A Fast Randomness Test that Preserves Local Detail

Tim Ebringer – The university of Melbourne, Australia
Li Sun – RMIT university, Australia
Serdar Boztas – RMIT university, Australia
VB 2008
Ottawa, Canada, 1-3rd Oct

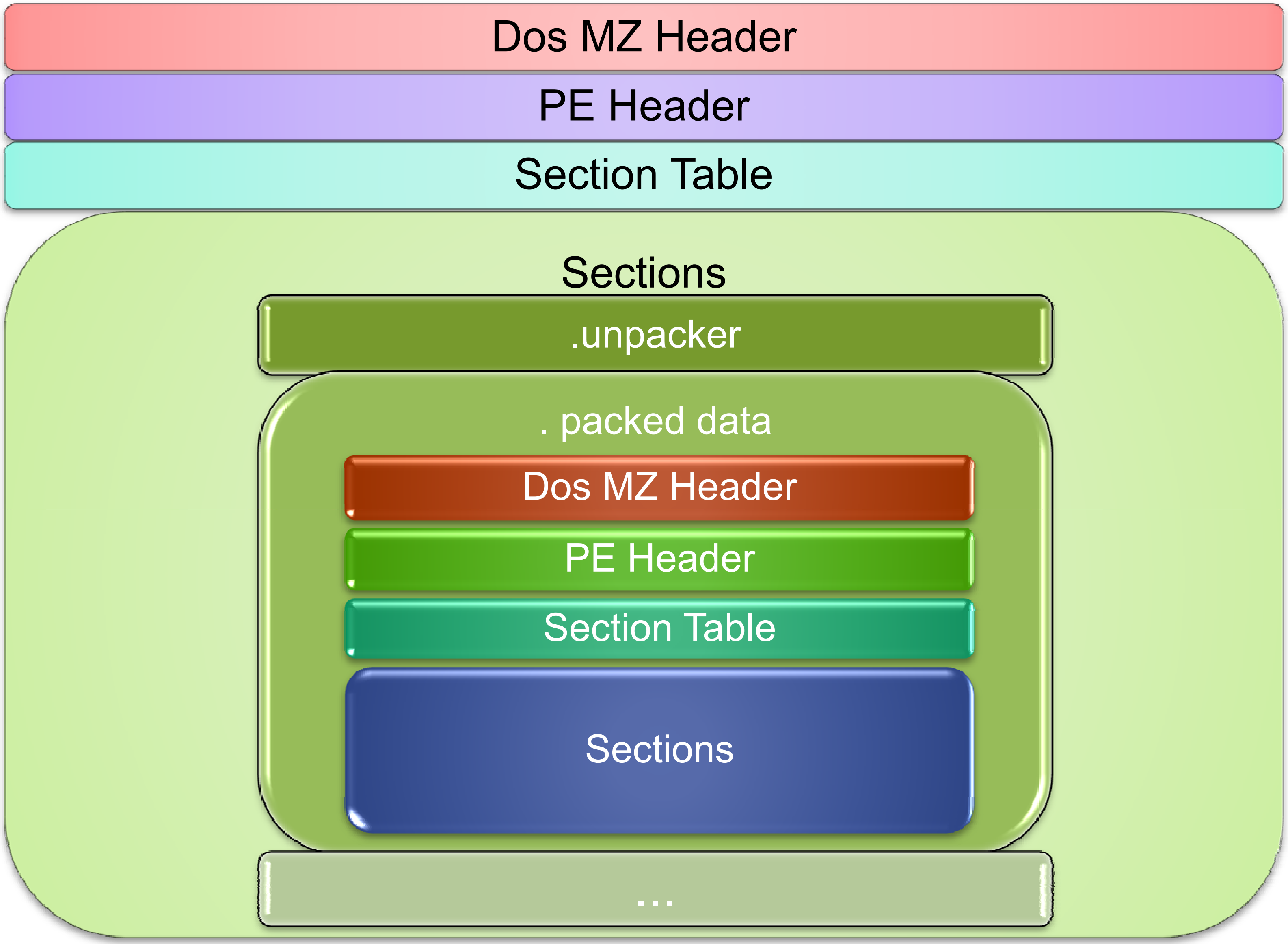
Acknowledgment

- This research was supported and sponsored by CA Labs, the research division of CA.
  - Ms Sun is presently a Ph.D. candidate sponsored by CA Labs
  - Dr Ebringer was an employee of CA, working in the CA Labs

Overview

- Packer and its characteristic
- Randomness test
- Algorithms
- Experiments and results
- Further work
- Conclusion

