On the Intractability of Designing an Efficient Entropy Brute Forcer

Mohamad F. Mokbel

{mfmokbel@live.com} - http://www.mfmokbel.com

June 09, 2012

Abstract

The problem of finding a contiguous set of bytes that have a given entropy value in a binary is a multifaceted undertaking. This is due to unpredictable and hard to 'patternize' formations of various bytes distribution in a given search space. This is an inherit limitation in the way entropy works, since without prior knowledge about the distribution of bytes, the expectancy of determining any subsequent byte is not guaranteed to converge to any imposed predictor. Thus, scaling (decrease or increase) and extrapolating on multiple spaces does not meet any expected entropy value. In this regard, designing a heuristic approach to tackle this problem is not possible with high accuracy as we will show in this paper. This is important in case where finding arrays of bytes in a binary that have given entropy value helps in determining if its encrypted or packed as discussed in [1]. Moreover, it provides deep inspection capabilities to identify where in the data, blocks of bytes have given entropy value for finding cryptographic elements such as, keys, certificates and others.

We have implemented an entropy brute forcer with five modes of operation, each supporting different accuracy level. More importantly, we provide an optimized brute forcer algorithm which exploits the entropy equation size limit in order to reduce the time complexity when searching for a specific value. In addition, the computational complexity analysis is provided for major operation modes to illustrate the differences in the performance from a mathematical point of view. Furthermore, we used Entyzer (Advanced entropy Analyzer) [2] as the main framework for extending it with the brute force algorithm discussed in this paper. Various Windows binaries have been used in testing the brute forcer.

The results confirm the hypothesis that writing an entropy brute forcer is complex and computationally expensive.

Keywords: Brute Force; Entropy; Entyzer

I. Introduction

"As for me, all I know is that I know nothing"
- Socrates

In [3], Mokbel and Cambly proposed an unobtrusive entropy based compiler optimization comparator using Shannon entropy as a means to examine the statistical variations at 1-gram byte distribution and quantify the information contained in a binary. In this paper, we build on the extensive analysis presented in [3], especially regarding entropy in depth exploration from theoretical and practical point of views. Thus, interested reader is advised to refer to [3] for more information about entropy.

This paper aims to examine the problem of determining where in a given search space, the sought entropy value is located using flexible parameterized inputs.

For reference, Shannon entropy equation is:

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

The most relevant work to the research presented in this paper is the Entropy IDA Plugin tool developed by Zbitskiy [4]. The tool calculates the entropy for 32-bit PE, ELF and any binary files. In addition, it has the capability to search for a given entropy value based on a chunk and step sizes. However, the tool lacks the powerful input parameterization and the different algorithmic implementations presented in this paper. Thus, the accuracy in locating entropies as well as the performance impact it reveal shows noticeable differences between both works.

II. Algorithmic Analysis

"There are no facts, only interpretations"
- Nietzsche

In this section, we present two algorithms demonstrating various modes of operation. In Algorithm 1, three modes of operation are supported. The algorithm receives three parameterized inputs: ***N**egative **P**ermissible **R**ange* (NPR), ***M**ain **V**alue* (MV) and ***P**ositive **P**ermissible **R**ange* (PPR). The input conditions are stated in $\boxed{A}$ and the conditional operations on those inputs for satisfying given entropy value are located at $(L.13)$. These values represent the range of the entropy sought target. The output is the address(es) $(L.02)$ where the sought entropy(ies) Ң is/are located in the data. The reason behind such a flexible parameterization is to allow for greater possibilities when searching for entropy.

For mode **1** $(\text{ɱ} = 1)$, from lines $[10 - 15]$, the entropy brute forcer functions by enumerating through all the information Й̄**,** such that when a given entropy value is found, the algorithm starts again from the end address $b]$ of the last found entropy and so on until all the information is consumed. However, this doesn't constitute a true brute forcer.

