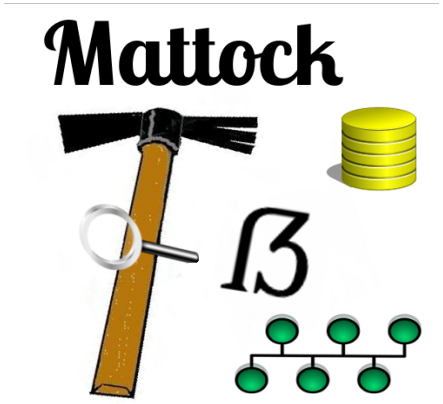
Schedule

- Asynchronous processing & tool-chain approach
- Integrity, privilege separation and capabilities.
- CarvFS & MinorFS
- MattockFS core design
- MattockFS as distributed-framework building block
- **Installation (hands on)**
- File-system as API (hands on)
- Python API (hands on)

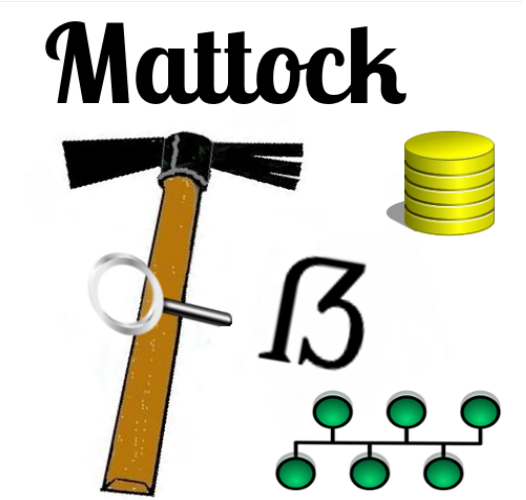


MattockFS

Computer-Forensics File-System

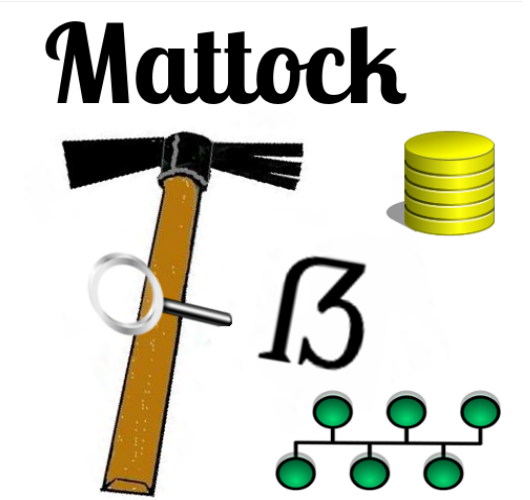
MattockFS hands-on

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- Python bindings
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

The hands-on part



- **Installation**
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- Python bindings
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

Required ubuntu packages

- fuse
- redis-server
- python-fuse
- python-redis
- python-xattr
- python-libewf
- sleuthkit
- python-exif
- git
- python-setuptools
- libpython-dev
- exif
- binutils
- python-demjson
- attr
- python-magic

sudo apt-get install <package>

Additional packages

- **wget http://dfrws.capibara.com/python-fadvise_6.0.0_amd64.deb**
- **wget http://dfrws.capibara.com/python-pyblake2_0.9.3_amd64.deb**
- **sudo dpkg -i python-fadvise_6.0.0_amd64.deb**
- **sudo dpkg -i python-pyblake2_0.9.3_amd64.deb**

Fetching MattockFS

- git clone https://github.com/pibara/MattockFS.git

- cd MattockFS
- git pull origin master

- sudo ./quick_setup
- cd ..