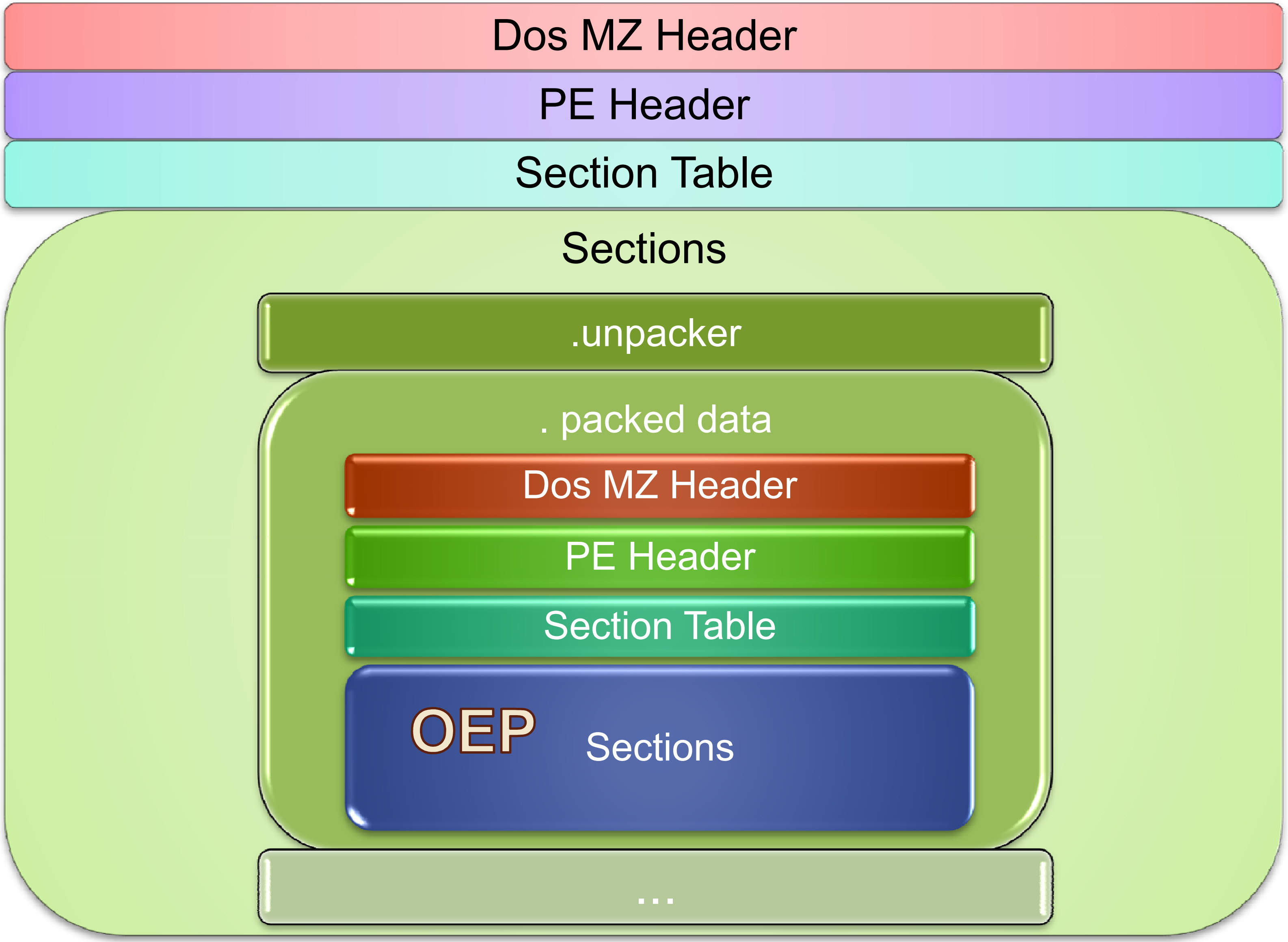
PE packing


Dos MZ Header
PE Header
Section Table
Sections
.unpacker
. packed data
Dos MZ Header
PE Header
Section Table
Sections
...