For mode **2** $(\text{ɱ} = 2)$, from lines $[10 - 17]$, the entropy brute forcer starts by first exercising mode 1, and if no entropy was found, it keeps advancing the starting offset in the search space by one until it hits the first sought entropy value (if any), and then switches back to mode **1**. Thus, mode **2** is more computationally expensive than mode **1**. This mode registers its worst case and functions as a true brute forcer in case not a single entropy instance was found.

For mode **3** $(\text{ɱ} = 3)$, from lines $[08 - 09]$, this is the divide and conquer mode. However, it doesn't have a functional implementation on its own. In addition to the previously discussed input parameters, this mode takes only an input ɖ which specifies the number of blocks required to divide the information Й̄ space. Modes **1**, **2, 4** and **5** (which will be discussed later in the paper) are all applicable for this mode. This mode represents a fine grained attack on the information space in an attempt to find entropy at the earliest point.

Note that because of the introduced range flexibility when seeking an entropy value, the algorithm is influenced by the earliest entropy match (based on the order of evaluation). Hence, any subsequent matches are subjected to the location of the prior match in the search space. This is due to the function of the indexing as mentioned above.

$$Alg.1 = \begin{cases} Exit, & \text{Ң}(\bar{\text{Й}}.Size) = 0.0 \quad \{\mathbf{1}\} \\ \forall (x \in Size_2^{256})\ \exists y \in \text{Ң}, & \text{Ң}(\bar{\text{Й}}.Size) = 8.0 \quad \{\mathbf{2}\} \\ |\text{Ң} - MV|_{\cong} \leq \varepsilon, & (NPR \land PPR) = 0\ \land\ (MV \in [0.0, 8.0] \in \mathbb{Q}) \quad \{\mathbf{3}\} \end{cases} \qquad \boxed{B}$$

Moreover, function $\boxed{B}$ adds more constraints to the algorithm. In case the entropy of the total search space is zero $\boxed{\text{B}.1}$, then the algorithm bails out immediately without any further computation. In addition, in case the entropy of the total search space is the maximum entropy value $\boxed{B.2}$, that is 8.0, then a complete holistic heterogeneous search space is detected, which enables finding every possible entropy value in the range between one and eight with respect to every possible length value between 2 and 256. This is illustrated in Table 1 which shows the hexadecimal distribution of a complete 1-gram byte. Furthermore, since

Algorithm 1. ENTROPY BRUTE FORCER

01. **Input**: $Information\ '\text{Й̄}',\ Operation\ Mode\ '\text{ɱ}', NPR, MV, PPR, \text{ɖ}$

$With\ the\ following\ definitions$: $\text{ɱ} = \{1,2,3\}$ // $Modes\ of\ operation$

$$\boxed{A}\ \begin{pmatrix} (NPR \wedge PPR) \in [0.0, 1.0] \\ MV\ \in [0.0, 8.0] \end{pmatrix}\ Such\ that \begin{cases} (MV + PPR) \ngtr 8.0 \\ (MV - NPR) \nless 0.0 \\ [(MV - NPR) \wedge (MV + PPR)] \neq 0.0 \end{cases}$$

02. **Output**: $Entropy(ies)\ '\text{Ḩ}'\ found\ at\ address(es)\ [a, b]$

03. **_begin**

04. $\text{Й̄}.SetStartOffset(0)$

05. $\_ShiftStartingOffset$: /* $Goto\ label\ for\ operation\ mode\ 2$ */

06. $\text{Й̄}.SetStartOffset(\text{Й̄}.StartOffset)$

07. $Idx = \text{Й̄}.StartOffset$ /* $Initialize\ Index\ to\ a\ starting\ offset$ */

08. **if** $(\text{ɱ} == \mathbf{3})$ **then** $\left\{ \text{Ɠ}_1^{\text{ɖ}-1} = \left\lfloor \frac{\text{Й̄}.Size}{\text{ɖ}} \right\rfloor\ and\ \text{Ɠ}_{\text{ɖ}} = \text{Й̄}.Size - \sum_1^{\text{ɖ}-1} \text{Ɠ} \right\}$ $such\ that\ 2 \leq \text{ɖ} \leq \text{Й̄}.Size$

