code pour gérer l'affichage :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
        case 88:
            insptr++;
            if (!done_reading) {
              tick_display_time = 180;
              if (begin_parse_buf) {
   displaytext = strtok(vstrace_buf, "\n");
                begin_parse_buf = 0;
                if (!displaytext) {
                  displaytext = "Error reading
vstrace file!";
                }
              } else {
                displaytext = strtok(NULL, "\n");
   if (!displaytext) {
                  displaytext = "Done reading
syscalls!";
                  done_reading = 1;
                }
              }
            }
            ps[g_p].actors_killed += *insptr;
            hittype[g_i].actorstayput = -1;
            insptr++;
            break;
```

Ce que nous faisons ici, c'est de vérifier que nous n'avons pas fini de lire le fichier, et nous utilisons la variable **tick_display_time** pour afficher le message pendant **180** centisecondes de l'appel système, ou indiquons que nous avons fini de lire le fichier.

S'il y a un problème de lecture du fichier d'origine, nous l'affichons ici de cette manière.

Ce qui nous permet du coup de pouvoir suivre l'ensemble des appels systèmes qui a été effectué par **vstrace** sur la commande **ls** (voir figure 4).

Avouez que cela est nettement plus agréable n'est-ce pas ?

9. UNE PETITE ÉNIGME AVANT DE SE DIRE AU REVOIR !

Mettons côte à côte les appels systèmes tracés depuis Valgrind, avec ceux de **strace** lorsque l'on exécute la commande **ls**.

Nous ne prenons ici que les vingt premiers appels afin de ne pas utiliser trop d'encre (voir tableau ci-contre).

Pour la partie Valgrind, vous aurez bien sûr remarqué que les noms étaient préfixés par **sys_**, car nous avons pris directement les noms tels qu'ils apparaissent dans le noyau Linux.

valgrind	strace
sys_brk	execve
sys_newuname	brk
sys_open	mmap
sys_read	access
sys_newfstat	open
sys_mmap	fstat
sys_mmap	mmap
sys_mprotect	close
sys_close	open
sys_access	read
sys_open	fstat
sys_close	mmap
sys_close	mprotect
sys_arch_prctl	mmap
sys_mprotect	mmap
sys_munmap	close
sys_set_tid_address	open
sys_set_robust_list	read
sys_rt_sigaction	fstat
sys_rt_sigprocmask	mmap

Pourquoi les appels ne se font pas dans le même ordre ? Pourquoi ce ne sont pas toujours les mêmes appels systèmes utilisés ? Un petit élément de réponse se situe dans le premier appel système côté **strace**...

CONCLUSION

C'est en faisant n'importe quoi que l'on fait n'importe quoi ! J'espère que vous serez motivé pour commencer à explorer l'univers Valgrind qui est fascinant et permet d'apprendre un tas de trucs super utiles. J'ai souhaité joindre l'utile à l'agréable à travers une exploration des appels systèmes que vous n'auriez jamais regardés un par un en dehors peut-être d'une session de débugging, hein ? Avouez ! ■

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

LE TEST DE PETER

ÉTIENNE DUBLÉ
[Ingénieur de Recherche CNRS au LIG]
Relectures par Henry-Joseph Audéoud et Narek Davtyan

MOTS-CLÉS : SYSTÈME BOOTABLE, INIT, INITRAMFS, LVM, HACKING, SÉRIE US

On a tous un ou deux amis bizarres. C'est le cas par exemple de mon ami Peter. Dès qu'il trouve quelque chose de nouveau, il ne peut s'empêcher de faire un test. Il teste donc les spécialités culinaires, les paradigmes de programmation, les techniques de drague... Il teste absolument tout. Avant-hier, quand je l'ai aperçu à la terrasse d'un café, je me doutais donc bien qu'il était en train de tester quelque chose...

Mardi, je suis sorti du boulot un peu moins en retard que d'habitude. En passant devant EVE, une petite cafét' étudiante sur le campus de Grenoble, j'ai repéré Peter en train de visionner une vidéo sur son ordi portable. Il faut dire qu'il a du temps en ce moment, il s'est fait licencier récemment, suite à une histoire de produit pas assez testé (c'est en tout cas ce qu'il m'en a dit). Bref, je me suis dit que j'allais passer voir ce qu'il mijotait.

« Tiens salut » a-t-il dit en m'apercevant. « Je suis en train de tester quelque chose. Tu connais cette série US ? » a-t-il enchaîné en tournant l'écran vers moi. Je lui ai répondu que oui. Je ne suis pas un grand fan, mais ma femme aime bien. Et il m'a proposé de visionner l'extrait qui lui posait problème.

1. L'ÉPISODE

Dans cet épisode, l'agent Devon, du bureau NCIS de Chicago, est soupçonné d'être une « taupe » à la solde d'une puissance étrangère. Plutôt que de l'arrêter, Gibbs (le chef d'équipe) décide de profiter de cette situation : Devon ne sait pas qu'on le soupçonne. Si on peut trafiquer son matériel pour intercepter ses communications, on pourrait obtenir des informations importantes. Voire même le

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

manipuler et l'entraîner vers de fausses pistes. McGee (l'expert technique) et DiNozzo (l'agent de terrain), travestis en agents d'entretien, pénètrent alors dans son bureau pendant la pause déjeuner. À ce moment-là, on voit que l'ordi de Devon est resté allumé. McGee l'arrête et le fait redémarrer sur une clé USB qu'il a sans doute préparée au préalable. On voit alors le bureau qui s'affiche puis une barre de progression, mais, déjà, l'agent Devon se dirige de nouveau vers le bâtiment. DiNozzo en sort et le croise devant la porte d'entrée. Pour gagner quelques secondes, il l'interpelle au sujet de la jolie collègue qui vient de passer. Mais l'agent Devon y prête peu d'attention et se dirige vers l'ascenseur. On voit alors la barre de progression qui se termine, McGee qui arrache aussitôt la clé USB, sort du bureau et referme la porte, juste au moment où l'agent Devon sort de l'ascenseur.

Peter positionne ensuite le curseur de la vidéo un plus loin, à un moment où l'agent Devon utilise à nouveau son ordi. On comprend alors que la ruse a fonctionné : il n'a visiblement remarqué aucun changement sur son système. En répondant à quelques e-mails sur son ordi trafiqué, il révèle rapidement plusieurs de ses complices.

2. « IMPOSSIBLE » OU JUSTE « INVRAISEMBLABLE » ?

- *Tu as vu ? Tu crois que c'est possible ? me demande alors Peter.*
- *De réinstaller un OS et que la personne ne se rende compte de rien ? Ben non, mais c'est qu'une série, tu sais...*
- *Il n'y a pas que ça. Le gars, McGee, il a bien rebooté sur sa clé, au début de la séquence. Par contre, pour une install normale il aurait dû rebooter une deuxième fois à la fin, pour démarrer sur l'OS fraîchement réinstallé. Et là il n'a rien fait, il a juste débranché la clé USB et il est parti en laissant le système tourner.*
- *Effectivement, mais pour le public moyen ça passe inaperçu ce genre de détail.*
- *Peut-être. Mais nous on n'est pas le public moyen. Et à bien y réfléchir, je crois qu'on pourrait très bien programmer un système d'install qui suive ce comportement. C'est ça que je suis en train de tester.*
- *Ah. Et tu as aussi quelque chose pour expliquer que la réinstallation d'OS passe complètement inaperçue ?*
- *Ben ouais. Par exemple l'OS du gars pourrait être un partage NFS (Network File System). Si c'est le cas, le fait de booter temporairement sur une clé USB en local ne change rien aux données, puisqu'elles sont hébergées ailleurs.*
- *Du NFS ? Ah ouais pourquoi pas. Et c'est vrai que les gars sont censés bosser dans une agence gouvernementale avec pas mal de personnel... Donc il est tout à fait possible qu'il y ait un système de ce genre pour centraliser les données et le boulot d'administration qui va avec.*
- *Tout à fait.*
- *Mais si c'est un partage NFS et que rien n'est modifié sur l'OS au final, comment tu expliques que le système soit trafiqué ?*
- *Eh bien euh, il y a deux solutions. Le plus simple serait que McGee bidouille directement des trucs sur le serveur NFS distant. Mais dans ce cas, il aurait pu le faire depuis son bureau tranquillou, plutôt que de monter toute cette opération pour s'introduire dans le bureau de Devon. Le souci à mon avis, c'est que le serveur central est sans doute hautement sécurisé et surveillé : si McGee tente de s'y introduire et se fait repérer, les complices de l'agent Devon pourraient en avoir connaissance. À l'inverse, le poste local de Devon est sûrement beaucoup plus vulnérable.*
- *Je vois. Mais tu n'as pas répondu à ma question...*
- *Et bien, si on peut bidouiller la procédure de boot en local, on peut par exemple empiler le partage NFS avec une autre couche de filesystem qui contiendrait les hacks en question. Je pourrais te simuler ça sur mon portable. Mais si tu n'as pas le background ça va te paraître un peu nébuleux...*

3. LES ÉTAPES DE DÉMARRAGE D'UN OS

Peter se met visiblement à réfléchir sur la façon dont il va pouvoir m'expliquer la chose. Puis il reprend :

- *As-tu une vision claire des grandes étapes de démarrage d'un OS GNU/Linux ?* **[1]**
- *Oui je crois. Quand on met la machine sous tension, le firmware s'exécute (BIOS ou UEFI sur un PC), il passe la main au bootloader, qui lance le noyau Linux. Celui-ci s'initialise, puis monte la racine du système de fichiers et lance le premier processus (ce qu'on appelle un « système d'init »). Ce premier processus est l'ancêtre de tous ceux qui seront créés par la suite.*
- *C'est à peu près ça, mais ce que tu me décris, c'est le cas le plus simple. En réalité on ne peut pas gérer tous les cas de cette manière. Par exemple, imagine le cas suivant :*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ton système de fichiers est un btrfs avec du RAID logiciel sur 2 disques en dessous. Le bootloader est déjà censé trouver le fichier du noyau linux (`/boot/vmlinuz-<version>` ou quelque chose du genre) pour pouvoir le lancer. S'il y arrive dans ce genre d'environnement, il est balaise. Grub2 y arriverait peut-être, mais je ne suis pas sûr. C'est pour ça qu'on place parfois `/boot` sur une autre partition, dans un environnement moins complexe (par exemple une partition physique en FAT). Ensuite, si on admet que le bootloader a pu lancer le noyau, celui-ci doit par contre forcément monter le système de fichier racine. Il y a alors 2 approches. La première est de considérer que le noyau doit savoir gérer ce genre de cas complexe. On implémente donc toutes les technologies concernées (ici btrfs et RAID) dans le noyau. Mais ensuite, comment aller chercher un module `/lib/modules/.../btrfs.ko` sur une partition en btrfs ?? Il faut donc compiler ces différentes technos en dur dans le noyau, et non pas sous la forme de modules. Et l'éditeur de la distribution fournissant en général le même noyau pour tous, il faut aussi inclure tout ce que les autres utilisent, peut-être du NFS, du XFS, du je-ne-sais-quoi et gérer toutes les combinaisons possibles des différentes technos. Tu vois le souci. Donc, depuis un bout de temps, on préfère une autre approche, en tout cas dans les distributions grand public. Cette deuxième approche consiste à intercaler une étape supplémentaire entre le noyau et le système d'init. As-tu remarqué les fichiers `initrd.img-<version>` dans `/boot` ?

- *Euh oui... Cela concerne donc l'« étape supplémentaire » dont tu parles ?*

- *Oui. On appelle ça un initramfs. En fait il s'agit d'une archive compressée au format `cpio` [2] contenant une arborescence de fichiers. Regarde :*

```
$ mkdir /tmp/initramfs
$ cd /tmp/initramfs
$ zcat /boot/initrd.img-4.4.0-36-generic | cpio -id
166243 blocs
$ ls
bin  conf  etc  init  lib  run  sbin  scripts  usr  var
```

Tu vois, le contenu de cette archive ressemble à un mini-système, avec une arborescence classique. Et ce mini-système à beau être mini, il sait faire des choses. Par exemple :

```
$ find . -name btrfs.ko
lib/modules/4.4.0-36-generic/kernel/fs/btrfs/btrfs.ko
```

Ce mini-système contient, entre autres, le module noyau nécessaire pour gérer un système de fichier btrfs.

- *Intéressant... Et donc, tu dis que ce mini-système s'intercale entre le noyau et le système d'init ?*

- *Oui, grosso modo. Voilà comment ça se passe. En plus du noyau, le bootloader charge en RAM cette archive compressée, à une adresse donnée. Et quand il lance le noyau, il lui passe cette adresse via un registre CPU. Le noyau est donc capable de décompresser l'archive et se retrouve avec un système de fichiers stocké en RAM. Au final, après son initialisation, c'est à ce mini-système que le noyau va passer la main, en exécutant le fichier `/init`.*

- *Je vois... Et donc si on a du btrfs et du RAID logiciel, ce script pourra charger les modules correspondants. Il sera alors capable de trouver notre système de fichiers final, le monter et exécuter le système d'init. C'est bien ça ?*

- *Oui tout à fait.*

- *Dans ce cas, il y a un truc qui me chiffonne... Si j'ai bien compris, on a juste déplacé la complexité depuis le noyau linux vers cet initramfs, non ? Si c'est l'initramfs qui doit savoir gérer toutes les technos et toutes les combinaisons possibles dont tu parlais tout à l'heure, alors le noyau est effectivement plus léger, mais cet initramfs doit être énorme et super complexe !*

- *Non, pas du tout. Premièrement, il est beaucoup plus facile d'implémenter quelque chose en espace utilisateur plutôt que dans le noyau, donc la complexité est largement moindre. Et en plus, dans les distributions modernes, l'initramfs est automatiquement adapté à ton environnement. Par exemple, si tu installes le paquet `lvm2` sur une Ubuntu, l'archive compressée est automatiquement regénérée (via un script qui s'appelle `update-initramfs`) pour que l'initramfs puisse gérer cette nouvelle fonctionnalité. Au final, la taille et la complexité de l'initramfs sont donc relativement maîtrisées.*

- *Je vois.*

- *Dernière chose à ce sujet : tu as vu que le mini-système à beau être mini, il présente une arborescence très classique. L'avantage étant que, pour le noyau, démarrer le système final ou démarrer le mini-système, c'est la même chose. Et on pourrait éventuellement aller plus loin, enchaîner tour à tour plusieurs mini-systèmes de ce type avant d'arriver au système final...*

- *OK... Je crois que je commence à voir où tu veux en venir, et le lien avec notre épisode de série US. Tu penses qu'en rajoutant une étape de ce type dans le déroulement du boot, on pourrait introduire les hacks nécessaires ?*

- *Humm... non, pas exactement. Mais c'était bien tenté. En fait, l'initramfs de l'agent Devon est chargé de deux choses : d'abord, il doit monter le système de fichiers final, c'est-à-dire le partage NFS, et ensuite lancer le*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

système d'init de cet OS distant. Or, pour introduire les hacks tels que je les imagine, on va devoir bidouiller quelque chose entre ces 2 actions. Donc a priori la seule solution pour ajouter du code à cet endroit-là est de modifier l'initramfs existant.

4. DE L'INITRAMFS AU SYSTÈME D'INIT FINAL

- *Bon, je te montre.*

Pour un test plus réaliste, il me faudrait 2 machines, mais là je n'ai que mon portable. On va donc simuler à la fois le serveur NFS central et la machine locale de l'agent Devon. Commençons par le serveur NFS central :

```
$ su
# apt-get install nfs-kernel-server
# cd /tmp
# debootstrap --variant=minbase xenial os-devon
# ls os-devon/
bin  boot  dev  ...  tmp  usr  var
# echo "/tmp/os-devon *(rw)" >>/etc/exports
# exportfs -a
```

Voilà, on a préparé un OS Debian dans /tmp/os-devon et on l'a rendu accessible par NFS.

Passons à la partie « machine locale ». On considère donc à présent qu'on est sur le PC de l'agent Devon. Pour y croire un peu plus, on peut ajouter une ligne dans **/etc/hosts** :

```
# echo '127.0.0.1 nfs-norfolk' >> /etc/hosts
```

Maintenant, imagine que ce PC est en train de démarrer. Le noyau est chargé, il a lancé le script **/init** *de l'initramfs. Celui-ci doit alors monter le partage NFS :*

```
# mkdir -p mnt/os-devon
# mount -t nfs nfs-norfolk:/tmp/os-devon
mnt/os-devon
# ls mnt/os-devon/
bin  boot  dev  ...  tmp  usr  var
```

À ce stade, dans son fonctionnement normal, l'initramfs est censé passer la main à l'OS hébergé à distance, en faisant quelque chose comme :

```
# cd mnt/os-devon
# exec chroot . sbin/init
```

- *?? Qu'est-ce que ça fait, ce* **exec chroot** *machin ??*

- *Analysons la chose en commençant par la fin. Pour démarrer un OS, il faut lancer le premier processus, celui qui aura le PID 1, et ce processus sera chargé de lancer tous les autres. C'est ce qu'on appelle le « système d'init », et d'après ce que tu m'as dit tout à l'heure, tu connais déjà. Comme tu peux t'en douter, ce système d'init est démarré en exécutant /sbin/init. En fait, comme il y a plusieurs implémentations possibles, il s'agit d'un lien symbolique :*

```
# ls -l /sbin/init
lrwxrwxrwx 1 root root 20 sept. 29 04:01 /sbin/
init -> /lib/systemd/systemd
```

Tu vois, sur mon système c'est **systemd** *qui se cache derrière.*

- *OK... Tout s'éclaire.*

- *Revenons à notre cas d'étude, avec le système distant monté par NFS. Le problème à ce stade, c'est qu'on n'est pas dans un environnement adéquat pour lancer un système d'init. En effet, les chemins des fichiers ne sont pas ceux attendus : à la place de* **/sbin/init**, *on a* **[...]/os-devon/sbin/init**, *à la place de* **/bin/sh**, *on a* **[...]/os-devon/bin/sh**, *le répertoire* **/lib** *est en réalité* **[...]/os-devon/lib**, *etc. Il est clair que le programme d'init ne va pas fonctionner correctement dans ces conditions ! En fait, ce qu'il faudrait, c'est déplacer la racine du système de fichiers, pour que le symbole* **/** *pointe dorénavant vers le répertoire* **[...]/os-devon**.

- *Pourquoi pas... si je comprends bien, ça reviendrait à changer de référentiel sur un graphique... Mais c'est possible un truc pareil ?*

- *Ben oui, t'es mauvais en anglais ou quoi ? Dans l'instruction* **chroot**, *il y a root ce qui signifie racine, et ch c'est pour change !*

- *Ah oui, j'aurais pu deviner :)*

- C'est clair. Tu ferais bien de reprendre un café. Donc l'instruction **chroot** *permet de changer la racine du système de fichiers. Le premier paramètre correspond à la nouvelle racine, ici le répertoire courant. Le deuxième correspond à l'exécutable à lancer dans ce nouvel « environnement ».*

- *Ça va changer la racine de tous les processus lancés à partir de maintenant ?*

- *Heu... C'est un peu plus subtil que ça. En fait, la notion de racine du système de fichiers, c'est une notion propre à chaque processus. Mais quand un processus donné a besoin de créer un fils, il fait un appel système* **fork()**, *et le noyau duplique ce processus. Cet « attribut » se retrouve donc dupliqué, et le processus fils se retrouve avec la même racine que son père.*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- *Je vois... Et comme c'est le processus d'init qu'on manipule, s'agissant du premier processus, tous ceux créés par la suite hériteront de proche en proche de ce même attribut. C'est bien ça ?*

- *Oui. Je vois que tu as bien compris. Donc passons au* **exec**. *Tu sais à quoi ça sert ?*

- *Non, pas plus.*

- *En fait, quand tu lances une commande, dans un script shell, cela provoque la création d'un processus fils chargé d'exécuter cette commande. Le père se met alors en attente, et quand la commande est terminée (cela signifie donc que le fils s'est arrêté) il reprend la main pour poursuivre le script. Si tu préfixes la commande par* **exec**, *ce comportement est modifié. Aucun processus fils n'est créé, et c'est le père qui prend en charge la commande. Si on va au bout du raisonnement, on s'aperçoit que* **exec** *est toujours la dernière commande exécutée par le script. En effet, quand le père prend en charge la commande, il n'y a plus personne pour s'occuper du script, donc personne pour reprendre la main après le* **exec**. *C'est définitif.*

- *C'est bien bizarre comme instruction ! Et à quoi ça sert ??*

- *Et bien dans notre cas, le fils serait le système d'init. Ce processus étant censé vivre jusqu'à l'arrêt de la machine, il n'est pas utile de garder un processus père en attente pour rien.*

- *Donc c'est juste une question d'économie de ressources ?*

- *Non, il y a une autre raison, plus importante d'ailleurs. Pour bien comprendre, il faut se mettre à la place du noyau. Pour lui, un initramfs ou un système final, c'est la même chose : le script* **/init** *de l'initramfs, c'est déjà un « système d'init ». Et comme c'est son premier bébé, il le lance avec le PID* **1**. *Maintenant, à la fin de ce script, on doit passer la main au système d'init final. Si on préfixe avec* **exec**, *comme on reste sur le processus père, on reste avec ce même PID* **1**. *Sinon, s'agissant d'un fils, le système d'init final aurait un PID différent de* **1**.

- *Et alors ? Où est le problème ? Le fait d'avoir un PID différent de* **1** *pourrait déstabiliser le système d'init ??*

- *Oui, aussi étonnant que cela puisse te paraître... Cela vient du fait que la commande* **/sbin/init** *est utilisée pour 2 choses différentes. Soit elle assure son rôle de système d'init, soit elle contrôle ce système d'init. As-tu déjà tapé* **init 6** *pour rebooter ta machine ?*

- *Euh oui... C'est le même exécutable* **/sbin/init** *qui est appelé dans ce cas ?*

- *Oui, bien sûr ! Et tu vois bien que dans ce cas on ne veut pas lancer une autre instance du système d'init, on veut juste le contrôler. On a donc 2 fonctionnalités complètement différentes à gérer dans ce même exécutable. Et est-ce que tu devines comment cet exécutable détermine la fonctionnalité à assurer ?*

- *Ne me dis pas qu'il vérifie son numéro de PID ??*

- *Si !! Je suis d'accord avec toi que c'est un peu étrange et pas très intuitif, mais j'imagine qu'on a gardé ce fonctionnement pour des raisons historiques... Toujours est-il que, de ce fait, l'emploi du* **exec** *pour conserver le PID* **1** *entre l'initramfs et le système d'init final est primordial. Voilà, tu sais tout sur la commande* **exec chroot . sbin/init** *que j'ai introduite tout à l'heure.*

- *Merci pour les explications. Si je récapitule, cette commande permet donc de lancer l'OS sur le partage NFS, à la fin de l'initramfs. C'est bien ça ?*

- *Oui. En tout cas on peut imaginer que le système de l'agent Devon fonctionnait de cette manière, avant qu'il soit piraté.*

5. UNE UNION DE FILESYSTEMS

- *Je vois... Et donc, comment est-ce que tu comptais modifier ce fonctionnement ?*

- *Et bien, juste avant de passer la main au système d'init final, on pourrait monter une union avec nos hacks stockés sur la clé USB, via un montage avec le filesystem adéquat.*

- *Une union ? C'est-à-dire ?*

- *Le principe d'une union, c'est grosso-modo de pouvoir combiner plusieurs systèmes de fichiers au niveau d'un seul point de montage. Je te montre.*

```
# cd /tmp
# mkdir -p hacks unionwork mnt/union
# modprobe overlay
# mount -t overlay -o
upperdir=hacks,lowerdir=mnt/os-
devon,workdir=unionwork none mnt/union
```

Voilà, notre union est montée dans **mnt/union**. *J'ai utilisé le système de fichiers* **overlay** **[3]** *pour la construire. Comme tu le vois dans les options, on a construit l'union des répertoires* **hacks** *et* **mnt/os-devon**. *Cela signifie que tous les fichiers et répertoires que l'on trouve dans l'un ou l'autre de ces répertoires seront visibles dans notre point de montage* **mnt/union**. *Bien évidemment, comme* **hacks** *est vide pour l'instant, dans* **mnt/union** *on retrouve ni plus ni moins que le contenu de* **mnt/os-devon** :

```
# ls mnt/union
bin  boot  dev  ...  tmp  usr  var
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Mais si on ajoute des choses dans **hacks**...

```
# touch hacks/mytest
# ls mnt/union
bin  boot  dev  ...  mytest ... tmp  usr  var
```

... les modifs apparaissent dans l'union.

