A bloom filter library in C to support JTAN use-cases

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

Jean-Louis Huynen

Agenda

- Filters in security
- Bloom filters
- Hashlookup introduction
- fleur
- a-ray-grass
- APK impersonation detection pipeline
- Future works

Filters in security

- process a stream to produce another stream following some criteria

```
cat /etc/passwd | grep foo
```

- the criteria can be:
  - a string
  - a regex
  - a yara rule
  - a snort / suricata rule
  - a sigma rule
  - etc.

- parts of these criteria usually comes in the form of lists:
  - IoC and IoA
  - maltrail's data sources
  - MISP events and warning lists
  - lists of known files
  - etc.

Filters in security - Moving parts

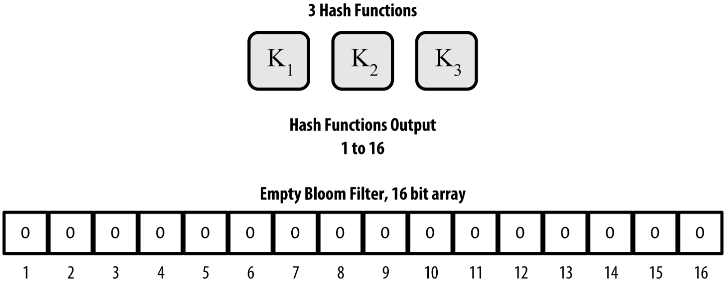
```
rule silent_banker : banker{
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
    $a or $b or $c
}
```

- $a, $b, $c definition look a lot like a list
- (and we have some really big ones)
- there is no easy way to update such lists
    - suricata's datasets is a good example of getting around such issues
    - datasets are lists that can be updated at runtime

Filters in security - Privacy

- These lists always expose their content
- There are use-cases where we'd rather not share openly (even hashes)
  - one can do searches to find the hash's source
  - bloom filters blind the content while keeping it usable
  - bloom filters provide deniability because of their inherent probabilistic nature

Bloom Filters?



3 Hash Functions

K_1 K_2 K_3

Hash Functions Output
1 to 16

Empty Bloom Filter, 16 bit array

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 1: 16 bits array, 3 hash functions

Bloom Filters?

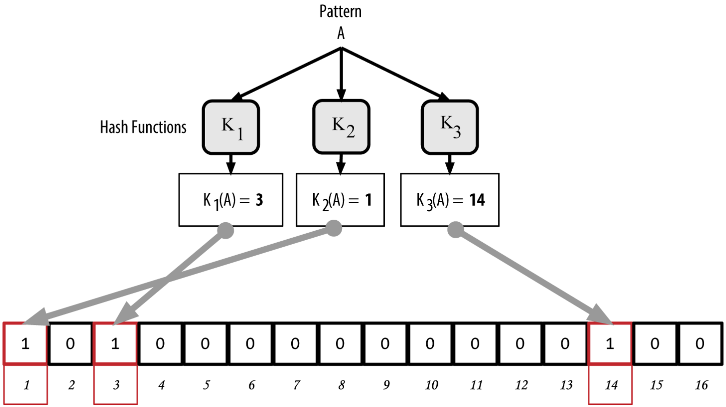
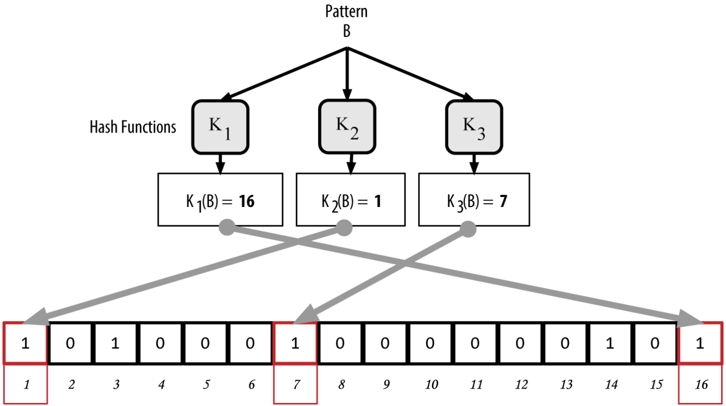


Figure 2: Inserting A

Bloom Filters?



1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*

Figure 3: Inserting B

Bloom Filters?

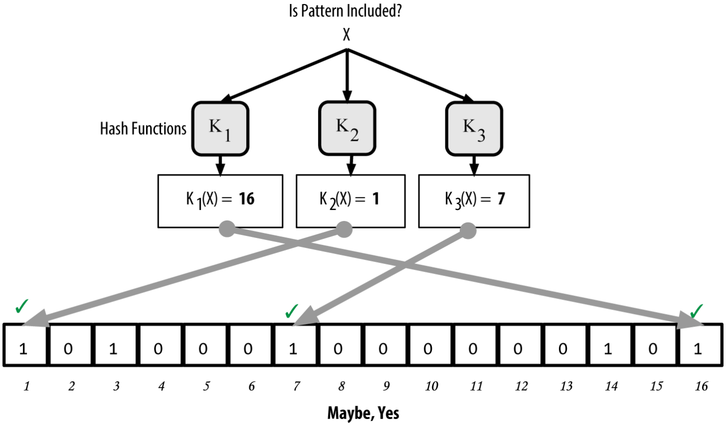
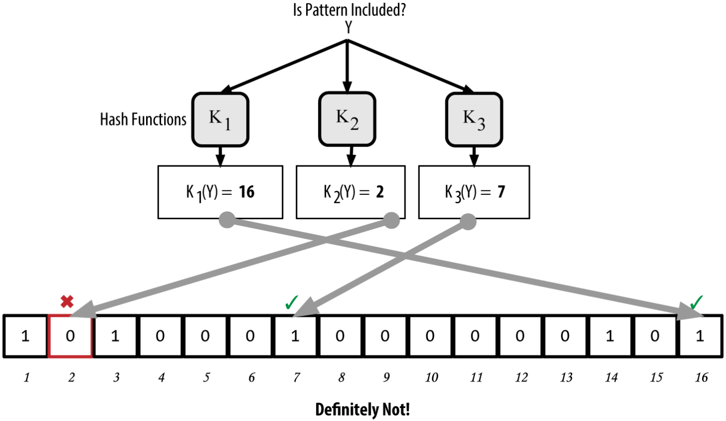


Figure 4: Testing X

Bloom Filters?