09. **else** $\{\ \text{Ɠ}_0 = \text{Й̄}.Size\ \}$

10. **for** $(Idx; Idx \leq \text{Ɠ}_r; Idx++)$ $such\ that\ r = \begin{cases} 0, & \text{ɱ} == \langle \mathbf{1|2|4|5} \rangle \\ [1, \text{ɖ}], & \text{ɱ} == \mathbf{3} \end{cases}$

11. $\text{Й̄}.SetEndOffset(Idx)$

12. $\text{Ḩ} = CalculateEntropy(\text{Й̄})$

13. **if** $\Big[\big((\text{Ḩ} \geq MV) \wedge (\text{Ḩ} \leq (MV + PPR)) \big) \vee \big((\text{Ḩ} \geq (MV - NPR)) \wedge (\text{Ḩ} \leq MV) \big) \Big]$

14. **then** $\left\{ \begin{matrix} Entropy\ is\ found\ at\ address\ \ [\text{Й̄}.StartOffset, Idx]; \\ IsEntropyFound = True \end{matrix} \right\}$

/* $If\ Entropy\ found\ and\ still\ more\ data\ to\ parse, then\ advance\ Idx\ by\ 1\ and\ continue$ */

15. **if** $(Idx < \text{Ɠ}_r)$ **then** $\{\ \text{Й̄}.SetStartOffset(Idx + 1)\ \}$

/* $This\ is\ for\ Operation\ Mode\ 2. If\ no\ Entropy\ was\ found\ in\ mode\ 1, then\ advance\ starting\ offset$ */

16 **if** $\begin{bmatrix} (IsEntropyFound == False) \wedge \big((Idx - 1) == \text{Ɠ}_r \big) \wedge \\ (\text{Ḩ}.StartOffset < \text{Ɠ}_r)\ \wedge (\text{ɱ} == \mathbf{2}) \end{bmatrix}$

17. **then** $\left\{ \begin{matrix} \text{Й̄}.StartOffset++; \\ \textbf{Goto}\ \_ShiftStartingOffset; (L.05) \end{matrix} \right\}$

18. **_end**

every single byte in the distribution is different which satisfies the maximum entropy length requirement, the result is a perfect search space which enables the computation of all possible entropy values. This enables us to generate a table containing all possible entropy values which will be fed selectively to Algorithm 2.

For $\boxed{B.3}$, if the MV value satisfies the condition presented, then it becomes difficult to compare two floating point values of different accuracies due to compiler and architectural limitations. Thus, if the absolute difference between the entered MV and the calculated entropy value Ḩ is less than a given small epsilon value, then the comparison is considered almost equal. However, this condition is not honored in the implementation. It is advised that a value for PPR would be chosen accordingly instead, as this allows greater flexibility when searching for Ḩ.

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Table 1. 1-Gram Byte Hexadecimal Distribution

Figure 1 shows the characterization of entropy with respect to a given number of bytes in a complete holistic heterogeneous search space based on the data in Table 1. As shown, the distribution satisfies the logarithmic equation shown in the *shadow* area. Moreover, the horizontal line, to the left of the graph, shows the minimum number of bytes (in ***bold***) that needs to be satisfied in order to get the equivalent entropy value (the complete range of values is shown in Appendix A).

Thus, the question becomes, what is the minimum number of bytes required that satisfy a given entropy value? In Algorithm 2, we address this question (ɱ $= 4$ *and* ɱ $= 5$), which exploits the characterization presented in Figure 1 in order to reduce the number of steps required when seeking an entropy value. Line $[05]$ shows the minimum number of bytes required for a given entropy value. In the case presented at Lines $[06-10]$ (ɱ $= 5$), the value of the step size is relative to the MV or $\lfloor MV - NPR \rfloor$ sought with respect to the $EntSizeLimit[\ \lfloor MV -$

$NPR\rfloor$] array's values at Line [05]. In another words, it advances the index of every round by $EntSizeLimit[\ \lfloor MV + PPR \rfloor\ ]$ and changes the starting offset only for the first block that hits $EntSizeLimit[\ \lceil MV + PPR \rceil\ ]$.