- *Et en cas de conflit ? Si on trouve le même chemin de fichier dans* **hacks** *et* **mnt/os-devon** ?

- *J'allais y venir. La priorité est sur la « couche supérieure » (l'option de montage* **upperdir**) *donc* **hacks**. *Regarde :*

```
# mkdir hacks/bin
# echo hacked >> hacks/bin/bash
# cat mnt/union/bin/bash
hacked
```

- *OK... Par contre il y a encore des trucs pas clairs dans ta ligne de commandes* **mount**... *L'option de montage* **workdir=unionwork**, *tu n'en as pas parlé... Et le* **none**, *avant le point de montage, à quoi ça sert ?*

- *Oh ça ce n'est pas très important. Le* **workdir** *c'est, comme son nom l'indique, un répertoire de travail pour le montage* **overlay**. *Si je me souviens bien, il en a besoin pour implémenter certaines opérations de manière atomique. Et le* **none** *est là uniquement pour respecter la syntaxe du* **mount**. *Parce qu'historiquement, on montait toujours un device sur un répertoire. Alors que maintenant, il existe plusieurs systèmes de fichiers de ce genre pour lesquels on n'a pas de device associé. Donc on met juste un mot quelconque à cet emplacement pour respecter la syntaxe.*

  Bon, je ne vais pas m'étendre davantage sur ce sujet, mais sache que ça marche plutôt pas mal, ce genre de filesystem. Sur l'union, tu peux modifier les fichiers à loisir, les modifications seront reportées sur la couche **upperdir**, *donc* **hacks**. *La couche* **lowerdir** (ici **mnt/os-devon**) *restera intacte, quoi que tu fasses. Pour des cas pathologiques (suppression dans l'union d'un fichier de* **mnt/os-devon**), *le filesystem crée même des fichiers spéciaux dans la branche* **hacks**. *Tu pourras essayer si ça t'intéresse.*

  En tout cas, tu vois ce que j'imaginais : plutôt que de lancer le système d'init final sur **mnt/os-devon**, *on le lance sur l'union, ça nous permet d'inclure toute une couche de hacks pour pirater la machine.*

- *Oui je vois bien. Par contre, d'où tu sors cette techno de derrière les fagots ? Je veux dire, OK ça a l'air sympa et tout, mais à quoi ça peut bien servir ? C'est juste pour hacker des OS au démarrage ?? Il y a un gars qui s'est tapé des jours et des jours de programmation noyau pour ça ??*

- *Non, bien sûr que non ! En fait, ça existe depuis un bout de temps ! On a introduit ça pour les live-CDs : en faisant une union, l'OS pouvait combiner tous les fichiers en lecture seule sur le CD avec une zone de stockage en RAM, donc accessible en écriture. Quand on modifiait ou supprimait un fichier sur l'union, la modif était enregistrée côté RAM. Ainsi, on avait vraiment l'impression d'avoir booté un média en lecture/écriture...*

- *OK, je vois le principe... Mais j'imagine que la techno a presque disparu avec l'omniprésence des clés USB. Plus besoin d'union sur un média accessible en lecture/écriture...*

- *Détrompe-toi. On a gardé grosso modo le même principe. La clé USB a beau être réinscriptible, on l'utilise en grande partie en lecture seule. Il y a deux raisons à cela, je crois. La première, c'est de fabriquer des images compatibles à la fois avec les CDs et les clés USB. La deuxième, c'est que les systèmes de fichiers en lecture seule peuvent être beaucoup mieux compressés.*

Peter commença alors à griffonner un schéma au dos de la note de son café (voir figure 1).

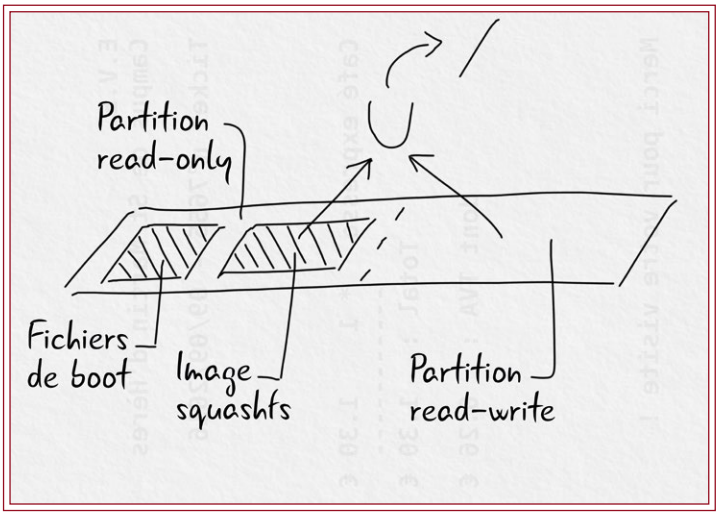


Fig. 1 : Structure usuelle d'un LiveUSB (au dos de la note de chez EVE).

Puis il reprit :

- *Tu vois, on utilise un système de fichiers compressé,* **squashfs** *en général, pour stocker autant de fichiers que possible sur la clé USB. Et on peut faire une union avec un autre système de fichiers, stocké sur une petite partition qu'on utilisera en lecture/écriture. Ainsi, contrairement au CD qui enregistre en RAM, les modifs sont cette fois-ci stockées de manière pérenne sur cette partition, et donc tu les retrouveras si tu redémarres l'OS.*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- *Je vois. N'empêche que la moindre clé USB fait 8 Gio de nos jours... Est-il nécessaire de garder une structure aussi complexe ? Et concernant la compatibilité avec les CDs, l'argument me paraît discutable... Qui aujourd'hui grave un CD pour booter un OS ?*

- *Tu as sans doute raison. Surtout que cette complexité a des inconvénients, par exemple on ne peut pas mettre à jour le noyau ou le bootloader sur ce genre de LiveUSB. Mais bon. Il faut du temps pour que les choses changent.*

6. MIGRATION DE DISQUE À DISQUE

Avant même la fin de cette phrase, Peter était visiblement reparti dans ses pensées. Il lançait de temps en temps une phrase explicative, mais je n'étais pas sûr qu'elles m'étaient destinées. J'avais plutôt l'impression qu'il réfléchissait tout haut. À un moment, il finit quand même par se tourner vers moi.

- *Tu vois ce que je veux dire ?*

- *Euh... à vrai dire...*

- *Reprenons. Tu es d'accord qu'en modifiant l'initramfs, on peut introduire ce système d'union et hacker l'OS sans que l'agent s'en aperçoive ?*

- *Oui, ça me semble plausible.*

- *OK, donc passons à l'autre question. Dans l'épisode, l'ordi de l'agent Devon démarre sur la clé USB, jusqu'à afficher un bureau, puis une barre de progression. À la fin de cette barre de progression, l'agent McGee débranche la clé USB, à chaud, et laisse tourner l'OS. Ça c'est inhabituel.*

- *C'est vrai. On dirait un genre d'install sans redémarrage.*

- *Oui, grosso modo. En fait, rien ne nous empêche de rajouter encore quelques instructions dans notre script initramfs, de façon à copier l'archive initramfs modifiée sur la machine locale. Car sinon, au premier reboot sans la clé USB, l'OS retrouvera son comportement initial.*

- *Oui c'est clair.*

- *Le souci, c'est que si l'OS tourne effectivement sur une union, alors on a la majeure partie de l'OS sur le serveur NFS distant, mais on a aussi un répertoire* **`hacks`** *en local, sur la clé USB. Il faudra donc également copier ce répertoire sur la machine locale, pour que notre bidouille résiste au prochain reboot. Et a priori, ce répertoire* **`hacks`** *doit bien peser quelques centaines de mégas, au bas mot.*

- *Tant que ça ?*

- *Écoute, d'après l'épisode, dans les programmes hackés il y a au moins le client mail, et ce n'était visiblement pas un programme très léger... En plus, j'imagine que McGee n'a pas pris de risques et a compilé tous ces programmes en* **`static`**. *Sinon, il prend le risque d'incompatibilités avec les librairies partagées stockées côté NFS, et dont la version n'est peut-être pas celle attendue...*

- *Je vois... Donc oui pourquoi pas, quelques centaines de mégas, au moins. Ça expliquerait la barre de progression.*

- *Oui. Mais cette idée de copie au moment de l'initramfs ne me semble pas du tout cohérente avec ce qu'on voit dans l'épisode. Parce que la barre de progression apparaît vraiment tard, après l'affichage du bureau. Et il y a même autre chose qui me chiffonne. On a vu qu'avec un système d'union, les modifs sont stockées dans la « couche supérieure ». Cela veut dire que si l'agent Devon crée un nouveau document LibreOffice par exemple, il sera stocké quelque part dans* **`hacks`**. S'il reçoit un mail, c'est pareil, il sera stocké dans **`hacks`**. *Donc au final, pas de souci pour copier la version initiale de* **`hacks`** *au boot, mais ensuite ? L'OS va continuer à faire des modifs dans le répertoire* **`hacks`** *de la clé USB, pas sur la copie locale ! Et si on retire la clé USB, ça va clairement planter !*

- *C'est clair... Mais attends ! Je crois que ça me rappelle un truc ton histoire... un outil justement capable de « migrer à chaud » l'OS depuis la clé USB vers le disque local... comment il s'appelle déjà ?...* **`debootstick`**, *oui c'est ça ! On pourrait regarder comment il fonctionne !*

- **`debootstick`** ?

- *Oui, c'est un outil qui permet de créer des OS bootables. L'auteur a écrit un article dans GNU/Linux Magazine récemment* **[4]**... *Ce n'était pas le cœur de l'article, mais il a parlé de* **`debootstick`** *dans l'intro. Ça avait l'air sympa, alors je l'ai testé pour voir. Et il me semble bien avoir vu une option de ce type dans la page de manuel.*

- *C'est open source ?*

- *Je crois oui...*

Après une petite recherche **Google**, Peter s'était mis à explorer le dépôt de **`debootstick`** sur **GitHub [5]**. Et avec une rapidité qui m'étonnera toujours, il avait déjà trouvé le bout de code qui l'intéressait. Je vais finir par croire que les neurones de ce gars sont organisés en table de hashage...

- *Mais oui ! Évidemment !! C'est ça la solution, il suffit de jouer sur les volumes physiques LVM !!*

- *Si tu le dis. Et c'est quoi LVM ?*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

RMLL
SAINT-ÉTIENNE

17ème | Libre et Change

Rencontres Mondiales du Logiciel Libre

01- 07 juillet 2017

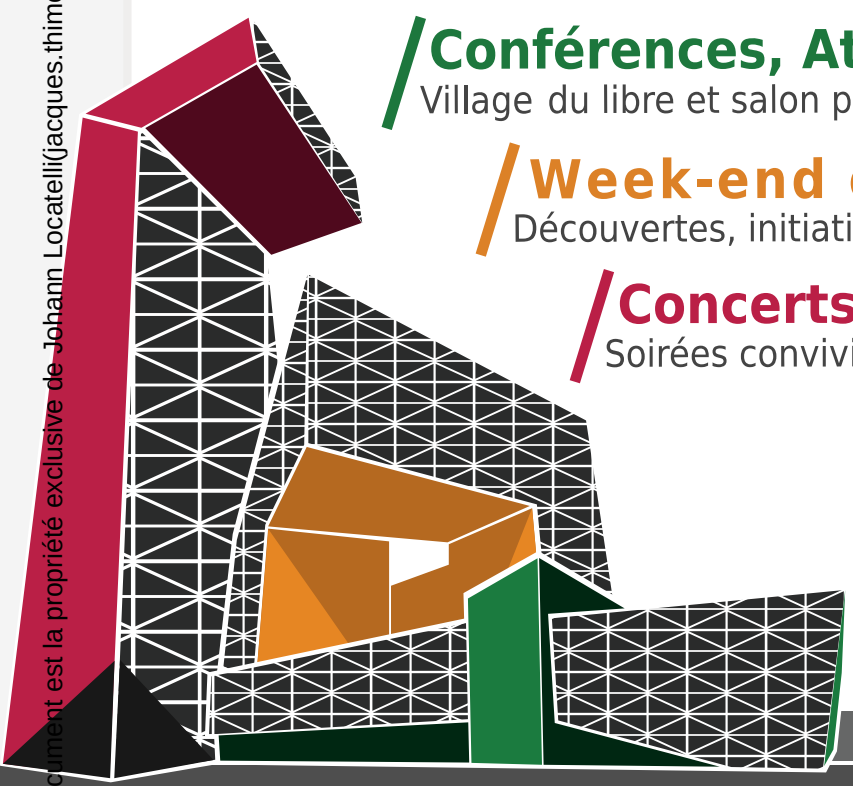
www.rmll.info | #RMLL2017

Place Jean Jaurès et Quartier Manufacture

/Conférences, Ateliers, Découvertes
Village du libre et salon professionnel

/Week-end découverte
Découvertes, initiations, activités ludiques

/Concerts et animations
Soirées conviviales, concerts, animations



ENTRÉE LIBRE

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Saint-Étienne L'expérience design
Saint-Étienne métropole
Loire Le Département
La Région Auvergne-Rhône-Alpes
/saintétienne Tourisme & Congrès
Liberté • Égalité • Fraternité
République Française
Université de Lyon
Université Jean Monnet Saint-Étienne
Jean Louis Gagnaire Député de la Loire Réserve Parlementaire
Adista

- *« Logical Volume Management ». LVM, en gros, c'est une abstraction qui te permet de manipuler tes disques avec beaucoup de souplesse. Normalement, ce que tu manipules, c'est des partitions. Et manipuler des partitions c'est un peu comme demander un peu de fantaisie à un comptable allemand rigoureux : tu es quand même très limité. Prenons par exemple le cas classique où tu manques d'espace disque sur une de tes partitions. Si, par un heureux hasard, cette partition est suivie d'un espace libre non utilisé, alors tu peux l'étendre assez facilement, mais c'est un cas super rare. Alors qu'avec LVM, on travaille sur des « volumes logiques » plutôt que sur des partitions. Et un volume logique ne correspond pas forcément à un espace contigu sur le disque, on est donc beaucoup plus souple. On peut même étendre un volume logique sur plusieurs disques.*

- *Je vois... Et donc, c'est à quel endroit dans ce code que* **debootstick** *utilise LVM ?*

- Attends, il faut que je t'explique un peu plus de trucs sur LVM, et sa structure en « couches ». Sur la couche la plus basse, on a la notion de « volume physique ». Un volume physique, c'est un « espace libre sur un disque ». Ça peut donc être une partition du disque, ou bien un disque complet. Ensuite on a une couche intermédiaire qui nous permet de créer des « groupes de volumes physiques ». Et enfin, dans chaque groupe de volumes, on peut créer 1 à n « volumes logiques », volumes qu'on pourra utiliser comme de bêtes partitions (donc en général mettre un *filesystem* dessus).

- *Ah ouais, quand même. Je me demande toujours quel genre de cerveau peut nous pondre une techno de ce genre...*

- *Réfléchis, c'est relativement logique... Imagine que tu doives stocker beaucoup de données sur un serveur. Tu comptes mettre l'OS sur un SSD et pour les données tu as 4 disques de 4 Tio chacun. Eh bien, pour faire simple, tu peux formater chaque disque comme un « volume physique » LVM. Ensuite tu mets le SSD tout seul dans un groupe de volumes que tu appelles* **VG_OS**, *et les 4 autres disques dans un groupe de volumes* **VG_DATA**. *Ensuite, tu crées tes « volumes logiques » : par exemple 3 volumes* **LV_ROOT**, **LV_BOOT** *et* **LV_SWAP** *dans* **VG_OS**, *et un seul volume* **LV_DATA** *dans* **VG_DATA**. *Tu formates tous ces volumes avec le filesystem qui va bien, et c'est parti. Et quand tu auras épuisé tes 16 Tio de disque avec ta grosse appli, tu me remercieras. Parce que tu auras juste à brancher 1 disque supplémentaire dans ta super baie, l'ajouter à* **VG_DATA** *et tu pourras alors tranquillement étendre* **LV_DATA** *(et ensuite le filesystem) sur l'espace libre.*

- *Effectivement sur ce genre d'exemple ça paraît censé.*

- *Ah, tu vois. Et ce n'est pas tout. Imagine, 6 mois plus tard tu reçois une alerte remontée par ton système de monitoring. Le contrôleur de disques t'annonce que ton deuxième disque de données (disons* **/dev/sdc**) *est passé en « predictive failure » : il va probablement bientôt rendre l'âme. Dans ce cas, avant que cela n'arrive, voilà ce que tu dois faire :*

  1) brancher un nouveau disque dans la baie pour s'assurer qu'on a de l'espace libre. Admettons que ce nouveau disque s'appelle **/dev/sdf** ;

  2) formater **/dev/sdf** *lui aussi comme un volume physique LVM ;*

  3) ajouter **/dev/sdf** *au groupe* **VG_DATA** ;

  4) indiquer à LVM de ne plus allouer de nouveaux blocs sur **/dev/sdc** *(le disque en « predictive failure »). LVM sera donc obligé d'utiliser les autres disques de* **VG_DATA** *pour stocker les nouvelles données ;*

  5) indiquer à LVM de libérer les blocs alloués sur **/dev/sdc** *: cette instruction va provoquer la migration des blocs alloués vers les autres disques de* **VG_DATA** ;

  6) une fois la migration terminée, enlever **/dev/sdc** *du groupe* **VG_DATA** ;

  7) débrancher **/dev/sdc** *de la baie.*

  Et voilà. Tu as résolu ton souci à chaud, sans redémarrage d'OS, et sans le moindre impact sur tes applications, même celles qui lisaient ou écrivaient dans **LV_DATA** *pendant nos manipulations.*

- *Impressionnant ! Mais on ne s'éloigne pas un peu de notre propos, du code de* **debootstick** *et de nos soucis de boot ?*

- *Non, pas du tout ! Regarde, ici, dans les sources de* **debootstick** **[6]** :

```
yes | pvcreate -ff ${TARGET}3
vgextend $LVM_VG ${TARGET}3
pvchange -x n ${ORIGIN}3
pvmove -i 1 ${ORIGIN}3 | while read pv action percent
do
    echo REFRESHING_MSG "$percent"
done
vgreduce $LVM_VG ${ORIGIN}3
```

D'après ce que j'ai compris, au moment où on arrive sur ce code, **${ORIGIN}** *c'est la clé USB et* **${TARGET}** *c'est le disque interne. D'autre part, apparemment, le filesystem de l'OS embarqué sur la clé est en réalité hébergé sur un volume LVM. Ce volume repose sur le groupe de volumes* **$LVM_VG**, *qui ne contient au départ qu'un seul volume physique,* **${ORIGIN}3**, *donc la troisième partition*

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

de la clé USB. Le but de ce bout de code est alors de forcer LVM à migrer les données depuis **`${ORIGIN}3`** *vers* **`${TARGET}3`**, *donc depuis la clé USB vers le disque interne.*

- Un peu comme tu expliquais juste avant, avec la migration des données d'un disque défaillant vers les autres disques ?

- Oui tout à fait. En réalité, les 5 instructions LVM que tu vois dans ce bout de code, c'est l'exacte traduction des étapes 2 à 6 que j'ai listées.

- D'accord...

- Et ce qui est puissant avec cette technique, c'est qu'on peut migrer à n'importe quel moment de la vie de l'OS. Donc l'histoire de la barre de progression qui s'affiche bien tard, alors que le bureau est déjà lancé, ça paraît tout à fait plausible avec cette technique.

Et accessoirement, tu vois aussi dans ce code qu'il n'est pas difficile de récupérer un pourcentage d'avancement, pour par exemple afficher cette barre de progression !

7. LE DÉROULÉ FINAL

Il avait failli renverser mon reste de café en voulant mimer la barre de progression. Il s'arrêta à nouveau pour réfléchir quelques secondes, puis reprit :

- A priori on peut valider tout ce qui se passe dans l'épisode. Tu es d'accord ?

- Oui, je crois aussi...

- Alors vas-y, explique-moi comment boote la clé USB « magique » de McGee.

- Eh bien, au départ, tout à fait normalement... Jusqu'au montage NFS. Ensuite, on peut penser qu'elle monte une union avec des données stockées sur la clé (ce qu'on a appelé le répertoire **`hacks`**). *Elle lance le système d'init final sur cette union. Et elle commence aussi à migrer tous les éléments de cette clé USB pirate vers le disque interne de la machine, pour que ce système soit conservé au prochain reboot. En utilisant LVM, a priori.*

- Oui, c'est ça. Dernière précision : quand on passe d'un processus d'init (initramfs) au suivant (init final), la bienséance voudrait qu'on passe la main dans un état le plus propre possible. En particulier, on évite de garder en vie des processus créés dans l'initramfs. Mais rien ne nous empêche de le faire en réalité. Donc, juste avant de passer la main, notre initramfs peut très bien lancer le processus de migration en arrière-plan.

- Bon. Sur ce, je rentre chez moi alors, il est temps.

- OK, merci pour l'info concernant **`debootstick`**. *Je regarderai ce soft plus en détail.*

- De rien.

Je n'ai rien dit sur le coup, mais j'estimais plutôt que c'était moi qui avais appris tout un tas de trucs, en une demi-heure de temps !

CONCLUSION

À présent, si je regarde un film ou une série, il suffit qu'un personnage fasse quelque chose d'inhabituel avec un ordi et je ne peux m'empêcher de me poser la question : « Impossible ou juste invraisemblable ? »...

Souvent on reste dans le classique, avec par exemple le logiciel de reconnaissance faciale d'une lenteur abominable parce qu'il affiche tous les visages qu'il analyse... Mais il y a pas mal d'autres scènes qui posent davantage question, et mériteraient clairement un petit test de faisabilité... Et là on commence à y réfléchir, jusqu'à perdre le fil de l'histoire. Tout ça à cause de « Peter Le Testeur ». Il m'a passé le virus. J'espère donc que je ne vous ai pas contaminé à votre tour, chers lecteurs ;)

Merci à Henry et Narek pour les relectures ! ■

NOTES ET RÉFÉRENCES

[1] Dans cet article, j'insiste principalement sur le « pourquoi » des différents éléments d'un OS. Une approche alternative et complémentaire pour bien comprendre est de consulter l'article de F. ENDRES, *« Live-System from Scratch »*, que l'on peut trouver dans *GNU/Linux Magazine n°202* (mars 2017, p.54 à 61). L'auteur y explique comment intégrer les différents éléments d'un OS pour créer une clé USB bootable.

[2] L'archiveur **`cpio`** est un vieux concurrent de **`tar`**, qui à ma connaissance n'est plus utilisé, sauf dans ce cas précis des archives *initramfs*.

[3] Le système de fichiers **`overlay`** (ou **`overlayfs`**) est disponible dans le noyau *mainline* depuis la version 3.18, on le trouve donc dans la plupart des distributions modernes.

[4] DUBLÉ É., *« Constructions with en langage bash »*, *GNU/Linux Magazine n°204*, mai 2017, p.70 à 78.

[5] Page GitHub de debootstick : https://github.com/drakkar-lig/debootstick

[6] À l'heure où ces lignes sont écrites, il s'agit de **`scripts/live/init/migrate-to-disk.sh`**, lignes 62 à 69. L'URL http://tiny.cc/debootstick-migrate pointe sur ce passage.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

SURVEILLEZ UN DOSSIER DE SOURCES POUR EXÉCUTER UNE COMMANDE AUTOMATIQUEMENT

STÉPHANE MOUREY
[Mousse sur le Seeraiwer]

MOTS-CLÉS : BASH, INOTIFY, MONITORING, SURVEILLANCE, FICHIER, DOSSIER

Lorsque l'on développe avec certains outils, un site web avec Jekyll par exemple, on vous propose une option qui se révèle bien pratique, souvent appelée --watch. Lorsqu'elle est activée, l'outil surveille le dossier contenant votre code source pour le recompiler dès qu'une modification intervient. Mais lorsque le processus de compilation fait intervenir plus d'un outil, ou si celui qui a votre préférence ne vous offre justement pas cette option, il paraît difficile de continuer à fonctionner comme cela. La solution : inotifywait !

