# Code Type Revealing Using Experiments Framework

Rami Sharon, Sharon.Rami@gmail.com, the Open University,
Ra'anana, Israel.
Ehud Gudes, ehud@cs.bgu.ac.il, Ben-Gurion University,
Beer-Sheva, Israel.

**Abstract.** Identifying the type of a code, whether in a file or byte stream, is a challenge that many software companies are facing. Many applications, security and others, base their behavior on the type of code they receive as an input.

Today's traditional identification methods rely on file extensions, magic numbers, propriety headers and trailers or specific type identifying rules. All these are vulnerable to content tampering and discovering it requires investing long and tedious working hours of professionals. This study is aimed to find a method of identifying the best settings to automatically create type signatures that will effectively overcome the content manipulation problem.

In this paper we lay out a framework for creating type signatures based on byte N-Grams. The framework allows setting various parameters such as N-Gram sizes and windows, selecting statistical tests and defining rules for score calculations. The framework serves as a test lab that allows finding the right parameters to satisfy a predefined threshold of type identification accuracy. We demonstrate the framework using basic settings that achieved an F-Measure success rate of 0.996 on 1400 test files.

**Keywords:** File Type; Content type revealing framework; Code type; Byte N-Gram statistical analysis.

## 1 Introduction

In today's connected environment, most businesses increasingly rely on the Internet as a source of information and a platform for communication and Electronic commerce. One of the main motivating factors driving the increased use of the net is the ability to use technologies based on active content such as Active-x, Java applets, Java Script and Executable files, in order to implement Web Based Applications. The flexibility of these technologies convey great benefits, but at the same time, allow using Web based applications as Malware carriers, capable of harming the organization by damaging its infrastructures, stealing information or performing other illegal activities.

One of the most difficult parts of the attack is to penetrate and infuse the code into the system. Attackers develop new approaches to disguise the true nature of the penetrating file, if by using naive methods such as changing File Extensions or by manipulating file content such as the file header. Recent research in this area was undertaken in order to find efficient ways to identify the true nature of a code in a file, without relying on external characteristics. One of the most common methods applied for this purpose is the analysis of the N-Grams, which are variable sequences of bytes (usually consecutive but not necessarily), present in the file [1, 2, 8].

The main contribution of this paper is a new Framework, called CTR – Code Type Revealing, that enable in a convenient manner the finding of the most efficient parameters (such as N-Gram size, Statistical classifier and other qualifiers), for the creation of characteristic signatures for different file types. The CTR Framework serves as an infrastructure for automatic creation of a signature for every file type, based on training data constructed from files of that type. Moreover, the CTR Framework can scan unknown files and determine their type based on that signature. Experiments that were made using the CTR Framework, presented very good results with 1394 files out of 1400 that were correctly identified. This is when taking into consideration closely related files, such as EXE and DLL, as one type. The F-Measure value, based on these tests, was 0.996.

The rest of this paper is structured as follows. In Section 2 we discuss related work, in section 3 we describe the CTR framework and in Section 4 the results of the evaluation. Section 5 is the summary.

## 2 Related Work

Using statistical measures on the content of a file has been investigated as a text classification technique, and later on was used to explore new methods for type classification and malware detection. Some of these techniques are based on statistical measurements and analysis of N-Gram distribution. McDaniel and Heydari [1] focus their early paper on automatic methods of creating file signatures, called file type “fingerprints”. They suggest three algorithms based on 1-gram frequency distribution. The first algorithm created a fingerprint using training data of files from the same type. The second algorithm used similar methods but based the fingerprint on cross correlation between byte pairs. The third one simply tested the file header and trailer for repeating patterns and their correlation strength. The first algorithm achieved a success rate of 27.5%, which is not far from a random guess. The second one did a better job (45.83%) but performed much slower. The third algorithm did much better (95.83%) but is much more vulnerable to content manipulation. Wei-Jen Li et al. [2] continued this approach and claimed reaching better results by refining the fingerprints using a set of centroids, which were carefully selected by using clustering methods to find the minimum set that provides good enough performance. Overall, the results were improved compared to [1]. Karresand and Shahmehri [3] continued with developing a similar method using Centroids with the goal of identifying file type based on binary data fragments. The Centroids were based on the Mean and Standard Deviation of byte frequency distribution. Later they extended their work, introducing the interesting concept of ‘Rate of Change’ between consecutive bytes [4].The calculation of the distance to the Centroid were made using two methods, 1-norm (also known as Manhattan Distance) and the frequency distribution of the Rate of Change of the fragment to be identified. The results were not conclusive. JPEG files achieved the best results, gaining more than 86.8%, but with 20% of False Positives. For other types the results were much worse. For instance, executable file gained 45% - 85% with up to 35% of FP.

