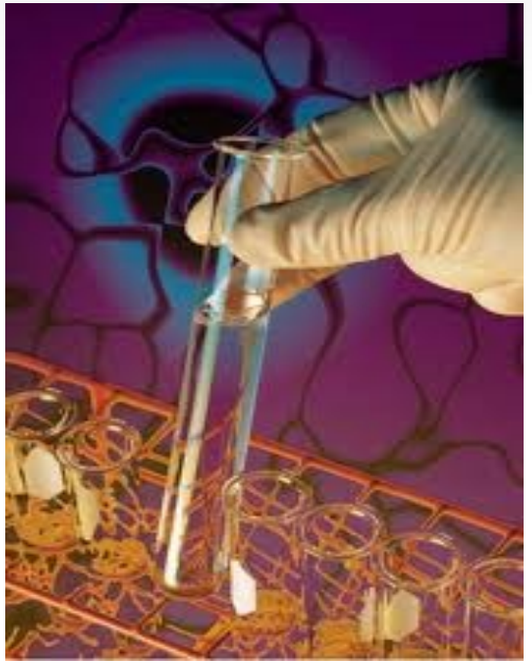
Basic Forensic Techniques

Davide Balzarotti
davide@iseclab.org

Summary

- Entropy
- Crypto and Similarity Hashes
- Strings and Encoding
- Bulk Data Analysis (carving)



Entropy

- In information theory, entropy is used to represent the amount of information (in bits) contained in a message
  - Low entropy means the data is very predictable and it can easily be compressed
  - High entropy means the data is very random and it is therefore hard to compress
- Different types of files have different entropy
  - **High**: compressed or encrypted data
  - **Medium**: machine code, programming languages
  - **Low**: repeated patterns, uncompressed media

Computing Entropy

Shannon Entropy

$$H_b(S) = -\sum_{i=1}^{n} p_i \log_b p_i,$$

Binary entropy (b=2)

```
H = 0
for s in alphabet:
    H += freq(s) * log(freq(s), 2)
H = -H
```

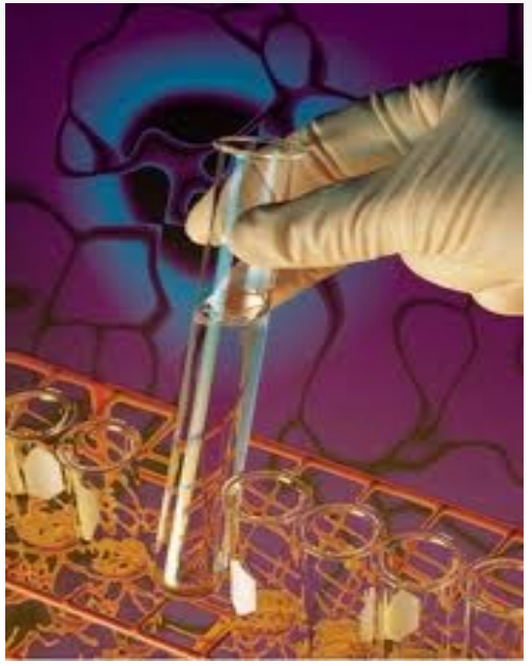
Or, from the command line

```
$ ent file
Entropy = 4.584778 bits per byte
```

`-c` also prints the byte frequency

Summary

- Entropy
- Crypto and Similarity Hashes
- Strings and Encoding
- Bulk Data Analysis (carving)



Cryptographic Hashes

- *Hashing* – deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string (the hash)
- For a *Cryptographic hash*, it should be:
  - infeasible to generate a message that has a given hash
  - infeasible to modify a message without changing the hash
  - infeasible to find two different messages with the same hash
- Used everywhere in forensic examinations
  - To validate media acquisition
  - To find duplicated / renamed files
  - To distribute White and Blacklists

Cryptographic Hashes

- MD5 – 128bit

  Status: BROKEN
  2008 – creation of a rogue SSL certificate
  Best paper award at Crypto conference: http://www.win.tue.nl/hashclash/rogue-ca/

- SHA1 – 160bit

  Status: BROKEN
  2017 – first practical collisions: https://shattered.io/

- SHA2 – (SHA256, SHA384, and SHA512)

For more info about existing attacks against all existing hash functions check: http://www.larc.usp.br/~pbarreto/hflounge.html

Cryptographic Hashes

```
$ md5sum /bin/ls
8a119032efff263f83a5905142959d36  /bin/ls

$ sha256sum /bin/ls
C7c73c51a659682eba186fd13d9f25814cd46273a0e4e
14a0affa266b4dd80dd  /bin/ls
```

- Remember: a crypto hash can tell you that two files are different, but not *how much*

Actually, it is designed to intentionally hide that information:
by design, a difference of 1 bit in the input should change at least half of the bits in the output !!

Hash-based File Identification

- Looking for bad files
  - Child pornography
  - Stolen documents
  - Malicious executables

Hash-based File Identification

- Looking for bad files
  - Child pornography
  - Stolen documents
  - Malicious executables
- Filtering out well-known files
  - Standard operating system files
  - Known good programs
  - NIST maintain a National Software Reference Library (NSRF) updated four times at year and downloadable in DVD format (http://www.nsrl.nist.gov/)

md5deep

- Suite of programs to
  - recursively compute file hashes (md5, sha1, sha256, tiger, whirlpool)
  - match files against a known list of hashes and report either the matches (`-m`) or the ones that do not match (`-x`)

```
# Build a list of MD5s
$ md5deep -e -r ./download > md5.list
$ cat md5.list
c14eb9df081431dbeb735c7723c533fb  /download/a.ppt
acaede97e67d2a4d75dbaa9d259468e9  /download/boh.pdf
...

# Match a list of MD5s against a list of files
$ md5deep -w -m md5.list /data/*.jpg
/data/malet.jpg matched /home/x/download/topo.jpg
```

Problems

- A file may contains very small modifications
  - By design, a change of one bit results in a completely different hash
- The file may be fragmented
  - On different sectors on the disk
  - On different packets in the network
- The file may have been partially overwritten
  - But the majority of it can still be recovered
- The file may be stored in different ways
  - E.g., on disk vs on memory

