Classifying malware using network traffic analysis.

Or how to learn Redis, git, tshark and Python in 4 hours.

CIRCL
Computer Incident
Response Center
Luxembourg

Alexandre Dulaunoy

January 10, 2014

Problem Statement

- We have more 5000 pcap files generated per day for each malware execution in a sandbox
- We need to classify[1] the malware into various sets
- The project needs to be done in less than a day and the code shared to another team via GitHub

[1] Classification parameters are defined by the analyst

File Format and Filename

```
...
0580c82f6f90b75fcf81fd3ac779ae84.pcap
05a0f4f7a72f04bda62e3a6c92970f6e.pcap
05b4a945e5f1f7675c19b74748fd30d1.pcap
05b57374486ce8a5ce33d3b7d6c9ba48.pcap
05bbddc8edac3615754f93139cf11674.pcap
05bf1ff78685b5de06b0417da01443a9.pcap
05c3bccc1abab5c698efa0dfec2fd3a4.pcap
...
<MD5 hash of the malware).pcap
```

MD5 values[2] of malware samples are used by A/V vendors, security researchers and analyst.

[2] https://www.virustotal.com/ as an example

MapReduce and Network Forensic

- MapReduce is an old concept in computer science
  - The **map** stage to perform isolated computation on independent problems
  - The **reduce** stage to combine the computation results
- Network forensic computations can easily be expressed in map and reduce steps:
  - parsing, filtering, counting, sorting, aggregating, anonymizing, shuffling...

Processing and Reading pcap files

```
ls -1  | parallel --gnu 'tcpdump -s0 -A -n -r {1}'
```

- Nice for processing the files but...
- How do we combine the results?
- How do we extract the classification parameters? (e.g. sed, awk, regexp?)

tshark

```
tshark -G fields
```

- Wireshark is supporting a wide range of dissectors
- tshark allows to use the dissectors from the command line

```
tshark -E separator=, -Tfields -e ip.dst -r mycap.cap
```

Concurrent Network Forensic Processing

- To allow concurrent processing, a non-blocking data store is required
- To allow flexibility, a schema-free data store is required
- To allow fast processing, you need to scale horizontally and to know the cost of querying the data store
- To allow streaming processing, write/cost versus read/cost should be equivalent

Redis: a key-value/tuple store

- Redis is key store written in C with an extended set of data types like lists, sets, ranked sets, hashes, queues
- Redis is usually in memory with persistence achieved by regularly saving on disk
- Redis API is simple (telnet-like) and supported by a multitude of programming languages
- `http://www.redis.io/`

Redis: installation

- Download Redis 2.8.3 (stable version)
- tar xvfz redis-2.8.3.tar.gz
- cd redis-2.8.3
- make

Keys

- Keys are free text values (up to 2^{31} bytes) - newline not allowed
- Short keys are usually better (to save memory)
- Naming convention are used like keys separated by colon

Value and data types

- binary-safe strings
- lists of binary-safe strings
- sets of binary-safe strings
- hashes (dictionary-like)
- pubsub channels

Running redis and talking to redis...

- screen
- cd ./src/ && ./redis-server
- new screen session (crtl-a c)
- redis-cli
- DBSIZE

Commands available on all keys

Those commands are available on all keys regardless of their type

- TYPE [key] → gives you the type of key (from string to hash)
- EXISTS [key] → does the key exist in the current database
- RENAME [old new]
- RENAMENX [old new]
- DEL [key]
- RANDOMKEY → returns a random key
- TTL [key] → returns the number of sec before expiration
- EXPIRE [key ttl] or EXPIRE [key ts]
- KEYS [pattern] → returns all keys matching a pattern (!to use with care)

Commands available for strings type

- SET [key] [value]
- GET [key]
- MGET [key1] [key2] [key3]
- MSET [key1] [valueofkey1] ...
- INCR [key] — INCRBY [key] [value] → ! string interpreted as integer
- DECR [key] — INCRBY [key] [value] → ! string interpreted as integer
- APPEND [key] [value]

Commands available for sets type

- SADD [key] [member] → adds a member to a set named key
- SMEMBERS [key] → return the member of a set
- SREM [key] [member] → removes a member to a set named key
- SCARD [key] → returns the cardinality of a set
- SUNION [key ...] → returns the union of all the sets
- SINTER [key ...] → returns the intersection of all the sets
- SDIFF [key ...] → returns the difference of all the sets
- S....STORE [destkey key ...] → same as before but stores the result

Commands available for list type

- RPUSH - LPUSH [key] [value]
- LLEN [key]
- LRANGE [key] [start] [end]
- LTRIM [key] [start] [end]
- LSET [key] [index] [value]
- LREM [key] [count] [value]

Sorting

- SORT [key]
- SORT [key] LIMIT 0 4

Commands availabled for sorted set type

- ZADD [key] [score] [member]
- ZCARD [key]
- ZSCORE [key] [member]
- ZRANK [key] [member] → get the rank of a member from bottom
- ZREVRANK [key] [member] → get the rank of a member from top

Atomic commands

- GETSET [key] [newvalue] → sets newvalue and return previous value
- (M)SETNX [key] [newvalue] → sets newvalue except if key exists (useful for locking)

MSETNX is very useful to update a large set of objects without race condition.

Database commands

- SELECT [0-15] → selects a database (default is 0)
- MOVE [key] [db] → move key to another database
- FLUSHDB → delete all the keys in the current database
- FLUSHALL → delete all the keys in all the databases
- SAVE - BGSAVE → save database on disk (directly or in background)
- DBSIZE
- MONITOR → what's going on against your redis datastore (check also redis-stat)

Redis from shell?

```
ret=$(redis-cli SADD dns:${md5} ${rdata})
num=$(redis-cli SCARD dns:${md5})
```

- Why not Python?

```
import redis
r = redis.StrictRedis(host=’localhost’, port=6379, db=0)
r.set(’foo’, ’bar’)
r.get(’foo’)
```

How do you integrate it?

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

- Code need to be shared?