Kolter et al. [5] suggested the use of fixed length N-Grams to identify malicious content in executable files. They translated binary content to textual representation and extracted 4-Grams of known malicious and benign binaries, resulting in about 255M unique 4-Grams. A number of classification methods have been used, such as Naive Bayes, Support Vector Machines (SVM) and Decision Trees, which yielded the best results. Dash et al. [6] Continued this approach and introduced the use of variable length N-Grams to identify malicious code. They claimed better performance over fixed size N-Grams due to essential data loss of significant longer N-Grams. Indeed, the results indicate less errors of type I (FP). Irfan et al. [7] suggest two approaches for type identification. First approach uses the cosine similarity for byte frequency comparison; the second is divide-and-conquer approach to group similar files, based on repeating byte patterns, regardless of their type. The results were not conclusive and the later approach improved the classification accuracy of some types, while getting worse results on others, comparing to methods not using divide-and-conquer approach. Moskovitch et al. [8] deal with methods of identifying malicious code based on concepts taken from text categorization. They introduce the class imbalance problem, which basically arises when the classes (in this case, benign and malicious code classes) are not balanced so that most inspected objects belong to one larger class, while other classes remain much smaller. This may result, in some extreme cases, in mistakenly labeling all data as member of the larger class. The results showed that when selecting about 10% of malicious code in the test data, which closely represent real life according to the authors, they achieved over 95% accuracy when using 16.7% of malicious code in the training data. The overall conclusion was that 10% - 40% of malicious code in the training data will provide optimal results in a true life distribution.

As was shown in the cited papers, different file types may require different parameters of analyzing the file content, and the framework discussed next is based on this approach.

# 3 Identifying File Type by its Content

The challenge that we are facing, is to find an automatic procedure to create type identifiers, called signatures, which will be accurate enough and resistant to content manipulation up to some degree.

We can deduce, based on previous work, that there is some correlation between the type of a file and its content. To be more precise, we can find a direct link, very strong in some cases and weaker in others, between repeating N-Grams and the type. The problem is that the general N-grams method has several parameters (e.g. the size N) and they are not equivalent for the different file types.

The aim of this paper is to describe the CRT framework, acting as an experiments lab, which will allow performing experiments on automatic type signatures creation, while changing settings such as N-Gram size, Window size, Statistical measures, Score calculation rules and others. In this section we describe the idea and the architecture of the framework. In the following section we present an implementation of the idea, which will serve as a proof of concept, and the results of our tests.

## 3.1 Implementation

The Framework makes use of the N-Gram Statistics Package (NSP) [9]. The package, which was developed by Pedersen et al., is a suite of Perl utilities, aimed to analyze N-Grams in text files. It collects information on the appearance of N-Grams and allows running association tests such as Fisher's exact test and log likelihood ratio on the collected data. Since the Framework goal is to analyze files of any type, binary or text, a preliminary step was added to translate files content to their hexadecimal textual representation.

## 3.2 The CTR Framework Architecture

The Framework consist of 2 paths, Signature creating and File test paths, which are schematically presented in Figure 1. The right side flow describes the Signature creation path based on training data files, while the left one describes the test flow. Both paths use the same preparation procedure, which is the "Count N-grams and Calculate statistics" step (see 3.3). We next describe each step in the two paths.