Block-Based Hashing

- Simple and fast algorithm:
  - Compute a separate hash for each block of the file
    (e.g, 512 bytes or 4K)
  - Search for the individual hashes in the destination
  - Can be computed by using md5deep

```
$ md5deep -p 1k file
```

- Works fine for aligned data
  - But adding one character at the beginning of a file would completely destroy the similarity

Random Sampling

- If we hash each 512-byte sector, a disk of 500GB generates 1 billion MD5s (i.e. 32GB of hashes!)
  - Comparing all of them with a list of malicious hashes takes a loong time
- If the data we are looking for is big enough, random sampling can provide reasonable precision
  - 50MB of bad material

    10.000 sectors → ~10% probability of detection

    100.000 sectors → ~63% probability of detection

    400.000 sectors → ~98.2% probability of detection

Similarity Hashing *(aka Fuzzy Hashes)*

- Context-Triggered Piecewise Hashing
  - Two files are similar if they have some identical sub-parts
  - Tool: SSDEEP
- Statistically Improbable Features
  - Two files are similar if they share some improbable sequences of bytes
  - Tool: SDHASH
- Frequency Distribution of n-Grams
  - Two files are similar if they have similar distributions of substrings of n bytes
  - Tool: TLSH

Similarity Hashing *(aka Fuzzy Hashes)*

- Context-Triggered Piecewise Hashing
  - Two files are similar if they have some identical sub-parts
  - Tool: SSDEEP
- Statistically Improbable
  - Two files are similar if th
  - Tool: SDHASH
- Frequency Distribution
  - Two files are similar if they have similar distributions of substrings of n bytes
  - Tool: TLSH

- Split the file in chunks (based on markers) and generate a 6bit hash of each one
- Concatenate all the hashes in one string
- Use edit distance to estimate the similarity

Similarity Hashing *(aka Fuzzy Hashes)*

- Context-Triggered Piecewise Hashing
  - Two files are similar if th
  - Tool: SSDEEP
- Statistically Improbable
  - Two files are similar if th
  - Tool: SDHASH
- Frequency Distribution of n-Grams
  - Two files are similar if they have similar distributions of substrings of n bytes
  - Tool: TLSH

- Compute entropy of a sliding window (e.g., of 64 bytes)
- Discard the lower and higher values, and identify the “rarest” chunks
- Put the hash of the “rarest” chunks in a sequence of bloom filters
- Similarity based on a normalized entropy measure between two digests

Similarity Hashing *(aka Fuzzy Hashes)*

- Context-Triggered Piecewise Hashing
  - Two files are similar if they have some identical sub-parts
  - Tool: SSDEEP
- Statistically Improbable
  - Two files are similar if th
  - Tool: SDHASH
- Frequency Distribution
  - Two files are similar if th
  - Tool: TLSH

- Extracts all 5-grams and computes each set of three characters for each of them (e.g., “ACE” for “ABCDE”)
- Each sequence is into a value between 0 and 255 and an accumulator array holds the counts for each of these hashes
- The final hash is computed by concatenating 2 bits per hash value of the accumulator (based on their quartiles values)

How Well do They Work?

Scenario 1: embedded object

Tools run on <target> and <object>
Question: how big can be the target to detect an object of fixed size?

Table 1 – Embedded object detection (higher is better).

Object Size (KB)	Max Target Size (KB): 95% Detection	
	ssdeep	sdhash
64	192	65,536
128	384	131,072
256	768	262,144
512	1,536	524,288
1,024	3,072	1,048,576

"*An evaluation of Forensic Similarity Hashes*" Vassil Roussev

How Well do They Work?

Scenario 2: common block correlation

Tools run on <target1> and <target2>
Question: what is the smallest object that correlate the two targets?

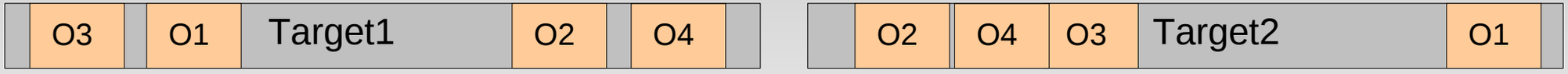
Table 3 — Single-common-block file correlation (lower is better).

Target Size (KB)	Min Object Size (KB): 95% Detection	
	ssdeep	sdhash
256	80	16
512	160	24
1,024	320	32
2,048	512	48
4,096	624	96

"*An evaluation of Forensic Similarity Hashes*" Vassil Roussev

How Well do They Work?

