Automatic file format analysis

Richard Ostertág, Martin Pašen

Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia

Abstract: During an analysis of the security of various devices, researchers often face the task of analyzing the format of an unknown binary file. There are various tools that can help them, but these tools are often limited in depth of their analysis (e.g. they just analyse the entropy). In this article, we will describe our effort to use machine learning to get a more detailed overview of individual parts of an unknown binary file without requiring user intervention.

1 Motivation

Various tools such as *Veles*[3] or *binwalk*[1] can be used to analyze the format of unknown binary files.

1.1 binwalk

Binwalk is usually used for its signature and entropy analysis. Signature analysis is depicted in the following table:

Table 1: Example of binwalk signature analysis output. [1] (`binwalk --signature firmware.bin`)

Decimal	Hex	Description
0	0x0	DLOB firmware header, ...
112	0x70	LZMA compressed data, ...
1310832	0x140070	PackImg section delimiter ...
1310864	0x140090	Squashfs filesystem, ...

Figure 1 shows the plot of entropy for Grandstream GXP2000 firmware file.

1.2 Veles

Veles [3] is interesting for his 3D visualisation of binary files. The intensity of point (x,y,z) in 3D cube represents the probability of byte sequence $[x,y,z]$ in the visualised file. Example of such visualisation is shown in figure 2.

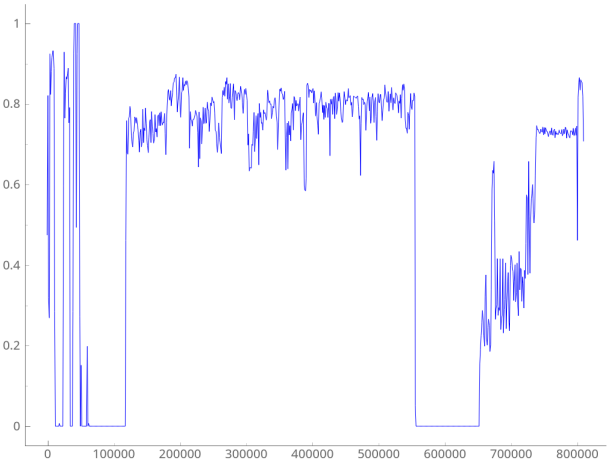

Figure 1: Example of binwalk entropy. [2]

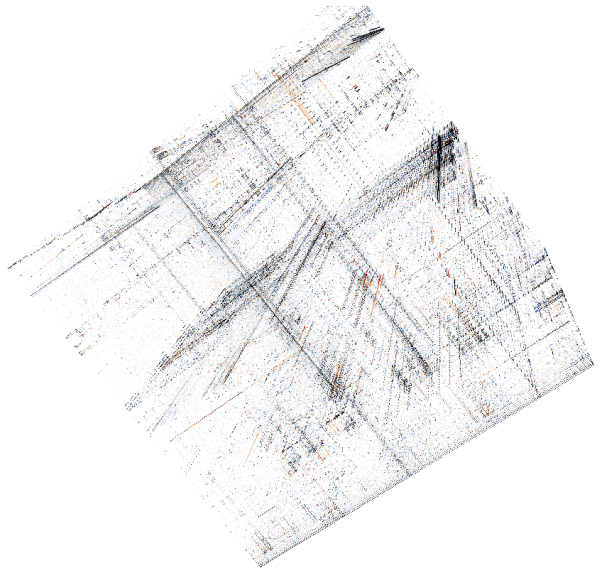

Figure 2: Example of Veles 3D visualisation of binary file consisting of compiled code (x86 32-bit). [4]

2 Our approach

Goal of our approach is to create algorithm that will take binary file and divide it into segments with captions, e.g., code - x86, text - ascii English, image - bmp, noise - zip, ... Our approach is composed of rounds. In every round we use same process for every segment. We are analyzing segments by sections (e.g. 100 bytes long), that can overlap. From now on we will refer to these sections as windows. Using extracting function, we get information about window, and then we feed this information to predicting model. Output of predicting model is prediction about window. If we are analyzing segment with caption raw data, than prediction can be: 0.95 code, 0.01 text, 0.02 image, 0.02 noise. If we are analyzing segment with caption image, prediction can be: 0.5 bmp, 0.5 jpg, 0 gif, 0 png, ... First prediction should be interpreted as : this win-