Nous allons écrire un script **Bash** qui surveillera le contenu d'un dossier, y compris ses sous-dossiers, et lancera une commande dès qu'une modification y sera détectée. Nous nous appuierons pour cela sur **inotifywait**, une commande shell permettant de recevoir les modifications envoyées par **inotify**, un mécanisme du noyau qui permet de surveiller les accès aux systèmes de fichiers.

1. INSTALLATION

inotifywait est fourni par le paquet **inotify-tools**, qu'on installe sur **Debian** et consorts à l'aide de la commande :

```
# apt-get install inotify-tools
```

2. LE SCRIPT

Notre script s'appelle **watchtree**. Placez-le dans **~/bin** si vous voulez le garder pour vous, ou dans **/usr/local/bin** si vous souhaitez rendre la commande disponible pour tous.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Voici son contenu :

```
01: #!/bin/bash
02:
03: # CONFIGURATION
04: EVENTS="modify,close_write,move,
create,delete"
05: FIFO=/tmp/watchtree-'date +%s%N'
06:
07: # FUNCTIONS
08: help() {
09:  cat <<eoh
10: Monitor directory and execute a
command each time a change occurs.
11: Usage: 'basename $0' directory
command
12:
13: eoh
14: }
15:
16: on_exit() {
17:   kill $INOTIFY_PID
18:   rm $FIFO
19:   exit
20: }
21:
22: # MAIN
23: if [ "$1" = "--help" ] ||
[ "$1" = "-h" ] ; then
24:  help
25:  exit
26: fi
27:
28: if [ $# -ne 2 ] ; then
29:  echo Invalid number of
arguments.
30:  echo
31:  help
32:  exit
33: fi
34:
35: echo Watching...
36: echo
37:
38: mkfifo "$FIFO"
39:
40: inotifywait -m -r -q -e
"$EVENTS" "$1" > "$FIFO" &
41: INOTIFY_PID=$!
42:
43: trap "on_exit" 2 3 15
44:
45: while read file
46: do
47:   "$2"
48: done < "$FIFO"
49:
50: on_exit
```

Voici quelques commentaires pour comprendre le code, ce qui permettra de l'adapter si le besoin se présente.

À la ligne 3, nous définissons les types d'événements qui doivent provoquer l'exécution de la commande : nous ne retenons que ceux correspondant à des modifications. Vous obtiendrez la liste complète des types disponibles en demandant l'aide d'**inotifywait --help**.

À la ligne 4, nous créons un nom de fichier temporaire, en nous appuyant sur la commande **date** avec assez de précision pour qu'il n'y ait aucune chance de produire deux fois le même nom.

Suivent deux fonctions, **help**, ligne 8 qui se contente d'afficher l'aide et sera appelée aux moments appropriés (demande explicite ou utilisation incorrecte) ; et **on_exit**, qui sera appelée lors de l'interruption de la commande pour faire le nettoyage nécessaire lorsque l'utilisateur demandera l'interruption de la surveillance. Les commandes ici sont très simples (interruption d'un processus ligne 17 et suppression d'un fichier ligne 18), mais elles ne prennent leur sens que plus loin, lors de la mise en place du monitoring.

Suit le corps principal de notre script. Après les vérifications d'usage et l'éventuel affichage de l'aide ainsi que d'un petit message de bienvenue, les choses sérieuses commencent à la ligne 38. Ici, à l'aide de la commande **mkfifo**, nous créons un *tube nommé*. Cette technique permet d'accéder à un tube d'une manière plus souple que les opérateurs fournis par le shell (**|**, **>**, **<**, etc.), en utilisant un nom de fichier. Ce fichier n'aura pas d'existence réelle sur le disque, mais son emplacement pourra être utilisé par plusieurs programmes pour écrire et lire directement en mémoire, de la même façon qu'avec un tube classique, les données n'étant pas « persistées » et la lecture devant être simultanée avec l'écriture.

Ce tube est alimenté à la ligne 40 par **inotifywait**. L'option **-m** permet de l'utiliser en mode surveillance, sans elle **inotifywait** quitterait dès le premier événement. Les autres options employées sont :

- **-r** qui fait que la détection d'événements a lieu également dans les sous-dossiers ;
- **-q** qui rend l'action silencieuse (ce que j'ai fait ici surtout pour éviter l'affichage d'un avertissement concernant l'utilisation de **-r**) ;
- **-e** qui permet de définir quels sont les types d'événements pour lesquels une notification doit avoir lieu.

Nous passons tout cela en arrière-plan, de façon à pouvoir lire les notifications dans un autre processus. Ligne 42, nous récupérons l'identifiant du processus **inotifywait** pour pouvoir le tuer à la sortie de notre script.

Ligne 43, nous utilisons la commande **trap** pour forcer l'exécution de la fonction **on_exit** en cas d'interruption du script, de façon à quitter proprement.

Enfin, aux lignes 45 à 48 nous définissons une boucle qui va lire sans fin le contenu de notre tube nommé, et exécuter la commande passée en deuxième paramètre de notre script à chaque fois qu'il y trouvera du contenu.

3. ACTIVATION

Si vous avez bien suivi le début de la section 2, le script est déjà placé dans le *path*. Il ne reste alors plus qu'à le rendre exécutable pour pouvoir l'utiliser depuis n'importe quel emplacement de l'arborescence. Ce sera fait grâce à la commande suivante :

```
# chmod +x ~/bin/watchtree
```

Ou, suivant l'emplacement que vous avez choisi :

```
# chmod +x /usr/local/bin/watchtree
```

CONCLUSION

Voilà, nous avons maintenant une commande qui nous permet de surveiller toute une arborescence et de déclencher l'exécution d'une autre au moindre changement. Pour reprendre l'exemple de notre introduction, en imaginant que nos sources se trouvent dans un dossier **src/** et que la compilation est lancée par la commande **./build**, nous pourrons utiliser notre script de la façon suivante pour compiler automatiquement à chaque modification des sources :

```
$ watchtree src ./build
```

Pour arrêter, il suffira de faire la combinaison de touches **<Ctrl>** + **<C>**. ■

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

BLOCKCHAIN : MODIFIER UN CONTRAT IMMUABLE

PHILIPPE PRADOS
[Consultant/Architecte OCTO Technology]

MOTS-CLÉS : BLOCKCHAIN, BITCOIN, ETHEREUM, SOLIDITY

Dans un monde immuable, comment corriger un bug ?

Les différentes technologies de **Blockchain** (**Bitcoin**, **Ethereum**, etc.) consistent à proposer une solution permettant de négocier un consensus, entre de nombreuses parties, sur un état stable. Pour Bitcoin, il s'agit de se mettre d'accord sur l'état d'un livre de compte ; pour Ethereum, sur l'état d'un ordinateur virtuel mondial.

Ethereum a la prétention de permettre la rédaction de contrats numériques, dont l'application est sous le contrôle du réseau et non d'une entité étatique. Des fonds peuvent être sous la responsabilité du contrat, dont le code décide ensuite sa distribution. Typiquement, le contrat peut servir de notaire, en gardant des fonds jusqu'à ce qu'une condition soit valide (délais, preuve de livraison, etc.).

Ethereum accueille des contrats, constitués d'une adresse et d'un code destiné à une machine virtuelle spécifique (**EVM**). Chaque instruction a un coût en gaz, lui-même converti en éthers, cotés sur différentes places de marchés. Il faut donc posséder quelques unités d'éthers pour pouvoir demander au réseau d'apporter des modifications à un contrat.

Pour invoquer un contrat, il est nécessaire d'écrire une transaction et de l'envoyer au réseau. Ce dernier se charge d'exécuter le traitement, d'apporter des modifications sur l'état du contrat, et d'écrire à la suite de la chaîne de bloc, le nouvel état du contrat. Tous les serveurs participant aux réseaux se mettent d'accord sur le nouvel état du contrat.

Le principe de base d'un contrat Ethereum est d'avoir un code immuable, impossible à modifier ou à supprimer. Si le développeur ne l'a pas prévu, il n'est pas possible d'arrêter un contrat. Il s'applique dès qu'il est sollicité.

En effet, lorsqu'un contrat est déposé dans la chaîne de bloc, il y est pour toujours. Toutes les instructions qui le compose peuvent être invoquées à tout moment, entraînant éventuellement une modification d'état du contrat, d'autres contrats liés, voire transférant des éthers vers un autre compte ou un autre contrat.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- Un contrat déposé dans la blockchain ne peut être supprimé.
- Il peut être invoqué par n'importe quel utilisateur ou autre contrat.
- Le code d'un contrat ne peut pas évoluer.
- Seules les données le peuvent.

Modifier l'état d'un contrat, c'est écrire les différences dans le prochain bloc.

Cette immutabilité est une excellente chose, car elle permet de garantir qu'aucune des parties prenantes du contrat ne pourra revenir sur ses engagements. C'est un élément essentiel de la sécurité d'Ethereum et cela contribue à sa valeur.

Si j'accepte de signer un contrat où je m'engage à rembourser telle somme en éthers, si une condition ne s'est pas produite à telle date, il m'est impossible de répudier cela. Lorsque le délai sera passé, l'autre partie peut déclencher le versement des éthers présents en caution dans le contrat.

C'est super cool. Personne, pas même un pirate, ne pourra apporter des modifications au contrat. À tel point que si le contrat est mal écrit, des éthers peuvent être inaccessibles à tout jamais.

1. LES TRUCS À SAVOIR SUR LA EVM ET SOLIDITY

En pratique, les développeurs utilisent le langage de développement **Solidity [1]**, permettant la rédaction simple de contrats. Le code Solidity est compilé dans un byte-code destiné à la machine virtuelle Ethereum (EVM).

1.1 Ethereum VM (EVM)

L'Ethereum Virtual Machine est très rustique. Elle propose des calculs sur des données de 256 bits (32 bytes) à l'aide d'une pile d'exécution (opérations sur 64 bits prévues pour une prochaine version). C'est-à-dire que les valeurs sont déposées sur une pile et les instructions consomment les données au-dessus de la pile pour y déposer le résultat.

Par exemple, la suite d'instructions suivante effectue une addition et place le résultat sur le sommet de la pile.

```
PUSH1 0x42
PUSH1 0x24
ADD
```

Il y a également des zones mémoires spéciales. La RAM, dédiée à l'exécution de la transaction, commence à l'adresse zéro et augmente au fur et à mesure des besoins, par paquet de 32 bytes. Comme toutes les instructions de la machine virtuelle, cela coûte de l'énergie (1 gaz par 32 bytes).

Les données persistantes dans le contrat (tous les attributs) sont mémorisées dans une table d'association entre 32 bytes pour la clé et 32 bytes pour la valeur. Les modifications sont sauvegardées dans le prochain bloc de la blockchain, après validation par la communauté.

Le coût d'un attribut est simple : 20 000 gaz lorsqu'une valeur est créée (qu'elle passe de zéro à autre chose) ; 5 000 gaz lors de l'écriture d'un attribut déjà existant ; et une récupération de 15 000 gaz lorsqu'une valeur est effacée à zéro (`delete`).

Lorsqu'une méthode est invoquée, une transaction est envoyée au contrat. On y trouve :

- `msg.sender` : l'émetteur du message ;
- `msg.gas` : le gaz restant pour la suite du traitement ;
- `msg.data` : les paramètres de l'invocation du contrat ;
- `msg.sig` : la signature de la méthode invoquée. Il s'agit en fait d'un calcul de hash sur le nom de la méthode, intégrant la liste des paramètres. Cela correspond aux quatre premiers octets de `msg.data` ;
- `msg.value` : éventuellement, une quantité de Wei (sous unité d'éther).

Pour construire un contrat depuis l'extérieur de la blockchain, il faut invoquer le contrat zéro, avec en paramètre, le code et la description du nouveau contrat. Le code du contrat zéro se charge d'ajouter le nouveau contrat dans la chaîne de bloc et de retourner son adresse.

Pour construire un contrat depuis un autre contrat, il est possible d'utiliser l'instruction `CREATE` de l'EVM. Cela est moins coûteux que depuis l'extérieur.

Des instructions permettent à un contrat d'écrire des logs, dans quatre topics. Les logs sont des tableaux de bytes, associés à chaque contrat. Cela permet aux API **JavaScript** ou autre, de récupérer des informations lorsqu'une transaction est validée par la communauté. C'est le canal privilégié pour les communications asynchrones entre les contrats et l'extérieur d'Ethereum.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

1.2 Solidity

Par-dessus la machine virtuelle, le langage Solidity propose de nombreux concepts complémentaires, pouvant être traduits en instruction de la machine virtuelle. Souvent, le code rédigé en Solidity est très éloigné du code compilé pour la EVM. Il est parfois important de comprendre la relation entre les deux, pour exploiter au mieux les avantages des deux modèles de programmation.

Comme les instructions de la EVM fonctionnent en 256 bits, cela n'économise pas le coût de sauvegarde des informations. Pour optimiser cela, Solidity se charge d'effectuer des opérations de masque binaire, pour pouvoir manipuler quelques bytes à la fois de chaque zone mémoire de 32 bytes. Par exemple, la méthode **f()** suivante :

```
contract Test {
    byte b;
    function f() {
        b=0xAA;
    }
}
```

est compilée en cette longue suite d'instructions. Tout cela pour ne modifier QUE le premier octet des 32 bytes :

```
// b = b AND NOT(0xFF) OR 0xAA
PUSH 0 // Offset de l'attribut
DUP1
SLOAD  // Lecture de l'attribut de position 0
PUSH FF // Masque binaire pour un octet
NOT   // Inversion du masque (0xFFFFF....FF00)
AND   // Masque entre la donnée de l'attribut et le
masque
PUSH AA // Valeur à écrire
OR  // Or entre 0xAA et l'attribut moins le premier octet
SWAP1
SSTORE  // Ecriture de l'attribut
```

Cela fait beaucoup d'instructions, mais c'est bien moins cher que de mémoriser chaque octet dans un espace de 32 bytes en termes d'éther.

Comme il n'y a qu'un seul point d'entrée pour un contrat, Solidity a décidé de consacrer les quatre premiers octets des paramètres de la transaction à l'identification de la méthode à invoquer. La valeur des octets correspond à un **bytes4(sha3("f(uint)")** sur le nom de la méthode, complété par le type des différents paramètres.

Le code du contrat commence par une sorte de **switch**, avec la signature de chaque méthode. Si aucune méthode ne correspond, alors la méthode par défaut est utilisée. C'est une méthode spéciale, n'ayant pas de nom ni de paramètre (**function () {}**).

```
switch (msg.sig) {
  case e1c7392a : // function init()
    ...
  case 2db12ac4 : // function changeToV2()
    ...
  case 82692679 : // function doSomething()
    ...
  default : // function ()
    ...
}
```

La machine virtuelle n'a pas de notion de constructeur. Pour construire une instance, il faut envoyer une transaction vers le contrat zéro. Les données sont alors considérées comme du code à exécuter.

Pour chaque contrat, le compilateur Solidity génère alors le code du constructeur, qui se charge de fournir le code du contrat à produire.

Attention, si le code complet du constructeur est trop volumineux, le contrat ne peut être construit. Il faut alors séparer la construction en deux, en ajoutant une méthode **init()** par exemple.

Solidity propose la notion de **modifier**. C'est un morceau de code encadrant un autre code. Il suffit d'annoter une méthode d'un **modifier**, pour que le contenu de la méthode soit encadré par le contenu du ou des **modifier**.

```
modifier onlyOwner {
    if (isOwner(msg.sender))
        _;
}

function f() onlyOwner {
  ...
}
```

Solidity propose une instruction **throw**, pour signaler une erreur dans un contrat. Comme il n'existe pas d'équivalent dans la machine virtuelle, le code généré invoque un saut vers une adresse mémoire invalide. Cela génère une erreur lors de l'exécution, ce qui est le but recherché. Ne vous étonnez pas alors, de recevoir une erreur de type « *saut à une adresse invalide* ».

Solidity propose la notion d'**event**. Ce sont en fait des logs pour la machine virtuelle. Ils permettent d'informer le code JavaScript ou autre, à l'écoute de la chaîne, lorsqu'une méthode d'un contrat est invoquée. Des filtres permettent d'attendre l'acquittement d'un traitement dans la blockchain.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Le hash de la signature de l'événement est utilisé comme nom de topic, pour l'indexation des événements.

Parmi les instructions avancées de la machine virtuelle Ethereum, il y a trois instructions spéciales :

- **`call`** pour invoquer un autre contrat et le modifier ;
- **`callcode`** pour utiliser le code d'un autre contrat, sur l'état du contrat appelant, en indiquant le contrat actuel comme à l'origine de l'invocation ;
- **`delegatecall`** pour utiliser le code d'un autre contrat, sur l'état du contrat appelant, en gardant l'identité de l'invocation de la méthode d'origine.

En fait, **`delegatecall`** est un bug fixe de **`callcode`**, afin de garder le **`msg.sender`** valide.

Un dernier point à savoir. Pour maîtriser la mémoire nécessaire à l'invocation des méthodes, l'attribut avec la clé **`0x40`** est utilisé par Solidity. La valeur indique la plus haute adresse mémoire utilisée par les méthodes du contrat.

Nous allons utiliser toutes ces particularités pour atteindre notre objectif.

2. AMENDER UN CONTRAT ?

Un contrat est immuable, mais parfois, on aimerait bien pouvoir le modifier, avec l'accord de toutes les parties si nécessaires. On aimerait pouvoir publier une nouvelle version du contrat, pour corriger un bug par exemple.

Du point de vue scénario d'usage, on peut imaginer la situation suivante : deux personnes signent un contrat numérique qui est régi par la loi. La loi peut évoluer remettant en cause le contrat. Il faut donc le modifier, avec l'accord des deux personnes.

Il est également possible d'ajouter un utilisateur étatique : l'État.

Le contrat est alors construit avec trois propriétaires : les deux personnes et l'État. Il est paramétré pour que 2 propriétaires seulement soient nécessaires pour modifier le contrat.

Ainsi, les deux personnes peuvent le modifier ou bien l'une d'entre elles avec l'aval de l'État (voir figure 1).

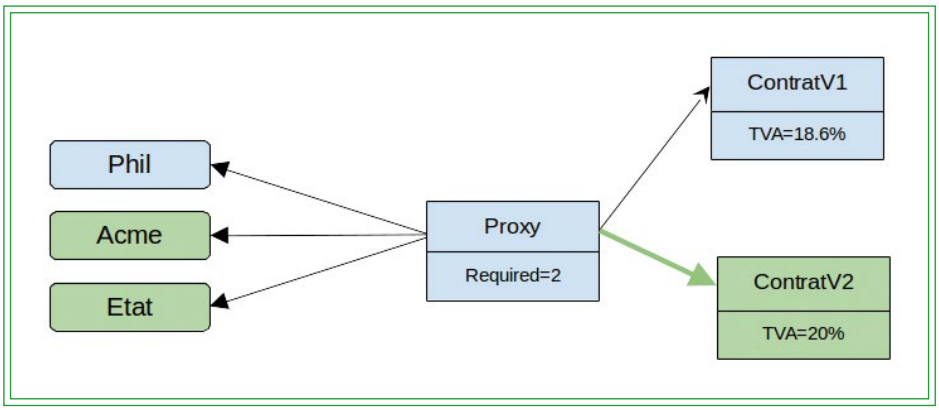


Fig. 1 : Contrat multi-owners.

C'est à ce chantier que nous nous sommes attelés. Comment modifier le comportement d'un contrat pourtant immuable ?

Plusieurs approches peuvent être envisagées :

- Supprimer le contrat actuel et le remplacer par un autre ;
- Permettre l'ajout d'avenant au contrat ;
- Utiliser un proxy spécifique ou générique ;
- Combiner toutes les approches.

Pour vous permettre de tester tout cela facilement, nous vous proposons d'utiliser **browser-solidity** présent à cette adresse **[2]**.

En ajoutant le plugin **MetaMask** à **Chrome** (voir figure 2), vous pouvez sélectionner la blockchain de test, et demander gratuitement quelques éthers sur ce réseau pour tester le code dans la vraie vie.

Sans le plugin, dans l'onglet représentant une boîte, sélectionnez ***Javascript VM***. Ainsi, vous pouvez compiler et tester tout le code, uniquement dans le navigateur ! Une implémentation de l'EVM est alors disponible localement (voir figure 3, page suivante). Et pas besoin d'Ether !

Fig. 2 : Plugin Chrome MetaMask.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

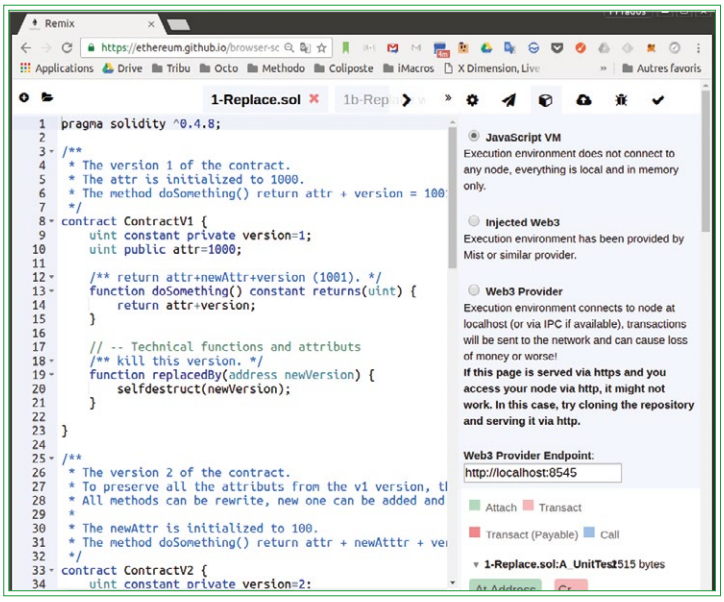


Fig. 3 : Sélection de l'EVM.

3. REMPLACER UN CONTRAT

Comme un contrat ne peut pas être modifié, il doit être possible de le remplacer. Avec l'accord de toutes les parties, le contrat précédent est annulé pour être remplacé par un nouveau.

Notez que dans les schémas, les textes en italique sont des propriétés ou des méthodes techniques, ne faisant pas partie du contrat d'origine. Les textes en gras sont des ajouts au contrat d'origine (voir figure 4).

Pour qu'il soit possible de modifier un contrat, il faut le prévoir dans le contrat d'origine.

```
/**
 * The version 1 of the contract.
 * The attr is initialized to 1000.
 * The method doSomething() return attr + version = 1001
 */
contract ContractV1 {
    uint constant private version=1;
    uint public attr=1000;

    /** return attr+newAttr+version (1001). */
    function doSomething() constant returns(uint) {
        return attr+version;
    }

    // -- Technical functions and attributs
    /** kill this version. */
    function replacedBy(address newVersion) {
        selfdestruct(newVersion);
    }

}

/**
 * The version 2 of the contract.
 * To preserve all the attributs from the v1 version, this version IS a ContractV1.
 * All methods can be rewrite, new one can be added
 * and some attributs can be added.
 *
 * The newAttr is initialized to 100.
 * The method doSomething() return attr + newAtttr + version = 1102
 */
contract ContractV2 {
    uint constant private version=2;
    uint attr;
    uint newAttr=100;

    /** return attr+newAttr+version (1102). */
    function doSomething() constant returns(uint) {
        return attr+newAttr+version;
    }

    /** return 42. Another method in version 2. */
    function doOtherThing() constant returns(uint) {
        return 42;
    }

    // -- Technical functions and attributs
    /**
     * Propagate the states from the v1 to v2.
     */
    function ContractV2(ContractV1 origin) {
        attr=origin.attr();          // Copy the current state of v1
        origin.replacedBy(this);
    }

    // -- Technical functions and attributs
    /** kill this version. */
    function replacedBy(address newVersion) {
        selfdestruct(newVersion);
    }
}
```

Le constructeur de la nouvelle version doit avoir accès aux attributs du contrat d'origine pour les récupérer.

J'ai omis les règles de sécurité, car nous les évoquerons plus loin. Il faut en effet ajouter des privilèges dans la méthode **replaceBy()**, pour que seules les parties prenantes du contrat puissent accepter la nouvelle version du contrat.

Vous pouvez tester cela en ligne ici **[3]**.

N'oubliez pas de sélectionner Javascript VM pour un test en local au navigateur. En cliquant dans l'ordre indiqué en figure 5, vous créez un contrat, l'initialisez et invoquez la méthode **doSomething()** première version, pour récupérer **1001** (**attr** + **version**).