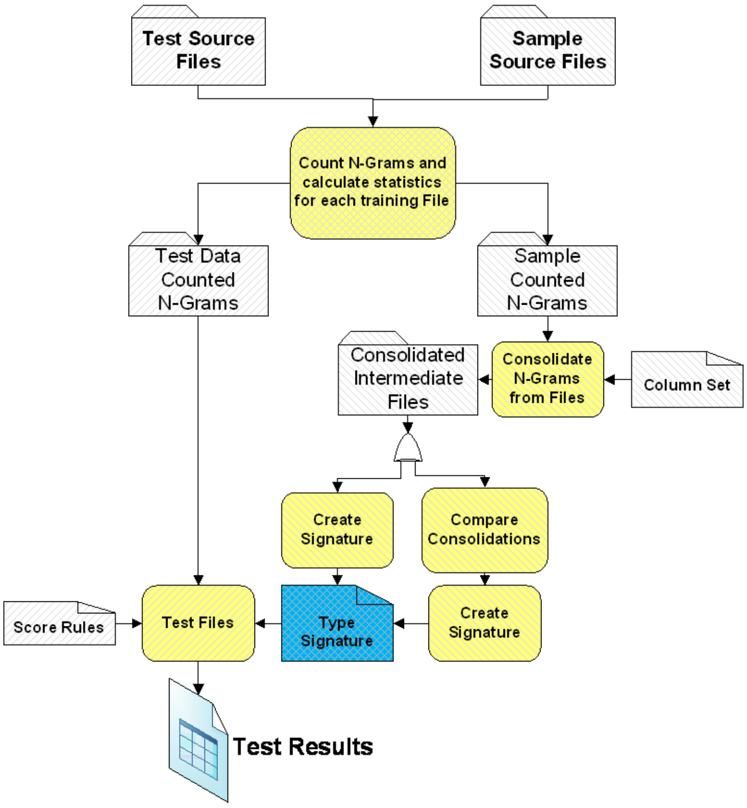
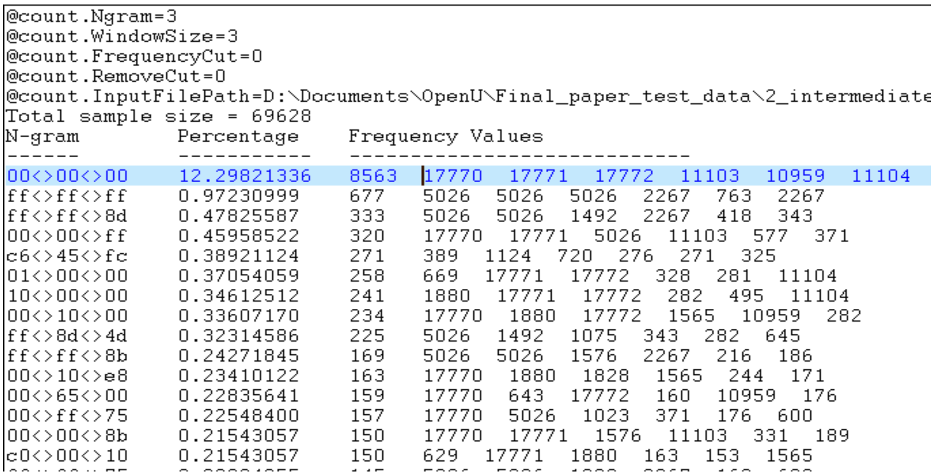


Figure 1: Framework Workflow

## 3.3 Count N-Grams and calculate statistics

This step is common to both paths. The input for the step is a folder containing files. In case of signature creation path, the files will be training data, representing a type. In the case of the test path, the folder contains files of unknown type. The step accepts, as an input, the following parameters:

1. N-Gram Size. The size of N-Grams, which will be collected.
2. Window Size. Allows extracting non-contiguous N-Grams. For instance, in case of N-Gram with size 2 and Window with size 3 for bytes XYZ, the extracted N-Grams will be XY, XZ, YZ. Window size should be greater or equal to the N-Gram size.
3. N-Gram Threshold. Allows settings the minimum support of the N-Grams that will be counted.
4. Statistical measures, e.g. Fisher's exact test, Log-likelihood, Chi-squared test etc.



```
@count.Ngram=3
@count.WindowSize=3
@count.FrequencyCut=0
@count.RemoveCut=0
@count.InputFilePath=D:\Documents\OpenU\Final_paper_test_data\2_intermediate
Total sample size = 69628
N-gram         Percentage     Frequency Values
------         -----------    ----------------------------
00<>00<>00     12.29821336    8563   17770   17771   17772   11103   10959   11104
ff<>ff<>ff     0.97230999     677    5026   5026   5026   2267   763   2267
ff<>ff<>8d     0.47825587     333    5026   5026   1492   2267   418   343
00<>00<>ff     0.45958522     320    17770   17771   5026   11103   577   371
c6<>45<>fc     0.38921124     271    389   1124   720   276   271   325
01<>00<>00     0.37054059     258    669   17771   17772   328   281   11104
10<>00<>00     0.34612512     241    1880   17771   17772   282   495   11104
00<>10<>00     0.33607170     234    17770   1880   17772   1565   10959   282
ff<>8d<>4d     0.32314586     225    5026   1492   1075   343   282   645
ff<>ff<>8b     0.24271845     169    5026   5026   1576   2267   216   186
00<>10<>e8     0.23410122     163    17770   1880   1828   1565   244   171
00<>65<>00     0.22835641     159    17770   643   17772   160   10959   176
00<>ff<>75     0.22548400     157    17770   5026   1023   371   176   600
00<>00<>8b     0.21543057     150    17770   17771   1576   11103   331   189
c0<>00<>10     0.21543057     150    629   17771   1880   163   153   1565
```

Figure 2: N-Gram Count file

The result of the step is a set of files, one for each source file, containing a list of extracted N-Grams, together with the collected statistics on each.

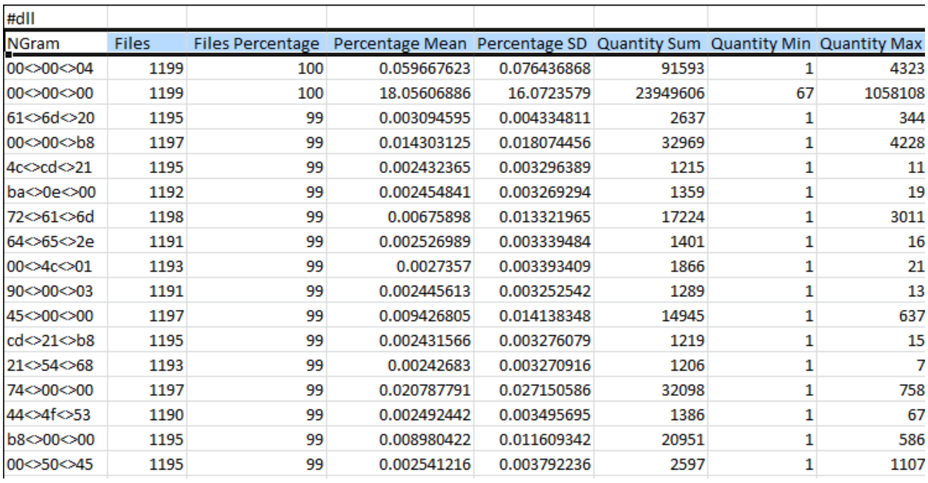
A sample output is presented in Figure 2. The first few lines describe the settings, such as N-Gram size (3-Gram, in this case) and required statistics measures. Following are column headers and the data. Each data line presents an N-Gram, followed by the results for the statistic calculation (percentage appearance in the file, in this case) and quantity information on the number of N-Grams that were found in the file. For each N-Gram that was found, the first frequency value states the number of occurrences that the N-Gram appears in the file. The following columns list the number of times that each subset of the N-Gram appeared in its current position. For instance, inspecting the marked line in Figure 2 reveals that ‘000000’ appeared 8563 times in the file, ‘00’ appeared 17770 times on the left hand side, 17771 times in the middle and 17772 times on the right hand side. Also, ‘0000’ appeared 11103 times on the left hand side, and so on.