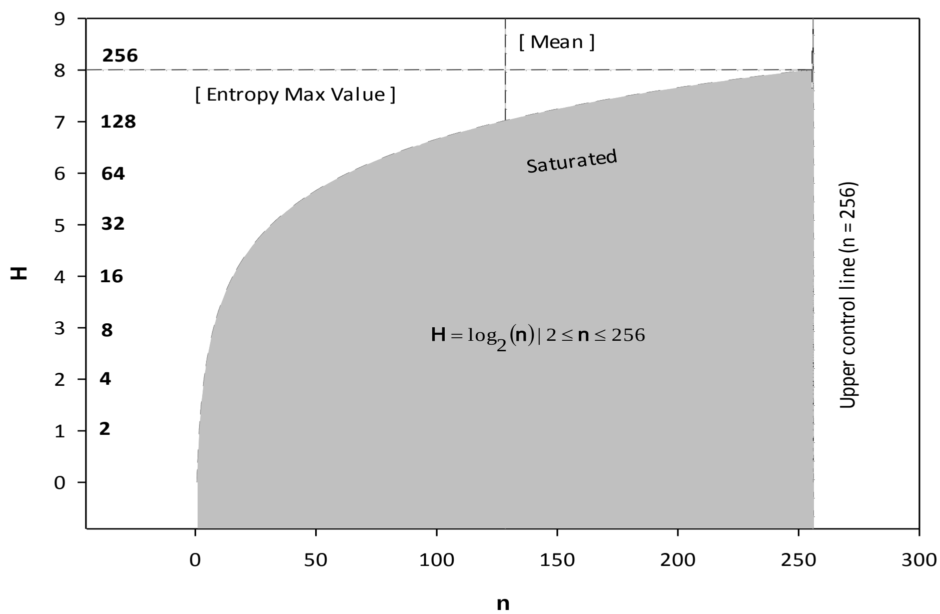

Figure 1. Characterization of Entropy (H) vs Min. Size (n)

Whereas, in the case presented at Lines $[11-16]$ ($\mathfrak{m} = 4$), the step size is established by taking the *floor* of $(MV - NPR)$, and the block size limit is determined by taking the *ceiling* of $(MV + PPR)$, all with respect to the *EntSizeLimit* array values. However, this is not intended to be a perfect solution; it is only meant to present a different attack vector (sacrificing accuracy) in order to reduce the processing time required when searching for an entropy value. On the other hand, the characterization array shown could be made more fine-grained by including other sizes as shown in Figure 1 and Appendix A.

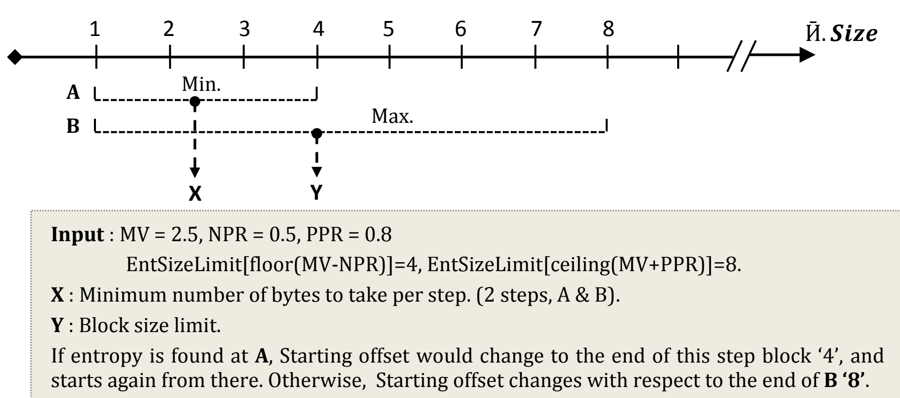

Figure 2. Illustration of Mode 4 ($\mathfrak{m} = 4$) operation

Figure 2 shows an example of how Algorithm 2 works in the case presented at Lines $[11 - 16]$.