Ensuite, il est temps de modifier la version et de s'assurer que le comportement de **doSomething()** est bien différent (**attr** + **newAttr** + **version** = **1102**). Il est alors possible d'invoquer une nouvelle méthode **doOtherThing()**, absente de la première version du contrat.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

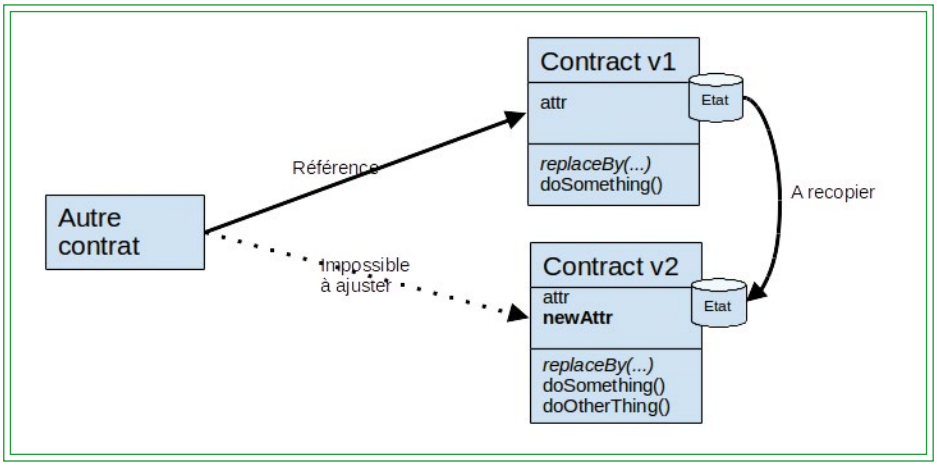


Fig. 4 : Proposer un nouveau contrat.

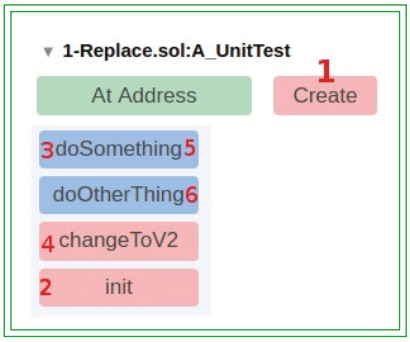


Fig. 5 : Tester le contrat.

Le code du test unitaire est celui-ci :

```
contract A_UnitTest {
  ContractV1 aContractV1;
  ContractV2 aContractV2;

  /**
   * Initialise useContract with a ContractV1.
   */
  function init() {
      aContractV1=new ContractV1();
  }

  /**
   * Invoke the method 'doSomething()', valide with a contract v1 or v2.
   * The caller must be known the reference of the current version.
   */
  function doSomething() constant returns(uint) {
      return ((address(aContractV2) == 0)
          ? aContractV1.doSomething()
          : aContractV2.doSomething());
  }

  /** Invoke a method 'doOtherThing()', only valide with a contract v2. */
  function doOtherThing() constant returns(uint) {
      if (address(aContractV2) == 0) throw;
      return aContractV2.doOtherThing();
  }


  /** Change to V2. */
  function changeToV2() {
      aContractV2=new ContractV2(aContractV1);
      aContractV1.replacedBy(aContractV2);
      delete aContractV1;
  }
}
```

Le premier inconvénient de cette approche est que la référence du contrat n'est pas maintenue lors du changement de version. Si d'autres contrats utilisent toujours une référence vers **ContractV1**, ils vont planter lorsqu'ils voudront l'invoquer. En effet, le contrat est détruit lorsqu'il est remplacé. Le test unitaire doit choisir la version du contrat à invoquer.

Le deuxième inconvénient est que les événements (**event**) ne sont plus émis du contrat v1, après le changement de version. Les applications en écoute d'événements doivent également traiter cela.

Le troisième inconvénient est que les données stockées dans le **ContractV1** doivent être déplacées dans le **ContractV2**. Cela peut coûter beaucoup de gaz, voire plus que ce qui est disponible pour une méthode. Même les soldes en éthers doivent être déplacées d'un contrat à l'autre, ce qui présente un grand risque de sécurité.

Une approche pour contourner cette difficulté consiste à proposer un contrat « *base de données* » suffisamment générique. Ce contrat est référencé par les versions 1 et 2 pour y stocker les données persistantes (voir figure 6, page suivante).

Le code est testable ici **[4]**.

Il n'est plus possible d'avoir des états dans les contrats v1 et v2. Tout passe par le **ContractDB**. Les fonds restent dans les contrats. Il faut encore les déplacer. Cherchons une meilleure approche.

4. AVENANT SPÉCIFIQUE

Finalement, si on cherche à faire l'analogie avec le monde juridique, modifier un contrat c'est ajouter un avenant, signé par toutes les parties. Pourquoi ne pas proposer ce modèle ? (voir figure 7, page suivante).

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

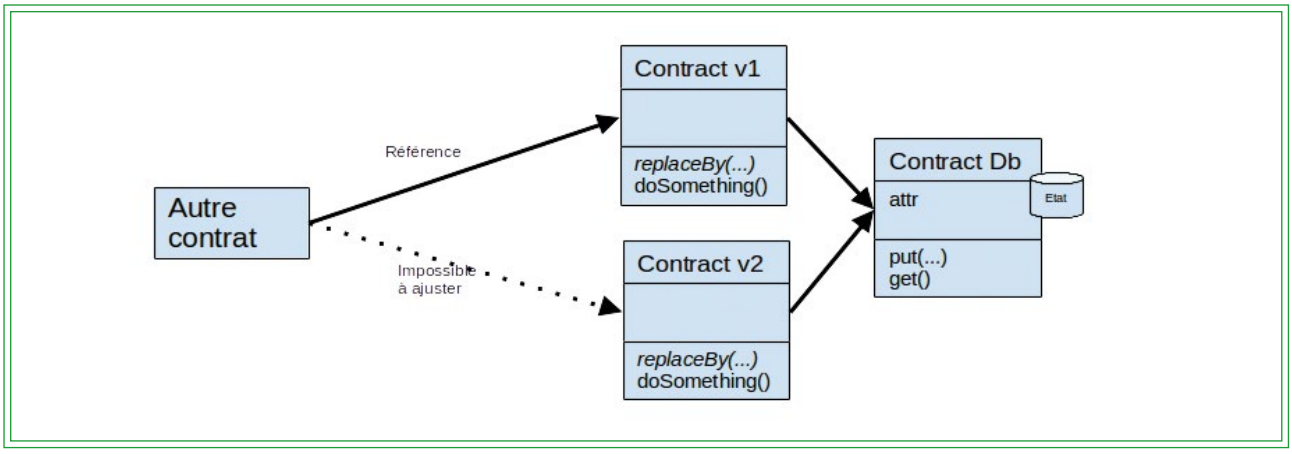


Fig. 6 : Contrat avec base de données.

L'idée est la suivante. S'il n'y a pas d'avenant, les clauses du contrat s'appliquent. S'il y a un avenant, alors il faut appliquer les clauses de l'avenant. Ce dernier peut éventuellement considérer les clauses d'origines comme inchangées. Si un avenant est disponible, il est interdit d'invoquer directement la clause du contrat. Il faut obligatoirement passer par l'avenant.

Dans chaque méthode du contrat, il faut ajouter un code pour s'assurer de la présence de l'avenant.

```
function doSomething() constant returns(uint) {
    if ((address(amendment) != 0) && (msg.sender != address(amendment))) {
        return amendment.doSomething();
    }
    else
        return attr+version;
}
```

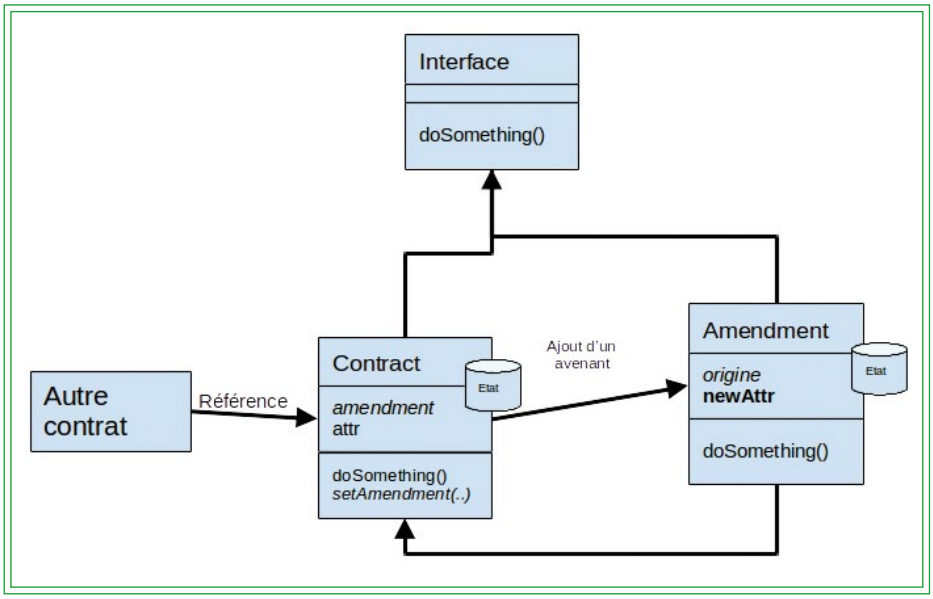


Fig. 7 : Contrat avec base de données.

En effet, il n'est pas possible d'utiliser un `modifier` Solidity pour cela, car chaque méthode a une signature spécifique.

Dans ce scénario, l'avenant possède son propre état, différent de l'état du contrat. L'avenant et le contrat doivent travailler en étroite collaboration pour se partager les attributs.

La démonstration est ici **[5]**.

Malheureusement, les événements peuvent être émis du contrat ou de l'avenant et il n'est pas possible d'ajouter de nouvelles méthodes au contrat d'origine.

5. PROXY SPÉCIFIQUE

Une fois dans la blockchain, un contrat possède une adresse unique. Les différents partenaires connaissent cette adresse. Ils l'utilisent pour manipuler le contrat.

Et si le contrat référencé n'est qu'un proxy vers l'implémentation courante du contrat ? Il suffit de modifier la référence vers le contrat cible dans le proxy pour modifier le comportement de ce dernier (voir figure 8, page 72).

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°15 !

SÉCURITÉ DES OBJETS CONNECTÉS

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

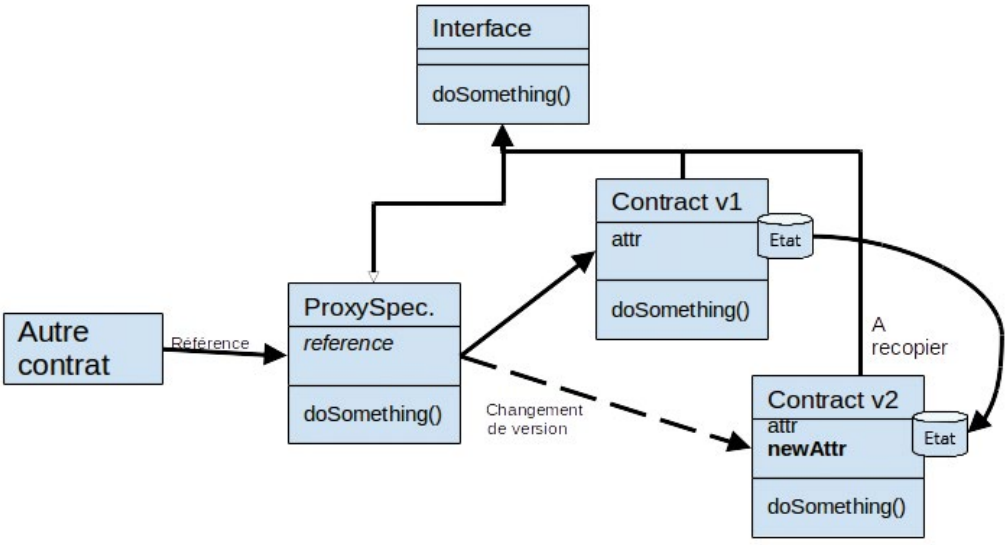


Fig. 8 : Proxy spécifique.

C'est cool, bien que très classique. Mais comment implémenter cela ? La première idée est de proposer une interface commune entre le proxy et les implémentations.

```
/**
 * Interface shared by Proxy, ContractV1 and ContractV2.
 */
contract Interface {
    /** return uint, depend of the current version. */
    function doSomething() constant returns(uint);
}
```

Ensuite, le proxy ainsi que les différentes versions, implémentent cette interface.

```
/** Delegate to the current version. */
function doSomething() constant returns(uint) {
   return currentVersion.doSomething();
}
```

Il est facile de tester cela avec browser-solidity ici **[6]**.

Le test unitaire se charge de *caster* correctement les contrats.

```
/**
 * Unit test.
 *
 * After created an instance,
 * - call 'init()'
 * - call 'doSomething()' return 1001
 * - call 'changeToV2'
 * - call 'doSomething()' return 1102
 * - call 'doOtherthing()' return 42
 */
contract A_UnitTest {
    /** Current proxy. */
    Interface aContract;

    function init() {
        aContract=Interface(new Proxy(new
ContractV1()));
    }

    /** Invoke the method 'doSomething()', valide
with a contract v1 or v2. */
    function doSomething() constant returns(uint) {
        return aContract.doSomething();
    }

    /** Invoke a method 'doOtherThing()', only
valide with a contract v2. */
    function doOtherThing() constant returns(uint) {
        throw; // Not implemented
    }
    /** Change to V2. */
    function changeToV2() {
        ContractV1 contractV1=ContractV1(Proxy(aCont
ract).currentVersion());
        Proxy(aContract).changeVersion(new
ContractV2(contractV1));
    }
}
```

Nous avons résolu le problème de la référence du contrat d'origine par d'autres contrats. Le monde extérieur référence le proxy et ce dernier ne bouge pas.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Nous n'avons pas à distribuer les attributs entre le contrat et l'avenant.

Mais, nous ne pouvons plus ajouter de nouvelles méthodes dans la nouvelle version du contrat. En effet, le proxy définit les méthodes décrites dans l'interface et ne peut plus évoluer. Il est possible de modifier l'implémentation des méthodes dans la version 2 du contrat, mais pas d'ajouter des méthodes.

De plus, il est encore nécessaire de faire migrer les données entre les contrats, avec les limites que nous avons évoquées.

Comment résoudre cela ? C'est possible, mais il va falloir sortir l'artillerie lourde.

6. PROXY GÉNÉRIQUE

Il est tentant de chercher à propager le message de la transaction vers la version du contrat ! Ainsi, quel que soit le message reçu, c'est le contrat cible qui peut le traiter. Il est ainsi possible d'ajouter de nouvelles méthodes sans devoir revoir le proxy.

Bon, pour cela, il faut sortir du code en assembleur. Solidity permet cela. Cool.

Nous devons récupérer le message, l'adresse du contrat cible, puis donner une zone mémoire libre pour récupérer le résultat de la méthode invoquée afin de le propager à l'appelant du proxy. Voici ce code.

```
function callCode(address target, int returnSize) internal {
    assembly {
      let brk := mload(0x40)              // Special solidity slot with top
memory
      calldatacopy(brk, 0, calldatasize) // Copy data to mem at offset brk
      let retval := call(sub(gas,150)
                         ,target          //address
                         ,0               //value
                         ,brk             //mem in
                         ,calldatasize    //mem_insz
                         ,brk             // reuse mem
                         ,returnSize)     // arbitrary return size
       // 0 == it threw (jump to bad destination)
       jumpi(0x00,iszero(retval))         // Throw (access invalid code)
       return(brk,returnSize)             // Return returnSize to the caller
  }
}
```

Comme indiqué, il faut savoir que Solidity identifie la taille mémoire qu'il utilise pour le contrat à l'emplacement **`0x40`**. C'est l'équivalent de **`brk(2)`** en **C**. La valeur stockée à cette référence permet de trouver une zone mémoire qui n'est utilisée par aucune méthode du contrat.

Le code assembleur commence par récupérer le top de la mémoire pour y copier les *datas* du message de la transaction (**`calldatacopy`**). Ensuite, il invoque le contrat **`target`** en lui indiquant la zone mémoire pour l'input. Il indique également la même zone mémoire pour récupérer le résultat. Comme il n'est pas possible de savoir à l'avance la taille maximum d'un retour possible d'une méthode du contrat cible, nous laissons l'utilisateur définir le paramètre **`returnSize`**. La valeur **`32`** est généralement suffisante. Enfin, le code retourne directement cette valeur à l'appelant.

Comment proposer un proxy générique maintenant ? En utilisant la fonction *fall-back*. Lorsqu'une fonction ne possède aucun paramètre ni nom, elle est invoquée pour toutes les méthodes dont la machine virtuelle ne trouve pas d'implémentation. C'est comme cela que fonctionne le code généré par Solidity. C'est d'ailleurs cette méthode qui est invoquée lors du dépôt d'éther dans le contrat. Nous n'avons qu'à invoquer notre méthode en assembleur pour toutes les méthodes du proxy ! (voir figure 9, page suivante).

```
/** Delegate all call to the current
version. */
function () payable {
  callCode(currentVersion,32);
}
```

Et voilà, nous avons maintenant un proxy générique, capable d'invoquer de nouvelles méthodes d'une version 2 de notre contrat.

Vous pouvez tester cela ici **[7]**.

Cette une approche sympathique. Elle présente néanmoins plusieurs défauts :

- La version 2 du contrat ne possède pas les données de la version 1. Il faut transférer l'état de la version 1 dans la version 2 pour reprendre le contrat.
- Les événements émis par le contrat v1 ne viennent pas de la même origine que les événements venant du contrat v2.

7. LA SOLUTION ULTIME

Nous pouvons essayer de mélanger différentes approches pour répondre à tous les besoins. Et si le Proxy se charge de maintenir les données de toutes les versions ? Utilisons **`delegateCall`** à la place de **`call`**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

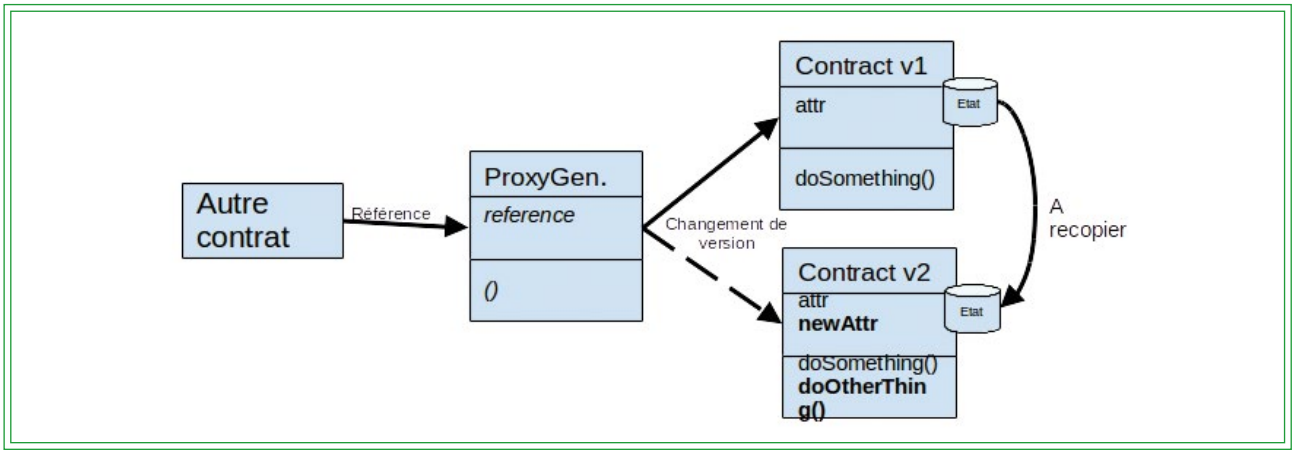


Fig. 9 : Proxy générique.

```
function propagateDelegateCall(address target, int returnSize)
internal {
  assembly {
    let brk := mload(0x40) // Special solidity slot with top
memory
    calldatacopy(brk, 0, calldatasize) // Copy data to memory
at offset brk
    let retval := delegatecall(sub(gas,150)
      ,target                          //address
      ,brk                             // memory in
      ,calldatasize                    // input size
      ,brk                             // reuse mem
      ,returnSize)                     // arbitrary return size
    // 0 == it threw, by jumping to  bad destination (00)
    jumpi(0x00,iszero(retval)) // Throw (access invalid code)
    return(brk,returnSize) // Return returnSize from memory to
the caller
  }
}
```

Pour cela, il faut bien faire attention au fait que le **Proxy**, et les différentes versions du contrat possèdent les mêmes attributs, dans le même ordre. La super-classe **Versionable** permet de mutualiser les attributs entre le **Proxy** et les contrats. De même, faire hériter le contrat v2 du contrat v1 permet de garantir que tous les attributs de la version 1 seront présents dans la version 2 (voir figure 10).

Attention, il faut bien comprendre ce qui se passe. Le proxy ne possède pas de méthode, mais va gérer les attributs des contrats v1 et v2, dont les méthodes sont présentes dans les implémentations correspondantes. Si on inspecte l'instance **ContractV1**, aucun attribut n'est valorisé. De même pour **ContractV2**.

La construction des contrats s'effectue en deux étapes, car le constructeur n'est pas une méthode comme les autres. En effet, construire une instance est un traitement spécial. Il est envoyé au contrat de numéro zéro de la blockchain. Le code du constructeur n'est pas disponible avec le contrat. Il n'est donc pas possible de le réutiliser pour initialiser le proxy.

Nous devons alors utiliser une méthode **init()** qui jouera le rôle de constructeur. Il ne faut pas oublier de l'invoquer, juste après la création de l'instance du contrat v1 et du contrat v2.

```
/**
 * Unit test.
 *
 * After created an instance,
 * - call 'init()'
 * - call 'doSomething()' return 1001
 * - call 'changeToV2'
 * - call 'doSomething()' return 1102
 * - call 'doOtherthing()' return 42
 */
contract A_UnitTest {
    event VersionChanged(Versionable version);

    /** A reference to a version of contract, via a
proxy. */
    ContractV1 private aContract; // FIXME: aContract

    /**
     * Initialise useContract with a ContractV1
encapsulated by a proxy.
     */
    function init() {
        // Create an instance of version 1
        ContractV1 v1=new ContractV1();
        // Encapsulate this instance in a proxy
        Proxy proxy=new Proxy(v1);
        // Cast the proxy to ContractV1
        aContract=ContractV1(proxy);
        // Init the instance via the proxy
        aContract.init();
    }
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

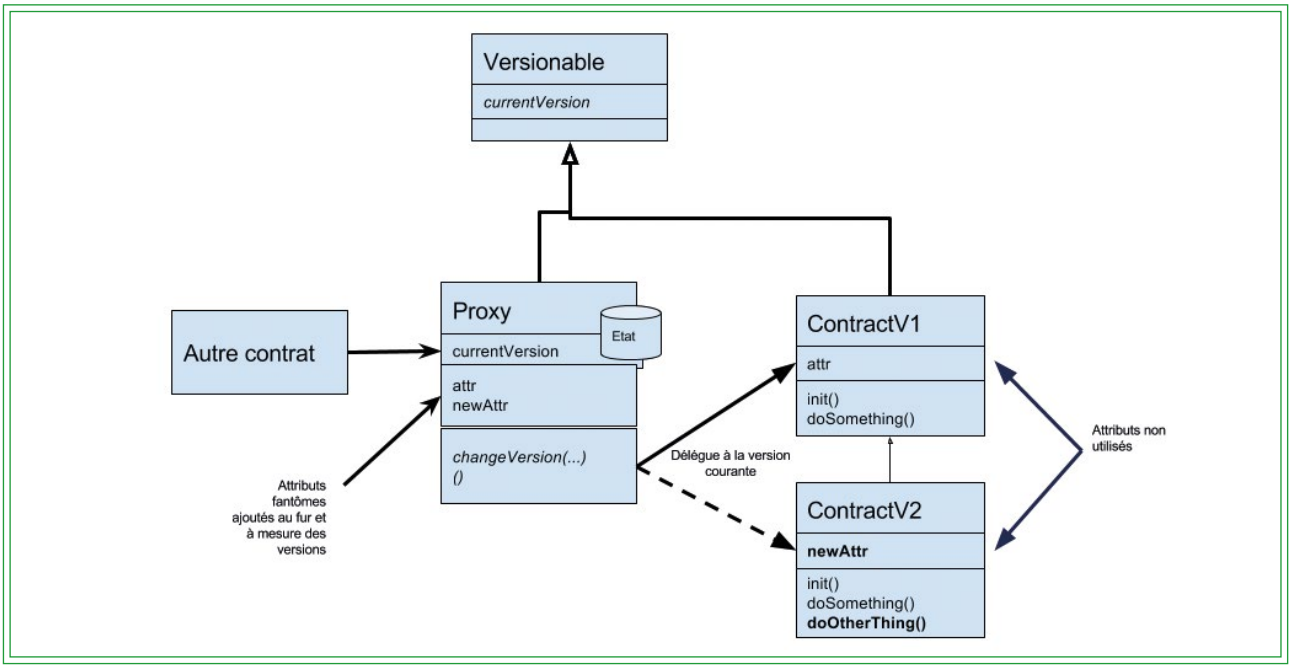


Fig. 10 : Proxy versionnable.

