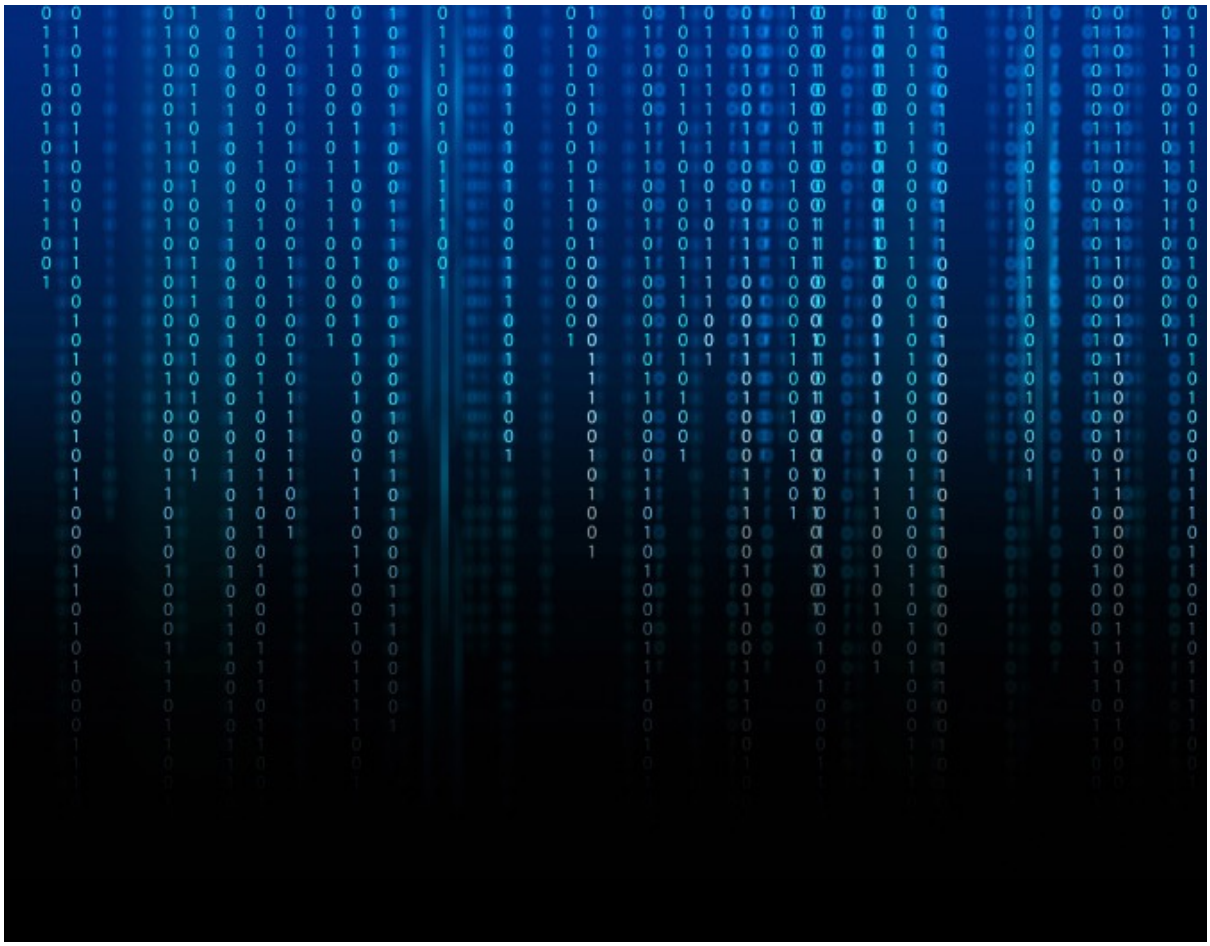
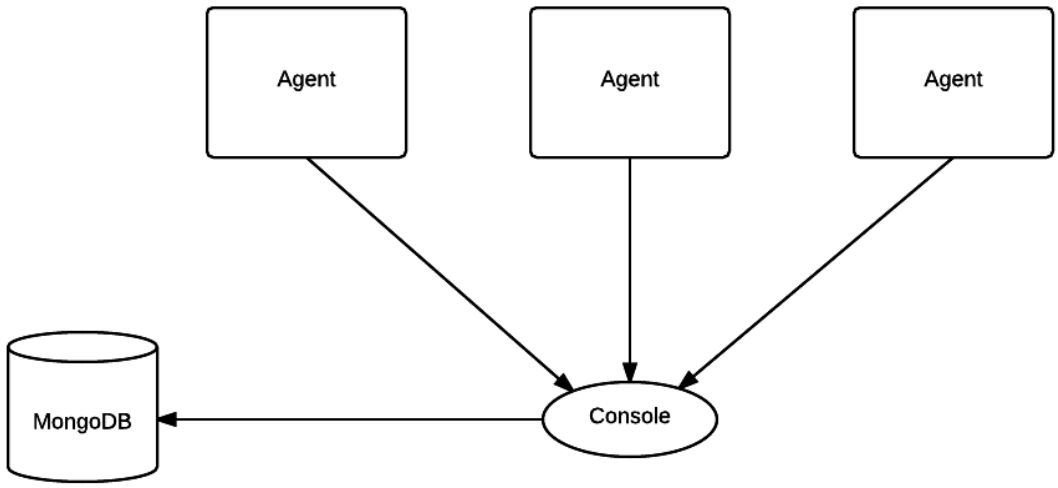
Documentation technique