## 3.4 Create Signature path

### 3.4.1 Consolidate N-Grams from files

In this step, result files from the Count N-Grams step are collected and consolidated into one summary file per type. The step accepts, as an input, a file describing the columns to be collected from the N-Grams count files, together with some statistical measures that should be made on them. These measures can be Minimum and Maximum value, Standard Deviation etc. The step accepts, the following parameters:

1. Files Threshold. This value set the threshold, in percentage, of files that should contain an N-Gram, in order for it to be counted as part of the signature.
2. Type Name. This value assigns the type to a summary file.



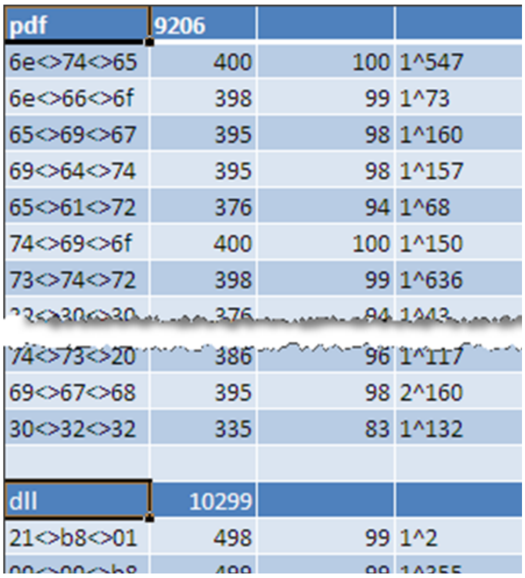
| #dll | | | | | | | |
|---|---|---|---|---|---|---|---|
| NGram | Files | Files Percentage | Percentage Mean | Percentage SD | Quantity Sum | Quantity Min | Quantity Max |
| 00<>00<>04 | 1199 | 100 | 0.059667623 | 0.076436868 | 91593 | 1 | 4323 |
| 00<>00<>00 | 1199 | 100 | 18.05606886 | 16.0723579 | 23949606 | 67 | 1058108 |
| 61<>6d<>20 | 1195 | 99 | 0.003094595 | 0.004334811 | 2637 | 1 | 344 |
| 00<>00<>b8 | 1197 | 99 | 0.014303125 | 0.018074456 | 32969 | 1 | 4228 |
| 4c<>cd<>21 | 1195 | 99 | 0.002432365 | 0.003296389 | 1215 | 1 | 11 |
| ba<>0e<>00 | 1192 | 99 | 0.002454841 | 0.003269294 | 1359 | 1 | 19 |
| 72<>61<>6d | 1198 | 99 | 0.00675898 | 0.013321965 | 17224 | 1 | 3011 |
| 64<>65<>2e | 1191 | 99 | 0.002526989 | 0.003339484 | 1401 | 1 | 16 |
| 00<>4c<>01 | 1193 | 99 | 0.0027357 | 0.003393409 | 1866 | 1 | 21 |
| 90<>00<>03 | 1191 | 99 | 0.002445613 | 0.003252542 | 1289 | 1 | 13 |
| 45<>00<>00 | 1197 | 99 | 0.009426805 | 0.014138348 | 14945 | 1 | 637 |
| cd<>21<>b8 | 1195 | 99 | 0.002431566 | 0.003276079 | 1219 | 1 | 15 |
| 21<>54<>68 | 1193 | 99 | 0.00242683 | 0.003270916 | 1206 | 1 | 7 |
| 74<>00<>00 | 1197 | 99 | 0.020787791 | 0.027150586 | 32098 | 1 | 758 |
| 44<>4f<>53 | 1190 | 99 | 0.002492442 | 0.003495695 | 1386 | 1 | 67 |
| b8<>00<>00 | 1195 | 99 | 0.008980422 | 0.011609342 | 20951 | 1 | 586 |
| 00<>50<>45 | 1195 | 99 | 0.002541216 | 0.003792236 | 2597 | 1 | 1107 |

**Figure 3: Type Consolidation Summary File**

A sample output is presented in Figure 3. The type is specified on the first line. Lines, following the column headings, list N-Grams, together with their relevant collected statistics. The second column presents the number of files, the N-Gram was found in and the third column presents this value in percentage of files. Next values present the collected columns for the Count N-Gram step, with statistical measures. For instance, the fourth column presents the Mean of the percentage value, taken from the second column in Figure 2, and the fifth column presents the Standard Deviation of this value.