Starting

- sudo start_mattockfs

Lets put in some data

- ewf2mattock macwd.E01

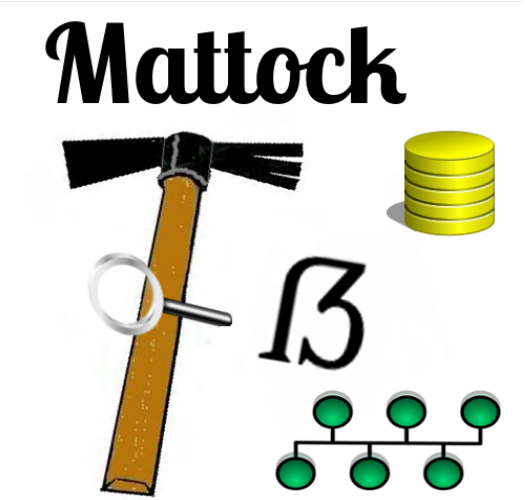
- cd /var/mattock
- ls -la archive/
- cd log
- ls -la 0*
- cat 0.provenance | head -1 | json_pp

What happened to the actual data?

- cat 0.journal | grep disk-img | head -1 | json_pp
- Look at the key.
- cat 0.journal | grep *d3b5d* | *grep UPD* | *json_pp*
- *cat 0.refcount*
- *cat 0.ohash*

- *cd ../mnt/0*
- *attr -g anycast_status actor/mmls.inf*
- *cat carvpath/0+346.json | json_pp*

The hands-on part



- Installation
- **File-system as API**
  - **CarvPath usage**
  - Actors, workers and jobs.
- Python bindings
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

Exploring the image

- IMG=”carvpath/346+103219200”
- IMGF=$IMG.dd
- mmls $IMGF

- Now consider how you would approach the first partition.

Solution

- PRT=$IMG/$(expr 64 * 512)+$(expr 32952 * 512)
- PRTF=$PRT.dd
- ls -la $PRTF

Looking deeper

- fsstat $PRTF|grep “Block Size”
- fls $PRTF
- istat $PRTF 24

- Now consider how we can approach the first JPEG file.

Solution

- JPG=$PRT/$(expr 2599 * 4096)+578956.jpeg
- echo $JPG
- file -L $JPG
- exif $JPG

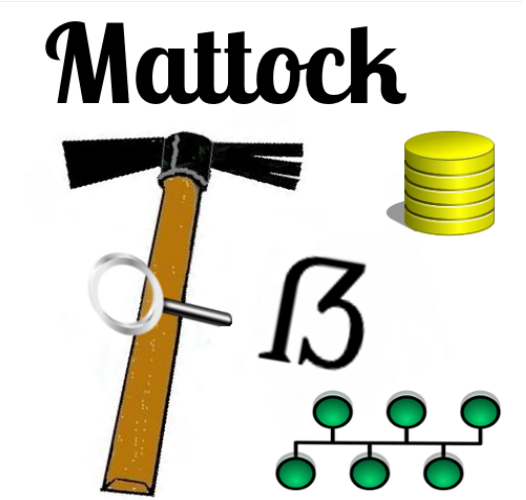
Opportunistic hashing

- export JPG
- python
  - import os
  - f=open(os.environ[“JPG”],”r”)
  - CTRL-Z
- bg
- attr -Lg opportunistic_hash $JPG

Opportunistic hashing (2)

- attr -Lg opportunistic_hash $JPG
- exif $JPG
- attr -Lg opportunistic_hash $JPG
- strings -16 $JPG
- attr -Lg opportunistic_hash $JPG
- cat ../../log/0.ohash

The hands-on part



- Installation
- **File-system as API**
  - CarvPath usage
  - **Actors, workers and jobs**.
- Python bindings
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

Being a worker

- attr -l actor/dsm.inf
- attr -g anycast_status actor/dsm.inf
- attr -l actor/dsm.ctl
- WORKER=$(attr -g register_worker actor/dsm.ctl | tail -1)
- echo $WORKER

