On the Efficiency of Mathematics in Intrusion Detection: the NetEntropy Case

Jean Goubault-Larrecq[1] Julien Olivain[1,2]

[1] ENS Cachan goubault@lsv.ens-cachan.fr
[2] INRIA olivain@lsv.ens-cachan.fr

Abstract. NetEntropy is a plugin to the Orchids intrusion detection tool that is originally meant to detect some subtle attacks on implementations of cryptographic protocols such as SSL/TLS. NetEntropy compares the sample entropy of a data stream to a known profile, and flags any significant variation. Our point is to stress the *mathematics* behind NetEntropy: the reason of the rather incredible precision of NetEntropy is to be found in *theorems* due to Paninski and Moddemeijer.
Keywords: sample entropy, Paninski estimator, malware, intrusion detection.

1 Introduction

In 2006, we described a tool, NetEntropy, whose goal is to detect subverted cryptographic network flows [16]. We had initially developed it as a plugin to the Orchids intrusion detection tool [8] to help detect attacks such as [11] or [22] where network traffic is encrypted and therefore cannot be inspected—unless we rely on key escrows [12, Section 13.8.3], but NetEntropy is much easier to install and use.

What NetEntropy does is estimate whether a source of bytes is sufficiently close to a random, uniformly distributed source of bytes. Encrypted data, random keys and nonces, compressed data should qualify as close. Plain text, but also shellcodes, viruses and even polymorphic viruses should not.

To this end, NetEntropy computes the sample entropy H_N of the source, and compares it to an estimator $\hat{H}_N$—a good enough approximation of what the average value of H_N should be if the source were indeed drawn at random, uniformly. We use the *Paninski estimator* (to be introduced later), and show that it gives an extraordinarily precise statistical test of non-randomness.

The purpose of this paper is to stress the mathematics behind this extraordinary precision. Before we had researched the mathematics, the best we could say was that NetEntropy worked well in practice, and this was supported by experiments. How well it fared was beyond us: it was only when we discovered the theorems in the literature that we realized that our entropy estimation technique was in fact precise up to levels we had never even dreamed of.

Outline. We start by reviewing the attack that led us into inventing NetEntropy in Section 2. As we said, NetEntropy evaluates the sample entropy of a flow of bytes. We review the known estimators of sample entropy in Section 3, and justify our choice of the Paninski estimator. In Section 4, we attack the central twin questions of this paper: how do we compute the value $H_N(\mathcal{U})$ the sample entropy should have on a uniformly distributed random N-byte flow? (Section 4.1) and how far away from $H_N(\mathcal{U})$ should the sample entropy be for us to conclude that the flow is not random? (Section 4.2) We shall see that entropy-based detection is extremely precise, already for small values of N (Section 4.3). We conclude in Section 5.

2 The `mod_ssl` Attack

Our primary example will be the `mod_ssl` attack [11]. Similar attacks include the SSH CRC32 attack [22].

The `mod_ssl` attack uses a heap overflow vulnerability during the key exchange (handshake) phase of SSL v2 to execute arbitrary code on the target machine. A normal (simplified) execution of this protocol is tentatively pictured in Figure 1, left. Flow direction is pictured by arrows, from left (client) to right (server) or conversely. The order of messages is from top to bottom. The handshake phase consists of the top six messages. Encrypted traffic then follows. We have given an indication of the relative level of entropy by levels of shading, from light (clear text, low entropy) to dark gray (encrypted traffic, random numbers, high entropy).

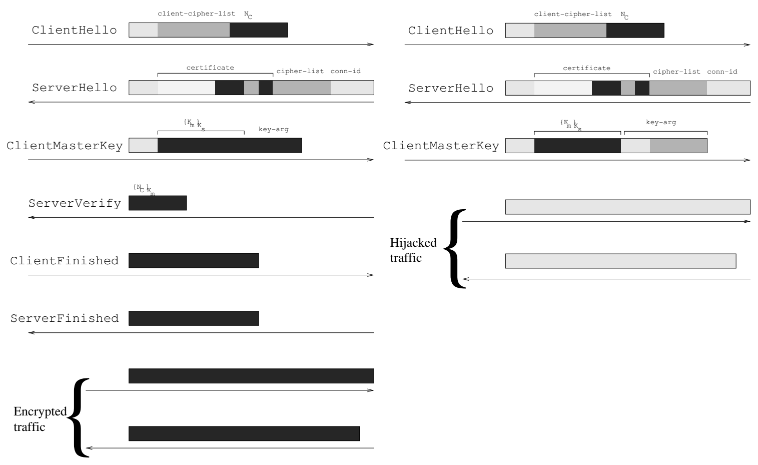

Fig. 1. Normal SSL v2 session (left), hijacked (right)

Encrypted traffic is (using state-of-the-art cryptographic algorithms) indistinguishable from random traffic. The byte entropy of a random sequence of characters is 8 bits per byte, at least in the limit $N \to +\infty$. On the other hand, the byte entropy of a non-encrypted sequence of characters is much lower. According to [4, Section 6.4], the byte entropy of English text is no greater than 2.8, and even 0-order approximations do not exceed 4.26.

