Entropia – pomiar i zastosowanie

Michał *Gynvael Coldwind* Składnikiewicz

stopień trudności

Entropia, rozumiana jako wielkość losowości pewnych danych, może służyć wprawnemu badaczowi jako narzędzie pomocne w wyszukiwaniu ukrytych danych oraz interpretacji budowy nieznanych plików. Niniejszy artykuł ma na celu przedstawić Czytelnikowi zagadnienia związane z entropią.

Na początek trochę teorii, czyli matematyczna definicja entropii omawianej w dalszej części tego artykułu: entropia jest funkcją pewnego ciągu danych, którą można wyrazić jako sumę iloczynów prawdopodobieństw wystąpienia danego znaku oraz ilości jego wystąpień w ciągu danych. W pseudokodzie obliczanie entropii wygląda następująco:

```
entropia = 0
dla każdego znaku 'i' w danym alfabecie:
   entropia = entropia + p(i) * ilość(i)
```

gdzie *p(i)* to prawdopodobieństwo wystąpienia znaku *i* w danym zbiorze danych, a *ilość(i)* to ilość jego wystąpień w tym ciągu. Tak zdefiniowana entropia przyjmuje wartości od 1 do wielkości równej co do wartości ilości znaków w ciągu.

Tyle teorii. W praktyce alfabetem są zazwyczaj wszystkie znaki ASCII (konkretnie wszystkie 256 wartości, które może przyjąć jeden bajt), a ciąg danych to po prostu pewne dane binarne, których entropię chcemy zmierzyć – na przykład plik. Jeżeli chodzi o praktyczne wartości zwracane przez funkcję entropii, to wyobraźmy sobie dwa przypadki. W pierwszym z nich załóżmy, że mamy plik o wielkości 256 bajtów, który zawiera same litery *A*. W takim wypadku prawdopodobieństwo wystąpienia litery *A* jest stuprocentowe, a prawdopodobieństwo wystąpienia jakiegokolwiek innego znaku jest zerowe – tak więc entropia będzie wynosić 1 * 100% * 256 bajtów + 255 * 0% * 0, czyli (pozbywając się jednostek) 256. W drugim, skrajnie przeciwnym przypadku, wyobraźmy sobie plik, również o wielkości 256 bajtów, który zawiera wszystkie znaki ASCII (tj. każdy możliwy znak ASCII wystąpi w nim dokładnie raz). Wtedy prawdopodobieństwo wystąpienia każdego znaku wynosi 1 / 256, a entropia będzie sumą dwustu pięćdziesięciu sze-

Z artykułu dowiesz się

- co to jest entropia,
- jak wykorzystać pomiary entropii do lokalizowania ukrytych danych,
- jak interpretować wykresy entropii danych.

Co powinieneś wiedzieć

- mieć ogólne pojęcie o matematyce,
- mieć ogólne pojęcie o informatyce.

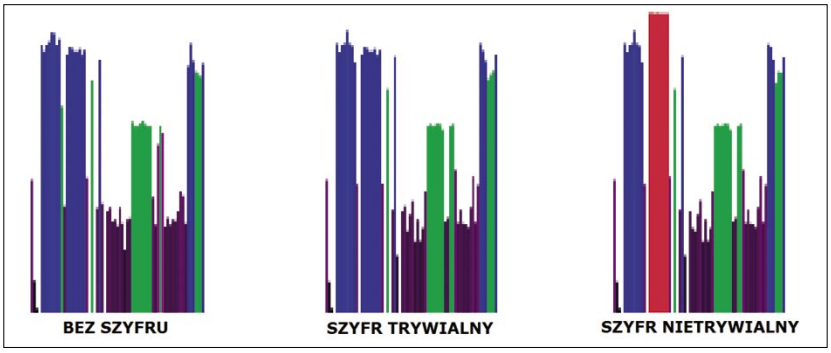
ściu iloczynów 1 / 256 * 1 bajt, co sprowadzi się do 256 / 256 czyli po prostu do entropii równej jeden.