Scenario 3: multiple common blocks correlation

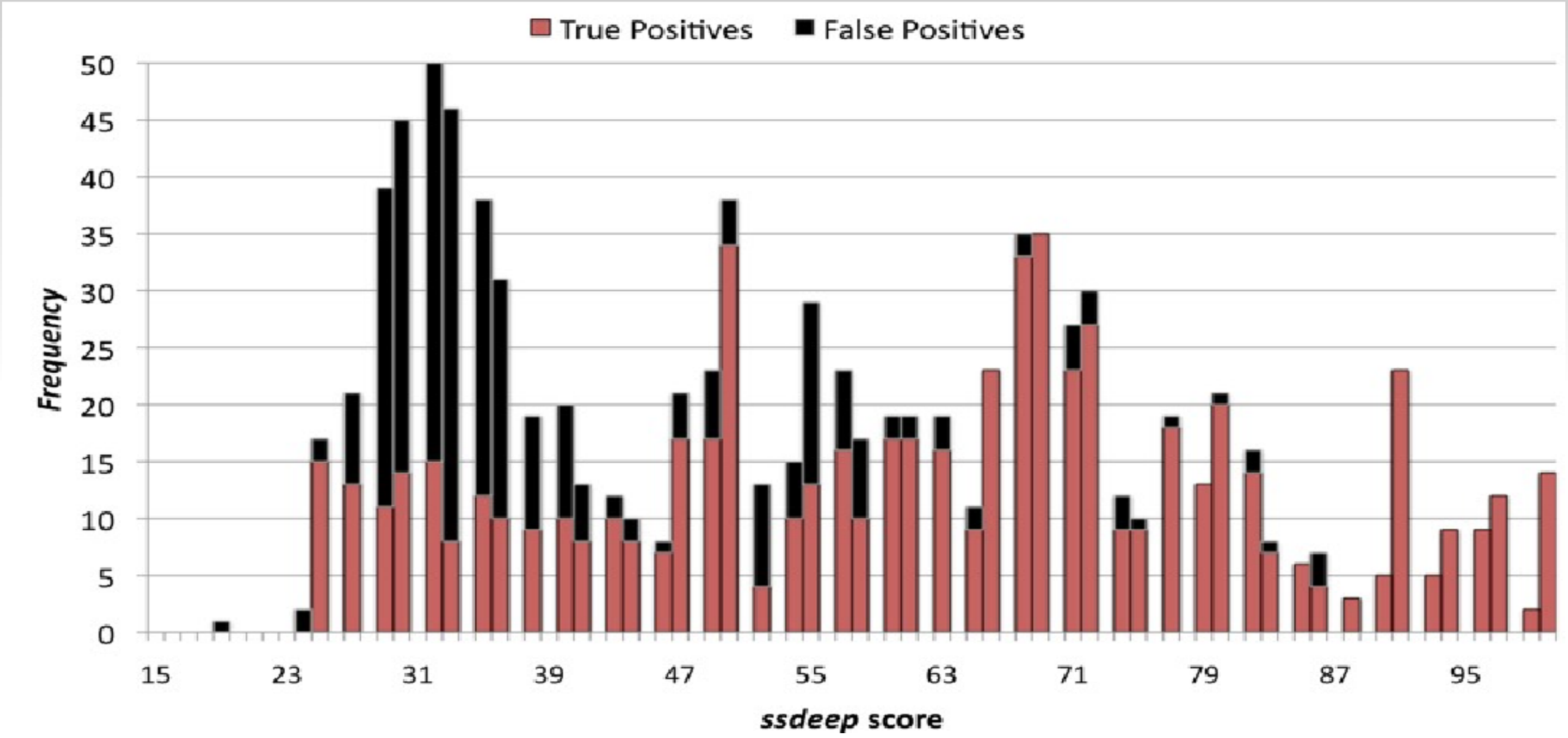


Tools run on <target1> and <target2>
Question: what is the probability that the the two targets get correlated if object is 50% the size of the target?

Table 4 — Multiple-common-blocks correlation probability (higher is better, 1.0 is optimal).

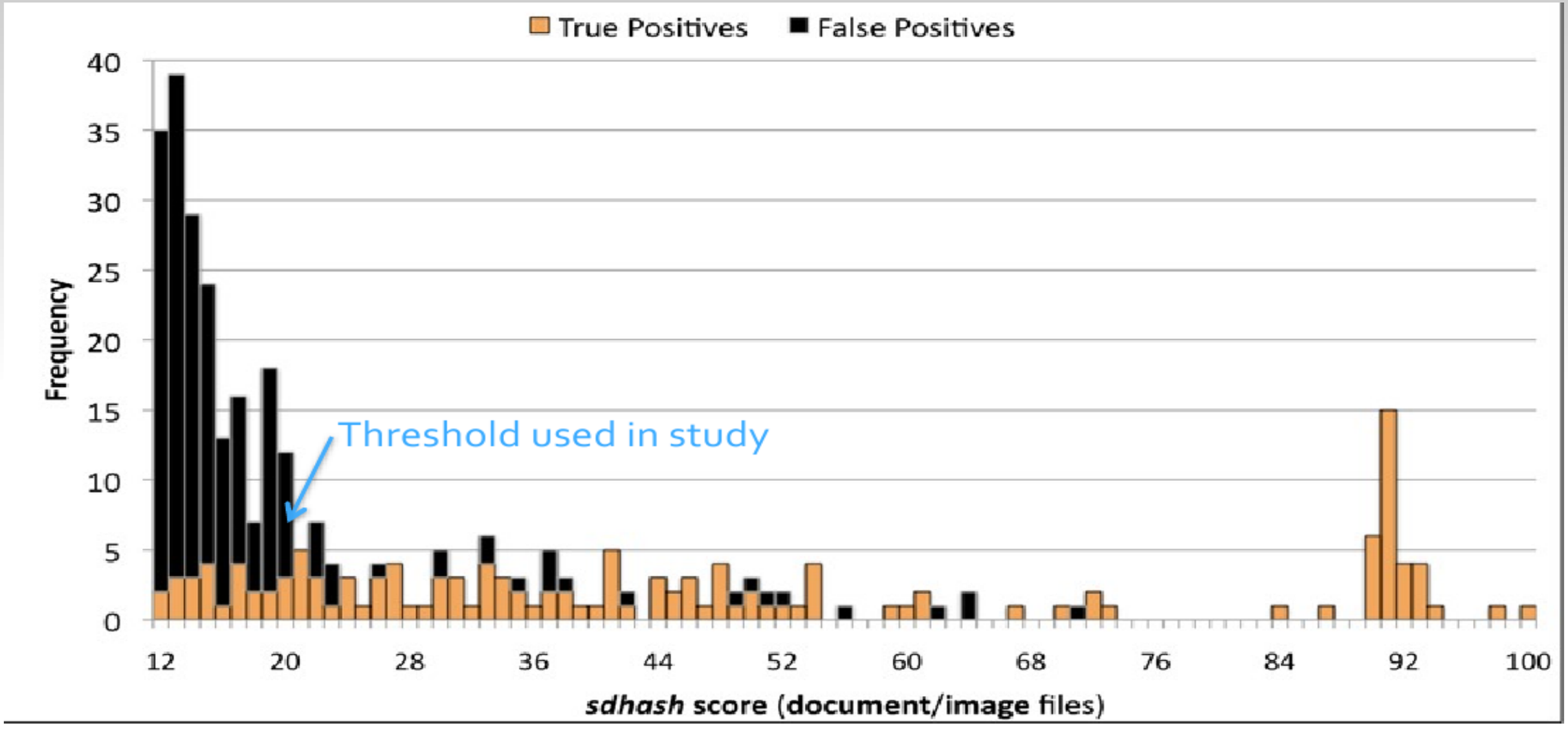
Targets	Common	4 pieces		8 pieces	
(KB)	(KB)	ssdeep	sdhash	ssdeep	sdhash
256	128	0.35	1.00	0.04	1.00
512	256	0.48	1.00	0.04	1.00
1,024	512	0.39	1.00	0.07	1.00
2,048	1,024	0.59	1.00	0.17	1.00
4,096	2,048	0.96	1.00	0.54	1.00