Shellcodes that are generally used with the `mod_ssl` attack hijack one session, and reuse the https connection to offer basic terminal facilities to the remote attacker. We detect this by realizing that the byte entropy of the flow on this connection, which should tend to 8, remains low. We can therefore detect the attack when we see the final payload in clear. Since the shellcode itself, whose entropy is low, is sent in lieu of a session key, the entropy is already low in some parts of the key exchange. This can be used to detect the attack even if the shellcode does not communicate over the https channel, which is also common.

While NetEntropy was meant to detect attacks, it can also be used, and has been used, for other purposes, typically network policy enforcement: Skype detection [5], detecting encrypted botnet traffic or encrypted malwares themselves [23, 10, 19]. Entropy checking had already been used for traffic analysis before [7], and our contribution is to show how extraordinarily precise this technique is even in undersampled situations. In particular it is much more precise than distribution identification tools such as PAYL [21]: we only seek to compare a byte stream to *uniform* random byte streams, and this is a setting where a mathematical miracle happens (Moddemeijer's Theorem in the so-called degenerate case, see Section 4.2).

Entropy checking can also be used as a randomness test, e.g., in checking the adequacy of pseudo-random number generators to cryptographic applications [20, 2.12]. In this setting as well as in ours, it has a number of advantages over other approaches. First, it is *fast* and requires *little memory*: we do not need to store the data to test for randomness, only the current distribution (256 integer registers). Second, it is extremely *precise*, as we shall show and demonstrate. Finally, it can be computed *online*, that is, as the input data come in, which makes it ideal in our network-sniffing situation. In cryptographic applications, it would still be fair to complement entropy-checking with other tests: see [9] for a theory of statistical tests suited to the domain.

3 Sample Entropy and Estimators

First, a note on notation. We take log to denote base 2 logarithms. Entropies will be computed using log, and will be measured in *bits*. The notation ln is reserved for natural logarithms.

Let w be a word of length N, over an alphabet $\Sigma = \{0, 1, \ldots, m-1\}$. We may count the number n_i of occurrences of each letter $i \in \Sigma$. The *frequency* f_i of i in w is then n_i/N. The *sample entropy* of w is:

$$\hat{H}_N^{MLE}(w) = -\sum_{i=0}^{m-1} f_i \log f_i$$

(The superscript MLE is for *maximum likelihood estimator*.) The formula is close to the notion of entropy of a random source, but the two should not be confused. Given a probability distribution $p = (p_i)_{i \in \Sigma}$ over Σ, the *entropy* of p is

$$H(p) = -\sum_{i=0}^{m-1} p_i \log p_i$$

In the case where each character is drawn uniformly and independently, $p_i = 1/m$ for every i, and $H(p) = \log m$.

It is hard not to confuse H and $\hat{H}_N^{MLE}$, in particular because a property known as the asymptotic equipartition property (AEP, [4, Chapter 3]) states that, indeed, $\hat{H}_N^{MLE}(w)$ converges in probability to $H(p)$ when the length N of w tends to $+\infty$, as soon as each character of w is drawn independently according to the distribution p. In the case of the uniform distribution, this means that $\hat{H}_N^{MLE}(w)$ tends to $\log m$ ($= 8$, for bytes).

However, we are *not* interested in the limit of $\hat{H}_N^{MLE}(w)$ when N tends to infinity, essentially because we would like to flag a anomalous value of the sample entropy as soon as possible, and also because the actual messages we monitor may be short: the encrypted payload is only a few dozen bytes long in short-lived SSH connections, for example. In practice, we only compute the sample entropy for words w of length $N \leq N_{\max}$. $N_{\max} = 65\ 536$ is sufficient in the intended intrusion detection applications.

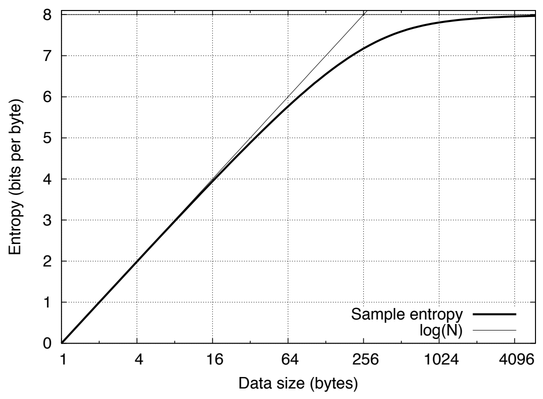

Fig. 2. Average sample entropy $\hat{H}_N^{MLE}(w)$ of words w of size N

Let us plot the average $\hat{H}_N^{MLE}$ of $\hat{H}_N^{MLE}(w)$ when w is drawn uniformly among words of size N, for $m = 256$, and N ranging from 1 to 4 096 (Figure 2). The x-axis has logarithmic, not linear scale. ($\hat{H}_N^{MLE}$ was evaluated by sampling over words generated using the `/dev/urandom` source.) The value of $H(p)$ is shown as the horizontal line $y = 8$. As theory predicts, when $N \gg m$, typically when N is of the order of roughly at least 10 times as large as m, then $\hat{H}_N^{MLE} \sim H(p)$. On the other hand, when N is small (roughly at least 10 times as small as m), then $\hat{H}_N^{MLE} \sim \log N$. Considering the orders of magnitude of N and m cited above, clearly we are interested in the regions where $N \sim m\dots$ precisely where $\hat{H}_N^{MLE}$ is far from $H(p)$.

