Using Redis for data processing in a incident response environment.

Practical examples and design patterns.

CIRCL
Computer Incident
Response Center
Luxembourg

Raphaël Vinot

January 23, 2016

Devops & Incident Response

- Time constraints
- Similarity of the problems...
- ... but also very fast evolution
- High flexibility, modularity of the code
- Need to keep track of the evolution of the landscape
- Low maintenance

Devops & Incident Response - Solutions

- Be lazy.
- strings, cat, grep, cut, tr, grep, sort, uniq, parallel
- Simple and readable programming language
- Keeping track of the evolution
- One problem $\rightarrow$ one solution
- Chaining the tools
- Refactoring
- Do not overthink

Devops & Incident Response - Questions

- What am I looking for?
- What is the format of my source?
- Can I grep it?
- Did someone else already searched that? Can I reuse code?
- How much time do I have?
- How much data do I have to process? Protip: going to be much more.
- How often do I have to do it? Protip: as soon as it works, very often

Devops & Incident Response - Writing code

- Do I have to write code?
- data $\rightarrow$ parsable feed $\rightarrow$ less data $\rightarrow$ output
- Am I done now?
  - Do I process my data faster than my input?
  - Am I comfortable processing all the input everytime I need it?
  - Should I keep my output? How long?

Example

```
tshark -E header=yes -E separator=, -r file.pcap | \
cut -d, -f4 " | sort | uniq -c
```

- Many PCAPs, we want to compate them together
- Not really practicable to go through all of them

import.py (first iteration)

```
import sys
for line in sys.stdin:
    print (line.split(","))
```

```
tshark -E header=yes -E separator=, -Tfields \
    -e http.server -r file.pcap | python import.py
```

- Still one single file.
- We don't keep the file name (malware hash)

import.py (second iteration)

```
# Need to catch the MD5 filename of the malware (from the pcap filename)
import argparse
import sys
argParser = argparse.ArgumentParser(description='Pcap classifier')
argParser.add_argument('-f', action='append', required=True, help='Filename')
args = argParser.parse_args()
filename = args.f
for line in sys.stdin:
    print (line.split(",")[0])
```

```
ls -1 ./pcap/*.pcap | parallel --gnu "cat {1} | \
    tshark -E header=yes -E separator=, -Tfields \
    -e http.server -r {1} | python import.py -f {1} "
```

- No history of the files we processed

import.py (third iteration)

```
# Need to know and store the MD5 filename processed
import argparse
import sys
import redis

argParser = argparse.ArgumentParser(description='Pcap classifier')
argParser.add_argument('-f', action='append', required=True, help='Filename')
args = argParser.parse_args()

r = redis.StrictRedis(host='localhost', port=6379, db=0)

md5 = args.f.split(".")[0]
r.sadd('processed', md5)
for line in sys.stdin:
    print (line.split(",")[0])
```

- No flexibility on the field type
- Need to re-run the whole processing every time

import.py (fourth iteration)

```
import argparse
import sys
import redis

argParser = argparse.ArgumentParser(description='Pcap classifier')
argParser.add_argument('-f', action='append', required=True, help='Filename')
args = argParser.parse_args()
r = redis.StrictRedis(host='localhost', port=6379, db=0)

md5 = args.f.split(".")[0]
if md5 not in r.smembers('processed'):
    r.sadd('processed', md5)
    first_line = True
    fields = None
    for line in sys.stdin:
        if first_line:
            first_line = False
            fields = line.strip().split(",")
        else:
            for field, element in zip(fields, line.rstrip().split(",")):
                try:
                    r.sadd('e:'+field, element)
                except IndexError:
                    print("Empty fields")
```

import.py (fourth iteration)

- No Track of fields stored
- High memory use
- Try / except should be avoided is possible