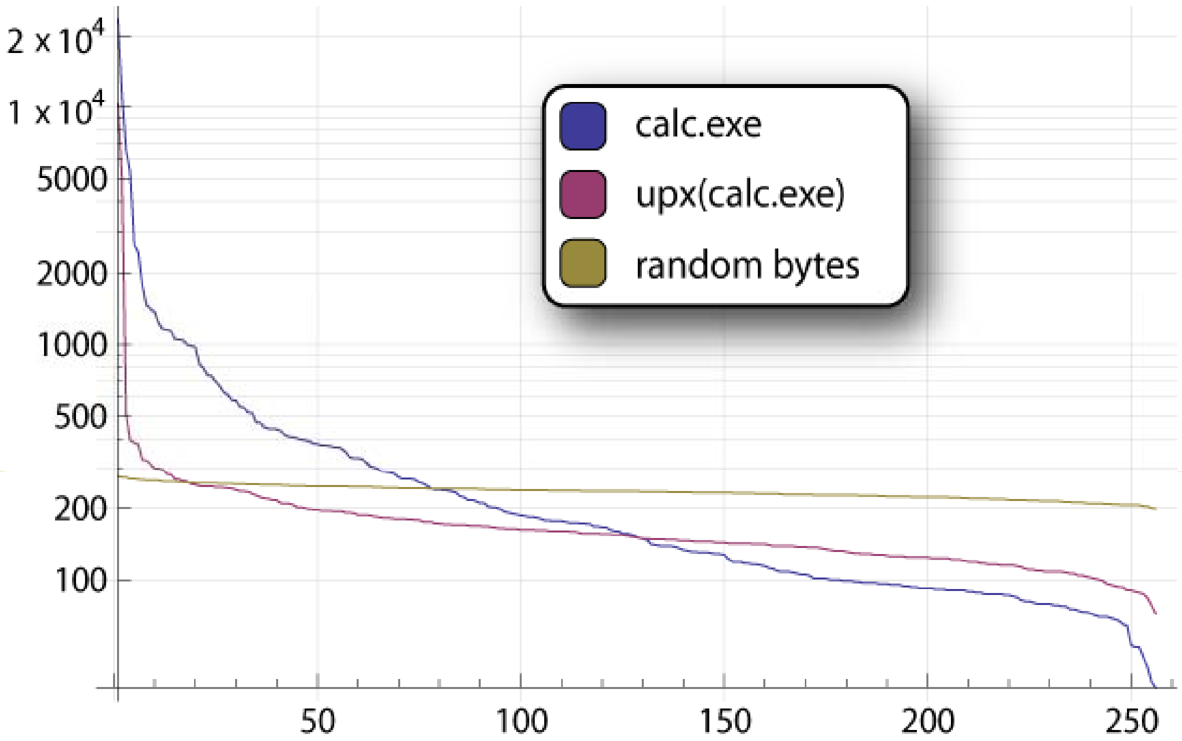
Packers

- UPX
- FSG
- MEW
- Upack
- PCShrinker
- PECompact
- Morphine
- ASPack
- ASProtect
- tElock
- Armadillo
- Themida
- VMProtect
- …

PE unpacking



Byte frequency distribution of a packed file



How random are the data ?

- Shannon entropy – measures the amount of uncertainty in a variable

$$H(X) = -\sum_{i=1}^{n} p(X = x_i) \log_2(p(X = x_i))$$

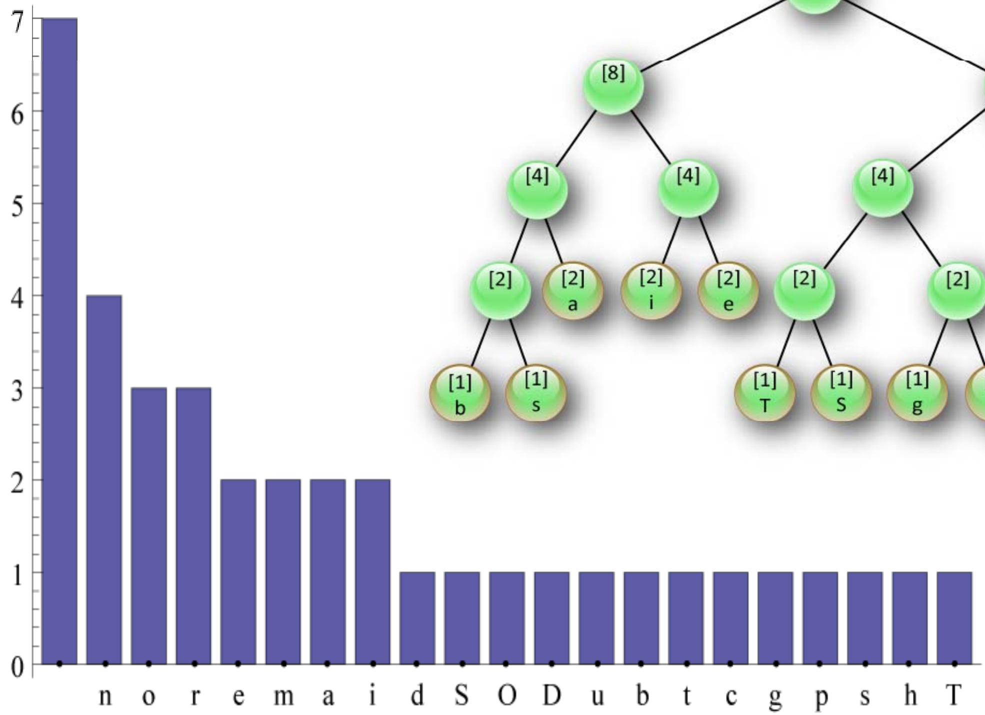
- Randomness test
  - TAOCP (Art of Computer Programming, Knuth)
  - DIEHARD (Marsaglia)
  - DIEHARDER (Brown)