Let us turn to the mathematical literature. The field of research most connected to this work is called *entropy estimation* [2], and the fact that $N \sim m$ or $N < m$ is often characterized as the fact that the probability p is *undersampled*. Note that for, say, $N = 32$ and $m = 256$, there is absolutely no chance we may have seen all bytes! This is as far as we can imagine to classical statistical experiments, with large sample sets. Despite this, we shall still be able to obtain informative statistical results.

In classical statistics, our problem is often described as follows. Take objects that can be classified into m *bins* (our bins are just bytes) according to some probability distribution p. Now take N *samples*, and try to decide whether the entropy of p is $\log m$ (or, in general, the entropy of a given, fixed probability distribution) just by looking at the samples. The papers [1, 17, 18] are particularly relevant to our work, since they attempt to achieve this precisely when the probability is undersampled, as in our case.

The problem that Paninski tries to solve [17, 18] is finding an *estimator* $\hat{H}_N$ of $H(p)$, that is, a statistical quantity, computed over randomly generated N-character words, which gives some information about the value of $H(p)$. Particularly interesting estimators are the *unbiased estimators*, that is those such that $E(\hat{H}_N) = H(p)$, where E denotes mathematical expectation (i.e., the average of all $\hat{H}_N(w)$ over all N-character words w).

Our task is:

- first, find an unbiased estimator $\hat{H}_N$ (or one with a small bias): we shall do this in Section 4.1;
- second, evaluate the confidence intervals (how close to $\hat{H}_N$ should $H_N(w)$ be for w be to be classified as random with, say, 95% confidence?); we shall do this in Section 4.2.

The sample entropy $\hat{H}_N^{MLE}$, introduced above, is an estimator, sometimes called the *plug-in estimate*, or *maximum likelihood estimator* [17]. As Figure 2 demonstrates, it is biased, and the bias can in fact be rather large. So $\hat{H}_N^{MLE}$ does not fit our requirements for $\hat{H}_N$.

Comparing $\hat{H}_N^{MLE}$ to the entropy at the limit, $\log m$, is wrong, because $\hat{H}_N^{MLE}$ is biased for any fixed N. Nonetheless, we may introduce a correction to the estimator $\hat{H}_N^{MLE}$. To this end, we must estimate the bias. Historically, the first estimation of the bias is the Miller-Madow bias correction [13] $(\hat{m} - 1)/(2N \ln 2)$, where $\hat{m} = |\{i \mid f_i \neq 0\}|$ is the number of characters that do appear at all in our N-character string w, yielding the *Miller-Madow estimator*:

$$\hat{H}_N^{MM}(w) = \hat{H}_N^{MLE}(w) + \frac{\hat{m} - 1}{2N \ln 2} = -\sum_{i=0}^{m-1} f_i \log f_i + \frac{\hat{m} - 1}{2N \ln 2}.$$

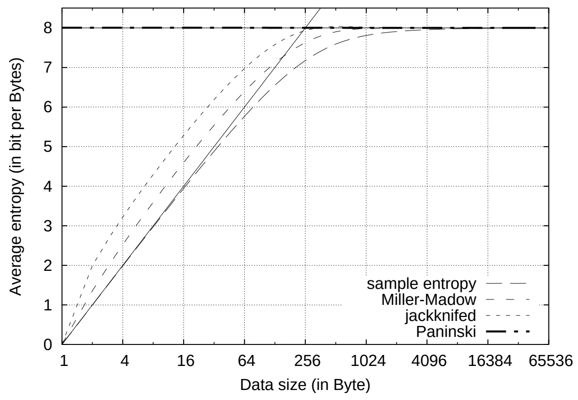

Fig. 3. Sample entropy estimators

Another one is the *jackknifed MLE* [6]:

$$\hat{H}_N^{JK}(w) = N\ \hat{H}_N^{MLE}(w) - \frac{N-1}{N}\sum_{j=1}^{N}\hat{H}_N^{MLE}(w_{-j})$$

where w_{-j} denotes the $(N-1)$-character word obtained from w by removing the jth character. While all these corrected estimators indeed correct the $1/N$ term from biases at the limit $N \to +\infty$, they are still far from being unbiased when N is small: see Figure 3, where the closer to the constant curve with y value $\log m = 8$ the better.