Tuning the Threshold



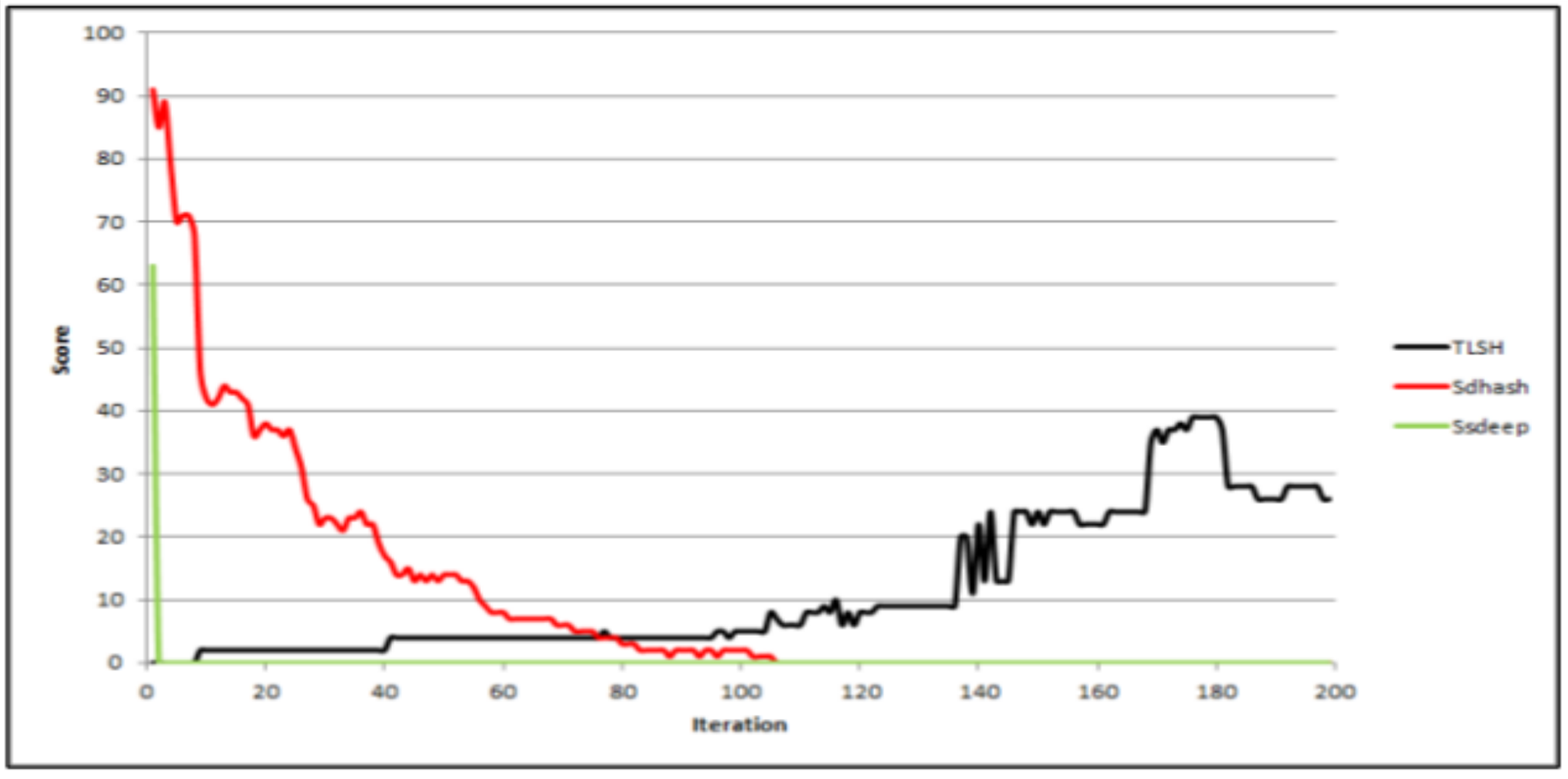
On a set of ~4.400 public files

Tuning the Threshold



On a set of ~4.400 public files

TLSH



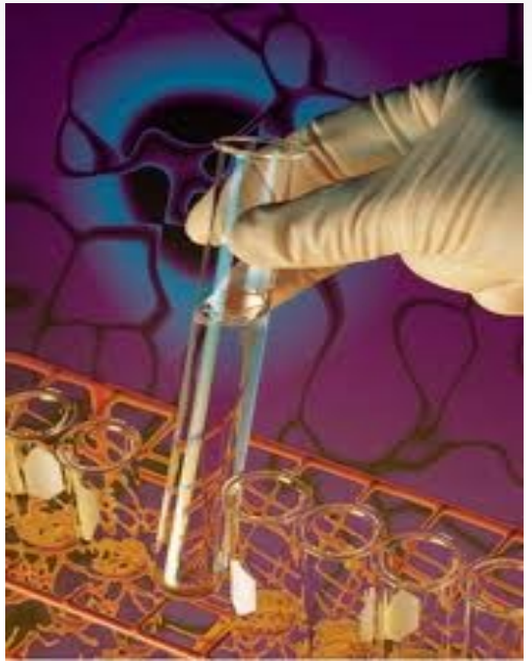
Incremental mutations of Pride and Prejudice

To each Hash its own Job

- `ssdeep` is good where large chunks of data remain exactly the same
- `ssdeep` always return zero similarity if the size of one object is more than double than the other
- `sdhash` is almost always superior to ssdeep
- `sdhash` is good at finding different data with a common core
- `tlsh` is the only algorithm that can cope with small differences distributed all over the data