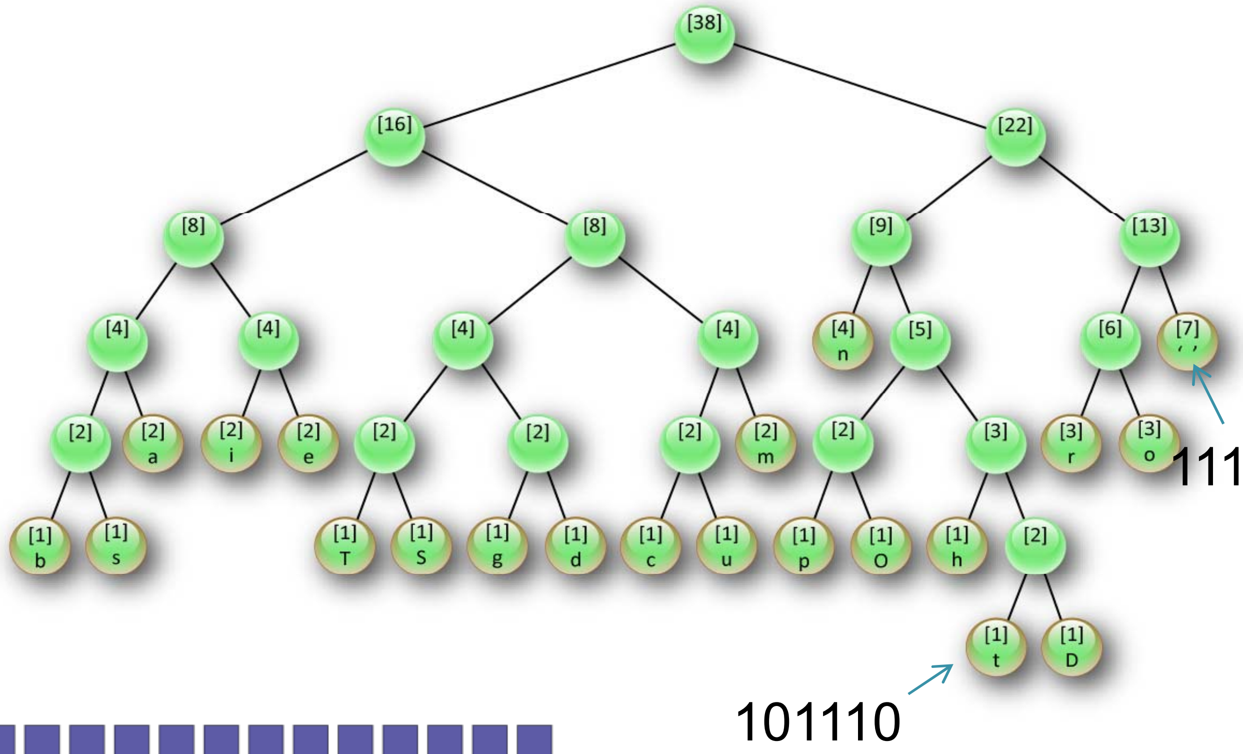
Algorithms

- Build a byte-frequency histogram;
- Construct the Huffman tree by inserting bytes into the tree in the order of their frequency;
- Construct a “length-encoding” array, which gives the distance to the top of the tree for each element. This is the number of bits needed to encode this byte.
- Use the total code length to represent the corresponding data
  - Fixed sample count
  - Sliding window
- Very fast!

Huffman coding

- Variable length coding -- fast
- Example “ This program cannot be run in DOS mode”