In the case that interests us here, i.e., when p is the uniform distribution over m characters, an exact asymptotic formula for the bias is known as a function of $c > 0$ when N and m both tend to infinity and N/m tends to c: the result is due to Liam Paninski [17, Theorem 3]. The corrected estimator is:

$$\hat{H}_N^{P}(w) = \hat{H}_N^{MLE}(w) - \log c + e^{-c}\sum_{j=1}^{+\infty}\frac{c^{j-1}}{(j-1)!}\log j \qquad (1)$$

where the correction $-\log c + e^{-c}\sum_{j=1}^{+\infty}\frac{c^{j-1}}{(j-1)!}\log j$ is the *Paninski bias*.

While the formula is exact only when N and m both grow to infinity, in practice $m = 256$ is large enough for this formula to be relevant. On our experiments, the difference between the average of $\hat{H}_N^{P}(w)$ over random experiments and $\log m = 8$ is between -0.0002 and 0.0051 for $N \leq 100\ 000$, and tends to 0 as N tends to infinity. (On Figure 3, it is impossible to distinguish $\hat{H}_N^{P}$—the "Paninski" curve—from the constant curve with y-value $\log m = 8$.) $\hat{H}_N^{P}$ is an estimator of $H(p)$ with a very small bias, when p is the uniform distribution.

4 Evaluating the Average Sample Entropy

Instead of trying to compute a correct estimator of the actual entropy $H(p)$, which is, as we have seen, a rather difficult problem, we turn the problem around.

Let $H_N(p)$ be the N*-truncated entropy* of the distribution $p = (p_i)_{i\in\Sigma}$. This is defined as the average of the sample entropy $\hat{H}_N^{MLE}(w)$ over all words w *of length* N, drawn at random according to p. In other words, this is what we plotted in Figure 2. A direct summation shows that

$$H_N(p) = \sum_{\substack{n_0,\ldots,n_{m-1}\in\mathbb{N} \\ n_0+\ldots+n_{m-1}=N}} \left[\binom{N}{n_0,\ldots,n_{m-1}} p_0^{n_0}\ldots p_{m-1}^{n_{m-1}} \times \left(\sum_{i=0}^{m-1} -\frac{n_i}{N}\log\frac{n_i}{N} \right) \right]$$

where $\binom{N}{n_0,\ldots,n_{m-1}} = \frac{N!}{n_0!\ldots n_{m-1}!}$ is the multinomial coefficient.

When p is the uniform distribution $\mathcal{U}$ (where $p_i = 1/m$ for all i), we obtain the formula

$$H_N(\mathcal{U}) = \frac{1}{m^N} \sum_{\substack{n_0,\ldots,n_{m-1}\in\mathbb{N} \\ n_0+\ldots+n_{m-1}=N}} \left[\binom{N}{n_0,\ldots,n_{m-1}} \times \left(\sum_{i=0}^{m-1} -\frac{n_i}{N}\log\frac{n_i}{N} \right) \right] \qquad (2)$$

By construction, $\hat{H}_N^{MLE}$ is then an unbiased estimator of H_N. Our strategy to detect non-random text is then to take the flow w, of length N, to compute $\hat{H}_N^{MLE}(w)$, and to compare it to $H_N(\mathcal{U})$. If the two quantities are significantly apart, then w is not random.

Not only is this easier to achieve than estimating the actual entropy $H(p)$, we shall see (Section 4.2) that this provides us much narrower confidence intervals, that is, much more precise estimates of non-randomness.

For example, if w is the word

```
0x55 0x89 0xe5 0x83 0xec 0x58 0x83 0xe4
0xf0 0xb8 0x00 0x00 0x00 0x00 0x29 0xc4
0xc7 0x45 0xf4 0x00 0x00 0x00 0x00 0x83
0xec 0x04 0xff 0x35 0x60 0x99 0x04 0x08
```

of length $N = 32$ (so $N \ll m = 256$, a very much undersampled situation), then $\hat{H}_N^{MLE}(w) = 3.97641$, while $H_N(\mathcal{U}) = 4.87816$, to 5 decimal places. Since 3.97641 is significantly less than 4.87816 (about 1 bit less information), one is tempted to conclude that w above is *not* random. (This is indeed true: this w is the first 32 bytes of the code of the `main()` function of an ELF executable, compiled under `gcc`. However, we cannot yet conclude, until we compute confidence intervals, see Section 4.2.)

Consider, on the other hand, the word

```
0x85 0x01 0x0e 0x03 0xe9 0x48 0x33 0xdf
0xb8 0xad 0x52 0x64 0x10 0x03 0xfe 0x21
0xb0 0xdd 0x30 0xeb 0x5c 0x1b 0x25 0xe7
0x35 0x4e 0x05 0x11 0xc7 0x24 0x88 0x4a
```

This has sample entropy $\hat{H}_N(w) = 4.93750$. This is close enough to $H_N(\mathcal{U}) = 4.87816$ that we may want to conclude that this w is close to random. And indeed, this w is the first 32 bytes of a text message encrypted with `gpg`. Comparatively, the entropy of the first 32 bytes of the corresponding plaintext is only 3.96814.