```
    /** Invoke a method valide with a contract v1 or v2. */
    function doSomething() constant returns(uint) {
        return aContract.doSomething();
    }}

   /** Invoke a method only valide with a contract v2. */
    function doOtherThing() constant returns(uint) {
        return ContractV2(aContract).doOtherThing();
    }

    /** Change to V2. */
    function changeToV2() {
        // Create an instance of version 2
        ContractV2 v2=new ContractV2();
        // Cast the current contract to be a Proxy
        Proxy proxy=Proxy(aContract);
        // Change the delegate instance to v2.
        proxy.changeVersion(v2);
        // Init the version 2 via the proxy.
        aContract.init();
    }
}
```

L'intégralité du code est ici **[8]**.

Ce dernier modèle répond à toutes les exigences :

- la référence du contrat n'évolue pas, même en cas de changement d'implémentation ;
- il est possible d'ajouter de nouvelles méthodes ou de nouveaux attributs dans une nouvelle version du contrat ;
- il n'est pas nécessaire de migrer les données entre les versions ;
- les événements émis par le code des contrats V1 ou V2 viennent bien du proxy.

Les inconvénients de cette approche sont les suivants :

- il n'est pas possible de supprimer un attribut (sauf à faire un **delete** sur ce dernier, dans le contrat V2) ;
- il est nécessaire de séparer le constructeur de la méthode **init()** ;
- cela présente un surcoût de 735 gaz pour chaque invocation.

8. MODIFIER UN CONTRAT AVEC L'ACCORD DE TOUS

Un contrat est généralement signé entre deux ou plusieurs parties. Conceptuellement, il est dangereux de permettre à une seule des parties de pouvoir apporter des modifications.

Par exemple, imaginons un contrat entre une entreprise Acme et un utilisateur Phil. Si Acme peut modifier le contrat qui lie Acme et Phil sans l'accord de Phil, cela casse toute la sécurité proposée par la blockchain. Acme peut unilatéralement récupérer tous les fonds,

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

sans que Phil puisse s'y opposer. Même sans volonté de nuire, si Acme se fait voler sa clé privée, le pirate peut modifier le contrat et voler les fonds, en se faisant passer pour Acme.

Donc, il est indispensable de protéger le contrat de toute modification sans l'accord de *n* parties parmi *m*.

Le *wallet* **Myst** proposé par Ethereum propose un portefeuille pouvant posséder plusieurs signataires. Il est possible de le paramétrer pour qu'un minimum de signataires soit nécessaire pour invoquer certaines méthodes.

L'idée est la suivante. Lorsqu'une méthode protégée est invoquée, la transaction décrivant l'appel est mise de côté. Si le même appel est effectué par une autre partie du contrat, avec exactement les mêmes paramètres, un compteur est incrémenté. Lorsque suffisamment de parties confirment leur souhait d'invoquer la même méthode, avec les mêmes paramètres, alors elle est réellement invoquée.

Le code permettant de gérer de multiples signatures est disponible dans les sources d'Ethereum. La lecture de ce code est très instructive. Un tableau de bits est construit lorsqu'une requête est demandée, avec un bit par signataire.

Comment utiliser ce code ? Il suffit d'hériter de `MultiOwned`, d'enrichir le constructeur et d'ajouter une protection aux méthodes sensibles.

Vous retrouverez ici **[9]** la version protégée du `Proxy`, permettant à plusieurs `owners` de se mettre d'accord sur la nouvelle version du contrat.

Pour tester ce code, il faut :

- créer une instance du test unitaire ;
- invoquer `init()` ;
- invoquer `doSomething()` pour récupérer `1001` (version 1 du traitement) ;
- demander le changement de version via l'utilisateur 1 (`user1_changeToV2()`) ;
- confirmer la demande de changement en invoquant de même `changeVersion()` avec strictement les mêmes paramètres, mais via l'utilisateur 2 (`user2_changeToV2()`) ;
- invoquer `doSomething()` pour récupérer `1102` (version 2 du traitement) ;
- et enfin, `doOtherthing()` pour confirmer qu'il est possible d'ajouter une nouvelle méthode.

Pour retrouver toutes les versions, c'est ici **[10]**.

CONCLUSION

Notre recherche de solutions, vers différentes pistes, nous a finalement amené à proposer une solution générique simple et de bon goût. Elle utilise tout plein de spécificités de la machine virtuelle et des choix d'implémentations de Solidity :

- Utiliser le fait qu'un Cast est possible vers n'importe quelle adresse de contrat. Cela permet de faire passer le `Proxy` comme un `ContractV1` ou `ContractV2`.
- Utiliser la méthode par défaut, lorsqu'une méthode n'est pas identifiée par le contrat.
- Utiliser l'attribut à l'adresse `0x40` pour identifier une zone mémoire disponible pour déléguer le traitement.
- Utiliser l'assembleur pour invoquer une méthode d'un autre contrat, et récupérer la valeur de retour avant de la propager à l'appelant.
- Utiliser la délégation pour que les événements des différentes implémentations viennent bien du Proxy.

Nous vous proposons une solution générique de quelques lignes, permettant de limiter au maximum les impacts de la mise à jour d'un contrat.

Finalement, Ethereum propose des contrats immuables, si on veut. Avec un peu d'effort, on peut également faire autrement. ■

RÉFÉRENCES

[1] Documentation de Solidity : http://solidity.readthedocs.io

[2] Test en ligne : http://ethereum.github.io/browser-solidity

[3] Demo Remplace.sol : https://goo.gl/9vbjw7

[4] Demo Remplace with DB.sol : https://goo.gl/8L1XSE

[5] Demo avenant_specifique.sol : https://goo.gl/7GFj0U

[6] Demo proxy_specifique.sol : https://goo.gl/s5n2ei

[7] Demo proxy_generique.sol : https://goo.gl/N3liZD

[8] Demo propagate_proxy.sol : https://goo.gl/cQLrFr

[9] Demo propagate_proxy_secu.sol : https://goo.gl/eJFbQU

[10] Toutes les demos : https://goo.gl/eJFbQU

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Professionnels, Collectivités, R & D...

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail : abopro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

UTILISEZ DROPBOX DEPUIS PYTHON

SÉBASTIEN CHAZALLET
[Ingénieur en logiciels libres]

MOTS-CLÉS : PYTHON, DROPBOX, TRANSFÉRER, TÉLÉCHARGER, PARTAGER

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

DropBox est un service de stockage en ligne extrêmement populaire qui vous permet de sauvegarder des fichiers, ou des répertoires, d'en gérer des versions successives, de les visualiser en ligne et les partager. Cet article va vous présenter un moyen d'automatiser toutes ces tâches.

L'entreprise **DropBox** fonctionne sur un modèle assez courant : elle offre un espace de stockage limité avec des fonctionnalités séduisantes, puis propose de payer pour plus d'espace ou plus de fonctionnalités. Elle offre également une série d'outils *open source* pour permettre la connectivité avec sa solution et ainsi contribuer à son adoption et à sa popularité.

En ce qui nous concerne, nous allons nous pencher sur son API qui nous permet de faire un certain nombre d'opérations par le code, ce sur quoi nous allons nous concentrer en précisant que nous nous intéresserons à la V2 de cette API et sur son utilisation pour Python 3 (certains des exemples du tutoriel que vous trouverez sur le site ne fonctionnent qu'en Python 2).

1. MISE EN PLACE

1.1 App

Pour pouvoir faire quoi que ce soit avec DropBox, il faut commencer par créer un compte. Une fois ceci fait, on peut créer une **app**. Cette notion d'app représente un point d'accès à DropBox.

Il faut vous demander ce que vous souhaitez faire avec le code que vous allez écrire : si vous souhaitez gérer

toutes vos données, vous pouvez choisir l'option **Full dropbox** ; si au contraire vous souhaitez compartimenter les données de votre app, il vous faut choisir l'option **App folder**.

La première figure montre la page permettant de gérer cette première étape.

Une fois ceci fait, on peut accéder à une page qui nous permet d'avoir un *token* d'accès (voir figure 2, page suivante). Il faut cliquer sur le bouton mis en évidence dans l'image suivante pour générer ce code. Pour information, les champs **App key** et **secret** ne sont plus importants, ils sont surtout utiles pour la V1 de l'API, que nous ne présenterons pas ici.

Dans la suite de l'article, nous appellerons cette valeur **APP_TOKEN**. Bien entendu, vous vous assurerez de garder cette clé secrète, ce qui signifie aussi ne pas l'écrire en dur dans votre code et la valider et pousser sur GitHub...

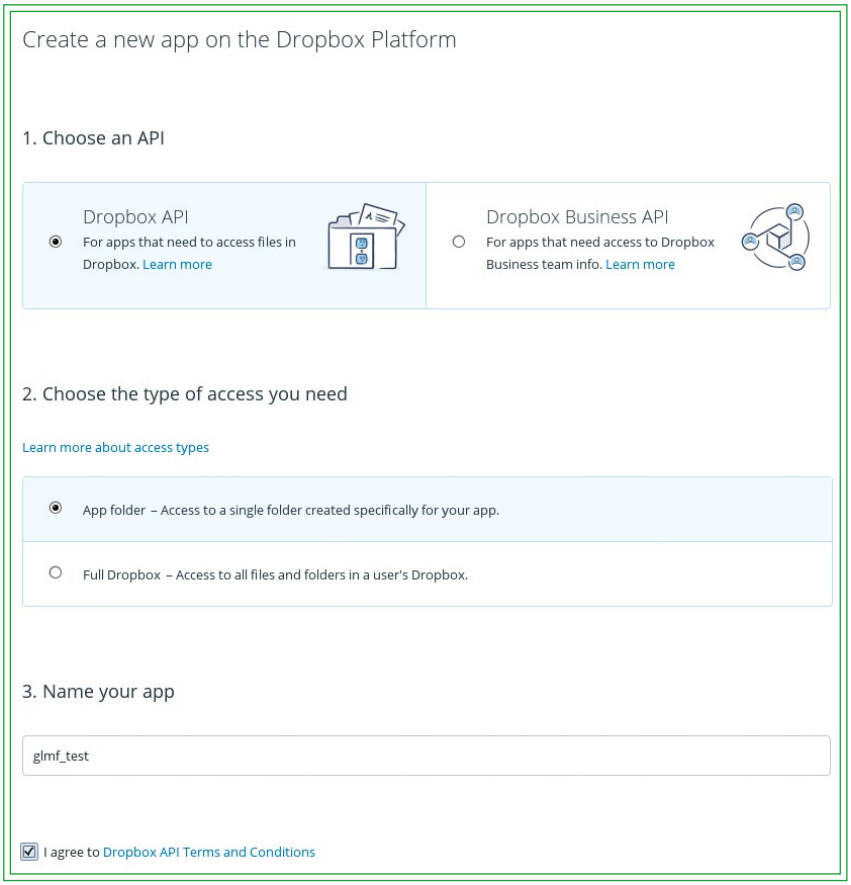


Fig. 1 : Choix de l'option de gestion des données.

1.2 Python

Il nous faut maintenant installer la bibliothèque de DropBox pour Python, ce qui est aussi simple que d'habitude. Il suffit de faire, en tant qu'administrateur :

```
# pip3.6 install dropbox
```

On pourra vérifier que l'installation s'est bien passée :

```
$ python3.6
>>> import dropbox
```

Si cela fonctionne, c'est que tout va bien. On peut maintenant se connecter, ce qui se fait en une instruction :

```
>>> dbx = dropbox.Dropbox("APP_TOKEN")
```

En l'absence de message d'erreur, vous êtes connecté. Sinon, il faut se référer au message pour savoir comment résoudre le problème. DropBox utilise **OAuth** pour gérer la sécurité de l'authentification.

Voici maintenant comment obtenir les informations relatives à votre compte :

```
>>> dbx.users_get_current_account()
FullAccount(account_id='dbid:XXX',
name=Name(given_name='Sébastien',
surname='Chazallet', familiar_name='Sébastien', display_name='Sébastien Chazallet',
abbreviated_name='SC'),
email='s.chazallet@mail-server.com', email_verified=True,
disabled=False, locale='en',
referral_link='https://db.tt/XXX', is_paired=False, account_type=AccountType('basic',
None), profile_photo_url=None,
country='FR', team=None, team_member_id=None)
```

Deux remarques ici, lesquelles seront vraies pour tout l'article. D'une part, la première partie du nom d'une fonction désigne un module de fonctionnalités de DropBox : (**users**, **files**, **oauth**, **async**, **sharing**...). Ceci vous permettra de vous y retrouver dans la documentation officielle **[1]**.

D'autre part, la réponse des différentes fonctions est un objet spécifique, dont le nom est particulièrement explicite sur

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

le type de contenu, et dont le contenu est exprimé sous une forme assimilable à un dictionnaire.

Dans l'exemple ci-dessus, vous pouvez visualiser les différentes valeurs associées à mon compte (altérées légèrement pour des questions évidentes).

Il existe aussi une fonction qui permet très simplement de vérifier l'espace qu'il nous reste :

```
dbx.users_get_space_usage()
```

2. GÉRER DES FICHIERS

2.1 Lister le contenu d'un répertoire

Voici comment lister le contenu du répertoire racine de l'application :

```
>>> dbx.files_list_folder('')
ListFolderResult(entries=[],
cursor='XXX', has_more=False)
```

Le contenu du répertoire se trouve dans l'objet **entries**. Nous verrons tout à l'heure de quel type d'objet il s'agit. En l'état, notre répertoire est simplement vide. Petite précision qui a son importance : il s'agit d'une application réseau et donc avoir un temps de réponse acceptable est extrêmement important. D'où le fait que la quantité de données que le serveur va nous renvoyer est limitée. D'où la propriété **has_more**. Si cette propriété est vraie, alors il y a plus sur le serveur et il faut appeler la fonction **files_list_folder** en lui passant en paramètre le curseur. D'où la présence de ces deux objets dans la réponse donnée par le serveur DropBox.

Cette manière de fonctionner se retrouve aussi dans d'autres fonctionnalités et c'est la raison pour laquelle nous avons des curseurs.

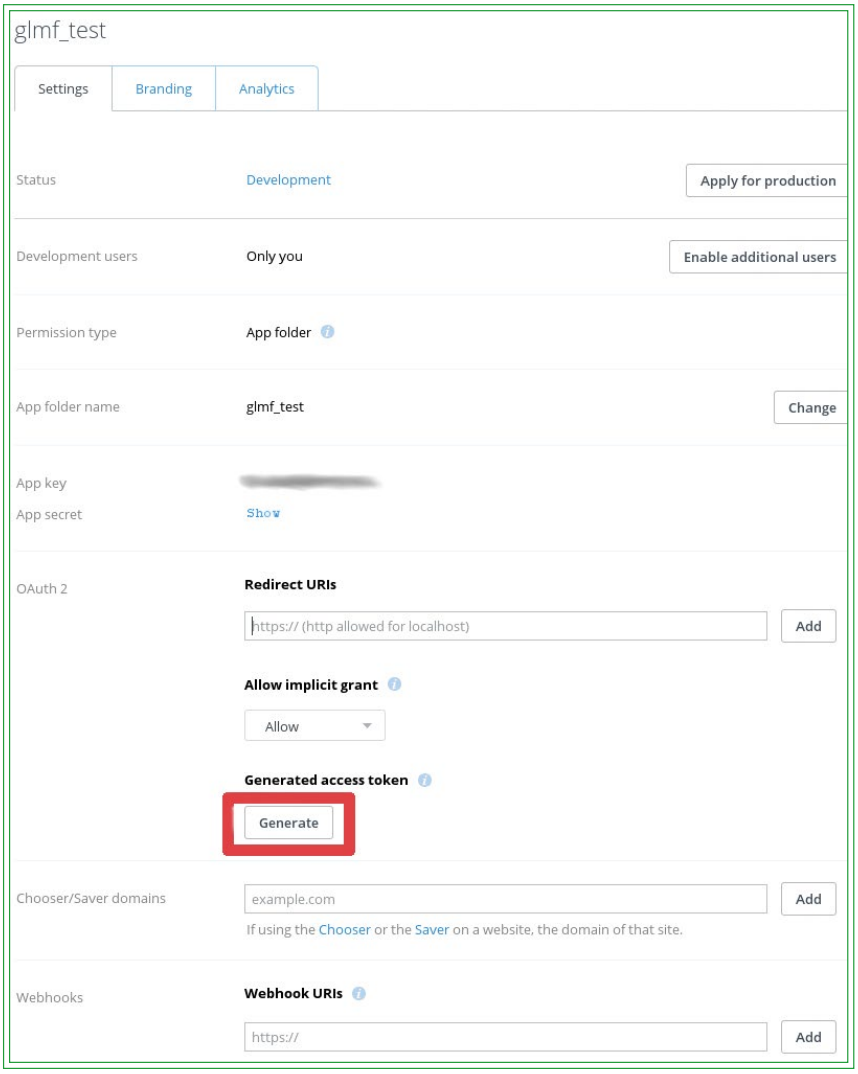


Fig. 2 : Génération du token d'accès.

2.2 Uploader un fichier

Nous allons maintenant ajouter à ce répertoire un nouveau fichier :

```
with open("/home/inspyration/todolist.txt", "rb") as f:
    dbx.files_upload(f.read(), "/ma_todolist.txt")
```

Vous aurez remarqué que le fichier qui est téléchargé est un bête fichier texte. Pourtant, il est ouvert en mode binaire. La raison en est simple : le contenu est envoyé sur le réseau, et le réseau ne travaille qu'avec des octets. Pas des nombres ou des chaînes de caractères ou quoi que ce soit d'autre, des octets. Nous devons donc d'une manière ou d'une autre transférer des octets et quel moyen plus évident de le faire qu'en ouvrant notre fichier avec l'option **rb** ?

De plus, on utilise la méthode **files_upload** qui prend comme premier paramètre le contenu du fichier et pour second paramètre le chemin de ce dernier sur le serveur.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Le chemin d'un fichier (ou d'un répertoire, par ailleurs) commence toujours par **/**, le **/** représentant la racine du répertoire de l'application (si paramétré en **App Folder**) ou la racine de votre DropBox sinon.

On termine enfin cette partie en précisant que la méthode **files_upload** renvoie ceci :

```
FileMetadata(name='ma_todolist.txt', id='id:XXX',
client_modified=datetime.datetime(2017, 4, 17, 9, 2, 23),
server_modified=datetime.datetime(2017, 4, 17, 9, 2, 23),
rev='111', size=115, path_lower='/ma_todolist.txt', path_
display='/ma_todolist.txt', parent_shared_folder_id=None,
media_info=None, sharing_info=None, property_groups=None,
has_explicit_shared_members=None, content_hash='1b5bd00ad
14518be019d14c49435a6cebd30ff8bc8b9b0833336829dc86081b2')
```

Il s'agit d'un objet contenant toutes les métadonnées de notre fichier : son identifiant, son nom, son emplacement, sa date de modification... mais surtout le **content_hash** qui va nous permettre de vérifier que le fichier n'a pas été corrompu pendant son transfert (depuis ou vers le serveur).

Si à cet instant, on décide à nouveau de parcourir la liste des fichiers à la racine, cet objet serait celui que l'on trouverait dans la propriété **entries** évoquée précédemment.

On peut aussi la récupérer ainsi :

```
>>> metadata = dbx.files_get_metadata("/ma_todolist.txt")
```

2.3 Vérifier le transfert

Pour vérifier qu'un transfert du client vers le serveur ou du serveur vers un client a bien fonctionné, il suffit de créer un hashage du fichier des deux côtés et de comparer le résultat. Il est à peu près impossible qu'une erreur de transfert génère néanmoins la même hash.

De son côté, le serveur DropBox va générer ce hash. Il nous suffit donc de générer le même hash (en utilisant la même méthode) côté client. Pour cela, on a la documentation de DropBox qui nous explique comment faire **[2]** ou encore un petit projet dédié sur GitHub **[3]**.

Nous allons utiliser ce dernier, en le clonant :

```
$ git clone https://github.com/dropbox/dropbox-api-
content-hasher
```

Il nous suffit maintenant d'utiliser la ligne de commandes pour vérifier un fichier (ce qui peut être utile hors contexte de la création de votre application) :

```
$ cd dropbox-api-content-hasher/python
$ python3 hash_file.py ~/todolist.txt
```

Si on souhaite pouvoir faire la même chose en Python, il suffit de regarder le code de ce dernier fichier :

```
from dropbox_content_hasher
import DropboxContentHasher
hasher = DropboxContentHasher()
with open(fn, 'rb') as f:
    while True:
        chunk = f.read(1024)
# or whatever chunk size you
want
        if len(chunk) == 0:
            break
        hasher.update(chunk)
print(hasher.hexdigest())
```

2.4 Créer une version

Si nous décidons de remplacer le fichier précédent par un nouveau, portant le même nom, nous risquons de tomber sur cette erreur :

```
>>> with open("/home/inspyration/
todolist.txt", "rb") as f:
    metadata = dbx.files_upload(f.
read(), "/ma_todolist.txt")
Traceback (most recent call last):
[...]
dropbox.exceptions.ApiError:
ApiError([...], UploadError('path',
UploadWriteFailed(reason=WriteError('
conflict', WriteConflictError('file',
None)), [...])))
```

Première remarque : les objets exception sont également assimilables à des dictionnaires et comportent pas mal d'informations.

Ensuite, le message nous explique qu'il existe déjà un fichier portant ce nom. Et par défaut, dans DropBox, en cas de conflit, on génère une exception.

Dans notre cas, on veut réellement remplacer notre fichier. Nous allons donc le préciser :

```
with open("/home/inspyration/
todolist.txt", "rb") as f:
    metadata = dbx.files_
upload(f.read(), "/ma_todolist.
txt", mode=dropbox.files.
WriteMode("overwrite"))
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Nous allons ainsi créer une seconde version de notre fichier et cela se verra dans les métadonnées par le fait que l'on a un numéro différent dans la propriété **rev**.

Voici maintenant comment lister les différentes versions d'un fichier :

```
>>> dbx.files_list_revisions("/ma_todolist.txt")
ListRevisionsResult(is_deleted=False,
entries=[...])
```

3. OPÉRATIONS COURANTES

3.1 Créer/supprimer

On peut facilement créer des répertoires :

```
dbx.files_create_folder("/new")
```

On peut supprimer un fichier ou un répertoire ainsi :

```
dbx.files_delete("/ma_todolist.txt")
```

Cette opération aura pour résultat de changer la métadonnée pour y indiquer que le fichier n'est plus disponible. Le fichier ne sera plus visible dans l'interface web. Cependant, une sauvegarde en est conservée.

On peut aussi supprimer de manière permanente un répertoire ou un fichier ainsi :

```
dbx.files_permanently_delete("/ma_todolist.txt")
```

Cette opération n'est pas réversible.

Enfin, on peut aussi travailler par lot :

```
lauch = dbx.files_delete_batch([metadata1, metadata2])
```

Ceci a l'avantage d'être asynchrone. Vous recevez une notification comme quoi le serveur a compris ce qu'il devait faire ainsi que l'identifiant de la tâche et vous pouvez vaquer à vos occupations. Lorsque cela vous sied, vous pouvez lui demander où il en est :

```
dbx.files_delete_batch_check(lauch.get_async_job_id())
```

3.2 Copier, couper, coller

Ces opérations font partie de notre quotidien. Avec DropBox, elles sont également réalisables, et ceci de manière assez directe :

```
dbx.files_copy("/ma_todolist.txt", "/ma_
todolist2.txt")
```

Pour déplacer un fichier :

```
dbx.files_move("/ma_todolist.txt", "/ma_
todolist2.txt")
```

De la même manière que tout à l'heure, il existe un moyen de faire plusieurs opérations, de manière asynchrone :

```
dbx.files_copy_batch([dropbox.files.
RelocationPath("/ma_todolist.txt", "/ma_
todolist2.txt") , ...])
dbx.files_move_batch([dropbox.files.
RelocationPath("/ma_todolist.txt", "/ma_
todolist2.txt") , ...])
```

Là encore, il existe **files_copy_batch_check** et **files_move_batch_check** pour vérifier le statut de la copie ou du déplacement.