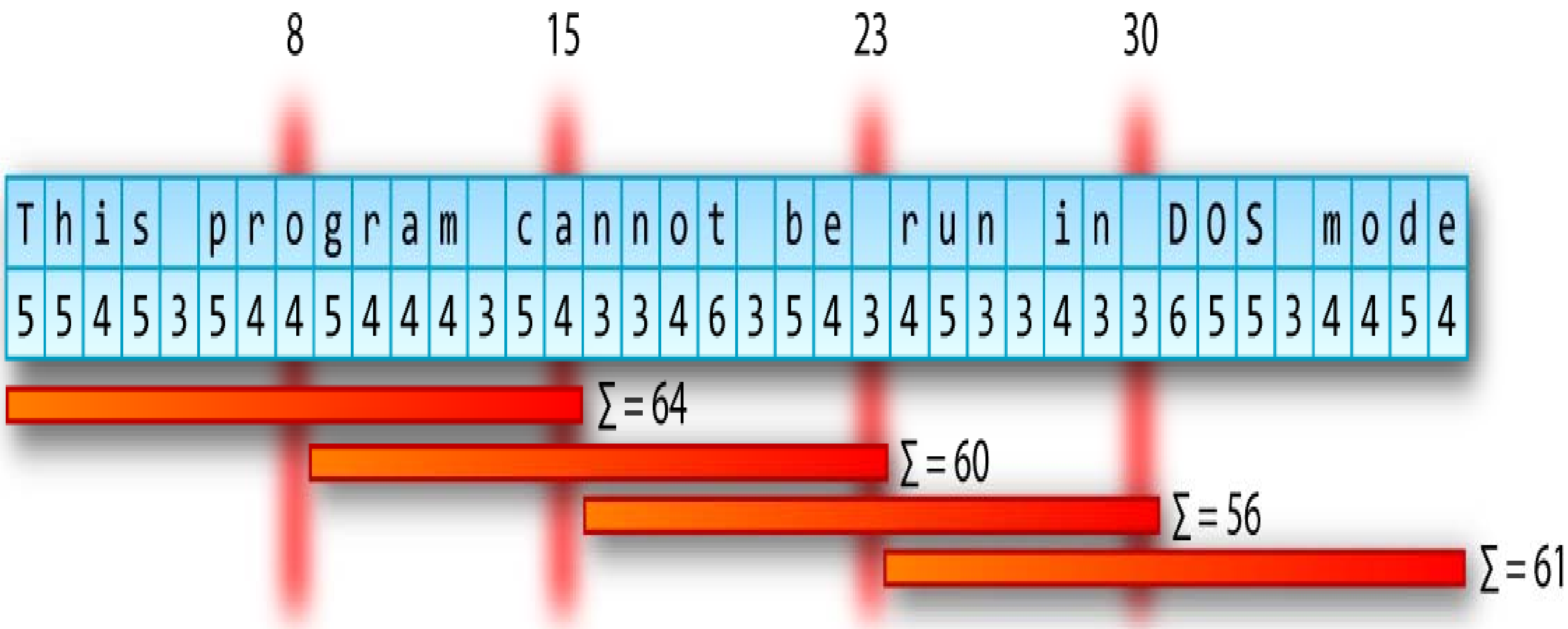


Fixed sample count

- Set a number of sample points, equally spaced throughout the file
- Windows overlap
- Sum the “length encoding” of the bytes within each window
- Advantages:
  - Files of dis-similar length can be easily compared
- Disadvantages:
  - Long files will lose detail because of the very large window
  - Short files will be over detailed because of the very small window