### 3.4.2 Create Signature

A Signature can be constructed using two methods. One uses consolidation files as a direct input for the step. The second one allows comparing consolidation files, in order to eliminate N-Grams that repeat in more types then a predefined threshold. Both methods accept a list of data columns to be collected from consolidation files. In case of choosing the ‘Compare Consolidation’ method, a threshold value should be added. For instance, if N-Gram 00<>00<>00 appears in 5 different types and the threshold is 4, it will not be included in the Signature. This feature was designed to get more unique type signatures.



| pdf | 9206 | | |
|---|---|---|---|
| 6e<>74<>65 | 400 | 100 | 1^547 |
| 6e<>66<>6f | 398 | 99 | 1^73 |
| 65<>69<>67 | 395 | 98 | 1^160 |
| 69<>64<>74 | 395 | 98 | 1^157 |
| 65<>61<>72 | 376 | 94 | 1^68 |
| 74<>69<>6f | 400 | 100 | 1^150 |
| 73<>74<>72 | 398 | 99 | 1^636 |
| 23<>30<>30 | 376 | 94 | 1^43 |
| 74<>73<>20 | 386 | 96 | 1^117 |
| 69<>67<>68 | 395 | 98 | 2^160 |
| 30<>32<>32 | 335 | 83 | 1^132 |
| | | | |
| dll | 10299 | | |
| 21<>b8<>01 | 498 | 99 | 1^2 |
| 00<>00<>b8 | 499 | 99 | 1^255 |

Figure 4: Signature Sample File

A sample of a signature file is shown in Figure 4. It contains the signature information for all types. The first line on each section states the type on the first column and the maximum available score for it on the second column. Following lines list the N-Grams, together with their collected information. This information includes, in the second and third columns, the number of files and percentage that the N-Gram appears in. The fourth column contain the statistics collected from the consolidated files (see 3.4.1), separated by a '^' sign. In Figure 4, the min and max N-Grams appearance was collected. In the first row, for instance, the N-Gram 6e<>74<>65 appeared in each training file between 1 and 547 times.

## 3.5 Test Files path

In this path, we use the type signatures to test files and estimate their type. Tested files are processed through the count N-Grams step (see 3.3), which was also applied on training data. It is important to emphasize that, in order to get reliable results, the Count N-Grams step should be set with the same setting as used in the signature path.

The step accepts, as an input, the list of files to be tested, the signature file and a Score rules file.

### 3.5.1 Score Rules file

The score rules file is assembled from Rules, containing sequential Conditions. Each Rule applies to a specific file type. A Condition, within the rule, determines the score that a found N-Gram will contribute to the final score of a file, in case that the condition will be found to be true. Conditions are ordered and the first Condition that will be found as true will be applied.

Following is a sample set of two conditions in a rule:

```
For a specific N-Gram
  Condition 1:
    If the N-Gram was found X times in the file, where
    X >= 'Min times in the signature' and X <= 'Max times
    in the signature'
      Grant it the score Y.
  Condition 2:
    If the N-Gram was found X times in the file, where
    X > 'Max times in the signature'
      Grant it the score Z.
```