Algorithm 2. OPTIMIZED ENTROPY BRUTE FORCER

01. **Input**: $Information$ $'\bar{Й}'$, $Operation\ Mode$ $'ɱ'$, NPR, MV, PPR

$With\ the\ following\ definitions$: $ɱ = 4, 5$

$$\begin{pmatrix} (NPR \wedge PPR) \in [0.0, 1.0] \\ MV \in [0.0, 8.0] \end{pmatrix} Such\ that \begin{cases} (MV + PPR) \ngtr 8.0 \\ (MV - NPR) \nless 0.0 \\ [(MV - NPR) \wedge (MV + PPR)] \neq 0.0 \end{cases}$$

02. **Output**: $Entropy(ies)$ $'Ң'$ $found\ at\ address(es)$ $[a, b]$

03. **_begin**

04. $\bar{Й}.SetStartOffset(0)$

05. $EntSizeLimit[8] = \{2, 4, 8, 16, 32, 64, 128, 256\}$

06. **if** $(MV == \boldsymbol{X})$ **|** $\boldsymbol{X} \in \mathbb{N} \vee \in \mathbb{Q}$ **|**$0.0 \leq \boldsymbol{X} \leq 8, NPR = 0 \vee \neq 0, PPR = 0 \vee \neq 0$

07. **if** $(Idx == EntSizeLimit[\lceil MV + PPR \rceil])$

08. $\bar{Й}.SetStartOffset(Idx)$ $//\ Or\ if\ entropy\ found$

09. $\bar{Й}.SetEndOffset(Idx)$

10. $Idx \mathrel{+}= EntSizeLimit[\lfloor MV - NPR \rfloor]$

11. **While** $(Idx \leq Ɠ_r)$

12. **if** $((Idx\ \%\ EntSizeLimit[\lceil MV + PPR \rceil] == 0) \wedge NPR \neq 0 \wedge PPR \neq 0)$

13. $\bar{Й}.SetStartOffset(Idx)$ $//\ Or\ if\ entropy\ found$

14. $\bar{Й}.SetEndOffset(Idx)$

15. $//\ What\ follows\ is\ the\ same\ as\ in\ Algorithm\ 1.$

16. $Idx \mathrel{+}= EntSizeLimit[\lfloor MV - NPR \rfloor]$

17. **_end**

For example, in the scenario presented in Figure 2, the number of steps (worst-case) taken by $(ɱ = 4)$ (Lines $[11 - 16]$) is 2 and 12 for $(ɱ = 5)$ (Lines $[06 - 10]$. Whereas, in the worst-case scenario for $(ɱ = 1)$, the number of steps would be 7 and 28 for $(ɱ = 2)$. Thus, $(ɱ = 4)$ (Lines $[11 - 16]$) shows a better performance as compared to the other modes. However, the level of accuracy and computational time vary among all the modes as we will show in the experimentation section. Moreover, other possibilities exist to combine these modes together, for example, mode 1 can be integrated with mode 4.

Note that the order of the modes is different from the ones in the tool. Following is the mapping between the modes presented in the paper ‘P’ and the ones in the final release of the tool ‘T’: ‘P’ (m = 1) -> T (m = 1), ‘P’ (m = 2) -> T (m = 2), ‘P’ (m = 3) -> T (is a separate option invoked via the –b parameter for specifying the block size), ‘P’ (m = 4) -> T (m = 3) and ‘P’ (m = 5) -> T (m = 4).

It is important to note that none of the modes in both algorithms allow overlapping among the found entropies search space. The offsets (starting and ending addresses) of the bytes that constitute the range of entropies found are always in increasing order. And this is a design decision.