Zatem entropia jest średnią ważoną prawdopodobieństw wystąpień każdego ze znaków ASCII w określonym ciągu danych. Należy zauważyć, że czym mniejsza wartość entropii w powyższym przypadku, tym więcej rodzajów znaków zostało użytych. Pojawia się pytanie – czy są jakieś charakterystyczne wartości (lub ich przedziały) entropii dla różnego rodzaju danych? Przed przejściem do odpowiedzi na to pytanie należy jeszcze powiedzieć dwie rzeczy.

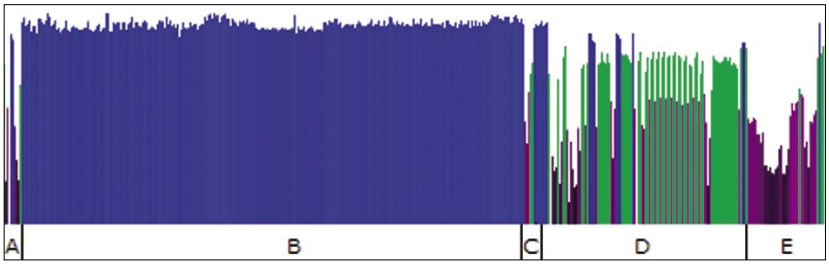
Po pierwsze, wartość entropii NIE ZALEŻY od kolejności wystąpień danych znaków w ciągu, a jedynie od ich ilości oraz całkowitej długości ciągu (można jednak temu prosto zaradzić wyliczając na przykład entropię z ciągu różnic kolejnych bajtów lub wykonując bardziej skomplikowane transformacje). Po drugie, entropia wyrażana w skali od 1 do długości ciągu wydaje się być mało wygodna, dlatego w dalszej części artykułu posługiwać się będę entropią wyrażoną procentowo. Przekształcenie na wynik procentowy uzyskać można korzystając z prostego wzoru:

```
Entropia = (1 - Entropia / Długość_
                        Ciągu) * 100%
```

Tak więc entropia 0% oznaczać będzie stały ciąg (na przykład same literki *A*), a entropia na poziomie 99,9% (100% nigdy tak naprawdę nie osiągnie) – ciąg całkowicie losowy. Dodatkowo można jeszcze podnieść uzyskaną wartość do kwadratu, aby zwiększyć różnicę między poszczególnymi (górnymi) poziomami entropii.



Rysunek 1. *Porównanie entropii programu z danymi jawnymi oraz zaszyfrowanymi*



Rysunek 2. *Wykres entropii^2 dla pliku calc.exe*

Rodzaj danych a entropia

Okazuje się że poziom entropii różnego rodzaju danych różni się od siebie. Tak więc tekst w języku naturalnym będzie miał inny poziom entropii niż kod maszynowy, a ten z kolei będzie miał inny poziom entropii niż skompresowane dane. Przeanalizujmy poszczególne przypadki. Tekst w języku naturalnym zawiera w sobie jedynie niewielki wycinek całego *alfabetu* ASCII. Przykładowo do napisania niniejszego artykułu wystarczyły jedynie małe i wielkie litery łacińskie, małe i wielkie polskie znaki diakrytyczne oraz pewne znaki specjalne, takie jak kropka, nawias czy znak pusty (*spacja*).

Tekst w języku naturalnym ma zazwyczaj entropię na poziomie od 94,5% do 95,3%, a entropię^2 na poziomie od 87% do 92%. Należy zaznaczyć, że entropia ta może być niższa w przypadku, gdy dane słowo (lub jakaś jego część) często się powtarza – na przykład w wierszach/tekstach piosenek lub jako często używany prefiks nazwy obiektu w programie. Kolejnym rodzajem danych może być kod maszynowy, czyli *skompilowany asembler.* Entropia kodu maszynowego zależy w dużej części od rodzaju operacji, które są w analizowanym fragmencie wykonywane. Przykładowo obliczenia arytmetyczne (ADD, SUB, DIV etc.) mają inne opkody (ang. *opcode*) niż obliczenia wykonywane na koprocesorze (FADD etc.), a te z kolei różnią się od opkodów instrukcji kontrolujących przebieg aplikacji (JMP, TEST, CMP, JNZ etc.). Tak więc ciąg, w którym program wykonuje jedynie obliczenia na koprocesorze, będzie miał niższą entropię niż ciąg, w którym jest wszystkiego po trochu. Kod maszynowy charakteryzuje się entropią na poziomie od 80% do 98%, a entropią^2 na poziomie od 64% do 97%.