Note that, provided a deviation of roughly 1 bit from the predicted value $H_N(\mathcal{U})$ is significant, the $\hat{H}_N^{MLE}$ estimator allows us to detect deviations from random-looking messages extremely quickly: using just 32 characters in the examples above. Actual message sizes in SSL or SSH vary from a few dozen bytes (usually for small control messages) to a few kilobytes (user data), with a maximum of 64KB.

4.1 Computing $H_N(\mathcal{U})$

There are basically three ways to compute $H_N(\mathcal{U})$. (Recall that we need this quantity to compare $\hat{H}_N^{MLE}(w)$ to.) The first is to use Equation (2). However, this quickly becomes unmanageable as m and N grow.

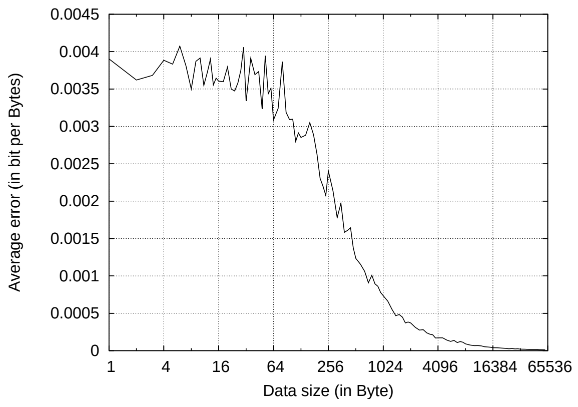

Fig. 4. Error term in (3)

A much better solution is to recall Equation (1). Another way of reading it is to say that, for each constant c, when N and m tend to infinity in such a way that N/m is about c, then $H_N(\mathcal{U})$ is equal to $\log m$ minus the Paninski bias, namely:

$$H_N(\mathcal{U}) = \log m + \log c - e^{-c} \sum_{j=1}^{+\infty} \frac{c^{j-1}}{(j-1)!} \log j + o(1). \tag{3}$$

As we have seen, when $m = 256$, this approximation should give a good approximation of $H_N(\mathcal{U})$. In fact, this approximation is surprisingly close to the actual value of $H_N(\mathcal{U})$. The $o(1)$ error term is plotted in Figure 4. It is never more than 0.004 bit, and decreases quickly as N grows. The series (3) converges quickly, too: the sum stabilizes after a number of iterations that is roughly linear in c. We implemented this using 64-bit IEEE floating-point numbers, and never needed more than 398 iterations for $N \leq 65\ 536$ ($c = 256$). This grew to 415 iterations with 96-bit IEEE floating-point numbers, and to 786 iterations with 512-bit floating-point numbers using the arbitrary precision floating-point arithmetic library MPFR [15]. Comparing the values of the Paninski bias computed with 64-bit and 512-bit numbers reveals a relative difference that never exceeds $1.1\ 10^{-11}$, which is negligible: 64-bit computations are enough.

The third method to evaluate $H_N(\mathcal{U})$ is the standard Monte-Carlo method consisting in drawing enough words w of length N at random, and taking the average of $\hat{H}_N(w)$ over all these words w. This is how we evaluated $H_N(\mathcal{U})$ in Figure 2, and how we defined the reference value of $H_N(\mathcal{U})$ which we compared to (3) in Figure 4. To be precise, we took the average over 100 000 samples for $N < 65\ 536$, taking all values of N below 16, taking one value in 2 below 32, one value in 4 below 64, $\ldots$, and one value in 4 096 below 65 536. The spikes are statistical variations that one may attribute to randomness in the source. Note that they are in general smaller than the error term $o(1)$ in (3).

In the end, the fastest implementation is just by using a pre-filled table of values of $H_N(\mathcal{U})$ for values of $c = N/m = N/256$, with N ranging from 0 to $N_{\max}$. One can then use any of the above three methods to fill the table. With $N_{\max} = 65\ 536$, this requires a table that takes less than one megabyte. We can also save some memory by exploiting the fact that $H_N(\mathcal{U})$ is a smooth and increasing [17, Proposition 3] function of c, storing only a few well-chosen points and extrapolating; and by using the fact that values of N that are not multiples of 4 or even 8 are hardly ever needed.

4.2 Confidence Intervals

Evaluating $\hat{H}_N^{MLE}(w)$ only gives a statistical indication of how close we are to $H_N(\mathcal{U})$. Recall our first example, where w was the first 32 bytes of the code of the `main()` function of some ELF executable. We found $\hat{H}_N^{MLE}(w) = 3.97641$, while $H_N(\mathcal{U}) = 4.87816$. What is the actual probability that w of length $N = 32$ is non-random when $\hat{H}_N^{MLE}(w) = 3.97641$ and $H_N(\mathcal{U}) = 4.87816$?

It is again time to turn to the literature. According to [1, Section 4.1], when N tends to $+\infty$, $\hat{H}_N^{MLE}$ is asymptotically Gaussian, in the sense that $\sqrt{N} \ln 2(\hat{H}_N^{MLE} - H)$ tends to a Gaussian distribution with mean 0 and variance $\sigma_N^2 = Var\{-\log p(X)\}$. In non-degenerate cases (i.e., when $\sigma_N^2 > 0$), the expectation of $(\hat{H}_N^{MLE} - H)^2$ is $\Theta(1/N)$, so the standard deviation will be proportional to $1/\sqrt{N}\ldots$ but precisely, the $p = \mathcal{U}$ case *is* degenerate.