Illustration of fixed sample count algorithm

- Sample count is 4

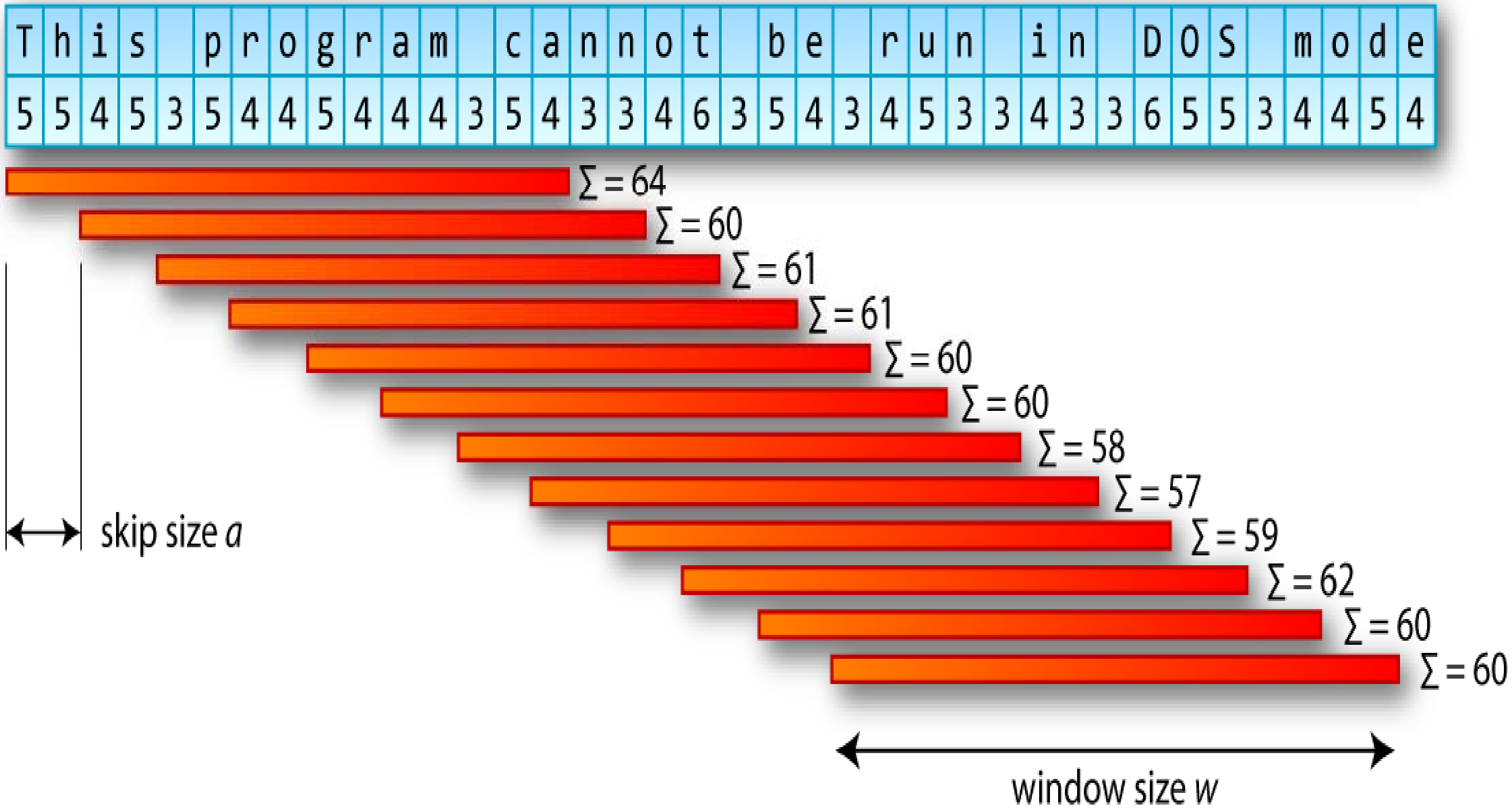


Sliding window

- Pick a fixed window size
- Move the window along the file by α bytes (skip size)
- Sum the “length encoding” of the bytes within each window
- Advantages:
  - Can look for areas of high entropy and fixed size (like crypto keys) in a sea of more structured data
- Disadvantages:
  - Can get a lot of data
  - It Is hard to compare files of different size

Illustration of sliding window algorithm

- Window size is 15 and skip size is 2



Pruning

- Simplify comparison between input samples of different length (the sliding window algorithm)
- Retain data of low randomness
- Eliminate data of high randomness

Proposed pruning heuristics

- *First*
  - Retrieves the first *N* values from the output
- *Smallest*
  - Sorts the output
  - Gets the first *N* smallest values
- *Ordered smallest*
  - Sorts the output
  - Gets the first *N* smallest values
  - Lists them in the order of its original position
- *Trunk*
  - Removes the middle part of the output
  - keeps *N*/2 values from the beginning and *N*/2 values from the end

Sample of pruning

- For an input {1, 3, 4, 9, 8, 10, 6, 7, 2, 5}
- If N = 6
  - First: {1, 3, 4, 9, 8, 10}
  - Smallest: {1, 2, 3, 4, 5, 6}
  - Ordered smallest: {1, 3, 4, 6, 2, 5}
  - Trunk: {1, 3, 4, 7, 2, 5}

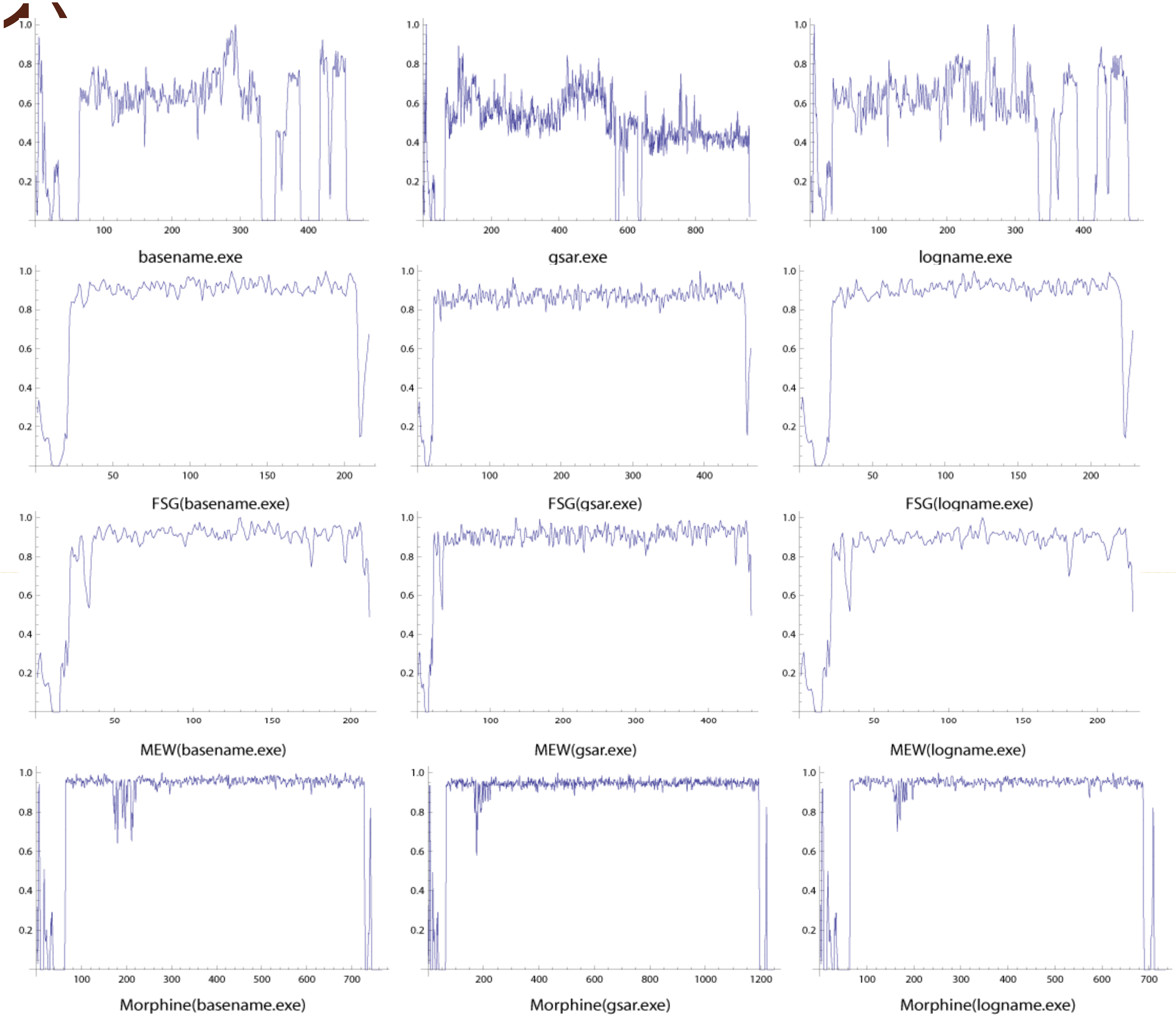
Randomness scanning (1)