Processing a job

- attr -l $WORKER
- JOB=$(attr -g accept_job $WORKER | tail -1)
- Echo $JOB

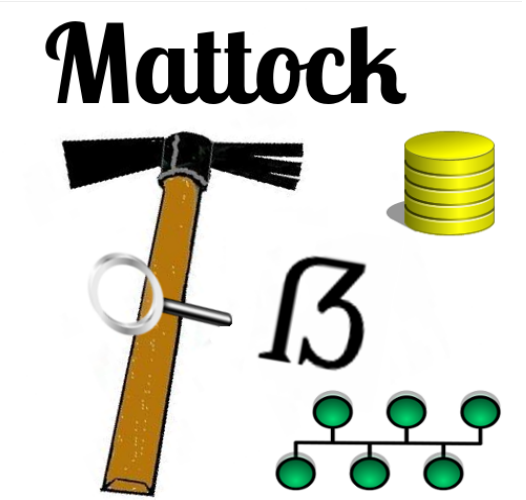
- CP=$(attr -g job_carvpath $JOB | tail -1)
- cat $CP|json_pp

- attr -s routing_info -V “;” $JOB
- JOB= $(attr -g accept_job $WORKER | tail -1)
- tail -1 /var/mattock/log/0.provenance | json_pp

Shutting down a worker

- ls $WORKER
- attr -s unregister -V “1” $WORKER
- ls $WORKER

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- **Python bindings**
  - **Carvpath**
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

Python API

- Simple language binding for FS-based API
- CarvPath dessignations

- cd ~/MattockFS
- python

  from mattock.api import MountPoint

  mp=MountPoint(“/var/mattock/mnt/0”)

CarvPath

```
img=mp[“346+103219200”]
fs=img["32768+16871424"]
pic=fs["10645504+578956"]
print pic.as_file_path()
```

CarvPath (2)

```
from mattock.carvpath import Fragment,Sparse

cp=mp.context.empty()+ Fragment(346,103219200) + Sparse(1000)

print cp

cp = cp + cp + cp + cp +cp + cp + cp + cp

print cp

cp = cp + cp

print cp
```

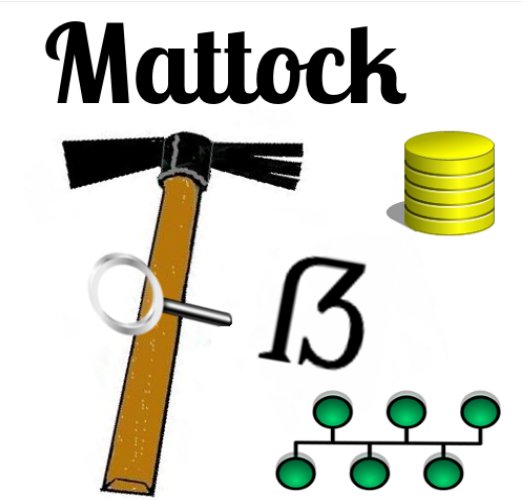
CarvPath (3)

```
print len(cp.fragments)
print cp[0]
print cp[1]
Print cp[0].offset
Print cp[1].size
```

Redis longpath map

```
print str(cp)
print mp.context.longpathmap[str(cp)]
```

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- **Python bindings**
  - Carvpath
  - **Jobs**
  - Carvpath child jobs
  - New data child-jobs
  - Making a naive worker

A worker

- from dfrwsdemo import mmls
- worker=mp.**register_worker**(“mmls”)
- job=worker.**poll_job**()

  ….
- job.**done**()

poll_job vs get_job

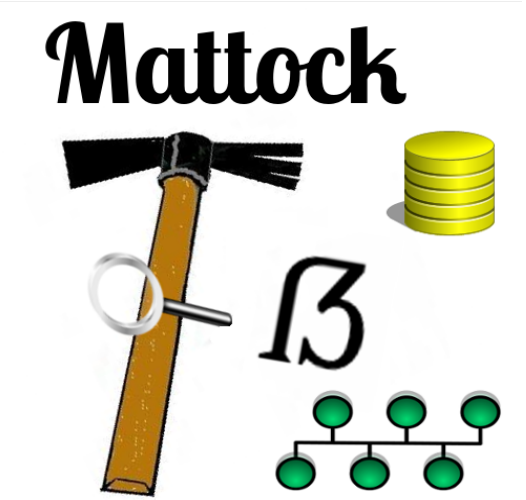
```
# Get the next job, if non is available, keep polling until one is.
def get_job(self):
    while True:
        job = self.poll_job()
        if job is None:
            sleep(0.05)
        else:
            yield job
```

```
mp = MountPoint("/var/mattock/mnt/0")
context = mp.register_worker("actorname")
for job in context.get_job():
    cp=job.carvpath

    ...

    job.done()
```