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1

Figure 5: Testing Y

Hashlookup introduction

- Current Know Files Filters are failing us (NSRL)
- Hashlookup is a attempt to build decent list of known files (benign files)
- Along with the tooling for such use-cases:
  - sort out interesting files during forensics investigations
  - create list of known files in a organisation (https servers, etc.)
  - create a list of resources used by an organisation (spot reuse and impersonation)
  - keep privacy while sharing

Hashlookup introduction

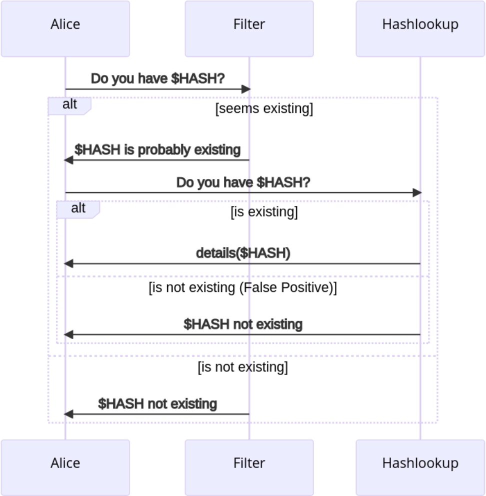
- Given a hashlookup db containing 500.000.000 sha1 digests (20 bytes)
- the list would be 10GB
- a bloom filter with 1.0E-4 would be roughly 1GB
- offline:
  - almost instant response compared querying the online service
  - offline queries remain private
  - false positives can be spotted by querying the online service
- We chose DCSO's bloom filter libraries (bloom and flor):
  - OSS, simple, and easily auditable
  - CLI tools
  - serialization on disk
  - can easily merge or update filters
- https://cra.circl.lu/hashlookup/hashlookup-full.bloom

Hashloop introduction - Offline

```
find /usr/bin/ -type f -print0 | xargs -0 sha1sum
| awk '{ print $1 }'  | tr a-f A-F | bloom c
/home/jlouis/hashlookup-full.bloom
```

```
curl -X 'GET'   'https://hashlookup.circl.lu/lookup/sha1/1939E2A00F90F3A
-H 'accept: application/json' | jq .
```

Hashlookup introduction - Online queries to check FP



fleur

- fleur comes from the need to use hashlookup-like filters in yara
- fleur has been developed in the frame of JTAN
- fleur is an implementation of bloom in language C
- it features:
  - the same features as bloom (only a tad faster)
  - a C API to interface with other tools like yara

a-ray-grass / WIP

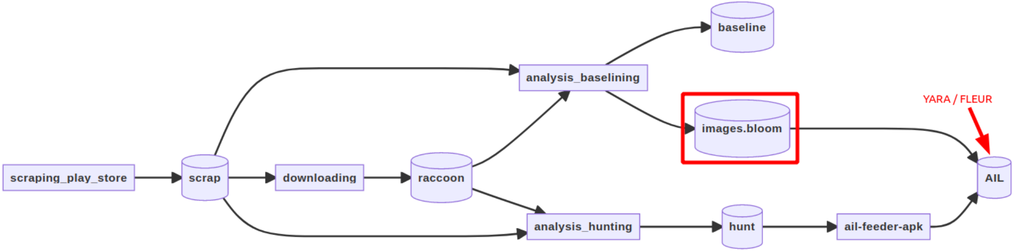
- a-ray-grass is yara module developed in the frame of JTAN
- it allows for the query of a bloom filter in yara rules
- the primary use is filtering known files

```
import "araygrass"
import "hash"

rule HashlookupMatching
{
    condition:
        araygrass.check_string(hash.sha1(0, filesize), 1) == 1
}
```

APK impersonation detection pipeline / WIP

- ail-feeder-apk has been developed in the frame of JTAN
- make use of work done on hashlookup / fleur / a-ray-grass



Future works

- have a hashlookup filter for benign files (remove malshare)
- have hashlookup for different trust levels, and purposes
- use hashlookup services for baselining APK
- run the APK impersonation detection chain in production
- have AIL natively use yara and fleur bloom filters
- make a-ray-grass thread safe on insertion operations
- create smarter detection filters (on images for instance)
- create privacy-aware detection rules

Credits and References

- Mastering Bitcoin - Second Edition by Andreas M. Antonopoulos LLC is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
- Bloom Filter tutorial: https://llimllib.github.io/bloomfilter-tutorial/
- fleur https://github.com/hashlookup/fleur
- flor https://github.com/DCSO/flor
- bloom https://github.com/DCSO/bloom
- a-ray-grass https://github.com/hashlookup/a-ray-grass
- androfleur https://github.com/ail-project/yara/tree/androfleur
- ail-feeder-apk https://github.com/ail-project/ail-feeder-apk