Summary

- Entropy
- Crypto and Similarity Hashes
- Strings and Encoding
- Bulk Data Analysis (carving)



Extracting Strings

- Useful to extract phone numbers, email addresses, credit card numbers, addresses, or other strings with a fixed format:
  - `grep` will do the job if you know what are you looking for
  - `strings` may help if you do not know what you are looking for

    (check the `-a -n -t` and `-e` options)

- It may seem easy, but try that
  - on MS Windows
  - on a file containing Arabic characters
  - on a 100MB file

Characters Encoding

- A way to associate characters to certain bit patterns
  - ASCII is a 7-bit encoding that contains 95 printable characters
    - OK for English
  - 8-bits characters encodings (such as ISO 8859) extend ASCII by adding a localized page of 128 new symbols
    - Used for Latin, Greek, Cyrillic, ...

Characters Encoding

- UNICODE covers every character used in any language
  - Allows for 17 planes of 64K characters
  - Each symbol is mapped to a *code point* (a sequence of non-negative integers) – there are over 1 million code points
  - Last version (9.0) includes more than 128K symbols
  - To represent them, we would need a 21-bit per symbol encoding
    - Most systems maps them to a variable-length sequence of bytes (UTF-8) or to a variable-length sequence of 16 bits (UTF-16)

UTF

- UTF-8 encoding
  - If the code point is <128, it's represented by the corresponding ASCII byte value
  - If the code point is between 128 and `0x7ff`, it's turned into two byte values between 128 and 255
  - Code points > `0x7ff` are turned into three- or four-byte sequences, where each byte of the sequence is between 128 and 255
- UTF-16 uses instead sequences of two bytes
  - Original Unicode encoding
  - Endianness is important
  - Also the first 128 characters are not anymore compatible with ASCII
  - Used by Microsoft

Playing with Unicode

```
>>> u = unichr(40960)+ u'Hello World'+ unichr(1972)
>>> u8 = u.encode('utf-8')
>>> u16 = u.encode('utf-16')
>>> print "%r"%u16
# the first two bytes represent the endianness
'\xff\xfe\x00\xa0H\x00e\x00l\x00l\x00o\
```

Playing with Unicode

```
>>> u = unichr(40960)+ u'Hello World'+ unichr(1972)
>>> u8 = u.encode('utf-8')
>>> u16 = u.encode('utf-16')
>>> print "%r"%u16
# the first two bytes represent the endianness
'\xff\xfe\x00\xa0H\x00e\x00l\x00l\x00o\
```

```
$ grep Hello file.u8
Binary file file.u8 matches
$ grep Hello file.u16
$ strings file.u16
$ strings -e l file.u16
Hello World
$ iconv -f UTF-16 -t UTF-8 uni2
hello World
```

Strings in URLs

- Percent-encoding (aka urlencoding) is used to represent any reserved and unprintable character inside an URI
  - Anything not in the set **`[A-Za-z0-9-_.~]`** is replaced by a **%** sign followed by the hexadecimal representation of the character
  - Unreserved characters should not be encoded... but sometimes they are
  - E.g. “`Hello world!`” → “`Hello%20world%21`”

```
>>> import urllib
>>> urllib.encode("Hello world!")
'Hello%20world%21'
>>> urllib.unquote("Hello%20world%21")
'Hello world!'
```

Binary to ASCII Encodings

- Hexadecimal
  - 2 bytes per character
    (used to represent hashes or to transfer files over remote shells)

```
$ echo 'Hello world!' | xxd -ps
48656c6c6f20776f726c6421
$ echo '48656c6c6f20776f726c6421' | xxd -r -ps
Hello world!
```

- Unix-to-Unix Encoding (UUencoding)
  - Encode 6 bits at a time
  - Use special header and footer
  - Largely replaced by MIME encodings

```
$echo 'Hello World!' | uuencode -
begin 664 -
-2&5L;&\@5V]R;&0A"@``
`
end
```

Binary to ASCII Encodings

- Base64
  - Encode 6bit at a time, non human readable
  - One of the two MIME supported encodings
  - Largely used in email attachments

```
$ echo 'Hello World!' | base64
SGVsbG8gV29ybGQhCg==
$ echo 'SGVsbG8gV29ybGQhCg==' | base64 -d
Hello World!
```

- Quoted-printable
  - The other MIME binary to text encoding
  - Human readable, efficient if most of the characters are printable
  - Escape non-printable characters with a = sign (“`Hello=20world!`”)

Encoding in Python

`string.encode(...)` `string.decode(...)`

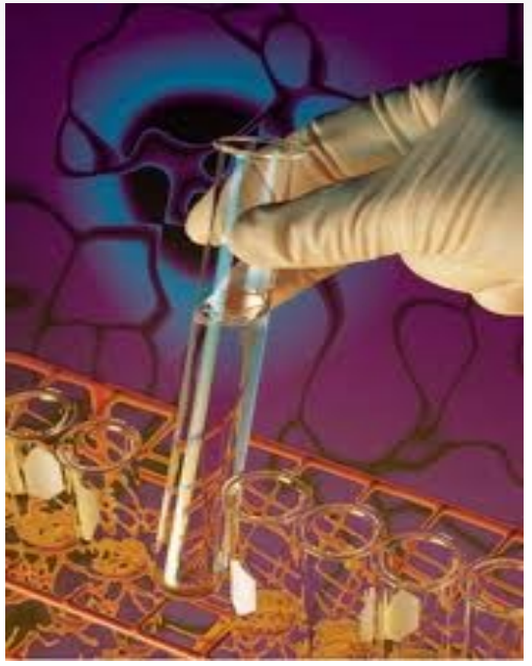
- More than 100 different formats supported !!

  http://docs.python.org/library/codecs.html

```
>>> "Hello world!".encode("quopri")
'Hello=20world!'
>>> "c2VjcmV0".decode("base64")
'secret'
>>> "Hello world!".encode("hex")
'48656c6c6f20776f726c6421'

>>> "Hello world!".encode("rot13")
'Uryyb jbeyq!'
>>> "Hello world!".encode("zlib")
'x\x9c\xf3H\xcd\xc9\xc9W(\xcf/\xcaIQ\x04\x00\x1d\t\x04^'
```