III. Computational Complexity Analysis

In this section, we present the computational complexity for the compute intensive modes introduced in section II. The computational complexity for calculating the entropy of a given search space is shown in $\boldsymbol{E}[\mathbf{1}]$ (note that B_h represents the size of one byte).

$$\bar{\text{Й}}.Size + (B_h)^2 \mid B_h = 16 \qquad \boldsymbol{E}[\mathbf{1}]$$

For mode 1 (worst-case):

$$\boldsymbol{E}[\mathbf{1}] \times (\bar{\text{Й}}.Size - 1) \qquad \boldsymbol{E}[\mathbf{2}]$$

For mode 2 (worst-case):

$$\boldsymbol{E}[\mathbf{2}] + \left[\left((\bar{\text{Й}}.Size - ShiftingOffset) + (B_h)^2\right) \times \left((\bar{\text{Й}}.Size - 1) - ShiftingOffset\right)\right]^2 \qquad \boldsymbol{E}[\mathbf{3}]$$

$$Such\ that\ 1 \leq ShiftingOffset \leq \bar{\text{Й}}.Size$$

“I want to know God’s thoughts; the rest are details”
- Einstein

IV. Experimental Evaluation

In order to provide a comparative analysis on the various modes of operation, 6 binaries (identified as A, B, C, D, E and F) from Windows 7 Professional (x64-bit) with SP 1 were analyzed on an Intel(R) Core(TM)2 Duo CPU P7350 @ 2.00GHz, L2 cache 3072 KBytes, 12-way set associative, 64-byte line size and 4 GBytes of memory (DD3). Moreover, the experiments were conducted using the 32-bit version of Entyzer. For every binary, 8 different entropy values ($MV, NPR\ and\ PPR$) covering almost all the major possible step values are exercised, recording the time it took to find every value as well as the number of entropy(ies) found. Note that the ‘*Total*’ row values represent the entropy of the whole file. The order of the sought entropy values parameters is (NPR, MV and PPR).

Across all the modes in tables 2, 4, 6 and 8, mode 4 registers the least amount of time it took to search for entropy. However, the degree of accuracy varies dramatically across all the modes. Though the computational time between modes 4 and 5 is close (except F4), the number of entropy(ies) found shows great differences. Thus, a fine-grained approach is almost always better for mining for entropies at the expense of time complexity. The reason why F4 took significantly more time in mode 5 compared to mode 4 is likely to be related to the difference in the number of entropies found. In mode 5, only 12% of the total number of entropies found was detected compared to those in mode 4 (88%).

Therefore, the change in the starting offset was less frequent (which would have reduced the search space at the earliest match) in mode 5.

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	0	0	0	0	1	4
2	0.1|2.0|0.6	0	0	0	0	0	2
3	0.4|3.0|0.6	0	0	0	0	7	3
4	0.4|4.0|0.3	0	0	0	0	94	15
5	0.7|5.0|0.8	0	0	0	0	141	42
6	0.2|6.0|0.1	1	4	5	5	674	623
7	0.2|7.0|0.2	5	20	31	21	674	4950
8	0.8|8.0|0.0	5	19	31	38	674	4950

Table 2. Mode 1 – Time taken to search for entropy

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	7524	15596	21003	44845	111813	264521
2	0.1|2.0|0.6	3159	6413	8766	20831	37818	113914
3	0.4|3.0|0.6	1572	3178	4447	11055	17498	57070
4	0.4|4.0|0.3	597	1211	1776	5346	6023	22972
5	0.7|5.0|0.8	259	526	800	3092	2308	10298
6	0.2|6.0|0.1	9	19	27	844	0	740
7	0.2|7.0|0.2	0	0	0	261	0	0
8	0.8|8.0|0.0	0	0	0	107	0	0

Table 3. Mode 1 – Number of entropy(ies) found

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	0	0	0	0	1	3
2	0.1|2.0|0.6	0	0	0	0	0	1
3	0.4|3.0|0.6	0	0	0	0	0	0
4	0.4|4.0|0.3	0	0	0	0	0	0
5	0.7|5.0|0.8	0	0	0	0	0	0
6	0.2|6.0|0.1	0	0	0	0	0	0
7	0.2|7.0|0.2	0	0	0	0	0	0
8	0.8|8.0|0.0	0	0	0	0	5	39