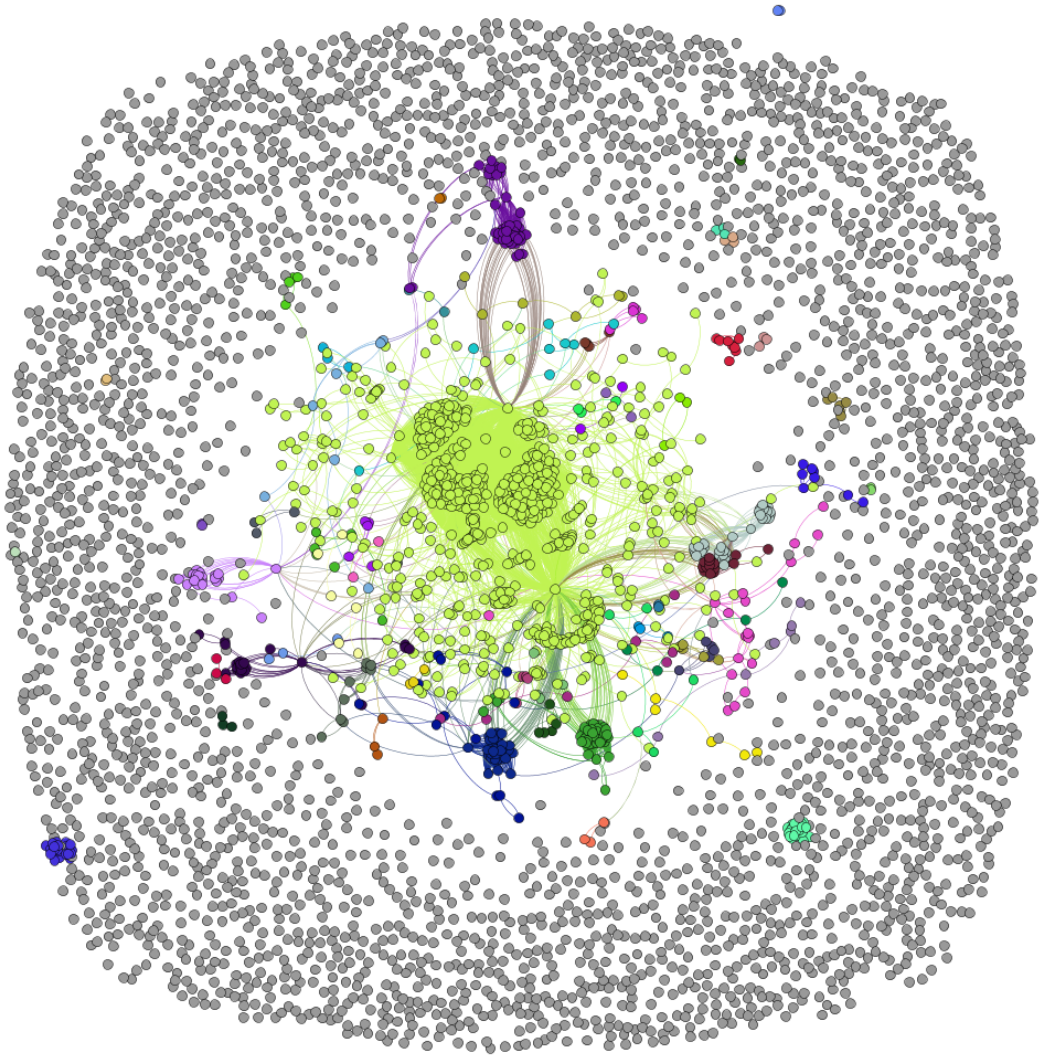
import.py (fourth iteration)

```
import argparse
import sys
import redis
import hashlib

argParser = argparse.ArgumentParser(description='Pcap classifier')
argParser.add_argument('-f', action='append', required=True, help='Filename')
args = argParser.parse_args()
r = redis.StrictRedis(host='localhost', port=6379, db=0)

md5 = args.f.split(".")[0]
if md5 not in r.smembers('processed'):
    r.sadd('processed', md5)
    first_line = True
    for line in sys.stdin:
        if first_line:
            first_line = False
            r.sadd('type', *line.rstrip().split(","))
        else:
            for type, element in zip(r.smembers('type'), line.strip().split(",")):
                r.sadd('e:{}'.format(type), element)
                ehash = hashlib.md5()
                ehash.update(element.encode('utf-8'))
                ehhex = ehash.hexdigest()
                if element is not "":
                    r.sadd('v:'+ehhex, md5)
```

graph.py - exporting graph from Redis

```
import redis
import networkx as nx
import hashlib

r = redis.StrictRedis(host='localhost', port=6379, db=0)

g = nx.Graph()

for malware in r.smembers('processed'):
    g.add_node(malware)

for fieldtype in r.smembers('type'):
    g.add_node(fieldtype)
    for v in r.smembers('e:'+fieldtype.decode('utf-8')):
        g.add_node(v)

        ehash = hashlib.md5()
        ehash.update(v)
        ehhex = ehash.hexdigest()
        for m in r.smembers('v:'+ehhex):
            print (m)
            g.add_edge(v,m)

nx.write_gexf(g,"/tmp/out.gexf")
```

graph.py - exporting graph from Redis



Design patterns

- Shared memory (this code, BGP Ranking, ssdc, url-abuse)
- Pipes (AIL Framework, MultiProcQueue)
- Cache (BGP Ranking, ssdc, uwhoisd)
- Long term storage & fast access (BGP Ranking)
- I don't know what I'm doing (All of them, at the beginning)

Design patterns - Sourcecodes

- https://github.com/CIRCL/bgp-ranking
- https://github.com/CIRCL/AIL-framework
- https://github.com/CIRCL/ssdc/tree/master/multiproc
- https://github.com/Rafiot/bgpranking-hilbert
- https://github.com/Rafiot/MultiProcQueue
- https://github.com/Rafiot/uwhoisd
- ...