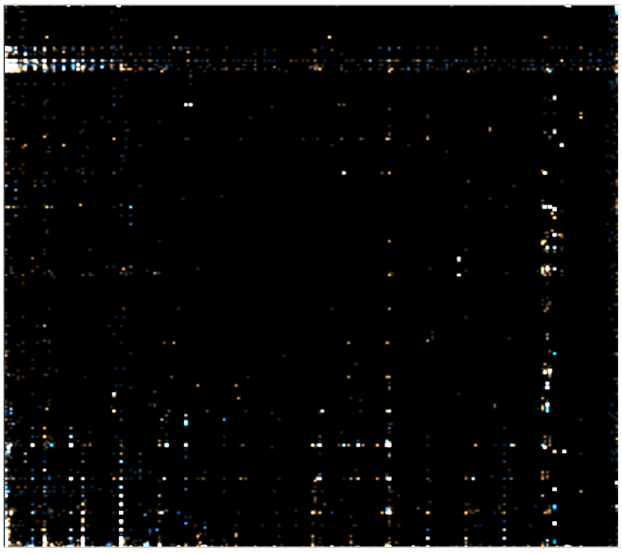

Figure 3: 2D visualization of code

Figure 4: 2D visualization of text

dow is filled with bytes representing code, second prediction should be interpreted as : this window is on the edge of two different things, one half is bmp and one is jpg. After that, using predictions about every window, we divide segment into subsegments. When segment is divided into subsegments, we use smoothing model to smoothen subsegments. It can decide wheather we shall connect some neighbouring subsegments and which caption shall be prevented. We are repeating this process in rounds, until all segments aren't in final state. They get to final state, when they have caption, that we dont know how to further analyze.

2.1 Information extracting function

To determine, whether binary file is code, text or bitmap using *Veles*[3] 2D (or 3D) visualization, is simple. Code 3 is identifiable by its vertical and horizontal lines (result of higher appearance of bytes, that represent frequently used instructions, or are part of longer representations of these instructions). Text 4 is recognizable for its square in area of ASCII letters and some vertical lines in are of space, dot, . . . Bitmap 5 has brighter diagonal, as a result of fact, that adjacent pixels have, most of the time, similar colors. So 2D visualization (from now on referenced as Double BFD) is good enough information and models should be capable to make correct predictions using this information. But its problem is too big size 256*256 and that makes this function almost unusable.

Identity Natural question is, why not just give window as input to model. One problem is, that models have fixed input shape, so model trained like this has to be connected

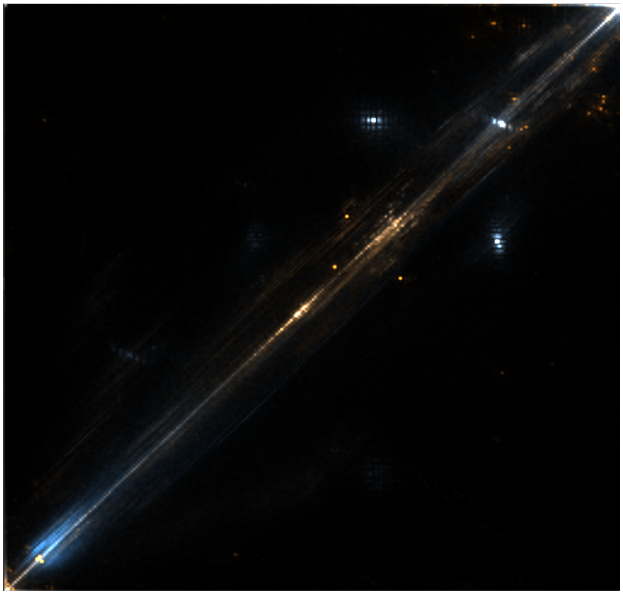

Figure 5: 2D visualization of bitmap

to fixed length of window. But that isn't that big of a problem. Bigger problem is its efficiency. As we can see in table 3, whatever learning model we use it's not even close, compared to other functions. That is bit of contradictory. It has more information than any other function, but it has the worst performance. Maybe problem is in architecture of neural net. Maybe using more complex net, bigger data set, techniques to fight over fitting and bigger computational power we could harvest this information and get best results. Or maybe whatever net we use, there will be just too many traps (local minimums) to get to efficient infor-

mation extraction.