Table 4. Mode 4 – Time taken to search for entropy

"A small error in the former will produce an enormous error in the latter"
- Poincare

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	4110	8445	11384	23066	59660	137978
2	0.1|2.0|0.6	1726	3464	4630	10781	16109	59222
3	0.4|3.0|0.6	832	1638	2236	5187	7593	28413
4	0.4|4.0|0.3	313	657	908	2358	3083	10991
5	0.7|5.0|0.8	120	268	405	1133	1382	4712
6	0.2|6.0|0.1	0	0	0	160	0	143
7	0.2|7.0|0.2	0	0	0	222	0	152
8	0.8|8.0|0.0	0	0	0	81	0	0

Table 5. Mode 4 – Number of entropy(ies) found

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	0	0	0	0	1	4
2	0.1|2.0|0.6	0	0	0	0	0	2
3	0.4|3.0|0.6	0	0	0	0	1	1
4	0.4|4.0|0.3	0	0	1	15	47	497
5	0.7|5.0|0.8	0	0	0	0	8	2
6	0.2|6.0|0.1	0	0	0	4	21	23
7	0.2|7.0|0.2	0	0	0	0	10	77
8	0.8|8.0|0.0	0	0	0	0	5	38

Table 6. Mode 5 – Time taken to search for entropy

		A	B	C	D	E	F
	Size/Bytes	[20480]	[40448]	[51200]	[102400]	[272896]	[651264]
	Total	5.64373	5.88127	5.94664	7.33376	5.34517	6.25283
1	0.3|1.0|0.2	7523	15595	21002	44844	111812	264520
2	0.1|2.0|0.6	2948	6014	8122	20514	36394	108571
3	0.4|3.0|0.6	1380	2822	3943	9845	15650	50539
4	0.4|4.0|0.3	212	522	484	522	332	1330
5	0.7|5.0|0.8	220	445	663	2153	1915	8240
6	0.2|6.0|0.1	9	19	27	1	0	276
7	0.2|7.0|0.2	0	0	0	228	0	0
8	0.8|8.0|0.0	0	0	0	81	0	0

Table 7. Mode 5 – Number of entropy(ies) found

	A7 - (0.2|7.0|0.2)		
Blocks	**M1**	**M2**	**M3**
0 - 5120	0	664	0
5120 - 10240	0	675	0
10240 - 15360	0	616	0
15360 - 20480	0	392	0

Table 8. Modes 1, 2 and 3 — Time taken to search for entropy

For mode 2, we took only the sample A7 with one entropy entry, and divided the search space according to mode 3. The results demonstrate the performance impact (time) mode 2 has on brute forcing for entropy. Since no entropy was found (not shown) for modes 1, 2 and 3, all the modes register their worst-case scenarios. The fact that mode 2 didn't reveal any entropy in the search space; it is by definition that none of the other modes would reveal anything.

Bibliography

[1] Lyda, R. and Hamrock, J. (2007) Using Entropy Analysis to Find Encrypted and Packed Malware. IEEE Security & Privacy, Vol. 05, Issue 02, pp. 40-45.

[2] Mokbel, M. F. (2011) Entyzer+ (Advanced Entropy Analyzer). Http://www.mfmokbel.com

[3] Mokbel, M. F. and Cambly, C. D. (2010) An Unobtrusive Entropy Based Compiler Optimization Comparator. In Technology Showcase at the 20th Annual International Conference on Computer Science and Software Engineering (CASCON 2010). Available at Http://www.mfmokbel.com.