Summary

- Entropy
- Crypto and Similarity Hashes
- Strings and Encoding
- Bulk Data Analysis (carving)



Bulk Data Analysis

- Process of searching an input data for files or other kinds of artifacts based on their content
- It is very useful when
  - The format in which the data is stored is unknown
    - Carve a MS Word document out of a disk image
    - Carve a jpeg picture out of a MS Word document
  - The structure or the metadata of the container are damaged or overwritten
    - Recover a deleted file
    - Recover data from a formatted partition
- It is ineffective if data encoding is unknown

Carving Files

- Headers & Footers -based carvers
  - Does not work on fragmented files
  - Lots of file positives if header and footer are too short
  - When files have no clear footers, a fixed amount of bytes are extracted

Carving Files

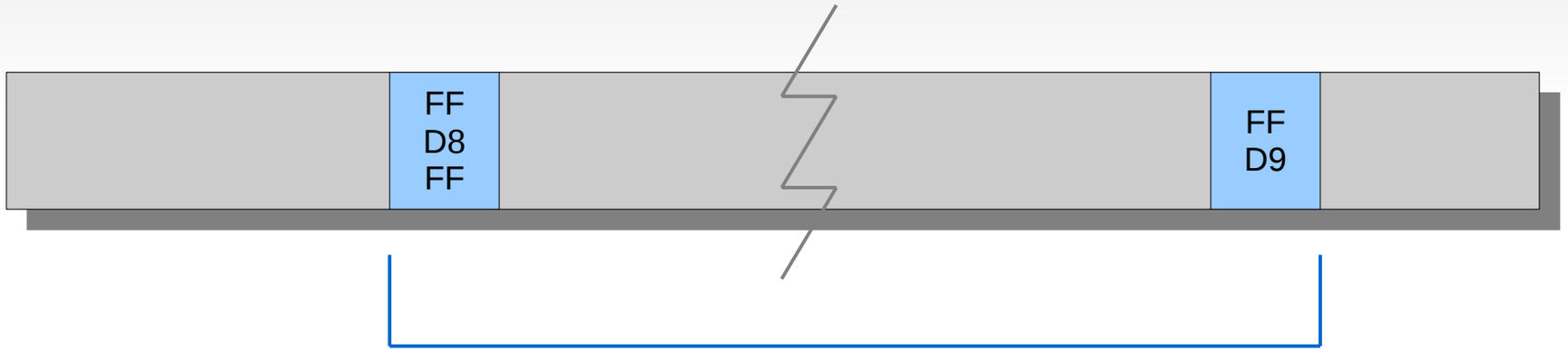
- Headers & Footers -based carvers
  - Does not work on fragmented files
  - Lots of file positives if header and footer are too short
  - When files have no clear footers, a fixed amount of bytes are extracted
- File Structure -based carvers
  - Parse the file internal structure to locate and identify information (file size, type, …)
  - Less false positives, but requires structured data

Carving Files

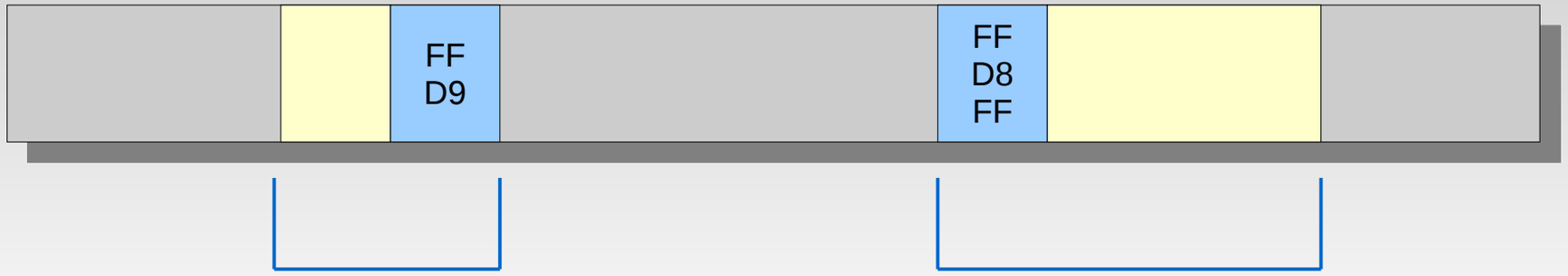
- Headers & Footers -based carvers
  - Does not work on fragmented files
  - Lots of file positives if header and footer are too short
  - When files have no clear footers, a fixed amount of bytes are extracted
- File Structure -based carvers
  - Parse the file internal structure to locate and identify information (file size, type, …)
  - Less false positives, but requires structured data
- Content -based carvers
  - Look at the content of each block to identify the file type
  - Often based on entropy and statistical information about the bytes distribution

Example: Carving JPG Images

- Files start with **`0xFFD8`** (start of image signature)
- Then, they contain a number of information blocks, each starting with **`0xFFxx`** (marker signature, where `xx` is the block type)
- After the markers there is the data stream, starting with **`0xFFDA`** and ending with **`0xFFD9`**



Fragmentation



- Approaches exist to reassemble fragmented files
  - Usually format dependent (e.g., for `jpg` or `avi`)
  - Often support only linear (i.e. with fragments in order) fragmentation
  - May rely on disk fragmentation heuristics
  - E.g., `Adroit Photo Forensics` (~2.000 $) is the state of the art to recover fragmented photos

General Purpose Carving Tools

- Scalpel
  - A spin-off of an old version of the `foremost` tool
  - Version 2.0: https://github.com/machn1k/Scalpel-2.0
  - Multi-threading and GPU (NVIDIA Cuda) support
  - A configuration file controls what kind of information you want to extract