Double BFD Output of this function is an array of length $256 * 256$ and i-th element in this array represents normalized occurence of adjacent bytes A B, where $A = i/256$ and $B = i\%256$.

BFD Output of this function has length 256 and i-th element represents normalized occurence of byte A where $A = i$.

shifted BFD In exploring stages of project, we had idea about Double BFD from Veles visualization, naturally BFD came up, as it was similar and used in problems like this (identifying language, ...). When exploring we decided to try shifted BFD. Shifted BFD means, that instead of counting normal bytes, we count newly created bytes like that: Lets say our window of 100 bytes looks like this : A B C .... First newly created byte X would be computed like this:

$$X = A << 4 + B >> 4$$

Second byte Y like this:

$$Y = B << 4 + C >> 4$$

We had lower expectations of this than BFD, because bytes in BFD naturally represent instructions, ..., so we thought BFD had higher informational value. But tests proved us wrong. When we saw, that shifted bfd works as well, we tried to connect these two, and we got better results. Increasing granularity of this idea we got function Shifts, that is currently working with best results in almost all tests.

Shifts Output of this function has length $256 * 8$. This function is 8 times used BFD with offsets 0, 1, ..., 7. Offset x means, that instead of couning bytes in window, we count shifted bytes by x bits. Shitfed Bfd is example of offset 4. So we create new byte by taking last x bits from first byte and 8-x bits from next byte.

Entropy, adjacency index, ... Some of our functions have only 1 number as output. For example entropy, evaluated on level of bytes, and adjacency index. Adjacency index represents average distance of neighbouring bytes.

Comparison of extracting functions Accuracy is result of many variables (length of window, number of training samples, ...), but most significantly of these 3:

- level (what are we trying to distinguish between : formats of image, language of text, ...)
- model
- extracting function

Table 2: Evaluation of extracting function

Model	1	2	3	4
Identity	0.46	0.50	0.50	0.17
Double BFD	0.72	0.99	0.46	0.10
BFD	0.82	0.98	0.92	0.12
shifted BFD	0.84	0.99	0.86	0.13
Shifts	0.89	0.99	0.96	0.12
Entropy	0.70	0.39	0.49	0.17

As it isn't simple to show tables with 3 variables and in next section we will discuss what learning model to use (and we will decide for neural network), in this section we will compare different extracting functions on different levels using neural net. We will use these levels:

1. code, text, image, noise
2. arm_linux, mips_linux, x86_linux, x86_windows, x86_64_linux, x86_64_windows
3. bmp, gif, jpg, png, png_greyscale
4. bztar, gztar, cypher_aes, random noise, xztar, zip

2.2 Predicting model

Job of predicting model is to take output of extracting function and predict what this window is. First we will use Orange for rough, but simple, evaluation 3 of potential of models. Shown evaluation is on level, where we are going from caption raw data (not yet analyzed data; first round) to one of : text, code, image, noise. After few similar evaluations on different levels we decided to use neural networks. They were most consistent and even most of the times they had the best performance. But we are not disqualifying possibility, that on some level, using some special extracting function, some model can be better than neural net. That's, why adding another learning models, is one of many possible improvements of our program. But for now we will work with neural nets.

2.3 Smoothing model

Let's say that window has size 100 bytes, and we move window by 5 bytes after every evaluation. So every byte is part of 20 windows. To get final prediction values for byte we average those twenty predictions. Then we take maximum of averaged predicted values and that is our prediction. By connecting neighbouring bytes with same prediction we create segments. The whole binary file is divided into these segments and every segment represents 1

Table 3: Evaluation of models using Orange

Model	BFD	Shifts	Identity
Neural Net	0.82	0.89	0.46
kNN	0.84	0.86	0.26
SVM	0.81	0.89	0.37
SGD	0.81	0.87	0.40
logistic Regression	0.74	0.87	0.29
Naive Bayes	0.78	0.87	0.65