As we shall see below, this is actually good news! In the degenerate case, the standard deviation will be proportional to $1/N$ indeed, which goes to zero much faster than $1/\sqrt{N}$.

This means that the confidence intervals will be remarkably small, of the order of $1/N$. Said differently, this means that to reach small confidence intervals, we shall only need very few bytes. Let us see how much. Much less is known about the variance of $\hat{H}_N^{MLE} = \hat{H}_N^{MLE}$ when $N \sim m$ or $N < m$ than about its bias. One useful inequality is that the variance of $\hat{H}_N^{MLE}$ is bounded from above by $\log^2 N/N$ [1, Remark (iv)], but this is extremely conservative. An improved bound is due to Moddemeijer [14, Equation (12)]: the variance of $\hat{H}_N^{MLE}$ evolves as $\sigma_N^2 + \frac{m-1}{2N^2 \ln^2 2}$ when $N \to +\infty$. In degenerate cases, where $\sigma_N = 0$, this means that the statistical standard deviation $SD(\hat{H}_N^{MLE})$ of $\hat{H}_N^{MLE}(w)$, on random words w of length N, is asymptotically equal to $\sqrt{\frac{m-1}{2}}\frac{1}{\ln 2}$ times $1/N$. With $m = 256$, this is about $16.29/N$. To confirm this, we have estimated the standard deviation of $\hat{H}_N$ by a Monte-Carlo method, see Figure 5. The curve on the right is the same as the one on the left, except the y-axis is in log scale, showing the relation between $SD(\hat{H}_N^{MLE})$ and $16.29/N$ more clearly.

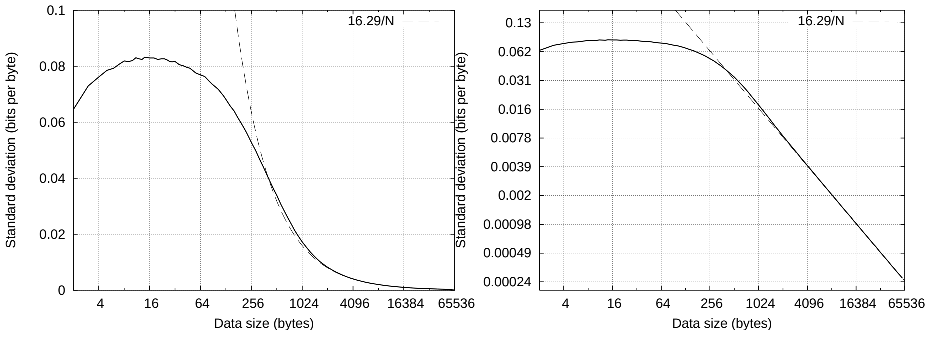

Fig. 5. Standard deviation of $\hat{H}_N^{MLE}(w)$ (linear scale left, log scale right)

For typical sizes of 1, 2, 4, and 8 KB, the standard deviation $SD(\hat{H}_N^{MLE})$ is 0.016, 0.008, 0.004, and 0.002 bit respectively. This is extremely small.

Let us estimate percentiles, again by a Monte-Carlo method, see Figure 6: the y values are given so that a proportion of all words w tested falls within $y \times SD(\hat{H}_N^{MLE})$ of the average value of $\hat{H}_N^{MLE}$. The proportions go from 50% (bottom) to 99.9% (top). Unless $N \leq 16$ (which is unrealistic), our estimate of $\hat{H}_N^{MLE}$ is exact with an error of at most $4 \times SD(\hat{H}_N^{MLE})$, with probability 99.9%. $4SD(\hat{H}_N)$ is at most $64/N$, and in any case no larger than 0.32 bit (for words of about 16 characters).

Let's return to our introductory question: What is the actual probability that w of length $N = 32$ is non-random when $\hat{H}_N^{MLE}(w) = 3.97641$ and $H_N(\mathcal{U}) = 4.87816$? For $N = 32$, $SD(\hat{H}_N^{MLE})$ is about maximal, and equal to 0.081156. So we are at least 99.9% sure that the entropy of a 32-byte word with characters drawn uniformly is $4.87816 \pm 4 \times 0.081156$, i.e., between 4.55353 and 5.20279: if $\hat{H}_N^{MLE}(w) = 3.97641$, we can safely bet that w is *not* random.