[4] Zbitskiy, P. (2010) IDA Entropy Plugin. Http://smokedchicken.org

APPENDIX A

Size	H
1	0
2	1
3	1.58496
4	2
5	2.32193
6	2.58496
7	2.80735
8	3
9	3.16993
10	3.32193
11	3.45943
12	3.58496
13	3.70044
14	3.80735
15	3.90689
16	4
17	4.08746
18	4.16993
19	4.24793
20	4.32193
21	4.39232
22	4.45943
23	4.52356
24	4.58496
25	4.64386
26	4.70044
27	4.75489
28	4.80735
29	4.85798
30	4.90689
31	4.9542
32	5
33	5.04439
34	5.08746
35	5.12928
36	5.16993
37	5.20945
38	5.24793
39	5.2854
40	5.32193
41	5.35755
42	5.39232
43	5.42626
44	5.45943
45	5.49185
46	5.52356
47	5.55459
48	5.58496
49	5.61471
50	5.64386
51	5.67243
52	5.70044
53	5.72792
54	5.75489
55	5.78136
56	5.80735
57	5.83289
58	5.85798
59	5.88264
60	5.90689
61	5.93074
62	5.9542
63	5.97728
64	6
65	6.02237
66	6.04439
67	6.06609
68	6.08746
69	6.10852
70	6.12928
71	6.14975
72	6.16993
73	6.18982
74	6.20945
75	6.22882
76	6.24793
77	6.26679
78	6.2854
79	6.30378
80	6.32193
81	6.33985
82	6.35755
83	6.37504
84	6.39232
85	6.40939
86	6.42626
87	6.44294
88	6.45943
89	6.47573
90	6.49185
91	6.50779
92	6.52356
93	6.52356
94	6.55459
95	6.56986
96	6.58496
97	6.59991
98	6.61471
99	6.62936
100	6.64386
101	6.65821
102	6.67243
103	6.6865
104	6.70044
105	6.71425
106	6.72792
107	6.74147
108	6.75489
109	6.76818
110	6.78136
111	6.79442
112	6.80735
113	6.82018
114	6.83289
115	6.84549
116	6.85798
117	6.87036
118	6.88264
119	6.89482
120	6.90689
121	6.91886
122	6.93074
123	6.94251
124	6.9542
125	6.96578
126	6.97728
127	6.98868
128	7
129	7.01123
130	7.02237
131	7.03342
132	7.04439
133	7.05528
134	7.06609
135	7.07682
136	7.08746
137	7.09803
138	7.10852
139	7.11894
140	7.12928
141	7.13955
142	7.14975
143	7.15987
144	7.16993
145	7.17991
146	7.18982
147	7.19967
148	7.20945
149	7.21917
150	7.22882
151	7.2384
152	7.24793
153	7.25739
154	7.26679
155	7.27612
156	7.2854
157	7.29462
158	7.30378
159	7.31288
160	7.32193
161	7.33092
162	7.33985
163	7.34873
164	7.35755
165	7.36632
166	7.37504
167	7.3837
168	7.39232
169	7.40088
170	7.40939
171	7.41785
172	7.42626
173	7.43463
174	7.44294
175	7.45121
176	7.45943
177	7.46761
178	7.47573
179	7.48382
180	7.49185
181	7.49985
182	7.50779
183	7.5157
184	7.52356
185	7.53138
186	7.53916
187	7.54689
188	7.55459
189	7.56224
190	7.56986
191	7.57743
192	7.58496
193	7.59246
194	7.59991
195	7.60733
196	7.61471
197	7.62205
198	7.62936
199	7.63662
200	7.64386
201	7.65105
202	7.65821
203	7.66534
204	7.67243
205	7.67948
206	7.6865
207	7.69349
208	7.70044
209	7.70736
210	7.71425
211	7.7211
212	7.72792
213	7.73471
214	7.74147
215	7.74819
216	7.75489
217	7.76155
218	7.76818
219	7.77479
220	7.78136
221	7.7879
222	7.79442
223	7.8009
224	7.80735
225	7.81378
226	7.82018
227	7.82655
228	7.83289
229	7.8392
230	7.84549
231	7.85175
232	7.85798
233	7.86419
234	7.87036
235	7.87652
236	7.88264
237	7.88874
238	7.89482
239	7.90087
240	7.90689
241	7.91289
242	7.91886
243	7.92481
244	7.93074
245	7.93664
246	7.94251
247	7.94837
248	7.9542
249	7.96
250	7.96578
251	7.97154
252	7.97728
253	7.98299
254	7.98868
255	7.99435
256	8