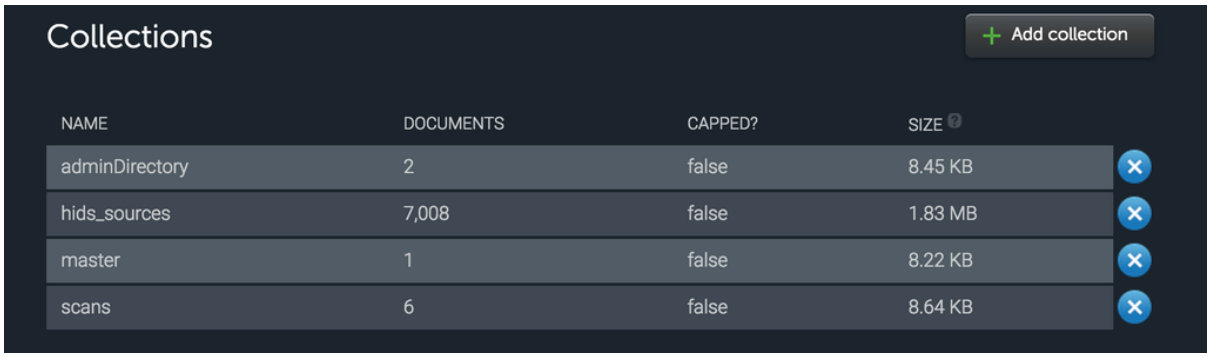
HIDS



L’architecture :



Cette architecture permet de travailler en **n agent**, c’est à dire que le fonctionnement du programme n’est pas influencé par le nombre d’agent, qu’il y en ai un ou 30 , le programme sait le gérer.
On utilise une base de donnée MongoDB par agent , et un agent par système à sécuriser. On choisit d’avoir un MongoDB sur un serveur distant afin d’augmenter l’intégrité du système.
Afin de sécuriser au maximum l’application, les échanges entre les agents et la console centrale devraient être chiffrés et conditionnés par un challenge échangé par la console et l’agent.
En MongoDB nous travaillons en « documents » et non en tables relationnelles, et cela s’adapte beaucoup mieux à notre projet.
En effet nous n’avons pas besoin de relations entre les collections.