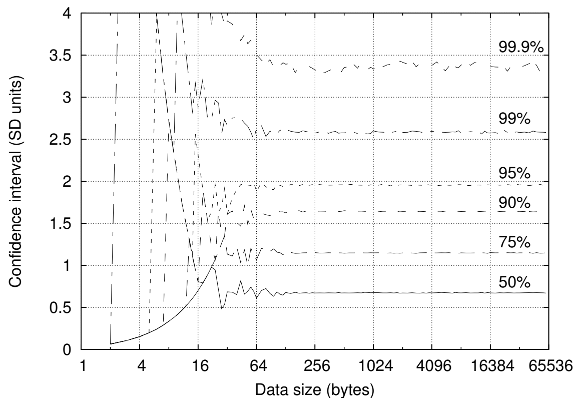

Fig. 6. Percentiles

Note that $N = 32$ is not only a terribly undersampled case, but is also close to the worst possible case we could dream of (see Figure 5). Still, $\hat{H}_N^{MLE}$ is already a reliable estimator of randomness here.

For sizes 1, 2, 4, and 8 KB, and a confidence level of 99.9% again, $\hat{H}_N^{MLE}$ is precise up to ± 0.0625, ± 0.0313, ± 0.0156, and ± 0.0078 bit respectively. These are remarkably small values.

4.3 Practical experiments

Data source	**Entropy** (bits/byte)	
	$\hat{H}_N^{MLE}$	H_N
Binary executable (elf-i386)	6.35	8.00
Shell scripts	5.54	8.00
Terminal activity	4.98	8.00
1 Gbyte e-mail	6.12	8.00
1KB X.509 certificate (PEM)	5.81	7.80 ± 0.061
700B X.509 certificate (DER)	6.89	7.70 ± 0.089
130B bind shellcode	5.07	6.56 ± 0.24
38B standard shellcode	4.78	5.10 ± 0.28
73B polymorphic shellcode	5.69	5.92 ± 0.27
Random 1 byte NOPs (i386)	5.71	7.99

Fig. 7. Sample entropy of some common non-random sources

We report some practical experiments in Figure 7, on non-cryptographic sources. This gives an idea of the amount of redundancy in common data sources.

The entropy of binary executables (ELF format, i386 architecture) was evaluated under Linux and FreeBSD by collecting all `.text` sections of all files in `/bin` and `/usr/bin`. Similarly, the entropy of shell scripts was computed by collecting all shell scripts on the root volume of Linux and FreeBSD machines (detected by the `file` command). Terminal activity was collected by monitoring a dozen `telnet` connections (port 23) on `tcp` from a given machine with various activity, such as text editing, manual reading, program compilation and execution (about 1 MB of data). As far as e-mail is concerned, the measured entropy corresponds to 3 years of e-mail on the first author's account. These correspond to large volumes of data (large N), so that H_N is 8 to 2 decimal places, and confidence intervals are ridiculously small.

The next experiments were made on smaller pieces of data. Accordingly, we have given H_N in the form $H \pm \delta$, where δ is the 99.9% confidence interval. Note that X.509 certificates are definitely classified as non-random. We have also tested a few shellcodes, because, first, as we have seen in Figure 1, it is interesting to detect when some random piece of data is replaced by a shellcode, and second, because detecting shellcodes this way is challenging. Indeed, shellcodes are typically short, so that H_N is significantly different from 8. More importantly, modern polymorphic and metamorphic virus technologies, adapted to shellcodes, make them look more random. (In fact, the one we use *is* encrypted, except for a very short prolog.) While the first two shellcodes in Figure 7 are correctly classified as non-random (even a very short 38 byte non-polymorphic shellcode), the last, polymorphic shellcode is harder to detect. The 99.9% confidence interval for being random is $[5.65, 6.19]$: the sample entropy of the 73 byte polymorphic shellcode is at the left end of this interval. The 99% confidence interval is 5.92 ± 0.19, i.e., $[5.73, 6.11]$: with 99% confidence, this shellcode is correctly classified as non-random. In practice, shellcodes are usually preceded with padding, typically long sequences of the letter `A` or the hexadecimal value `0x90` (the No-OPeration i386 instruction), which makes the entropy decrease drastically, so the examples above are a worst-case scenario. Detecting that the random `key-arg` field of Figure 1 (left) was replaced by a shellcode (right) is therefore feasible.

Another worst-case scenario in polymorphic viruses and shellcodes is given by mutation, whereby some specific instructions, such as `nop`, are replaced with other instructions with the same effect, at random. This fools pattern-matching detection engines, and also increases entropy. However, as the last line shows on a large amount of random substitutes for `nop` on the i386 architecture, this makes the sample entropy culminate at a rather low value compared to 8.

All in all, $\hat{H}_N^{MLE}$ is an extraordinarily precise estimator of $H_N(\mathcal{U})$, even in very undersampled cases. If you didn't believe the mathematics, we hope that these experimental data have convinced you. However, in the end, it is the mathematics which give you the right estimator (the Paninski estimator) and the bounds to expect of confidence intervals (Moddemeijer's theorem).

5 Conclusion

We have described the basic principles behind NetEntropy, a tool that we originally designed so as to detect attacks against cryptographic network flows, but can also be used for traffic analysis and malware detection. NetEntropy compares the sample entropy of the flow to an estimator $\hat{H}_N^{MLE}$ with a very small bias, and signals any deviation above some threshold of the order of $16.29/N$.