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- **Python bindings**
  - Carvpath
  - Jobs
  - **Carvpath child jobs**
  - New data child-jobs
  - Making a naive worker

A worker with CarvPath children.

```
from dfrwsdemo import mmls
worker=mp.register_worker(“mmls”)
job=worker.poll_job()
cp=job.carvpath
for poffset,psize in mmls(cp.as_file_path()):
    ccp=cp[str(Fragment(poffset,psize))].as_file_path()
    job.childsubmit(ccp,”fswalk”,”0”,”x-mattock/fs”,”dd”)
job.done()
```

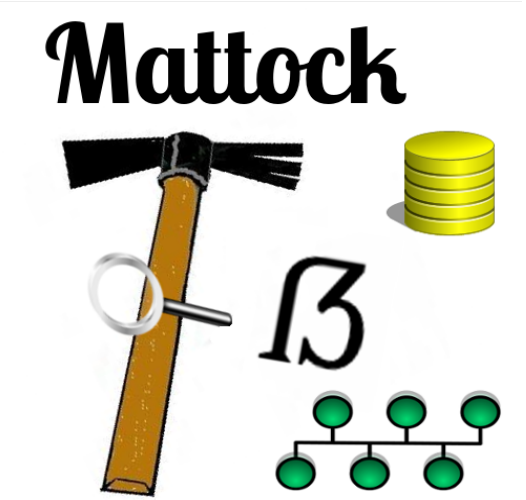
Routing logic

```
from dfrwsdemo import fswalk,is_jpg
worker=mp.register_worker(“fswalk”)
job=worker.poll_job()
cp=job.carvpath
for poffset,psize,i in fswalk(cp.as_file_path()):
    ccp=cp[str(Fragment(poffset,psize))].as_file_path()
    if is_jpeg(ccp):
        job.childsubmit(ccp,”exif”,””,”image/jpeg”,”jpeg”)
job.done()
```

Creating jobs from thin air

```
worker=mp.register(“resubmit”,”K”)
kickjob = worker.poll_job()
kickjob.childsubmit(carvpath=sys.argv[1],
                    nextactor=”mmls”,
                    routerstate="resubmit",
                    mimetype="application/unknown",
                    extension="dd")
kickjob.done()
```

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- **Python bindings**
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - **New data child-jobs**
  - Making a naive worker

Adding data to the archive

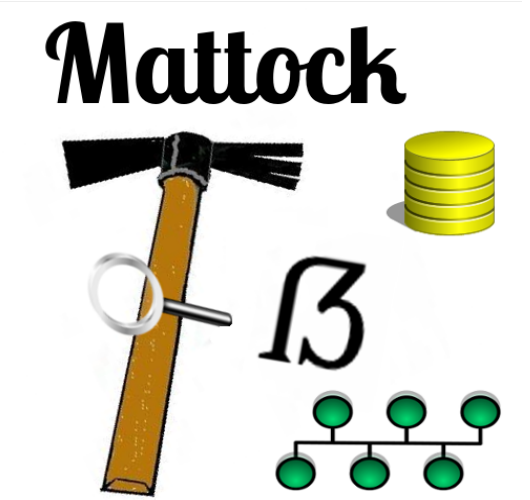
```
childdata = “some new data.”
mutable_path = job.childdata(len(childdata))
With open(mutable_path,”w+”) as f:
    f.seek(0)
    f.write(childdata)
child_carvpath = job.frozen_childdata()
```

A module that adds new data

```
#!/usr/bin/python
from mattock.api import MountPoint
from mattock.carvpath import Fragment
from dfrwsdemo import exif

mp = MountPoint("/var/mattock/mnt/0")
context = mp.register_worker("exif")
for job in context.get_job():
    cp=job.carvpath
    print "Processing",cp.as_file_path()
    exifjson = exif(cp.as_file_path())
    datasize = len(exifjson)
    mutable = job.childdata(datasize)
    with open(mutable, "r+") as f:
        f.seek(0)
        f.write(exifjson)
    exif_carvpath = job.frozen_childdata()
    job.childsubmit(exif_carvpath,"dsm","","x-mattock/exif-meta","json")
    job.done()
```