W przypadku skompresowanych danych sprawa jest prostsza, ponieważ istotą kompresji jest dążenie do upakowania jak największej ilości informacji w jak najmniejszej ilości bajtów, co wymusza znaczny wzrost entropii względem oryginału. Dokładny poziom entropii zależy od algorytmu kompresji, jednak prawdopodobnie będzie on na poziomie od około 99% wzwyż (czyli entropia^2 będzie na poziomie około 98% wzwyż). Ostatnim ciekawym przypadkiem są wszelkiego rodzaju protokoły oraz formaty binarne (takie, jak na przykład format symboli debuggera czy format nagłówków sekcji plików wykonywalnych). Dane w nich są wysoce uporządkowane, zazwyczaj składają się z offsetów, adresów i tym podobnych rzeczy, a także (co każdy programista oraz *reverse engineer* dobrze wie) znajduje się w nich bardzo dużo zer (zer binarnych – znaków ASCII o kodzie 0). Tego typu dane mają entropię na poziomie od 40% do 75%, a entropię^2 na poziomie od 16% do 55%. Pomiar entropii innych rodzajów da-

nych pozostawiam Czytelnikowi jako zadanie domowe.

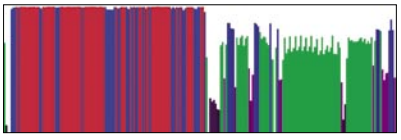
Szyfrowanie a entropia

Entropia zaszyfrowanego ciągu danych, w zależności od użytego algorytmu szyfrowania, może się w ogóle nie zmienić, lub znacznie wzrosnąć.

Pierwszy przypadek jest często spotykany w przypadku algorytmów, w których jeden znak jest zamieniany na inny – tak, jak w przypadku szyfru Cezara lub w części automatycznie generowanych algorytmów używanych do szyfrowania kodu w plikach wykonywalnych. Takie algorytmy, zazwyczaj trywialne, zbudowane są z szeregu odwracalnych instrukcji arytmetyczno-logicznych – takich jak ADD, SUB, ROR, ROL, XOR, INC czy DEC – oraz korzystają ze stałego, jednobajtowego klucza, który jest argumentem poszczególnych instrukcji. Należy zauważyć, iż w wypadku takich algorytmów zmienią się kody poszczególnych znaków, natomiast pary prawdopodobieństwa oraz ilości wystąpień pozostaną stałe, w związku z czym entropia nie ulegnie zmianie (jako, że we wzorze na entropie kod znaku nie ma znaczenia).

W przypadku algorytmów, które szyfrują całe bloki informacji lub szyfrują strumieniowo, ale w których istotne jest miejsce wystąpienia znaku (tj. w których *A* na pozycji trzeciej zostanie inaczej zaszyfrowane niż *A* na pozycji czwartej) poziom entropii się zmieni. W przypadku większości algorytmów szyfrowania bezpiecznie jest powiedzieć nawet, że entropia wzrośnie do poziomu 98% lub więcej (a entropia^2 do poziomu 96% lub więcej).

Z powyższych faktów płyną dwa bardzo ważne dla nas wnioski. Po pierwsze, w przypadku gdy ktoś próbuje ukryć (na przykład w pliku wykonywalnym) zaszyfrowane dane nietrywialnym algorytmem, wykres entropii pozwoli błyskawicznie zlokalizować miejsce ukrycia tychże danych.