The bias of the Paninski estimator $\hat{H}_N^{MLE}$ is extremely small, and the threshold $16.29N$ is minute, which allows us to correctly conclude that certain hard-to-qualify sources such as some polymorphic shellcodes are definitely non-random, even after having read only a few bytes (73 in our example). This is extraordinary: statistically, we are *undersampling* a probability distribution, to the point that we cannot have possibly seen all possible bytes; still, we can conclude.

The key to the miracle is *mathematics*: here, Paninski's estimator and Moddemeijer's theorem. In practical security, mathematics is often seen as an intellectual's game, which the rest of us don't need. We hope to have made a convincing statement that mathematics, as in most sciences, is essential.

Acknowledgments

Thanks to Mathieu Baudet, Elie Bursztein and Stéphane Boucheron for judicious advice. This work was partially supported by the RNTL Project DICO, and the ACI jeunes chercheurs “Sécurité informatique, protocoles cryptographiques et détection d'intrusions”.

Availability

NetEntropy is a free open source project. It is available under the CeCILL2 license [3]. The project homepage can be found at `http://www.lsv.ens-cachan.fr/net-entropy/`.

References

1. A. Antos and I. Kontoyiannis. Convergence properties of functional estimates for discrete distributions. *Random Structures and Algorithms*, 19:163–193, 2001.
2. W. Bialek and I. Nemenman, editors. *Estimation of Entropy and Information of Undersampled Probability Distributions—Theory, Algorithms, and Applications to the Neural Code*, Whistler, BC, Canada, Dec. 2003. `http://www.menem.com/~ilya/pages/NIPS03/`. Satellite of the Neural Information Processing Systems Conference (NIPS'03).
3. CEA, CNRS, and INRIA. Cecill free software license agreement, 2005. `http://www.cecill.info/licences/Licence_CeCILL_V2-en.html`.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & sons, 1991.

5. P. Dorfinger, G. Panholzer, B. Trammell, and T. Pepe. Entropy-based traffic filtering to support real-time skype detection. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 747–751, New York, NY, USA, 2010. ACM.
6. B. Efron and C. Stein. The jackknife estimate of variance. *Annals of Statistics*, 9:586–596, 1981.
7. X. Fu, B. Graham, R. Bettati, and W. Zhao. Active traffic analysis attacks and countermeasures. In *Proc. 2nd IEEE Int. Conf. Computer Networks and Mobile Computing*, pages 31–39, 2003.
8. J. Goubault-Larrecq and J. Olivain. A smell of Orchids. In M. Leucker, editor, *Proceedings of the 8th Workshop on Runtime Verification (RV'08)*, volume 5289 of *Lecture Notes in Computer Science*, pages 1–20, Budapest, Hungary, Mar. 2008. Springer.
9. D. Lubicz. On a classification of finite statistical tests. *Advances in Mathematics of Communications*, 1(4):509–524, 2007.
10. R. Lyda and J. Hamrock. Using entropy analysis to find encrypted and packed malware. *Security Privacy, IEEE*, 5(2):40–45, 2007.
11. J. McDonald. OpenSSL SSLv2 malformed client key remote buffer overflow vulnerability. `http://www.securityfocus.com/bid/5363`, May 2003. BugTraq Id 5363.
12. A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
13. G. A. Miller. Note on the bias of information estimates. In H. Quastler, editor, *Information Theory in Psychology; Problems and Methods II-B*, pages 95–100, Glencoe, IL, USA, 1955. Free Press.
14. R. Moddemeijer. The distribution of entropy estimators based on maximum mean log-likelihood. In J. Biemond, editor, *Proc. 21st Symp. Information Theory in the Benelux*, pages 231–238, Wassenaar, the Netherlands, May 2000.
15. The GNU MPFR library. `http://www.mpfr.org/`. Consulted December 02, 2013.
16. J. Olivain and J. Goubault-Larrecq. Detecting subverted cryptographic protocols by entropy checking. Research Report LSV-06-13, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2006. 19 pages.
17. L. Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15:1191–1253, 2003.
18. L. Paninski. Estimating entropy on m bins given fewer than m samples. *IEEE Transactions on Information Theory*, 50(9):2200–2203, Sept. 2004.
19. C. Rossow and C. J. Dietrich. Provex: Detecting botnets with encrypted command and control channels. In K. Rieck, P. Stewin, and J.-P. Seifert, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 7967 of *Lecture Notes in Computer Science*, pages 21–40. Springer Berlin Heidelberg, 2013.
20. A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. B. andAlan Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST, Apr. 2010. Revised, Lawrence E. Bassham III.
21. K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *RAID'2005*, pages 227–246. Springer-Verlag LNCS 3858, 2005.
22. M. Zalewski. SSH CRC-32 compensation attack detector vulnerability. `http://www.securityfocus.com/bid/2347`, Feb. 2001. BugTraq Id 2347.
23. H. Zhang, C. Papadopoulos, and D. Massey. Detecting encrypted botnet traffic. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 163–168, 2013.