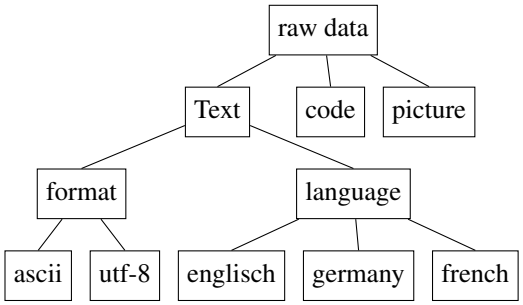

Figure 6: example of tree architecture of future predictions

group, for example text. Problem is, that this way we can get many segments with small length e.g. 5, 10, 15 bytes. Firstly this is problem for further rounds of evaluation and secondary output like that would be useless. So next step is smoothing of segments. Input of smoothing model will be information about actual, previous and next segment and output will be 0, 1 or 2. 0 means connect actual model with previous and remain caption of previous segment. 1 means to not connect anything and 2 means connect actual segment with next and remain caption of next segment. Again we face similar problem. We need to use some function to extract information of segment, and we need right model to evaluate this information and create prediction.

extracting information The most important information of segment is its caption and length. Another option is to average predictions of bytes that are in this segment. Now we are using combinations of this two options.

model As in previous similar situation, we have chosen to use neural networks. Advantage of this situation is, that our input shape is small (varying from 12 to 25). Disadvantage is, that automatic creation of samples takes too long and sometimes its almost impossible. So in these situations, when we have too small training set, we are incapable of training neural net, and we need to use different approach. That is simple model which connects too small segments (e.g. less than window size) with longer neighbor.

3 Implementation

Our program is capable of two things. One — given data sets in tree structure, it is able to train models and evaluate them (model to recognize picture from text was created and has 95 % chance to identify correctly ; model to recognise rar from zip using window with length 100 bytes has chance 50%). Two — given trained models, it can analyze any binary file. It is all done in Python using many libraries, such as *Keras*[7], *SciPy*[8] and *MatPlotLib*[9].

3.1 Training models

Let's say we have data for training in structure as is shown in figure 6. In the end, when program will be ready, it will work like this. First it will take binary file and divide it into segments of Text, code and picture. With segments code and picture it will be finished but with segment Text it will continue. It is really important that T is capital in Text. It means, that program is not supposed to divide segments with caption Text to segments of format and language, but it is supposed to fork and continue on two fronts. For every segment it is supposed to determine its language (and if there are more languages, then divide this segment into sub segments) and format. For every node except leaves and nodes with capital first letter program creates new predicting model and smoothing model. This training is well parametrized, meaning, that you can specify your own keras model for net, you can specify number of training samples, length of window, number of epochs, early stopping, ...

3.2 Analyzing

Program takes whatever binary file you give it and starts with creating one big segment with caption raw data. Then it will recursively divide and recaption segments, until they are not leaves. So it starts with 1 segment captioned raw data. After first round of analysis we will get for example 4 segments Text, code, picture, code. All segments but Text have caption that is a leaf, so they are final. In second round, segment Text will be analyzed, and we will get for example segments: ascii English and ascii french. On this picture 7 we can see output of 1 round of analysis. This binary file is composed of 250 bytes of BMP, 279 jpg, 233 png, 241 gives. Lines are output of predicting model. Background colors are output of smoothing model. As we can see it correctly guessed number of segments and it correctly guessed 3 of four segments. We can see, that precision of borders (produced by predicting model) of segments is mostly +- 10 bytes, only first one is out by 50 bytes. Before using smoothing model, from position 550 to 630 there was segment png_greyscale, from 760 to 820 gif and in area from 820 to 835 there is only few bytes captioned as jpg. Clearly in this situation smoothing

model failed. Png_greyscale connected to jpg instead of png and area from 760 to 835 ruled by gif and jpg was connected to png instead of gif.