4. TÉLÉCHARGER

4.1 Télécharger un fichier

Pour télécharger un fichier, rien de plus simple :

```
metadata, response = dbx.files_
download("/ma_todolist.txt")
```

On est déjà familier avec l'objet métadonnée, on l'est probablement également avec l'objet réponse, puisqu'il s'agit d'un objet obtenu via l'utilisation de l'excellent module **requests** de Python **[4]**. On peut donc l'utiliser comme suit :

```
with open("my_new_todolist.txt", "w") as f:
    f.write(response.content)
```

On notera qu'il est possible de faire plus direct :

```
dbx.files_download("/ma_todolist.txt",
"~/my_new_todolist.txt")
```

On notera qu'il est aussi possible de demander une révision particulière en utilisant cette URL :

```
dbx.files_download("/ma_todolist.txt",
"rev:aaaaaa")
```

D'ailleurs, on peut aussi faire ceci :

```
dbx.files_download("/ma_todolist.txt",
"id:aaaaaa")
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

5. PARTAGER

La fonctionnalité de partage est l'une des plus complexes de DropBox. D'une part parce qu'il faut savoir comment elle fonctionne, quelles sont les différentes possibilités et d'autre part, car il faut être capable d'identifier la personne avec qui l'on veut partager quelque chose.

Comme vous pouvez le remarquer plus haut, les métadonnées du fichier contiennent une propriété `shared_folder_id` qui est à `None`. Ceci signifie tout simplement que ce répertoire n'est pas partagé.

Pour y remédier, il faut commencer par ceci :

```
launch = dbx.sharing_share_folder("/new")
```

Un répertoire usuel sera partagé de manière synchrone. S'il s'agit d'un répertoire au contenu important, ceci sera fait de manière asynchrone. Il conviendra alors d'attendre que le travail soit terminé en vérifiant ceci :

```
lauch.is_complete()
```

Et en ne le vérifiant pas trop souvent non plus, on ne souhaite pas bombarder le serveur DropBox de requêtes inutiles.

Une fois que cela est terminé, on peut récupérer les nouvelles métadonnées du répertoire :

```
metadata = launch.get_complete()
```

On pourra constater que `shared_folder_id` n'est plus à `None`.

La première partie du travail est réalisée : le répertoire est prêt à être partagé.

Il faut maintenant identifier celui ou ceux avec qui on veut le partager. Cela se fait grâce à leur e-mail (celui avec lequel ils ont créé leur compte DropBox) :

```
member_selector = dropbox.sharing.
MemberSelector.email("lui@exemple.org")
```

Il faut maintenant vérifier dans la documentation les différents niveaux d'accréditation et en choisir un :

```
access_level = dropbox.sharing.AccessLevel.editor
```

Il existe aussi `owner`, `viewer` et `viewer_no_comment`.

On peut maintenant terminer le travail :

```
dbx.sharing_add_folder_member(metadata.shared_
folder_id,
                              [dropbox.sharing.
AddMember(member_selector, access_level)])
```

CONCLUSION

Nous avons présenté ici les principales fonctionnalités de DropBox : se connecter, *uploader* ou télécharger un fichier, faire quelques traitements par lot ou encore copier ou déplacer des fichiers.

Avec ce que nous avons vu ici, nous sommes capables de faire l'essentiel et de sauvegarder nos fichiers, les partager ou encore les organiser.

Pour aller plus loin, il faut d'une part être familier avec DropBox, puisque tous les principes que nous connaissons en utilisant l'interface web sont aussi vrais lorsque l'on code (on peut faire les mêmes actions, et il y a les mêmes conséquences).

Par exemple, on peut rajouter ou supprimer des utilisateurs sur un fichier, mais on doit aussi gérer le fait qu'il y a des utilisateurs qui héritent des droits positionnés sur le répertoire contenant le fichier.

La documentation en ligne est assez complète, mais ce n'est qu'une documentation purement technique. Cependant, si tout va bien, cet article aura réussi à vous mettre le pied à l'étrier et à donner un contexte et un cadre à cette documentation, ce qui devrait vous permettre de combler les vides. ■

RÉFÉRENCES

[1] Site officiel de la documentation de Dropbox : http://dropbox-sdk-python.readthedocs.io

[2] Technique de hashage : https://www.dropbox.com/developers/reference/content-hash.

[3] Hasher : https://github.com/dropbox/dropbox-api-content-hasher.

[4] Requests : http://docs.python-requests.org/en/master/.

POUR ALLER PLUS LOIN

Le SDK de DropBox est disponible sur GitHub et il contient un exemple montrant l'essentiel de ce qu'il faut savoir pour *uploader* du contenu. Il est disponible à cette adresse : https://github.com/dropbox/dropbox-sdk-python/blob/master/example/updown.py. Décortiquer ce code vous sera utile pour vous exercer sur cette API.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

PHP-RBAC : GÉREZ LES DROITS DE VOTRE APPLICATION À L'AIDE DE RÔLES

STÉPHANE MOUREY
[Mousse sur le Seeraiwer]

MOTS-CLÉS : DROITS, PHP, RÔLES, ACCÈS, AUTORISATION, RBAC

La gestion fine des droits des utilisateurs au sein des applications métier est un calvaire pour l'administrateur. Facilitez-lui la vie en adoptant le principe des rôles au sein de vos propres développements.

PHP-RBAC **[1]** est une bibliothèque **PHP** destinée à faciliter la mise en place d'une gestion des droits des utilisateurs basée sur l'attribution de rôles. Ce projet, lancé en 2008 au sein de **jframework**, un framework PHP maintenant inactif, évolue aujourd'hui au sein de la fondation **OWASP [2]** (*Open Web Application Security Project*), une organisation à but non lucratif créée en 2001 dont le but est l'amélioration de la sécurité des logiciels. PHP-RBAC obéit au *NIST* **[3]** *Level 2 Standard Hierarchical Role Based Access Control*, c'est-à-dire au standard de niveau 2 concernant le contrôle d'accès par rôles hiérarchiques du *National Institute of Standards and Technology*. Tout cela fait sérieux **[4]**. Mais quel problème s'agit-il de résoudre ?

La gestion classique des droits, par liste de contrôle d'accès (*ACL, Access Control List*), qui fonctionne en attribuant des autorisations aux utilisateurs ou à des groupes d'utilisateurs, se révèle inefficace dès lors que leur nombre devient conséquent et que l'application offre de multiples possibilités. Souvent, l'interface proposée pour ce faire prend la forme d'un tableau rempli de cases à cocher, avec une colonne pour chaque autorisation et une ligne pour chaque utilisateur en cours d'édition. Cela fait vite beaucoup de cases à cocher, et il peut arriver facilement de se tromper de ligne ou de colonne. Ainsi, ce système devient vite très chronophage et source d'erreurs, erreurs dont la gravité est relative à l'importance de l'application. Leurs administrateurs en savent quelque chose.

1. LES RÔLES

1.1 Concept de base

Une solution consiste donc à s'appuyer sur des rôles. Mais de quoi s'agit-il ? Partant du constat qu'au sein d'une

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

entreprise ou d'une administration, si le personnel évolue par recrutements, départs, mouvements internes, les fonctions à occuper obéissent à une logique beaucoup plus stable. Ainsi, si la personne chargée de l'accueil n'est pas toujours la même, la fonction et les droits afférents ne changent pas, et il est nécessaire de les attribuer à toutes les personnes en charge de la fonction. Au sein de l'application, les fonctions sont décrites à travers le concept de rôle, sans que les deux notions ne se confondent. Le rôle est un ensemble d'autorisations regroupées sous une appellation signifiante pour l'administrateur. Pour accorder les autorisations correspondantes à un utilisateur, l'administrateur lui affecte ce rôle. Selon sa fonction, un utilisateur peut cumuler plusieurs rôles.

La gestion par ACL telle que nous la connaissons ne disparaît pas pour autant, mais elle s'effectue désormais sur les rôles et non sur les utilisateurs : la mise en place des droits et des rôles est plus simple et plus rapide, il n'y a pas lieu de les modifier souvent, et le risque d'erreur en est diminué, d'autant plus qu'il est improbable ensuite de se tromper dans l'affectation d'un rôle à un utilisateur. Le bénéfice est donc double, on y gagne en temps et en sécurité. Par ailleurs, le principe des rôles est tel qu'il est compatible avec la gestion classique par ACL : il est ainsi possible de le mettre en place dans une application en production en maintenant les droits déjà attribués. Les gestions par ACL et par rôles peuvent d'ailleurs cohabiter au sein d'une même application, ce qui peut se révéler pratique à l'occasion, mais néfaste à long terme : on a tôt fait d'attribuer un droit hors d'un rôle pour aller vite, et d'oublier ensuite de le retirer lorsqu'il n'a plus lieu d'être. Il n'en reste pas moins que la compatibilité avec les concepts mis en œuvre antérieurement contribue à l'élégance de la solution.

1.2 Notion de hiérarchie

Ce que nous venons de dire suffit à définir un contrôle d'accès basé sur des rôles *plats* : les rôles n'entretiennent pas de relation entre eux. L'idée a été rendue plus subtile en introduisant la notion de *hiérarchie*. Bien qu'elles ne soient pas sans similarité et que bien souvent la première imite la seconde dans sa structure, la hiérarchie des rôles dans l'application n'est pas identique à celle des fonctions dans l'entreprise. En réalité, l'idée derrière la hiérarchie des rôles se rapproche bien plus du concept d'héritage en programmation objet, aussi emploierai-je ce vocabulaire pour décrire les relations entre les rôles.

L'idée est que certains rôles peuvent hériter des droits d'autres rôles. D'un point de vue méthodologique, il faut alors commencer par la définition des rôles qui ont le moins de droits pour aller vers ceux qui en ont le plus. Pour une application web, on commencera par un rôle *guest*, correspondant à un visiteur non authentifié. Au sein de cette application, il aura au moins le droit de consulter les pages d'accueil, d'authentification, d'erreur d'authentification et d'autres erreurs comme une page non trouvée ou une tentative d'accès non autorisée. Le rôle *user*, correspondant à un utilisateur authentifié, héritera des droits de l'utilisateur *guest*, auxquels il faudra au moins ajouter ceux de consulter la page de clôture de session, ainsi que celle affichant son profil. Les autres droits à attribuer à ces rôles, ainsi que les autres rôles nécessaires, dépendront de votre application. En dernier viendra le rôle *admin*, l'administrateur, qui lui fera exception et n'héritera de personne, car il ne faudrait pas lui supprimer un de ses droits par erreur en travaillant sur un des rôles dont il hérite. Naturellement, selon la complexité de votre application et celle de la structure dont il faut rendre compte, vous aurez à créer tous les rôles intermédiaires nécessaires.

Remarquez que le sommet de la hiérarchie des rôles ne correspond pas à celui de l'entreprise. En effet, on imagine mal le PDG prendre en charge la gestion des comptes des utilisateurs : cette tâche est celle de personnels techniques qui ont, au sein de l'application, une responsabilité plus importante que lui.

Une fois que vous aurez créé tous les rôles nécessaires, vous n'aurez plus qu'à les attribuer à vos utilisateurs. Notez que rien n'empêche un rôle d'hériter de plusieurs rôles, de même que rien n'interdit d'attribuer plusieurs rôles à un utilisateur. Dans ces cas, les autorisations se cumulent.

2. INSTALLATION DE RBAC

PHP-RBAC est donc la solution recommandée pour mettre en œuvre une solution de ce type au sein d'applications PHP. Voyons d'abord comment l'installer.

Vous pouvez obtenir les sources de différentes façons, soit en les téléchargeant directement **[5]**, soit en téléchargeant les sources depuis **GitHub [6]**. Mais, en tant que développeur PHP, vous devriez préférer l'installation par **Composer [7]**.

Pour cela, commencez par éditer le fichier **composer.json** de votre projet, créez-le au besoin, de façon à ce qu'il comporte au moins ceci :

```
{
  "require": {
    "owasp/phprbac": "2.0.*@dev"
  }
}
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Puis, lancez la commande **composer**. Si Composer est installé globalement, cela se fera ainsi :

```
$ composer install
```

Si Composer est installé localement à votre projet, cette commande sera plus appropriée :

```
$ php composer.phar install
```

Une fois cette étape réalisée avec succès, vous pouvez lancer l'installation proprement dite. Pour cela, ouvrez votre navigateur à l'adresse du script d'installation de PHP-RBAC. Si le dossier racine de votre projet est aussi celui de votre serveur web et si vous avez procédé à l'installation à l'aide de Composer, l'adresse sera la suivante : http://localhost/rbac/vendor/owasp/phprbac/PhpRbac/install.php.

Sur cette page, vous aurez simplement à entrer les paramètres de connexion à la base de données. Deux types sont supportés : **MySQL** (à travers l'adaptateur déprécié MySQL, ou MySQLi qui est recommandé) et **SQLite**. SQLite est la solution que je vais préférer ici, tout à fait adaptée pour réaliser des tests, comme ici, ou pour une application auto-hébergée. En production, pour une structure plus large, susceptible d'utiliser PHP-RBAC sur plusieurs sites et dans plusieurs applications, MySQL s'impose.

Les champs du formulaire qui vous sont présentés sont conçus pour une base MySQL, et il y a un manque de documentation pour SQLite. Qu'à cela ne tienne, après quelques menues recherches je suis en mesure de vous dire que le mieux est de les renseigner tous, sachant que le nom de la base de données sera utilisé pour définir le nom du fichier SQLite, en lui ajoutant simplement l'extension **.sqlite3**. Si vous optez pour une utilisation de SQLite en production pour PHP-RBAC, on ne saurait trop vous recommander de placer ce fichier dans un emplacement accessible à PHP, mais inaccessible à **Apache**...

Si une simple page blanche s'affiche à l'issue de ce formulaire, l'installation ne s'est pas déroulée correctement, et vous avez une configuration de votre serveur web dite « de production » et non « de développement », ce qui est le cas par défaut sur la plupart des distributions Linux aujourd'hui. En production, les messages d'erreurs ne sont pas destinés à être affichés à l'utilisateur et sont enregistrés dans des fichiers de logs. La procédure d'installation de PHP-RBAC, sous prétexte qu'il s'agit d'une bibliothèque de fonctions et non d'un produit fini, n'intercepte pas les exceptions. Du coup, au moindre problème non prévu, c'est la page blanche. La plongée dans les logs est alors votre seule issue. Si vous êtes comme moi, vous devrez modifier les droits sur le fichier SQLite où vous avez décidé de stocker vos données, ainsi que sur le fichier **/vendor/owasp/phprbac/PhpRbac/database/database.config**, de façon à permettre à l'utilisateur système qui fait fonctionner votre serveur web (**www-data** par exemple) d'y écrire.

Une fois cela fait, vous devrez supprimer le fichier **/vendor/owasp/phprbac/PhpRbac/install.php** afin de prévenir les risques de sécurité liés à cette procédure d'installation. Arrivé à ce point, si votre projet utilisait déjà Composer pour ses autres dépendances, vous ne devriez plus avoir rien à faire pour pouvoir commencer à utiliser les fonctionnalités proposées par PHP-RBAC. Si tel n'est pas le cas, vous devrez inclure l'autoloader défini par Composer à l'endroit le plus approprié de votre application, de façon à pouvoir charger automatiquement vos dépendances. Cela sera fait avec cette ligne :

```
require_once 'vendor/autoload.php';
```

À défaut, PHP-RBAC vous propose son propre autoloader :

```
require_once '/path/to/PhpRbac/autoload.php';
```

3. UTILISATION

À partir de là, nous pouvons créer un objet **Rbac** et commencer à travailler :

```
$rbac = new \PhpRbac\Rbac();
```

3.1 Création des rôles et des droits

Les créations de rôles et de droits suivent la même logique. L'objet **$rbac** possède deux objets en propriétés, **$Permissions** et **$Roles** destinés respectivement à recevoir les droits et les rôles. Ces deux objets partagent l'essentiel de leurs méthodes, et on les désignera par le terme d'« entité » lorsque l'on décrira un processus pouvant s'appliquer indifféremment à l'un ou l'autre des deux objets.

Au chapitre 1.3, nous avions évoqué, entre autres choses, la possibilité pour un utilisateur non identifié d'avoir accès à la page de *login*, et pour un utilisateur authentifié de pouvoir terminer sa session. Commençons donc par définir ces deux permissions et ces deux rôles.

```
$perms['login'] = $rbac->Permissions->
add('login', 'Can access authentification page');
$perms['logout'] = $rbac->Permissions->
add('logout', 'Can end working session');
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
$roles['guest'] = $rbac->Roles->
add('guest','non-authenticated user');
$roles['user'] = $rbac->Roles->
add('user','authenticated user');
```

Les entités proposent une méthode **add** qui permettent d'ajouter une entité. Cette méthode prend en arguments le titre de l'entité suivi de sa description succincte et renvoie son identifiant. Remarquez que j'ai pris soin de stocker ces identifiants dans un tableau : nous allons en effet en avoir besoin.

3.2 Association des rôles et des droits

Procédons maintenant à l'association entre ces permissions et ces rôles :

```
$rbac->assign($roles['guest'],$perms['login']);
$rbac->assign($roles['user'],$perms['logout']);
```

Cela est donc réalisé à l'aide de la méthode **assign** qui demande l'identifiant du rôle en premier argument, et l'identifiant de la permission en second.

3.3 Association des utilisateurs et des rôles

Maintenant que nos entités sont associées, il nous reste à affecter des rôles aux utilisateurs. PHP-RBAC ne prend pas en charge la gestion des utilisateurs proprement dite. Il présuppose son existence et en attend au moins un élément : un identifiant unique pour chaque utilisateur. Normalement, la base de données de PHP-RBAC attend un entier, mais si votre application utilise des UUID **[8]**, il suffira de modifier la nature de la colonne correspondante (**UserID** dans la table **PREFIX_userroles**) dans la base de données pour que tout fonctionne sans autre modification, avec tout de même un risque de dégradation des performances.

Donc, charge à vous de fournir les identifiants de vos utilisateurs. Supposons que notre application fonctionne avec un utilisateur fictif, utilisé tant que la personne utilisant le site ne s'est pas encore authentifiée. Son identifiant sera le **2**. Tant que nous y sommes, ajoutons un autre utilisateur, avec pour identifiant le **3**. Voici comment réaliser les associations correspondantes :

```
$rbac->Users->assign($roles['guest'],2)
$rbac->Users->assign($roles['user'],3)
```

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

ACTUELLEMENT DISPONIBLE !
LINUX PRATIQUE n°102

J'APPRENDS À PROGRAMMER
AVEC PROCESSING !

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

3.4 Autres manipulations

Une fois que tout est en place, il peut être utile d'effectuer des modifications. Comme le contexte d'exécution ne sera plus le même, nous ne disposerons plus de nos tableaux **$roles** et **$perms**. Il nous faut donc en premier lieu une méthode pour récupérer les identifiants des entités depuis leurs noms :

```
$permId = $rbac->Permissions->returnId('login');
$roleId = $rbac->Roles->returnId('guest');
```

Dès lors, il nous est possible de modifier leurs noms et leurs descriptions :

```
$rbac->Permissions->edit($permId,'auth','Accès à
la page d\'authentification');
$rbac->Roles->edit($roleId,'invite','Utilisateur
anonyme');
```

Ou encore de les détruire :

```
$rbac->Permissions->remove($permId);
$rbac->Roles->remove($roleId);
```

Pour briser l'association d'une permission à un rôle, il n'est par contre pas nécessaire de connaître leurs identifiants, leurs noms suffisent :

```
$rbac->Permissions-
>unassign('invite','logout');
// ou
$rbac->Roles->unassign('invite','logout');
```

Remarquez que cette opération peut se réaliser indifféremment sur l'objet **$Permissions** ou sur **$Roles**.

Il nous faut également pouvoir supprimer l'attribution d'un rôle à un utilisateur. Cela sera fait avec la méthode **unassignRoles** de l'objet **$Users** qui attend deux arguments : le titre du rôle et l'identifiant de l'utilisateur :

```
$rbac->Users->unassign('admin',$userId)
```

3.5 Validation d'accès

N'oublions pas le plus important, le but même de ce travail : autoriser ou interdire l'accès d'un utilisateur à une ressource. Pour cela, il faut utiliser la méthode **check**. Elle attend deux arguments : le titre de la ressource et l'identifiant de l'utilisateur :

```
if (!$rbac->check('admin',$userId)){
 die ('Vous n\'avez pas l\'autorisation
d\'accéder à cette ressource !');
}
```

Il est même possible d'envoyer une erreur 403 et d'interrompre le traitement en cas de tentative d'accès non autorisée avec une seule ligne de code :

```
$rbac->enforce('admin',$userId);
```

Il est également possible de vérifier si un utilisateur dispose d'un rôle ou de lister tous ces rôles :

```
if (!$rbac->Users->hasRole($rbac->Roles->
returnId('admin'),$userId)) {
 print 'L\'utilisateur ne dispose pas du rôle
"admin". Voici les rôles dont il dispose :</p>';
 foreach ($rbac->Users->allRoles($userId) as
$role) {
  print '<li>'.$role['Title'].'
('.$role['Description'].')</li>'
 }
}
```

4. ET LA HIÉRARCHIE ?

Nous sommes maintenant en mesure de réaliser une application où la gestion des droits s'appuie sur des rôles et les permissions qui leur sont affectées, mais celle-ci est encore *plate*, au sens où nous n'avons pas encore examiné comment mettre en place une hiérarchie des rôles et des permissions. Selon votre application, il ne sera peut-être pas nécessaire d'en venir là. Prenez toutefois le temps d'y réfléchir. En effet, la hiérarchisation s'effectue en même temps que la création des entités et PHP-RBAC ne fournit pas d'outil pour hiérarchiser des entités déjà existantes.

Pour créer des entités hiérarchisées, il nous faut abandonner la méthode **add** pour lui préférer la méthode **addPath** qui est un peu plus complexe. Celle-ci attend deux arguments.

Le premier est une liste des rôles ou de permissions dans leur ordre hiérarchique, en commençant par le plus élevé, en les désignant par leurs titres et en les séparant par une barre oblique. On appelle cette liste un chemin. Soyons plus clairs avec un exemple de chemin de rôles : **/admin/user/guest**. La création d'un tel chemin aura pour effet que le rôle **user** héritera de toutes les permissions de **guest** et le rôle **admin** de toutes celles de **users**. Le même principe vaut pour les permissions. Le chemin **/admin/logout/login** implique que la permission d'accès à **logout** vaut permission d'accès à **login** et la permission à **admin** vaut autorisation à **logout**.

Le second argument est un tableau contenant la description dans l'ordre du chemin des différentes entités définies. Pour le chemin de rôles **/admin/user/guest**, il faut fournir un tableau contenant successivement les descriptions des rôles **admin**, **user** et **guest**.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Cela est un peu laborieux à concevoir et, si les exemples donnés dans la documentation ont le mérite d'être explicites, ils paraissent lourds à utiliser. Toutefois, avec un peu de méthode, on parvient aisément à un résultat facilement exploitable :

```
// Définition des chemins
// pour les rôles
$cheminsRoles[] = array(
 'admin' => 'Administrateur',
 'user' => 'Utilisateur authentifié',
 'guest' => 'Utilisateur anonyme',
); // on peut en ajouter d'autres de la même
façon

// pour les permissions
$cheminsPermissions[] = array(
 'admin' => 'Interface d\'administration',
 'logout' => 'Fin de session',
 'login' => 'Formulaire d\'authentification',
); // on peut en ajouter d'autres de la même
façon

// Traitement des chemins
foreach ($cheminsRoles as $chemin) {
 $path = '/'.implode('/',array_flip($chemin));
 $rbac->Roles->addPath($path,$chemin);
}
foreach ($cheminsPermissions as $chemin) {
 $path = '/'.implode('/',array_flip($chemin));
 $rbac->Permissions->addPath($path,$chemin);
}
```

Une fois les choses clairement mises en place, il est assez facile d'écrire les entités sous forme de chemin pour que PHP-RBAC puisse les traiter.