- Packers
  - FSG 2.0
  - Mew 11
  - Morphine 2.7
  - RLPack 1.19
  - Upack 0.399
  - UPX 2.03w
- Data
  - UnxUtils
    - 116 files
    - File size 3KB – 191KB

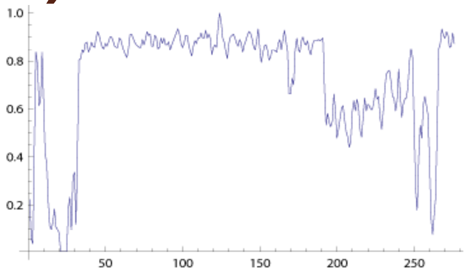
Randomness scanning (2)

- 6 x 116 packed files
- Fixed sample count
  - Sample count is 512
  - Balance the effect of over-represent detail of the small file and under-represent detail of the big file
- Sliding window
  - Window size is 32 bytes (256 bits)
  - Skip size is 16

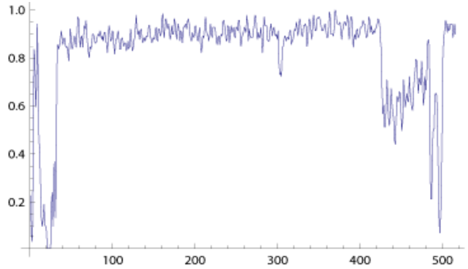
Randomness scanning results (1)



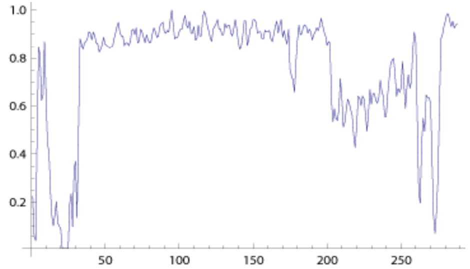
Randomness scanning results (2)



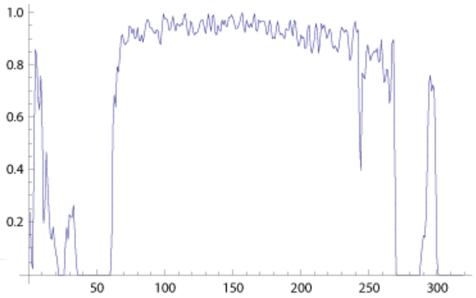
RLPack(basename.exe)



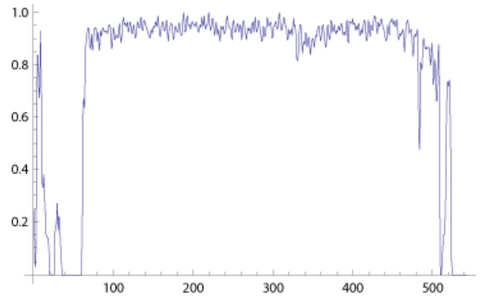
RLPack(gsar.exe)



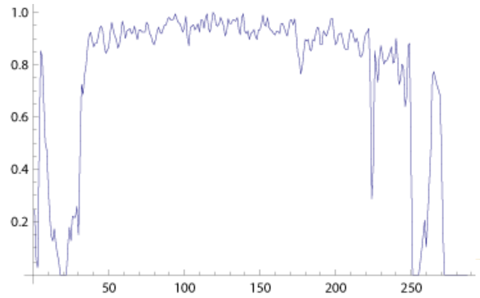
RLPack(logname.exe)