Nous avons donc quatre collections de données :

- adminDirectory : qui regroupe les admins du client, qui recevront les mails de l’application
- hids_sources qui archive les sources de l’application (ici un wordPress)
- master : qui historise les masterRelease , qui sont des versions validées par le clients et qui peuvent être utilisées pour des scans. A chaque chaque évolution des sources, une nouvelle masterRelease devra être crée.
- scans qui historise tous les sans

La console admin :

Premièrement l'URI de connexion à la base de donnée MongoDB, elle est de la forme :
« mongodb://<username>:<password>@<host>:<port>/<base> »

```
#####################################################################
MONGODB_URI = 'mongodb://hids:hids@ds039291.mongolab.com:39291/tp_secu'
#####################################################################
```

Ici on se connecte donc à une base hébergée sur mongolab, base nommée « tp_secu ».

```
#Fonction pour se connecter à la BDD en local
def bddConnect(tries=3):
    from pymongo import MongoClient
    myDB = None
    try:
        client = MongoClient(MONGODB_URI)
        myDB = client.tp_secu
    except pymongo.errors.ConnectionFailure:
        bddReconnect(tries)
    finally:
        return myDB

def bddReconnect(tries):
    print tries
    if(tries==0):
        print("Abandon")
        exit()
    print("Connection failed")
    time.sleep(10)
    print("Tentative de reconnexion")
    bddConnect()
    tries-=1
    return tries
```

Ici on peut voir la fonction de connexion à la base de donnée, avec en paramètre « tries » , le nombre de tentatives de reconnexion avant l'abandon.
On tente de se connecter au serveur de base de données , on choisit la base de données qui nous interesse : « myDB = client.tp_secu »
Si jamais la machine qui execute le script ne peut se connecter à internet, une exception « ConnectionFailure » est levée , et on rentre dans la fonction de reconnexion.

```
#Passer une requête afin de récuperer les données
def sqlRetrieveWorpressData():
    cursor = db.cursor()
    query = """SELECT post_title FROM i4_posts p
    JOIN i4_term_relationships r ON r.object_id = p.ID
    JOIN i4_term_taxonomy t ON r.term_taxonomy_id = t.term_taxonomy_id
    JOIN i4_terms terms ON terms.term_id = t.term_id
    WHERE t.taxonomy ='category'
    AND terms.name = 'procedure'
    AND p.post_type = 'post';"""
    lines = cursor.execute(query)
    data = cursor.fetchall()
    db.close()
```

Toujours dans l'axe base de données, voici la requête qui permet d'aller chercher les articles publiés sur WordPress dans sa propre base de données.
Remarque : Nous avons préfixé notre base de « i4_ », en relation avec notre promotion.
La requête récupère tous les articles dont le nom contient « Procedure » et qui ont une catégorie assignée.

Voici les fonctions d'ajout en base de données, comme on peut le voir on défini un format d'insertion qui correspond à notre format de données, puis on insère nos datas. On peut voir en dessous un scan en base de données.

```
#ajoute une nouvelle ressource en base
def addNewHashedRessource(collectionName,ressourceName,hashedDataToStore,database):
    target = database[collectionName]
    format = {"ressourceName":ressourceName,"hash":hashedDataToStore}
    posted = target.insert(format)


def addNewMasterRelease(releaseDate,releaseHash):
    format = {"releaseDate": releaseDate,"releaseHash":releaseHash}
    db = bddConnect()
    target = getAdminMails()
    posted = target.insert(format)
    mailUtil.mailNewMasterRealease(target)

def addNewAdmin(adminName,mail,phone,status="valide"):
    format = {"adminName":adminName,"mail":mail,"phone":phone,"created":chrono.getTime(),"valide":status}
    db = bddConnect()
    target = db.adminDirectory
    posted = target.insert(format)

def addNewScan(scanDate,scanType,files,status):
    format = {"scanDate":scanDate,"type":scanType,"files":files,"status":status}
    db = bddConnect()
    target = db.scans
    posted = target.insert(format)
```

```
{
    "_id": {
        "$oid": "55783d83d124862237bc95d7"
    },
    "files": 1245,
    "status": "Success",
    "type": "fast",
    "scanDate": "10/06/2015 15:37:06"
}
```