Dernier point, une entité peut appartenir à plusieurs chemins, ce qui permet des structures pyramidales, ou bien plus complexes encore.

CONCLUSION

Bien que les concepts mis en jeu paraissent un peu abstraits de prime abord, la gestion des droits par rôles apporte un confort d'utilisation auquel il est difficile de renoncer une fois que l'on y a pris goût. La hiérarchisation apporte une couche d'abstraction supplémentaire, qui peut effrayer, surtout quand on voit les différentes documentations sur le sujet : en effet, celles-ci partent toutes du schéma global de l'organisation ; mais nous avons vu que la logique est pourtant simple à appréhender en considérant un chemin unique plutôt qu'un organigramme entier.

Avec PHP-RBAC, il est facile d'implémenter une gestion des droits basée sur les rôles, hiérarchique ou non. Alors, pourquoi s'en priver ? ■

RÉFÉRENCES ET NOTES

[1] http://phprbac.net

[2] https://owasp.org

[3] http://www.nist.gov

[4] Apportons quelques nuances. J'ai eu l'occasion d'échanger avec l'un des principaux développeurs du projet PHP-RBAC à propos d'OWASP. Après s'être investi plus de dix ans dans cette organisation et avoir lutté pour en préserver l'esprit d'origine, il s'en est désolidarisé. Il reproche à certains membres d'avoir détourné cette fondation de ses objectifs premiers pour la mettre au service de leurs objectifs personnels. Du coup, difficile de voir dans OWASP une garantie de sérieux *aujourd'hui*. Pour autant, ce développeur considère PHP-RBAC comme un projet abouti, au développement arrêté sauf faille significative. Ce développeur reste toujours très réactif sur les tickets ouverts sur la page GitHub du projet : https://github.com/OWASP/rbac

[5] http://sourceforge.net/projects/phprbac/files/latest/download

[6] https://github.com/OWASP/rbac

[7] https://getcomposer.org, voir aussi MOUREY S., *"Enfin une gestion de dépendance correcte pour PHP : Composer !"*, *GNU/Linux Magazine n°162*, juillet 2013, p. 74 à 78

[8] MOUREY S., *"Bonne pratique : préférer un UID à une clé incrémentale"*, *Linux Pratique n°74*, novembre 2012, p. 80 à 81

POUR ALLER PLUS LOIN

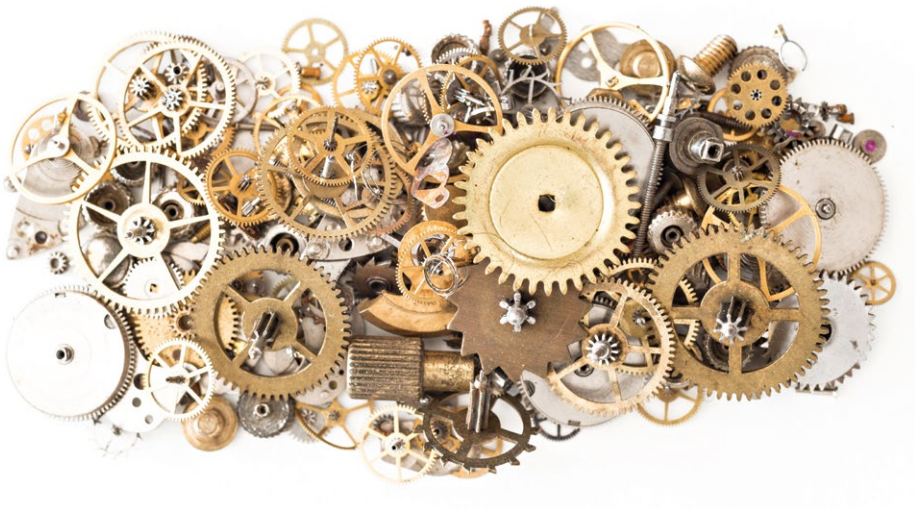
La page du NIST présentant les travaux sur le concept de RBAC : http://csrc.nist.gov/groups/SNS/rbac. En particulier, j'ai trouvé particulièrement utile ce document de travail par R. Sandhu, D. Ferraiolo et R. Kuhn : *« The NIST Model for Role-Based Access Control: Towards A Unified Standard »* (http://csrc.nist.gov/staff/Kuhn/towards-std.pdf).

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

L'ART DU REVERSE AVEC RADARE2

SYLVAIN NAYROLLES
[Security Software Engineer]

MOTS-CLÉS : REVERSE, GDB, RADARE2, ANALYSE STATIQUE, ANALYSE DYNAMIQUE, RÉTROCONCEPTION



Dans un article de MISC de mai 2013, Nicolas RUFF faisait une introduction au reverse engineering, en concluant sur la nécessité de développer en France cette compétence décrite comme le « Graal » en sécurité informatique. Dans cet article, il réalisait un éloge du logiciel IDA et de son géniteur, et ce à juste titre. Mais le monde Linux a aussi son IDA et il se nomme Radare2. Loin d'être aussi complet qu'IDA, il constitue un framework libre et avancé pour l'analyse de binaires sous Linux.

Nous n'allons pas réaliser une comparaison entre **IDA** et **Radare2**, tant ils sont incomparables de par leurs fonctionnalités et leurs philosophies. Le premier constitue une solution professionnelle et intégrale pour l'analyse de toutes sortes de binaires pour une multitude d'architectures. Au contraire, Radare2 est un *framework* constitué de nombreux outils, facilitant l'analyse de binaires sous **Linux**. Nous allons explorer ces fonctionnalités en les illustrant de cas concrets. L'art du reverse peut se diviser en deux grandes familles d'analyses ; les analyses statiques et les analyses dynamiques. Initialement, radare premier du nom était un projet initié comme un ensemble d'outils permettant de faciliter l'analyse de binaires. Le développement de la seconde version, remettant à plat l'ensemble du design de l'application, a gardé une grande partie de cette philosophie en offrant un ensemble d'outils se combinant très facilement. Pour poursuivre notre tour de ce fameux *framework*, il faut tout d'abord l'installer :

```
$ sudo apt-get install radare2
```

1. ANALYSE STATIQUE

L'analyse statique est l'art d'analyser un binaire sans son contexte d'exécution, ce qui complexifie un peu l'analyse. Une telle pratique consiste généralement en une extraction d'informations contenues dans le format exécutable de notre binaire, telles que les chaînes de caractères présentes dans le binaire, les symboles tels que les fonctions importées ou exportées, ou bien mieux encore les symboles de debug si ces derniers sont encore présents. Viens ensuite l'étape d'analyse des segments de

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

code exécutable avec bien sûr le reverse du code assembleur. Nous verrons qu'au-delà d'outils d'analyse, il nous sera aussi très utile d'avoir des outils permettant la manipulation de nombres hexadécimaux, de code assembleur, etc. Ceci représente la philosophie voulue par les développeurs de Radare2.

1.1 Rabin2

Rabin2 est un des outils basés sur le *framework* Radare2, réalisant des opérations simples dans la première phase de l'analyse.

Durant cette phase, nous allons tout d'abord essayer de comprendre l'exécutable comme une boîte noire, en essayant d'extraire les informations que ce dernier a bien voulu nous laisser. Nous allons donc essayer de lister le point d'entrée du programme, afin de savoir si ce dernier fût compilé de façon classique, avec quel type de compilateur, ou bien si c'est un *packer* fait « maison ». Ensuite, nous allons essayer de comprendre ce qu'il fait en listant les fonctions importées depuis d'autres bibliothèques. Et enfin, lister toutes les chaînes de caractères présentes dans ce dernier.

Rien de bien nouveau jusqu'ici, car il existe déjà dans l'écosystème Linux des outils réalisant les mêmes tâches telles que **readelf** pour les imports/exports, ou bien encore **strings** pour les chaînes de caractères. Radare2 tente par sa philosophie de centraliser la réalisation d'outils de retro autour de son *framework*.

Pour illustrer notre propos, nous allons tenter de comprendre ce que le programme **mon_programme** tente de réaliser sans l'exécuter, tout du moins dans un premier temps.

Afin de déterminer le point d'entrée d'un programme, il suffit de lancer le programme **rabin2** avec l'option **-e** :

```
$ rabin2 -e mon_programme
[Entrypoints]
addr=0x004004d0 off=0x000004d0
baddr=0x00400000
1 entrypoints
```

Ensuite, nous allons analyser les fonctions importées par notre binaire via la commande suivante :

```
$ rabin2 -i mon_programme
[Imports]
ordinal=001 plt=0x00000480 bind=GLOBAL
type=FUNC name=puts
ordinal=002 plt=0x00000490 bind=GLOBAL
type=FUNC name=printf
ordinal=003 plt=0x000004a0 bind=GLOBAL
type=FUNC name=__libc_start_main
ordinal=004 plt=0x000004b0 bind=GLOBAL type=FUNC
name=strcmp
ordinal=005 plt=0x000004c0 bind=UNKNOWN
type=NOTYPE name=__gmon_start__
5 imports
```

Nous observons que ce programme réalise des opérations sur des chaînes de caractères. Il ne semble pas du tout malveillant. Avec les précautions d'analyses habituelles, nous pouvons l'exécuter :

```
$ ./mon_programme
usage: ./mon_programme password
```

Ce programme constitue un simple *crackme*, où il faut trouver le mot de passe. En continuant d'utiliser rabin2, nous allons extraire les chaînes de caractères :

```
$ rabin2 -z mon_programme
[strings]
addr=0x004006b4 off=0x000006b4 ordinal=000 sz=11
section=.rodata string=Wond3rFU11
addr=0x004006bf off=0x000006bf ordinal=001 sz=18
section=.rodata string=[+] Well done :-)
addr=0x004006d1 off=0x000006d1 ordinal=002 sz=21
section=.rodata string=[-] Oh no, try again
addr=0x004006e6 off=0x000006e6 ordinal=003 sz=19
section=.rodata string=usage: %s password
4 strings
```

Cette commande permet de lister les chaînes de caractères contenues dans les sections utilisateur, et typiquement dans notre cas extraites de la section **.rodata** (*read only data*), section habituellement utilisée par **GCC** afin de stocker les chaînes de caractères constantes d'un programme. Ici la chaîne **Wond3rFU11** semble être la plus prometteuse afin de constituer un mot de passe :

```
$ ./mon_programme Wond3rFU11
[+] Well done :-)
```

1.2 Rax2

Rax2 est un utilitaire facilitant la manipulation des types souvent rencontrés lors d'une analyse statique :

```
$ rax2 10
0xa
```

L'opération inverse est bien sûr possible :

```
$ rax2 0x33
51
```

De nombreuses conversions sont disponibles au travers de cet outil, mais la gestion des chaînes est bien pratique :

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

```
$ rax2 -s 474e55204c696e7578204d6167617a696e65
GNU Linux Magazine
```

1.3 Rahash2

Rahash2 est l'outil permettant de calculer toutes sortes de *hash* sur n'importe quel fichier ou chaîne de caractères. Par exemple, il peut être très intéressant de calculer le *hash* d'une chaîne de caractères pouvant symboliser un mot de passe :

```
$ rahash2 -a all -s "admin"
0x00000000-0x00000005 md5: 21232f297a57a5a743894a0
e4a801fc3
0x00000000-0x00000005 sha1:
d033e22ae348aeb5660fc2140aec35850c4da997
0x00000000-0x00000005 sha256:
8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4
bb8a81f6f2ab448a918
0x00000000-0x00000005 sha384: 9ca694a90285c03
4432c9550421b7b9dbd5c0f4b6673f05f6dbce58052ba
20e4248041956ee8c9a2ec9f10290cdc0782
0x00000000-0x00000005 sha512: c7ad44cbad762a5
da0a452f9e854fdc1e0e7a52a38015f23f3eab1d80b93
1dd472634dfac71cd34ebc35d16ab7
fb8a90c81f975113d6c7538dc69dd8de9077ec
0x00000000-0x00000005 crc16: 9c99
0x00000000-0x00000005 crc32: 880e0d76
0x00000000-0x00000005 md4: f9d4049dd6a4dc35d4
0e5265954b2a46
```

Il est aussi possible de spécifier des offsets de fichiers afin de n'appliquer l'algorithme de *hash* qu'à une partie d'un fichier afin d'en déterminer la signature, comme certaines sections d'un binaire pouvant avoir été modifiées lors de l'exécution de ce dernier.

1.4 Rafind2

Rafind2 est un utilitaire permettant de réaliser des recherches de pattern hexadécimal, toujours orienté binaire, tout en permettant de borner notre recherche entre deux offsets :

```
$ rafind2 -s ELF /bin/true
0x1
```

Ici nous avons réalisé une recherche sur le *magic* **ELF** présent dans tous les binaires linux. Rafind2 nous répond qu'il a bien trouvé ce dernier à l'offset **0x1**, car le magic complet est **0x7F 0x45(E) 0x4C(L) 0x46(F)**.

Mais on peut imaginer de tenter de chercher d'autres *magics* dans un seul fichier, afin de détecter les *packers*, ou bien les ressources incluses dans un binaire, telles que des images, des fichiers audios, ou bien des archives.

1.5 Rasm2

Rasm2 est un petit utilitaire permettant de réaliser des opérations d'assemblage ou de désassemblage de code assembleur. Il peut nous permettre de compiler, par exemple des *shellcodes* :

```
$ rasm2 -a x86 -d 90
nop
```

Ici l'opcode **0x90** correspond bien en x86 à l'opération **nop**, c'est-à-dire ne rien faire.

L'opération inverse est bien sûr tout à fait possible :

```
$ rasm2 -a x86 'nop'
90
```

Cet outils étant orienté shellcode, il existe une option afin de générer notre code au format chaîne hexadécimal :

```
> rasm2 -a x86 'nop;nop;nop' -C
"\x90\x90\x90"
```

1.6 Radiff2

Radiff2 est un utilitaire bien plus évolué que ce que nous avons pu voir jusqu'ici. Il va, comme son nom l'indique, nous permettre de réaliser des tests de différences entre binaires afin de constater leur changement, ou bien leur évolution.

```
$ radiff2 prog1 prog2
0x00000198 7f8b1f2c4bf1bc002a5d70a764f41e31b
4d12c9b => d9f97568225e8a9a9277c5830056e06e0
b22e332 0x00000198
0x00000591 65 => 33 0x00000591
```

Ici la seule différence entre nos deux binaires est un caractère qui a changé, plus un *hash*. On est passé du caractère de code ascii **65** au caractère de code ascii **33**, soit le caractère **e** qui s'est changé en caractère **3**.

Radiff2 permet aussi d'utiliser des algorithmes bien plus complexes pour l'analyse de codes binaires. Il va être capable de calculer une différence sur le graphe d'exécution de ce dernier. On va par exemple récupérer un nouveau binaire similaire au précédent, mais ayant eu une correction. Nous allons donc analyser la différence de flux d'exécution de notre programme via la commande suivante :

```
$ radiff2 -g main bof1 bof2 > /tmp/main_graph
```

Puis via l'utilitaire **xdot** (**sudo apt-get install xdot**), nous allons afficher le graphe présenté en figure 1.

Nous constatons que notre flux d'exécution est changé au niveau du bloc réalisant le branchement conditionnel.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Il est souvent nécessaire en rétroconception d'étudier la différence entre plusieurs versions d'un binaire, ne serait-ce que pour évaluer la tâche de l'analyse et peut-être accélérer considérablement cette dernière. Quelques fois une nouvelle analyse n'est pas nécessaire surtout quand les différences ne portent que dans les ressources constantes du binaire, comme illustré dans le premier exemple.

Mais il peut être aussi intéressant d'étudier l'évolution d'un binaire afin, par exemple, de constater des mauvaises pratiques, comme dans le second exemple ou clairement la correction change considérablement la finalité du binaire ou potentiellement se cache un dysfonctionnement.

1.7 Un programme pour les gouverner tous

Vous allez me dire que pour un *framework* de désassemblage, adressé aux rétroconcepteurs sous linux, on ne voit pas beaucoup de code assembleur. Eh bien c'est que pour l'instant nous n'avons pas réellement utilisé le *framework* à 100 %. Le binaire **radare2** ou **r2** est le programme principal autour du *framework*, il permet de réaliser une analyse de *reverse* complète, via un mode console ou via une GUI web. Il regroupe toutes les fonctionnalités déjà présentées, en les regroupant autour d'un module d'analyse puissant.

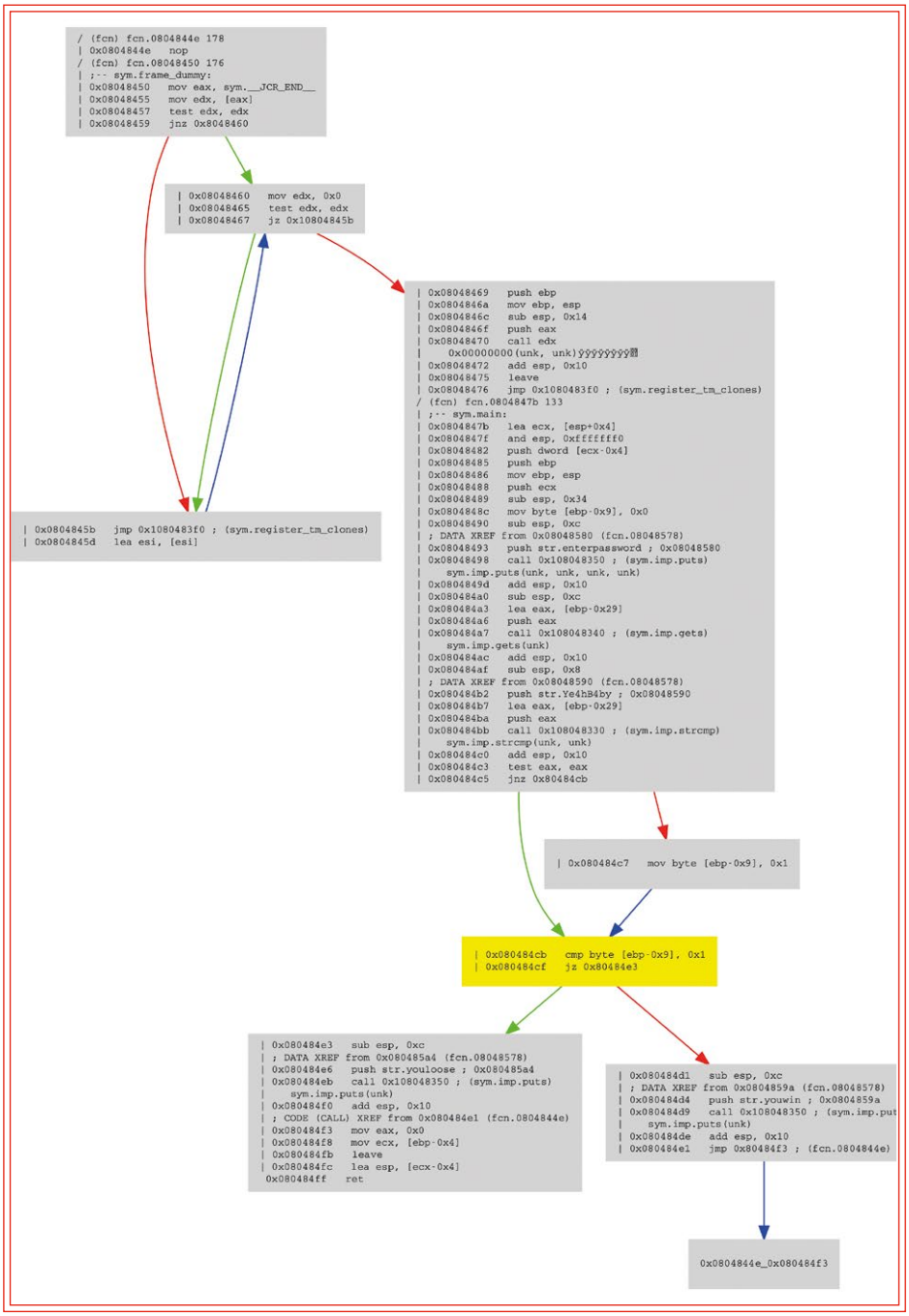


Fig. 1 : Exemple de graphe généré avec l'aide de radiff2 et xdot.

1.7.1 Premier pas...

Nous allons reprendre notre programme précédent afin de réaliser cette fois-ci une analyse complète au travers de Radare2.

Tout d'abord, nous allons lancer ce dernier afin qu'il charge notre binaire :

```
$ r2 mon_programme
[0x08048380] >
```

À ce stade, on entre dans la console interactive de r2. L'adresse **0x08048380** est le point d'entrée de notre programme, par défaut r2 se positionne sur ce dernier.

Une fois le désassemblage réalisé suite au chargement de votre binaire, Radare2 nous propose un module d'analyse de code très complet. Ce dernier permet de réaliser des tâches qui vont sensiblement faciliter la rétroconception. Il va tout d'abord tenter de reconstituer le graphe d'exécution du binaire.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ensuite, il va tenter de résoudre les symboles en remplaçant les adresses de ces derniers par le texte associé. Enfin, il peut tenter d'identifier les variables locales à une fonction selon le code assembleur, voire tenter un *mapping* de certaines variables en connaissant la signature des fonctions issues de bibliothèques standards.

Nous allons tout d'abord lui demander les informations collectées depuis la lecture des méta-informations du fichier via la commande **i** :

```
[0x08048380] > i
pic    false
has_va true
root   elf
class  ELF32
lang   c
arch   x86
os     linux
strip  false
type   EXEC (Executable file)
os     linux
bits   32
endian little
file   /home/sylvain/dev/articles/Linux
Magazine/radare2/src/mon_programme
uri mon_programme
```

Ceci nous permet d'afficher de nombreuses informations de compilation de notre binaire, telles que le PIC pour *Position Independent Code* à **false** ce qui signifie que l'ASLR (*Address Space Layout Randomization*) est désactivée.

Pour activer l'analyse de notre binaire, il suffit de lancer la commande **aa** (pour *analyse all*) :

```
[0x08048380] > aa
```

Une fois l'analyse terminée, on peut lister les fonctions via la commande **afl** :

```
[0x08048380] > afl
0x0804847b 25   1   sym.help
```

Par chance le binaire en cours d'analyse n'a pas été *strippé*, et inclut donc encore beaucoup de symboles inutiles à l'exécution, mais très pratiques lorsque l'on réalise une analyse statique du binaire. Typiquement le symbole associé à cette fonction aurait pu être retiré en exécutant une opération de *stripping*. Ici, r2 nous précise qu'il a déduit le nom de la fonction depuis la table des symboles via le préfixe **sym**.

Enfin pour analyser cette fonction, il suffit de le demander à r2 :

```
[0x08048380] > pdf@sym.help,
/ (fcn) sym.help 25
|           0x0804847b    55           push ebp
|           0x0804847c    89e5         mov ebp, esp
|           0x0804847e    83ec08       sub esp, 0x8
|           0x08048481    83ec0c       sub esp, 0xc
|           ; DATA XREF from 0x080485a0 (fcn.08048598)
|           0x08048484    68a0850408   push str.
Ammagicalpasswordchecker ; 0x080485a0
|           0x08048489    e8c2feffff   call 0x108048350 ;
(sym.imp.puts)
|              sym.imp.puts(unk, unk)
|           0x0804848e    83c410       add esp, 0x10
|           0x08048491    90           nop
|           0x08048492    c9           leave
\           0x08048493    c3           ret
```

Notre fonction étant une fonction somme toute très basique, elle ne comporte que très peu d'informations. Toutefois nous pouvons constater que r2 a détecté l'utilisation d'une chaîne de caractère issue de son analyse **str.Amagicalpasswordchecker**, ainsi que la référence à un symbole **sym.imp.puts**. Ceci nous simplifie considérablement l'analyse de notre binaire.

Nous pouvons afficher la chaîne de caractères référencée par l'analyseur via la commande **psz** (*print string zero terminated*) :

```
[0x08048380] > psz@str.Amagicalpasswordchecker
A magical password checker
```

Afin de poursuivre l'analyse de cette fonction, il va nous être possible de déplacer notre analyse depuis le point d'entrée de notre programme vers la fonction qui nous intéresse :

```
[0x08048480]> s sym.help
[0x0804847b]>
```

1.7.2 Base de connaissance

Lorsque l'on réalise une analyse de rétroconception, il est primordial de constituer une base de connaissance de notre binaire. Pour ce faire, nous allons souvent devoir nommer des variables, renommer des fonctions, ainsi qu'ajouter des commentaires dans le code assembleur. Ceci afin de reconstruire notre binaire en ayant une vision du bas vers le haut. On va décrire les briques élémentaires afin de pouvoir mieux comprendre ce que le programme fait à haut niveau.