Rysunek 3. *Wykres entropii^2 dla pliku calc.exe spakowanego UPX'em*

Listing 1a. *Program tworzący wykresy entropii*

```
// Entropy Chart Drawer "ent"
// Code by gynvael.coldwind//vx
#include <cstdio>
#include <cstdlib>
#include <cstring>
using namespace std;
// Typedef
typedef unsigned char byte_t;
// Naglowek TGA
#pragma pack(push, 1)
static struct TGA_HEADER {
  byte_t  id_field, c_map, img_type, color_map[5];
  unsigned short x_origin, y_origin, width, height;
  byte_t  bpp, flip;}
TgaHeader = { 0, 0, 2, { 0,0,0,0,0 }, 0, 0, 0, 0, 24, 0x20 };
#pragma pack(pop)
// Dane globalne
static int Height = 480, Width;
// Pobierz dane z pliku
byte_t *FileGetContent(const char *FileName, size_t *SizePtr){
  byte_t *Data;
  size_t Size;
FILE *f = fopen(FileName, "rb");
  if(!f) return NULL;
fseek(f, 0, SEEK_END); Size = ftell(f); fseek(f, 0, SEEK_SET);
Data = (byte_t*)malloc(Size);
  if(!Data){
    fclose(f);
    return NULL;}
 fread(Data, 1, Size, f);
  fclose(f);
 *SizePtr = Size;
return Data;  }
// Wrzuc dane do pliku
bool FileSetContent(const char *FileName, const byte_t *Data, size_t Size){
  FILE *f = fopen(FileName, "wb");
  if(!f) return false;
fwrite(Data, 1, Size, f);
  fclose(f);
return true;}
// Stworz histogram
```

Po drugie, w przypadku gdy wiemy, że jakieś dane są zaszyfrowane, ale ich poziom entropii jest stosunkowo niski (na przykład około 88%), to wywnioskować można, że dane te zostały zaszyfrowane trywialnym algorytmem zamieniającym poszczególne znaki na ich zakodowane odpowiedniki (co oznacza również, że taki algorytm podatny jest na atak za pomocą analizy statystycznej). Na Rysunku 1. przedstawione są 3 programy. W pierwszym (licząc od lewej) występuje pewna ilość danych, która jest jawna (niezaszyfrowana). W drugim dane te zostały zaszyfrowane trywialnym algorytmem (XOR), a w trzecim – nietrywialnym algorytmem (co spowodowało znaczny wzrost entropii).

Zastosowanie pomiaru entropii

Pomiar entropii stosować można przede wszystkim w inżynierii wstecznej (ang. *reverse engineering*) podczas analizy plików wykonywalnych. W takim przypadku najlepiej jest utworzyć wykres entropii kolejnych fragmentów (na przykład 256-bajtowych) pliku. Taki wykres może ułatwić zrozumienie budowy pliku, pomóc w lokalizacji poszczególnych składników pliku wykonywalnego, a także powiedzieć co nieco o jego zawartości dodatkowej (ukryte dane, ikony, bitmapy). Przykładowo Rysunek 2 przedstawia wykres entropii głównego pliku wykonywalnego kalkulatora dołączonego do Windows XP, stworzony za pomocą programu z Li-

stingu 1. Wykres został podzielony na 5 części (A, B, C, D oraz E). Pierwsza część (A) odzwierciedla nagłówki MZ oraz PE – wyraźnie na niej widać niską, zmienną entropię. Druga część (B), stanowiąca największą część wykresu, przedstawia entropię sekcji kodu (.text) kalkulatora. Cześć trzecia (C) to tablica importów – widać na wykresie wyraźny spadek entropii (z uwagi na uporządkowaną strukturę oraz dużą ilość zer). Cześć czwarta (D) to zasoby pliku PE, a konkretnie ikony zawarte w zasobach. Ostatnia część (E) to reszta zasobów – menu, tablica stringów oraz akceleratory. Dla porównania Rysunek 3 przedstawia wykres entropii tego samego pliku, ale po zastosowaniu pakera UPX (jest to program kompresujący pliki wykonywalne, ale do takiej postaci, że można je nadal uruchamiać – rozpakowują się one *w locie* podczas uruchomienia). Na wykresie wyraźnie widać obszar, w którym entropia wzrosła (kompresja kodu). Zasoby pliku najwyraźniej nie zostały skompresowane.

Pomiar entropii można wykorzystać również w systemach detekcji zachowań anormalnych (na przykład w protokołach sieciowych) lub w systemach wykrywających ukryte dane, a także przy analizie nieznanych formatów plików, szczególnie stosujących kompresję bądź szyfrowanie (lokalizacja danych oraz nagłówków).