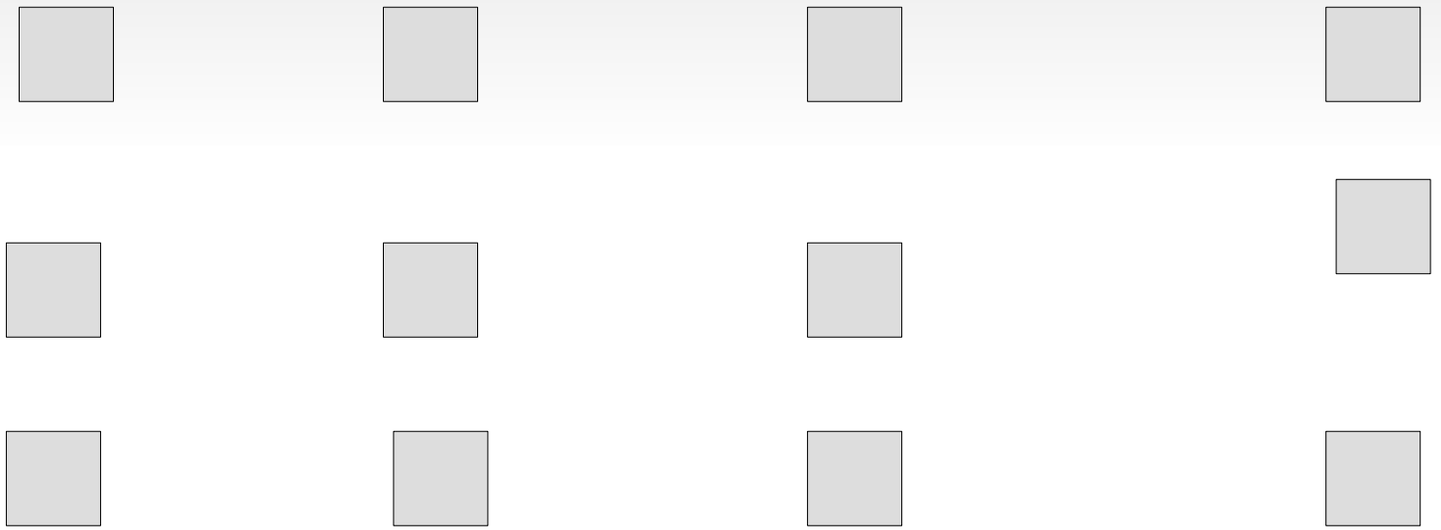
```
# file_ext case  [minSize:]maxSize  header_regex footer_regex

# PNG Files
png  y  2000000  \x50\x4e\x47?  \xff\xfc\xfd\xfe
```

Hash-Based Carving

- Idea: blocks with high entropy are likely unique and they are rarely found by chance in other documents
  - Multimedia files, encrypted data, original documents, …
- If a block is found in a device, it is very likely that the entire document is (or was in the past) there as well
- Tool: `frag_find` (http://afflib.org)

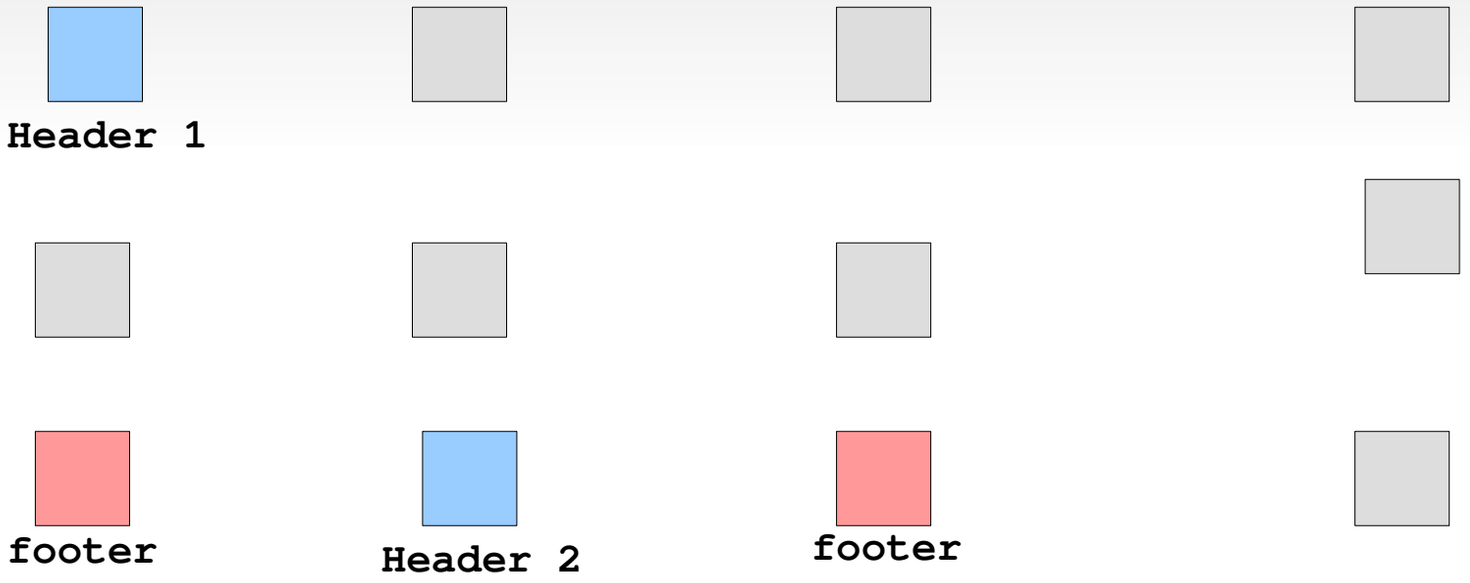
Format-Specific Carving Techniques

- Take advantage of the intrinsic characteristics of a particular file type to locate it and reassemble its fragment.
- Example: jpeg



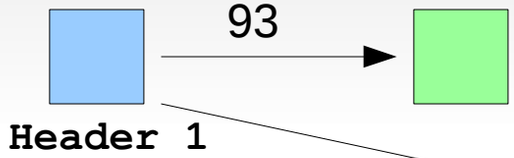
Format-Specific Carving Techniques

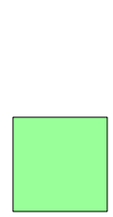
- Take advantage of the intrinsic characteristics of a particular file type to locate it and reassemble its fragment.
- Example: jpeg