4 Evaluation

Some evaluations have been already shown in table 3. But there we had easier job. We did not have to care about borders. We were considering only binaries of one type. Now we need some function to evaluate models. If we will have good evaluating function, than it will be easier to select best net from all possible. Nets with different architecture, functions, optimizers, using different extracting functions, window length and step by which we move window. Different smoothing model. Two most important properties are precision and speed. Speed is the easier one. Precision should consider information like number of correctly identified bytes, percentage of correctly identified segments, number of segments, ... Even thou at this moment we dont have good enough evaluating function, only basic one (correctly guessed bytes / all bytes), we can say it works pretty good as we could saw in previous example 7. Of course, it depends on what we are trying to evaluate. If we are trying to determine whether noise is.rar,.zip, encrypted file, or just random noise, it acts like it is flipping coin. But when we are working on .png, .jpg, .bmp, .gif, .png_greyscale we get precision from 50% to 60%.

5 Conclusion

Most of done work until this moment is connected with creating data sets, exploring working options and creating program, that will take care of all the work behind. Very few of it was connected with trying out more nets, different functions, evaluating functions, ... So even thou some % are not best numbers, there is a lot of possible and now much simpler work to be done to improve that and even thou most of the work was focused on creating tools to train nets and use them.

References

[1] ReFirmLabs: Binwalk
`https://github.com/ReFirmLabs/binwalk`

[2] Firmware hacking blog: Grandstream GXP2000 – binwalk entropy
`https://fwhacking.blogspot.com/2014/05/grandstream-gxp2000-7-binwalk-entropy.html`

[3] CodiSec: Veles
`https://codisec.com/veles`

[4] CodiSec: Binary data visualization
`https://codisec.com/binary-data-visualization`

[5] Christopher Domas: The Future of RE: Dynamic Binary Visualization. RECON 2013.
`https://www.youtube.com/watch?v=C8--cXwuuFQ`

[6] Bioinformatics Laboratory, Faculty of Computer and Information Science, University of Ljubljana, Slovenia: Orange
`https://orange.biolab.si`

[7] Chollet, François and others
`https://github.com/fchollet/keras`

[8] Eric Jones and Travis Oliphant and Pearu Peterson and others
`https://www.scipy.org/`

[9] Hunter, J. D.
`https://matplotlib.org`

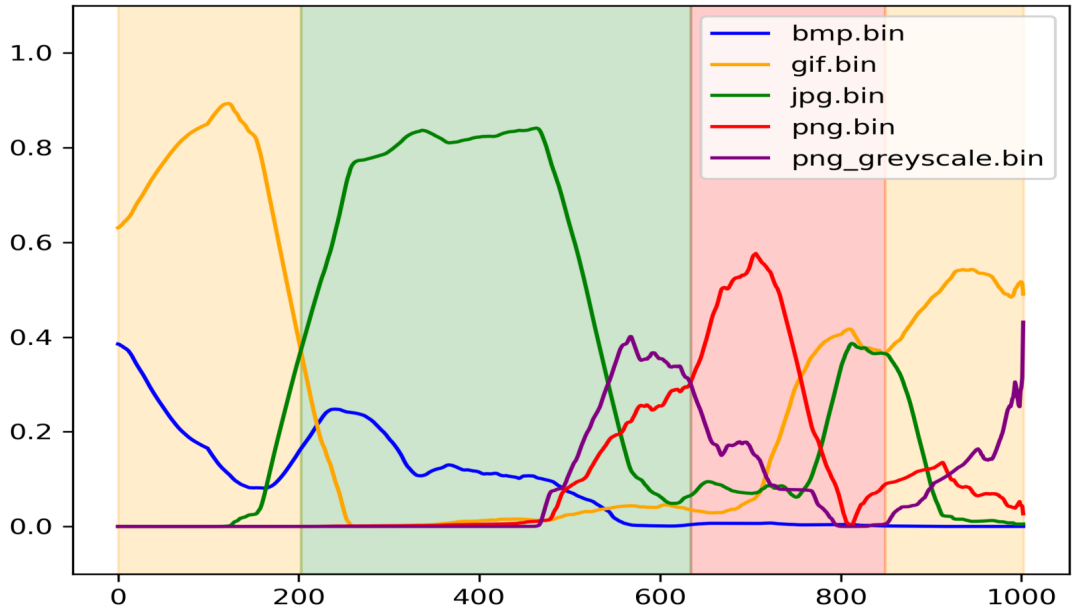

Figure 7: One round of analysis