Podsumowanie

Artykuł ten stanowi jedynie wstęp do zagadnień związanych z wykorzystaniem pomiaru entropii danych w inżynierii wstecznej. Gorąco zachęcam Czytelników do eksperymentów związanych z pomiarem entropii oraz, być może, nawet podzielenia się wynikami na łamach hakin9. ●

O autorze

Michał Składnikiewicz, inżynier informatyki, ma wieloletnie doświadczenie jako programista oraz *reverse engineer*. Obecnie jest koordynatorem działu analiz w międzynarodowej firmie specjalizującej się w bezpieczeństwie komputerowym.
Kontakt z autorem:
gynvael@coldwind.pl

Listing 1b. *Program tworzący wykresy entropii*

```
void MakeHistogram(const byte_t *Data, size_t Size, int Array[256]){
size_t i;memset(Array, 0, sizeof(int) * 256);
  for(i = 0; i < Size; i++) Array[Data[i]]++;}
// Wylicz entropie
double GetEntropy(const byte_t *Data, int Size){
  int Array[256];
  int i;
  double Entropy = 0.0;
MakeHistogram(Data, Size, Array);
  for(i = 0; i < 256; i++)
    Entropy += ((double)Array[i] / Size) * (double)Array[i];return Entropy;
                    }
// Funkcja rysujaca pionowa linie
void DrawLine(byte_t *Bitmap, int x, int LineHeight, byte_t R, byte_t G,
                     byte_t B){
  byte_t PixelColor[3] = { R, G, B };int i;
for(i = Height - 1; i >= LineHeight; i--)
    memcpy(Bitmap + (i * Width + x) *3, PixelColor, 3);}// Funkcja glownaint
main(int argc, char **argv){
  int SampleLength = 256;// Sprawdz argumenty
  if(argc < 2){
    fprintf(stderr, "usage: ent <FileName> [<SampleLength>]\n"
                    "SampleLength is 256 bytes by default\n");return 1; }
if(argc == 3) SampleLength = atoi(argv[2]);// Pobierz dane
  byte_t *BitmapData, *OutputData;
  byte_t *Data;
  size_t FileSize, Sz, i;
Data = FileGetContent(argv[1], &FileSize);if(!Data) {
    fprintf(stderr, "Error! File not found!\n");
    return 2; }// Zaalokuj pamiec dla bitmapy
  Sz = FileSize - SampleLength;
```

Listing 1c. *Program tworzący wykresy entropii*

```
  Width = FileSize / SampleLength;
OutputData = (byte_t*)malloc(sizeof(TgaHeader) + Width * Height * 3);
BitmapData = OutputData + sizeof(TgaHeader);
  memset(BitmapData, 0xff, Width * Height * 3);
int cnt = 0;// Narysuj wykres
 for(i = 0; i < Sz; i += SampleLength, cnt++) {
   int hi;
    double ent = GetEntropy(&Data[i], SampleLength);
    ent = 1.0 - (ent / SampleLength);//    ent = ent * ent;
    hi = (int)((1.0 - ent) * (float)Height);ent *= 100.0;
    printf("%.8x - %.8x: %.5f%%\n", i, i + SampleLength, ent);
if(ent >= 98.0)      DrawLine(BitmapData, cnt, hi, 0, 0, (byte_t)(ent *
                     2.55) );
    else if(ent >= 81.0) DrawLine(BitmapData, cnt, hi, (byte_t)(ent *
                     2.125), 0, 0);
    else if(ent >= 60.0) DrawLine(BitmapData, cnt, hi, 0, (byte_t)(ent *
                     2.55), 0 ); else                DrawLine(BitmapData,
                     cnt, hi, (byte_t)(ent * 2.55), 0, (byte_t)(ent * 2.55)
                     );}// Zapisz bitmapechar Name[512];
  snprintf(Name, sizeof(Name), "%s.tga", argv[1]);
TgaHeader.width = Width;
  TgaHeader.height = Height;
  memcpy(OutputData, &TgaHeader,
         sizeof(TgaHeader));
  FileSetContent(Name, OutputData, Width * Height * 3 + sizeof(TgaHeader));
//  Uwolnij danefree(Data);free(OutputData);
 return 0;}
```