R2 propose un mécanisme de *flags* permettant d'illustrer notre analyse, ainsi que d'une notion de projet permettant sa pérennisation.

Les *flags* sont regroupés en *namespace*. Les *namespaces* par défaut sont :

- **symbols** regroupant les textes issus de la table des symboles ;

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

- **functions** regroupant les noms des fonctions issues de la phase d'analyse. Les fonctions comportant un symbole n'apparaissent pas dans ce *namespace* ;
- **strings** issue de l'analyse des chaînes de caractères présentes dans le binaire ;
- **sections** représentant les sections du programme ;
- **relocs** pour l'ensemble des symboles de relocations.

Lors d'une analyse, nous allons surtout nous concentrer sur les *namespaces* **strings**, **symbols** et **functions**.

Reprenons l'exemple précédent, mais cette fois-ci nous *strippons* les symboles. Nous allons recharger notre binaire et relancer une analyse :

```
$ r2 mon_programme
```

Nous ne pouvons plus voir notre fonction **help**. Nous allons donc inverser notre analyse. Nous avons l'adresse de la chaîne de caractères associée à la fonction **help** :

```
[0x08048380]> fs strings
[0x08048380]> f
0x080485a0 28 str.
Ammagicalpasswordchecker
```

Ensuite, on va demander à r2 de chercher dans le code désassemblé cette adresse :

```
[0x08048380]> /c 80485a0
f hit_0 @ 0x08048484   #
5: push 0x80485a0
```

Là, il nous a trouvé la partie du code appelant. Enfin, on va lui demander d'analyser le code de la fonction déterminée après analyse autour de cette adresse :

```
[0x08048380]>
pdf@0x08048484
```

Nous observons que l'analyse n'a pas réussi à borner notre fonction (voir figure 2), donc nous allons essayer de l'aider. Visiblement, il y a un **leave** et juste après il y a le prologue typique d'une convention d'appel **cdecl**. Nous allons donc aider r2 en créant une nouvelle fonction :

```
[0x08048480]> af+ 0x0804847b (0x08048493-0x0804847b) help
```

Ici, on met à jour la base de connaissances de r2 en lui signalant la présence d'une fonction à partir de l'offset **0x0804847b** faisant une taille de (**0x08048493-0x0804847b**) et se nommant **help**. De plus, on va rajouter un tag pour simplifier l'analyse :

```
[0x08048480]> fs functions
[0x08048480]> f help @ 0x0804847b
```

Et on peut afficher notre fonction **help** :

```
[0x08048480]> pdf @ help
/ (fcn) help 24
|           0x0804847b    55          push ebp
|           0x0804847c    89e5        mov ebp, esp
|           0x0804847e    83ec08      sub esp, 0x8
|           0x08048481    83ec0c      sub esp, 0xc
|           ; DATA XREF from 0x080485a0 (fcn.08048598)
|           0x08048484    68a0850408  push str.Ammagicalpasswordchecker ;
0x080485a0
|           0x08048489    e8c2feffff  call 0x108048350 ; (sym.imp.puts)
|              sym.imp.puts(unk, unk)
|           0x0804848e    83c410      add esp, 0x10
|           0x08048491    90          nop
|           0x08048492    c9          leave
```

```
(fcn) fcn.0804844e 70
          0x0804844e    6690        nop
          0x08048450    b8109f0408  mov eax, 0x8049f10
          0x08048455    8b10        mov edx, [eax]
          0x08048457    85d2        test edx, edx
      ,=< 0x08048459    7505        jnz 0x8048460
     .--> 0x0804845b    eb93        jmp 0x1080483f0 ; (fcn.080483eb)
     ||   0x0804845d    8d7600      lea esi, [esi]
     |`-> 0x08048460    ba00000000  mov edx, 0x0
     |    0x08048465    85d2        test edx, edx
     `==< 0x08048467    74f2        jz 0x10804845b
          0x08048469    55          push ebp
          0x0804846a    89e5        mov ebp, esp
          0x0804846c    83ec14      sub esp, 0x14
          0x0804846f    50          push eax
          0x08048470    ffd2        call edx
             0x00000000(unk, unk)
          0x08048472    83c410      add esp, 0x10
          0x08048475    c9          leave
          0x08048476    e975ffffff  jmp 0x1080483f0 ; (fcn.080483eb)
          0x0804847b    55          push ebp
          0x0804847c    89e5        mov ebp, esp
          0x0804847e    83ec08      sub esp, 0x8
          0x08048481    83ec0c      sub esp, 0xc
          ; DATA XREF from 0x080485a0 (fcn.08048598)
          0x08048484    68a0850408  push str.Ammagicalpasswordchecker ; 0x080485a0
          0x08048489    e8c2feffff  call 0x108048350 ; (sym.imp.puts)
             sym.imp.puts(unk, unk)
          0x0804848e    83c410      add esp, 0x10
          0x08048491    90          nop
          0x08048492    c9          leave
          0x08048493    c3          ret
```

Fig. 2 : Désassemblage de la fonction avant analyse.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

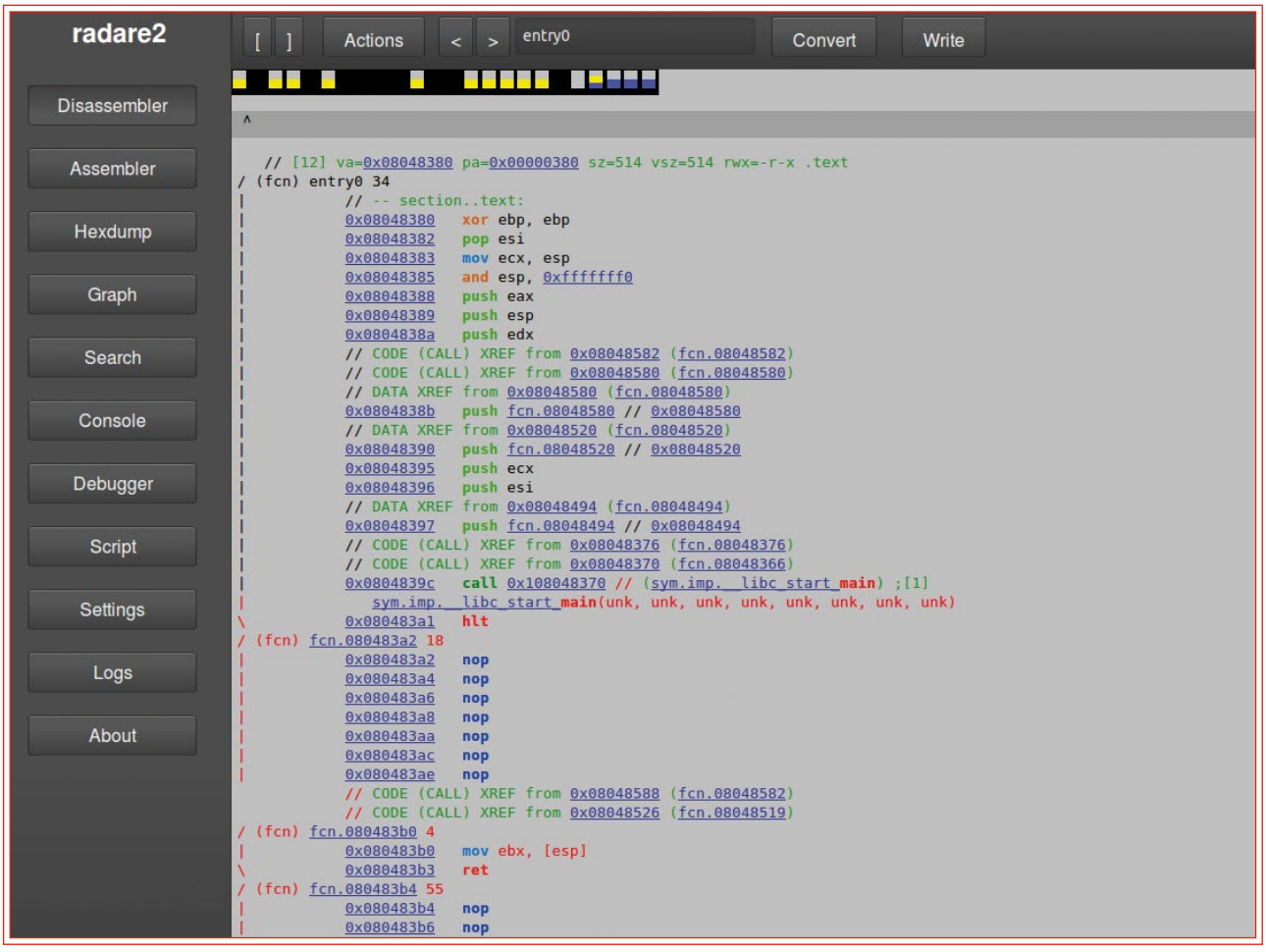


Fig. 3 : Graphe d'exécution de la fonction main de notre programme.

Enfin, nous pouvons sauvegarder notre base via la gestion des projets r2 :

```
[0x08048480]> Ps mon_programme
```

Et pour le rouvrir :

```
[0x08048480]> Po mon_programme
```

Attention r2 gère les projets depuis le répertoire de travail.

1.7.3 En mode console, vraiment ?

Comme vous avez pu le remarquer, pour l'instant il n'y a que très peu de captures d'écran, hormis celle d'une énième console. Historiquement, r2 est un outil en mode console pur, permettant une légèreté pour une première analyse. Il peut souvent exporter ses données dans des formats communs tels que **JSON** ou bien **.dot**, que l'on peut facilement visualiser après avec un utilitaire comme xdot.

Par exemple, afin de visualiser le graphe d'exécution d'une fonction en combinant la commande **ag** de radare2 et l'utilitaire **xdot** (voir résultat en figure 3) :

```
[0x08048380]> ag main > /tmp/main.dot
[0x08048380]> !xdot /tmp/main.dot
```

Mais récemment, les développeurs de r2 ont exaucé les vœux de certains utilisateurs, en incluant une IHM web. Pour cela, il suffit de passer en mode visuel par la commande **V** puis utiliser la commande **W**. (voir figures 4 et 5).

2. ANALYSE DYNAMIQUE

L'analyse dynamique est associée à un déboggage, mais sans le code source. R2 inclut un debugger afin de réaliser une telle analyse. De plus, sa *cli* est vraiment optimisée pour faciliter l'analyse, en plus de toutes les fonctionnalités que nous avons décrites.

Afin de démarrer l'analyse dynamique de notre binaire, il faut lancer notre programme cette fois avec l'option **-A** pour réaliser l'analyse au *loading* plus l'option

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

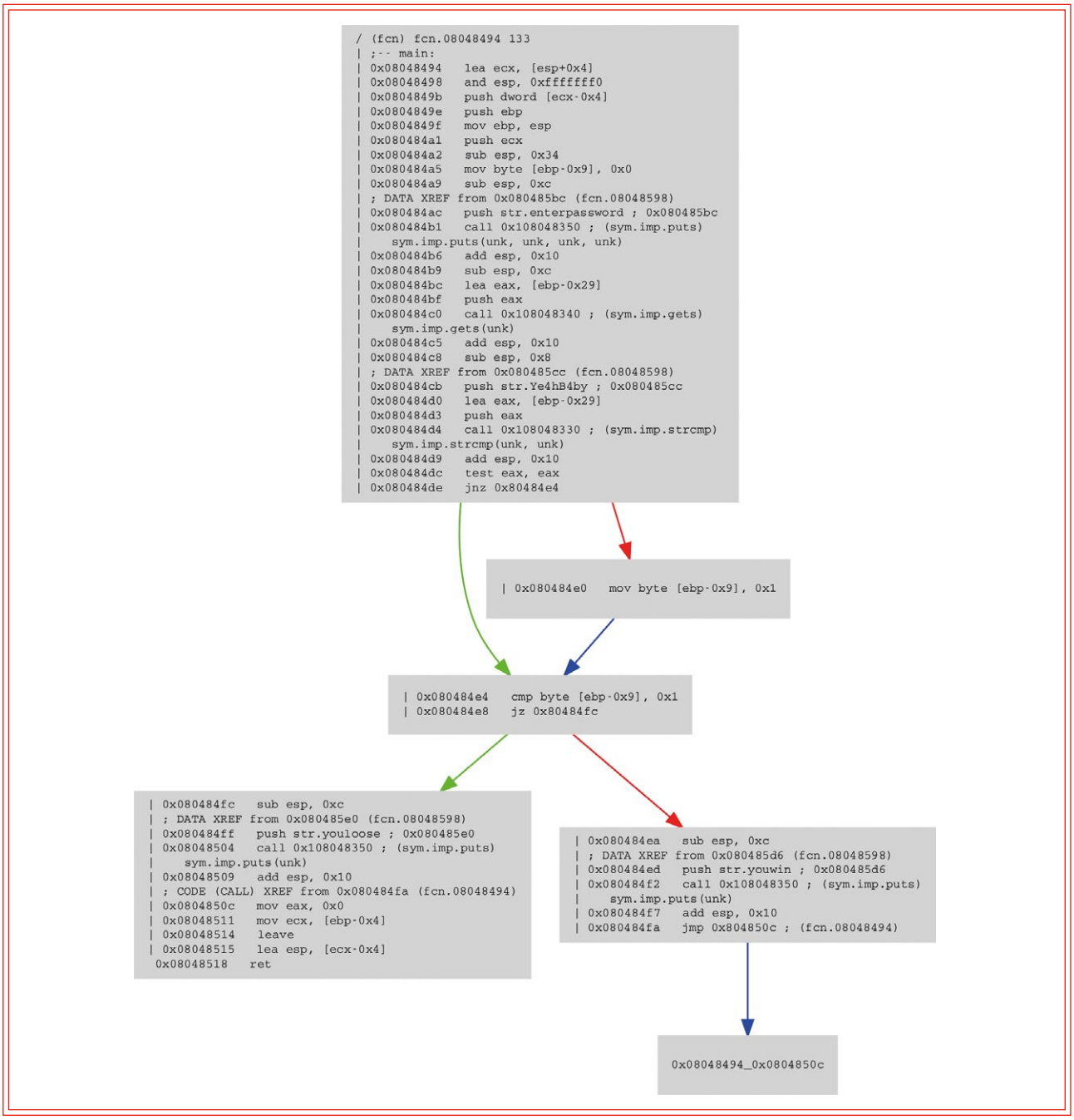


Fig. 4 : WebUI de Radare2.

-d pour lancer le *debugger*. Enfin, nous allons charger le projet que nous avons précédemment créé toujours à l'aide de la commande **Po**.

Il va donc nous être possible de positionner des points d'arrêt, d'analyser des zones mémoire, comme dans tout autre *debugger*.

Ici, nous allons positionner un point d'arrêt sur notre *flag* **help** représentant la fonction que nous avons analysée :

```
[0xf76f2a90]> db help
```

Puis nous allons continuer notre programme :

```
[0xf76f2a90]> dc
```

Pour plus de détails sur les commandes du *debugger*, je ne saurais trop vous conseiller de vous référer aux commandes *cheat sheets* **[1]**. C'est un *debugger* complet avec tout ce dont on attend de ce dernier.

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

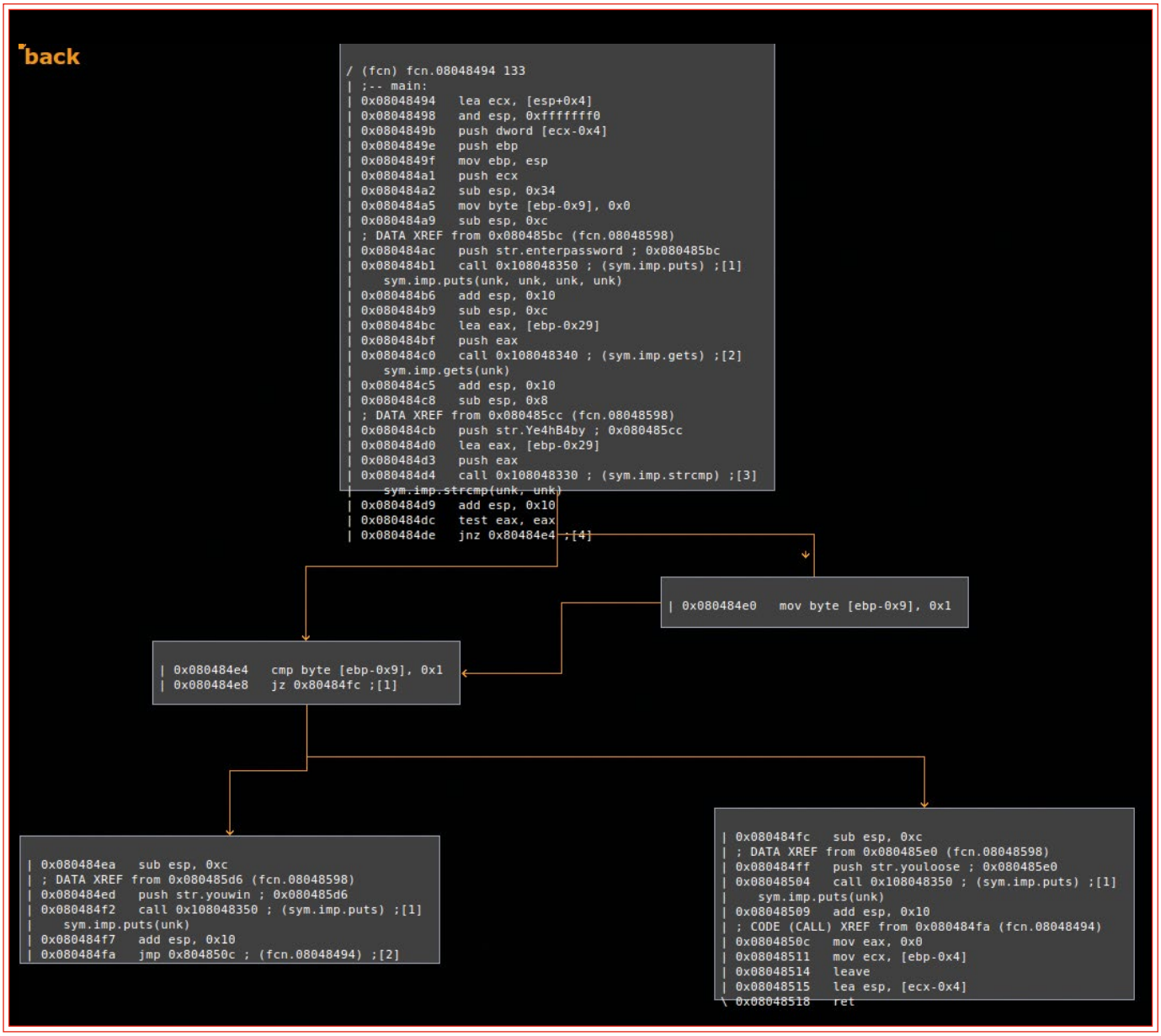


Fig. 5 : Graphe d'exécution de la fonction dans la WebUI.

CONCLUSION

Radare2 constitue un environnement complet de debuggage et d'analyse pour la rétroconception. Il permet de réaliser des analyses, disons artisanales, mais il est aussi facile de l'instrumenter pour réaliser des analyses automatiques. J'étais déjà un partisan, s'il en est, de radare2, mais au fil de cet article, j'ai encore découvert des fonctionnalités avancées et réellement pensées pour la rétroconception. Il s'inscrit de façon admirable dans un écosystème pas vraiment habitué à ce type de pratique, en prenant pour partie pris de réaliser un outil réellement orienté console, instrumentable et scriptable.

Amis « reverseurs », faites le grand saut, non pas dans un premier temps pour remplacer IDA, mais par exemple pour remplacer GDB en ligne de commandes, ou bien pour résoudre les challenges de *crackme* sur la plateforme **root-me**. ■

RÉFÉRENCE

[1] Cheat sheet de Radare2 : https://github.com/pwntester/cheatsheets/blob/master/radare2.md

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

SERVEURS DÉDIÉS Synology®

AVEC ikoula HÉBERGEUR CLOUD

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

POUR LES LECTEURS DE **LINUX MAG***

OFFRE SPÉCIALE -60 %

À PARTIR DE

5,99 € HT/MOIS

~~14,99 €~~

CODE PROMO **SYLIM17**

- 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.

CHOISISSEZ VOTRE NAS

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

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

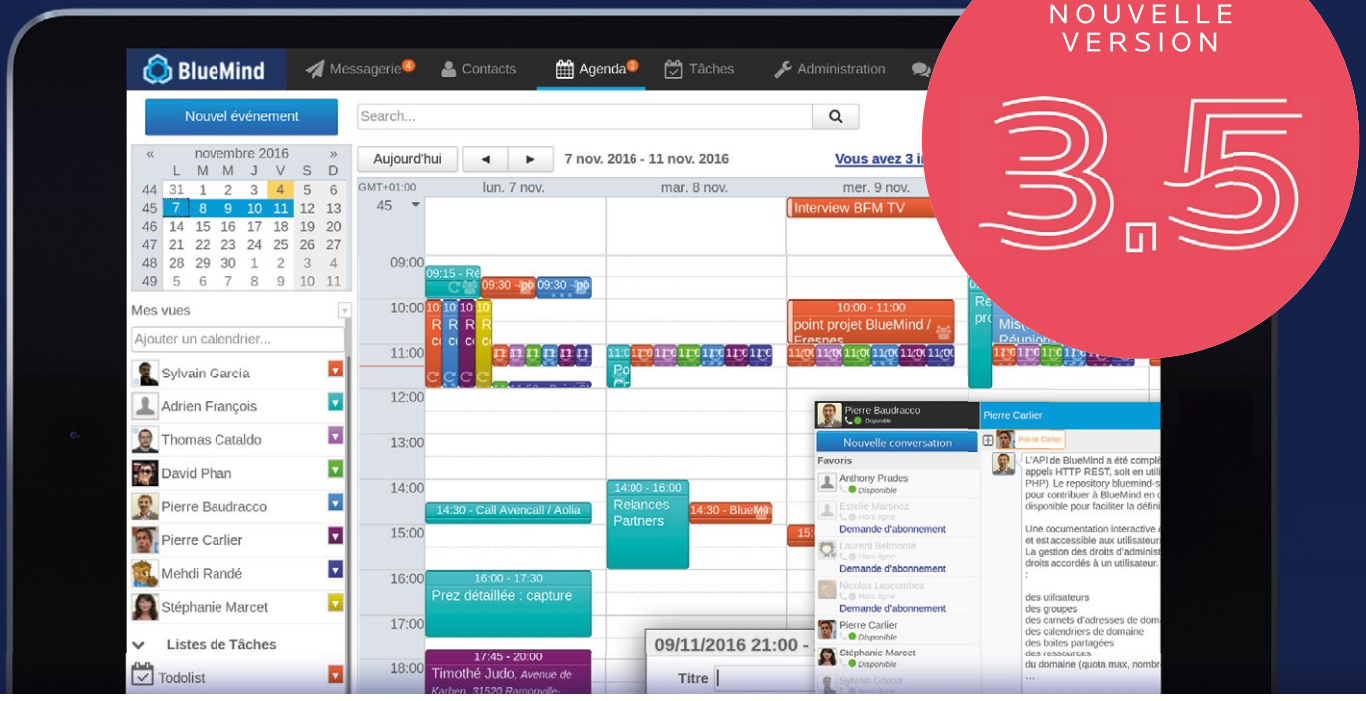
Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

BlueMind

SOLUTION OPENSOURCE
PROFESSIONNELLE DE MESSAGERIE
COLLABORATIVE

LIBÉREZ VOTRE MESSAGERIE

EMAILS
AGENDAS
CONTACTS
TÂCHES
CHAT
MOBILES
DOCUMENTS & PIÈCES JOINTES
ENVOI FICHIERS VOLUMINEUX
...
...



NOUVELLE VERSION
3.5

FRANCAIS / NOMBREUSES RÉFÉRENCES / ERGONOMIQUE / ÉVOLUTIF / ÉCONOMIQUE

Découvrez l'écosystème BlueMind et toutes les fonctionnalités sur
www.bluemind.net

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(jacques.thimonier@businessdecision.com)