Voici la fonction qui récupère les admins d'une base. Comme expliqué en soutenance, nous travaillons avec une base de donnée par clients, donc tous les admins de la base correspondent au client avec qui on travaille.

```
#Fonction qui récupère les mails des admins en base
def getAdminMails():
    db = bddConnect()
    mails=[]
    for admin in db.adminDirectory.find({"valide":"valide"},{'mail':True,'_id':False}):
        mailAddress = str(admin.get('mail'))
        mails.append(mailAddress)
    return mails
```

On a vu un peu plus haut, on travaille avec des « hash » , voici les fonctions qui leur sont relatives. Premièrement la fonctions de cash qui utilise le sha1, puis une fonction basique de comparaison afin de savoir si deux hash sont identiques ou non.

```
#Fonction basique de hash en sha1, testée
def hashData(data):
    hash_object = hashlib.sha1(data)
    hex_dig = hash_object.hexdigest()
    return hex_dig

#compare une data à une hashedData
def compareToHash(data, hashedData):
    newDataHashed = hashData(data)
    return newDataHashed == hashedData
```

Afin de manipuler des fichiers et de parcourir une arborescence, j'ai utilisé la librairie « os » de python. De plus afin d'obtenir le contenu d'un fichier pour le hasher il faut pouvoir l'ouvrir en lecture grâce à la fonction « openFile() ».

```
#retourne la liste des fichiers de manière récursive dans un repertoire
def listdirectory(path):
    fichiers=[]
    for root, dirs, files in os.walk(path):
        for i in files:
            fichiers.append(os.path.join(root, i))
    return fichiers

#ouvre un fichier afin de récupérer son contenu
def openFile(path):
    myFile = open(path,'r')
    data = myFile.read()
    return data
```

```python
#Fonction qui fait hydrate la BDD
def initiateSources(sourcePath=os.getcwd()+"/wordpress/"):
    if(debug):
        t0 = chrono.start()
    files = listdirectory(sourcePath)
    fileSize = len(files)
    iterator = 100/fileSize
    if(debug):
        print fileSize,"fichiers à traiter"
    hashArray = []
    database = bddConnect()
    #initiate la barre
    widgets = ['Traitement: ', Percentage(), ' ', Bar(marker=RotatingMarker()),' ', ETA(), ' ']
    pbar = ProgressBar(widgets=widgets, maxval=fileSize).start()
    for i in range(fileSize) :
        data = openFile(files[i])
        encodedData = hashData(data)
        varFile = files[i]
        hashArray.append(encodedData)
        addNewHashedRessource("hids_sources",varFile,encodedData,database)
        pbar.update(i+iterator)
    pbar.finish()
    releaseHash = hashData(''.join(hashArray))

    if(debug):
        print fileSize,"fichiers traités avec succès"
        chrono.stop(t0)
    return releaseHash

def initiateMasterRelease():
    releaseDate = chrono.getTime()
    releaseHash = initiateSources()
    addNewMasterRelease(releaseDate,releaseHash)
```

Ici deux fonctions : initiateMasterRelease() qui déclenche tout un process :

- parcours des fichiers , hash et insertions en base
- Création d'une nouvelle release de référence en base de données

InitiateSources() est la fonctions qui parcours l'arborescence de fichiers et à les hasher un par un . Une fois terminé la valeur hashée qui est envoyée en base de données.
Remarque : le chemin du fichier est inclut afin de gérer les déplacements de fichiers.
Voici une capture d'écran prise au cours d'un scan :

```
>>> HIDS.initiateMasterRelease()
1245 fichiers à traiter
raitement:  21% |\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\                                | ETA:  0:00:45
```

L’agent :