The hands-on part



- Installation
- File-system as API
  - CarvPath usage
  - Actors, workers and jobs.
- **Python bindings**
  - Carvpath
  - Jobs
  - Carvpath child jobs
  - New data child-jobs
  - **Making a naive worker**

Write a naive module

- Pick one of the following modules:
  - mmls
  - fswalk
- *import dfrwsdemo*
- Use mattock-resubmit to resubmit a carvpath to your module
- Check /var/mattock/log/0.journal to see if your module is working.

The naive mmls module

```
#!/usr/bin/python
from mattock.api import MountPoint
from mattock.carvpath import Fragment
from dfrwsdemo import mmls

mp = MountPoint("/var/mattock/mnt/0")
context = mp.register_worker("mmls")
for job in context.get_job():
    cp=job.carvpath
    print "Processing",cp.as_file_path()
    for poffset,psize in mmls(cp.as_file_path()):
        ccp=cp[str(Fragment(poffset,psize))].as_file_path()
        print "  * ", ccp
        job.childsubmit(ccp,"fswalk","","x-mattock/fs","dd")
    job.done()
```

The naive fswalk

```
#!/usr/bin/python
from mattock.api import MountPoint
from mattock.carvpath import Fragment
from dfrwsdemo import fswalk,is_jpg

mp = MountPoint("/var/mattock/mnt/0")
context = mp.register_worker("fswalk")
for job in context.get_job():
    cp=job.carvpath
    print "Processing",cp.as_file_path()
    for poffset,psize,inode in fswalk(cp.as_file_path()):
        ccp=cp[str(Fragment(poffset,psize))].as_file_path()
        if is_jpg(ccp):
            print "  ["+inode+"]  : " + ccp
            job.childsubmit(ccp,"exif","","image/jpeg","jpg")
    job.done()
```

Things not covered

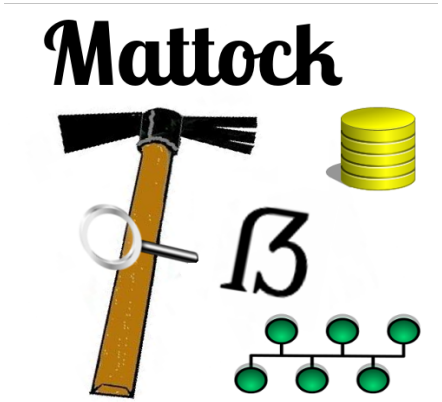
- Throttling
  - Querying potential fs-level fadvise info
  - Querying target module queue size and volume
  - Querying fadvise info on non-sub carvpaths.
- Implementing a worker on a multi-fs node
- Mandatory Access Controls for priv-sep

Schedule

- Asynchronous processing & tool-chain approach
- Integrity, privilege separation and capabilities.
- CarvFS & MinorFS
- MattockFS core design
- MattockFS as distributed-framework building block
- Installation (hands on)
- File-system as API (hands on)
- Python API (hands on)

Status

- Working reference implementation with three core features missing.
  - Restore from journal
    - **Your eyeballs are welcome on fixing this one.**
  - Quarantine of data that crashed modules.
  - Secondary opportunistic hashing.
- Low-level API for FS access in Python, other languages missing:
  - C++
  - **Please consider writing one for your favorite language!**
- Someone could build a full-fledged module library on top of the low level library/libraries. **You ?**
- Someone could build a kick-starting and load balancing mesh-up system on top of a low level API. **You?**
- Eventually: port reference implementation to C++.



Discussion & Questions

Computer-Forensics File-System

Rob J Meijer

pibara@gmail.com
rob.j.meijer@politie.nl

@RumpelstilkinFS
https://github.com/pibara
https://nl.linkedin.com/in/robjmeijer