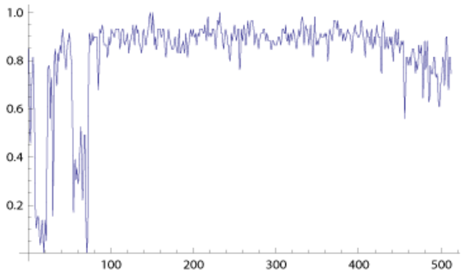
upx(basename.exe)



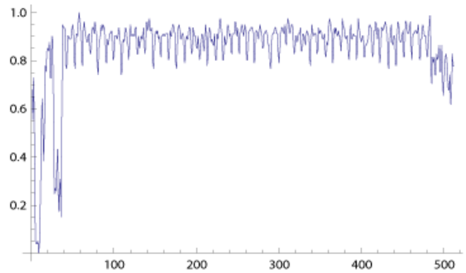
upx(gsar.exe)



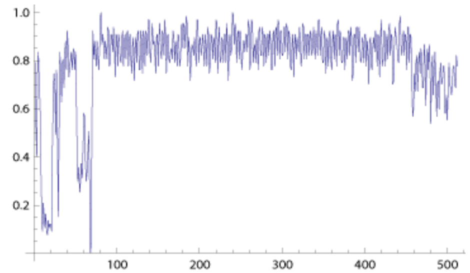
upx(logname.exe)



UPack(basename.exe)



UPack(gsar.exe)



UPack(logname.exe)

Randomness test applications

- Packer classification system
- Unpacking animation

Packer classification

- Characteristic extraction
  - A file is represented as an n-dimensional vector

    $e = \{r_{e1}, r_{e2}, \dots, r_{en}\}$
  - A packer's signature is also represented as an n-dimensional vector

    $S = \{r_{s1}, r_{s2}, \dots, r_{sn}\}$ where $r_{si} = \sum_{j=0}^{N} r_{eji}/N$ for a set of packed files $E = \{e_1, e_2, \dots, e_N\}$
- Identification
  - Distance measures (packed file & packer's signature)
    - Sum-of-squares distance (SSD)
    - Cosine distance

SSD vs. Cosine distance

- SSD

$$SSD\,(e,S) = \sqrt{(r_{e1} - r_{s1})^2 + (r_{e2} - r_{s2})^2 + \ \ldots + (r_{en} - r_{sn})^2}$$

- Cosine distance

$$Cosine\,(e,S) = \cos^{-1}\frac{e \cdot S}{|e||S|} = \ \cos^{-1}\frac{\sum_i (r_{ei} \cdot r_{si})}{\sqrt{\sum_i {r_{ei}}^2}\sqrt{\sum_i \,{r_{si}}^2}}$$

Also fast!

Packer classification (PC)

- *Full*
  - Whole set of randomness outputs from the previous “randomness scanning” experiment
- *Big*
  - Output from files that are over 20 KB
- *Medium*
  - Output from files in the range of 10-19 KB
- *Small*
  - Output from files less than 10 KB

PC results (1)

- Evaluation of two distance measures and four pruning strategies using the sliding window algorithm and full data set, *N=100*

Pruning method	Distance measure	Total files	Positive	False	Identification rate
First	SSD	691	561	130	81.19%
	Cosine	691	572	119	82.78%
Smallest	SSD	691	578	113	83.65%
	Cosine	690	639	51	92.61%
Ordered smallest	SSD	691	624	67	90.63%
	Cosine	690	676	14	97.97%
Trunk	SSD	693	662	31	95.53%
	Cosine	693	686	7	98.99%

PC results (2)

Algorithm (Pruning method)	Data set type	Total files	Positiv e	False	Identification rate
Fixed sample count	Full	696	396	300	56.90%
	Big	265	195	70	73.58%
	Medium	236	210	26	88.98%
	Small	185	162	23	87.57%
Sliding window (Trunk)	Full	693	686	7	98.99%
	Big	263	261	2	99.24%
	Medium	236	234	2	99.15%
	Small	185	184	1	99.46%

PC on malware samples

- Five samples for each packer
- Randomly picked from CA's zoo
- Sliding window algorithm
  - Cosine distance measure
  - Trunk pruning heuristic

Data set type	Total files	Positive	False	Identification rate
Full	30	24	6	80.00%
Big	24	18	6	75.00%
Small	6	6	0	100.00%

Unpacking animation

- Monitor the memory change
  - Place breakpoints on main loops
    - Use “Hump and dump” to identify main loop
  - Dump memory
    - IDA plugin to allow multi-dumping
  - Perform the detailed preserving randomness analysis on the dump
- Illustrate how a packer is working
- Demo

Further work

- Improve the algorithm performance by tuning parameters
- Develop new effective pruning strategies
- Evaluate various distance measures
- Build a large training data set

Conclusion

- Fast
- Preserves local detail
- Useful
  - Packer classification
  - Investigative tool

Thanks

- Any questions?