Format-Specific Carving Techniques

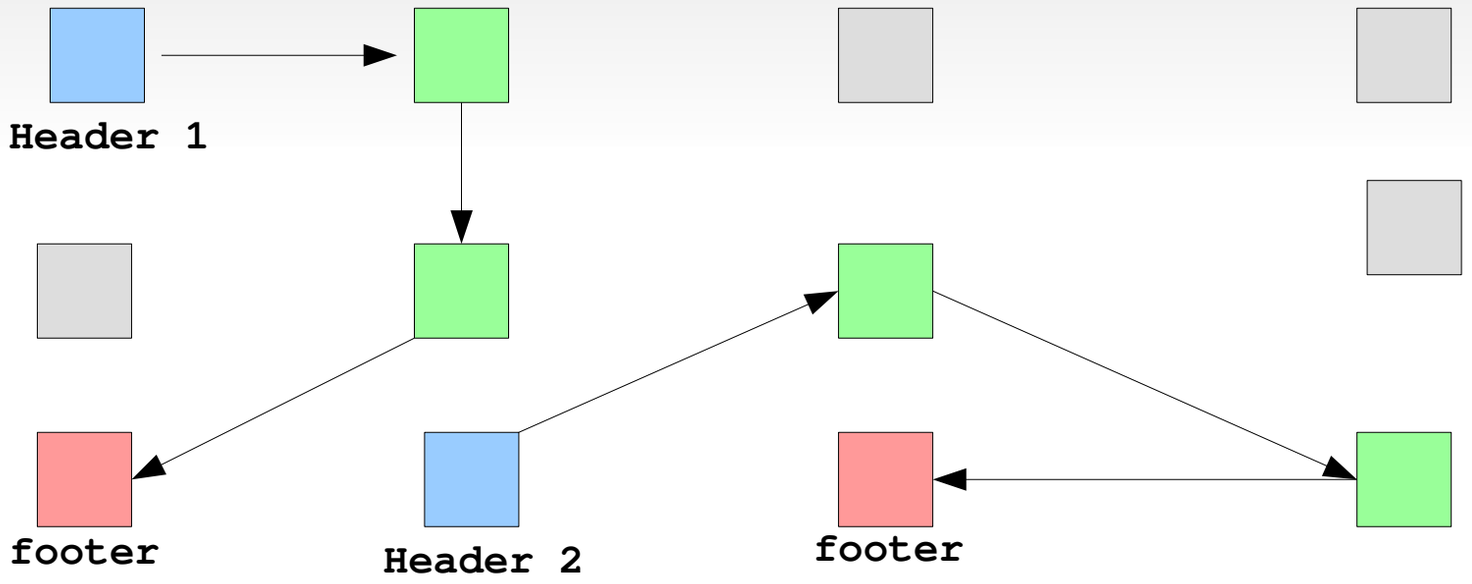
- Take advantage of the intrinsic characteristics of a particular file type to locate it and reassemble its fragment.
- Example: jpeg





Format-Specific Carving Techniques

- Take advantage of the intrinsic characteristics of a particular file type to locate it and reassemble its fragment.
- Example: jpeg



Extracting Forensic Features with BulkExtractor

- Stream-based analysis
- Useful to quickly process a lot of data to extract emails, credit card numbers, phone numbers, GPS coordinates...
  - The data is split in 16MB (partially overlapping) pages and the system processes one page on each available core
  - Scanners extract a number of features and save them in separate files
  - Histograms of the most common values are automatically computed
  - Automatically detects, decompresses, and recursively re-processes compressed data
    - ZLIB
    - Windows Hibernation files
    - PDFs
    - Browser cache

Bulk Extractor

- Tool: bulk_extractor

  https://github.com/simsong/bulk_extractor

- On my machine ~ 100 seconds to analyze a 850MB pcap trace

```
$ ls
aes_keys.txt       ether_histogram.txt  telephone.txt
alerts.txt         ether.txt            url_facebook-id.txt
ccn_histogram.txt  exif.txt             url_histogram.txt
ccn_track2_histogram.txt  find.txt      url_microsoft-live.txt
ccn_track2.txt     gps.txt              url_searches.txt
ccn.txt            json.txt             url_services.txt
domain_histogram.txt  kml.txt           url.txt
domain.txt         report.xml           winprefetch.txt
email_histogram.txt      rfc822.txt     zip.txt
email.txt          telephone_histogram.txt

$ cat telephone.txt
3169057 800-786-9199  ure by calling 800-786-9199 or
3170849 703-305-7785  </p><p> TTY: 703-305-7785 \012
3188827 800-786-9199  Services at 800-786-9199 or 703-308-
```

Stop Lists

- Running the tool on a clean computer would still extract A LOT of features
  - An Fedora 12 installation contains over 20.000 email addresses !!
- bulk_extractor supports list of known features to filter out
  - Can be prepared by the analyst based on the case
  - Can be generated by the tool itself by running it on a clean operating system
- These lists (called stop lists) contain feature values enriched with a bit of context (default 8 bytes per side)
  - An entry is skipped only when the entry appears in the same context
  - Stoplist available at: http://afflib.org/downloads/feature_context.1.0.zip

- “Identifying Almost Identical Files Using Context Triggered Piecewise Hashing”
  J. Kornblum – DFRWS 2006

- “TLSH – A Locality Sensitive Hash”
  J. Olivier, C. Cheng, Y. Chen – Cybercrime and Trustworth Computing Workshop 2013

- “An evaluation of Forensic Similarity Hashes”
  Vassil Roussev – DFRWS 2011

- “Data fingerprinting with Similarity Digests”
  V. Roussev – Research Advances in Digital Forensics 2010

- “Digital Media Triage with Bulk Data Analysis and Bulk Extractor”
  Simson L. Garfinkel – Journal of Computer Security 2013

- “Measuring and Improving the Quality of File Carving Methods”
  S.J.J. Kloet Master Thesis