L’agent est un simple fichier .py qui est déposé sur le serveur de sources à sécuriser, il sera activé de façon automatique par un CRON ou tâche planifiée.
Voici la fonction de scan « rapide » , elle permet en moins d’une seconde (cf capture n°2) de donner un résultat d’intégrité et d’archiver ce scan en base de données, afin d’en garder un historique.

```python
def fastScan(path=os.getcwd()+"/wordpress/"):
    HIDS.bddConnect()
    t0 = chrono.start()
    status = "Failed"
    #Cette fonction renvoie le répertoire d'exectution du fichier du CRON
    files = HIDS.listdirectory(path)
    fileSize = len(files)
    storedHashData = HIDS.getLastMasterReleaseHash()
    hashArray = []
    iterator = 100/fileSize
    timeNow = chrono.getTime()
    widgets = ['Traitement: ', Percentage(), ' ', Bar(marker=RotatingMarker()),' ', ETA(), ' ']
    pbar = ProgressBar(widgets=widgets, maxval=fileSize).start()
    for i in range(fileSize) :
        ressourceName = files[i]
        data = HIDS.openFile(ressourceName)
        encodedData = HIDS.hashData(data)
        hashArray.append(encodedData)
        pbar.update(i+iterator)
    pbar.finish()
    duration = chrono.stop(t0)
    releaseHash = HIDS.hashData(''.join(hashArray))
    if(HIDS.compareHashToHash(releaseHash,storedHashData)):
            status = "Success"
    else:
        contacts = HIDS.getAdminMails()
        mailUtil.mailScanFailed(contacts)
    print("Status :",status)
    HIDS.addNewScan(timeNow,'fast',fileSize,releaseHash,duration,status)
    if(status=='Failed'):
        print("Deep scan will start soon...")
        deepScan()
```

Voici un scan réussi, puis un échoué.

```
>>> CRON.fastScan()
Traitement: 100% |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| Time: 0:00:00
Temps execution : 0 h 0 min 0 secondes 827 ms
('actual', 'cad154ab4dc8db1ba934760556767e00026d7761')
('expected', 'cad154ab4dc8db1ba934760556767e00026d7761')
Success
```

Comme on peut le voir on affiche le statut , et puisque le scan est échoué on lance un deepScan pour identifier les failles.

```
>>> CRON.fastScan()
Traitement: 100% |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| Time: 0:00:00
Status :Failed
Deep scan will start soon...
Traitement:   8% |------------                                                                           | ETA:  0:04:26
```

Dans le cas d’un scan « failed » comme celui ci , une alerte mail doit être envoyée :
J’ai installé un serveur de mail local sur mon Mac (Postfix), et python l’utilise pour envoyer des mails :

Cette fonctionnalité permet de réagir rapidement en cas d’intrusion sur le système.
Attention ! Pour pouvoir tester cette fonctionnalité il faut :

- Avoir un serveur SMTP localhost


```
text = "Bonjour !\nLe robot de sécurité a détécté une faille dans le système !\nDes fichiers ont été alterés !\n"
html = """\
<html>
  <head></head>
  <body>
    <p>Bonjour !<br>
       Le robot de sécurité a détécté une faille dans le système !<br>
       Des fichiers ont été alterés !
    </p>
  </body>
</html>
"""

# Record the MIME types of both parts - text/plain and text/html.
part1 = MIMEText(text, 'plain')
part2 = MIMEText(html, 'html')

# Attach parts into message container.
# According to RFC 2046, the last part of a multipart message, in this case
# the HTML message, is best and preferred.
msg.attach(part1)
msg.attach(part2)

# Send the message via local SMTP server.
s = smtplib.SMTP('localhost')
# sendmail function takes 3 arguments: sender's address, recipient's address
# and message to send - here it is sent as one string.
s.sendmail(me, you, msg.as_string())
s.quit()
```

- s'être ajouté en tant qu'admin via HIDS.addNewAdmin('Thierry Meyer','thierry.meyer@tm-consultants.fr','000000')
- Avoir une master release
- Editer un fichier du wordpress
- Lancer un scan (qui sera donc failed)

Il existe aussi une fonction qui permet de lister les fichiers altérés, voici le mail envoyé :