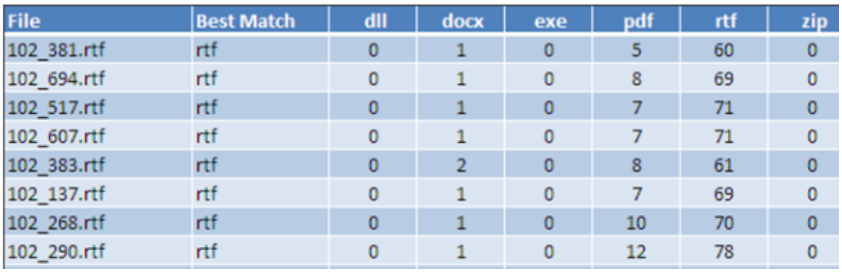
According to these sample conditions, if an N-Gram was found in a file between the min and max times, defined by the signature, it will add Y to the total score, otherwise, if it will be found more than the max, it will add Z to the total score. These Rules and the order between the conditions, which is important, can be created by setting preliminary Rules, creating a signature and using it on test data. Then, repeating this step again and again, using different settings, until the results accuracy is acceptable. Section 3.6 discusses this issue further. Since an N-Gram may be missing from some type training files because of the file threshold, a factor of N-Gram contribution weight is taken into consideration. It is calculated based on the percentage of files, the N-Gram was found in. The following algorithm describes the contribution of a specific N-Gram to the score:

```
For each $r \in \mathrm{Rules}$
  If $\mathrm{r.Type}$ matches current type
    For each $c \in \mathrm{r.Conditions}$
      If $c$ is true
        Add S = $(\mathrm{c.Score} * \mathrm{NGram\ weight})$ to the file
            type total score
        Break
```

S is the score that this N-Gram contributes to the total score of the file for the type. The total score that will be granted to a file for the type is based on the formula:

$$S_{Total} = \frac{\sum S_i W_i}{\sum W_i}$$

Where $S_i$ is the score for N-Gram i and $W_i$ is its weight.



| File | Best Match | dll | docx | exe | pdf | rtf | zip |
|---|---|---|---|---|---|---|---|
| 102_381.rtf | rtf | 0 | 1 | 0 | 5 | 60 | 0 |
| 102_694.rtf | rtf | 0 | 1 | 0 | 8 | 69 | 0 |
| 102_517.rtf | rtf | 0 | 1 | 0 | 7 | 71 | 0 |
| 102_607.rtf | rtf | 0 | 1 | 0 | 7 | 71 | 0 |
| 102_383.rtf | rtf | 0 | 2 | 0 | 8 | 61 | 0 |
| 102_137.rtf | rtf | 0 | 1 | 0 | 7 | 69 | 0 |
| 102_268.rtf | rtf | 0 | 1 | 0 | 10 | 70 | 0 |
| 102_290.rtf | rtf | 0 | 1 | 0 | 12 | 78 | 0 |

**Figure 5: Test Sample Output file**

A test output sample file is shown in Figure 5. The first column list all test files. Second one presents the best guess, meaning, types that received the best score. Following are one column per type, and the score that the file received for it.

## 3.6 Methodology

In this section we describe the methodology that can be used by the researcher, when creating a new type signature. The methodology can also be adopted by an automatic process. Following are the methodology steps. These steps should be processed for each type separately.

1. Set an accuracy threshold, which serves as an indication for success, when achieved.
2. Choose training and test files from reliable sources. All training set files must be of the same type, which the signature is created for. More files in the training set will result in a smaller and more accurate signature since each additional file in the set may lack of some N-Grams, which otherwise, could be part of the signature. This means that the removed N-Gram could negatively affected the accuracy of the signature. Test set should contain files from different types with existing signatures, in order to identify FP and FN errors.
3. For each set of parameters, run the following steps:
   (a) Choose settings, not tested before, such as N-Gram size, Window size, Statistical measures and Score Rules. For better performance, start from easy settings. e.g., start from smaller N value for the N-Gram size, choose simple statistical measures (min and max for example) before using complex ones etc.
   (b) Create a signature.
   (c) Run Tests.

(d) If results exceeds success threshold then stop.

The methodology is very simple and should be justified. Generally, if we take all possible options for N-Gram size, window size, possible score rules syntax, parameters of score rules and statistical measures and run an exhaustive test, it will take exponential time. One can use a Genetic algorithm for generating a close to best configuration, but as can be seen from Sections 4.2 and 4.3 below, the above simple methodology gave very good results. It's quite easy to replace it with a genetic algorithm or another learning method.

## 4 Experiments

We performed the Experiments with very basic tests settings. The goal was to demonstrate the strength of the process, while keeping it simple. All experiments were done on Windows, although the framework does not enforce specific OS. Performance was not taken into consideration, since the framework is not considered to be part of a runtime or production environment, but further development of the test file path, using C# on .NET environment, yielded an average of 70ms scan time for a 0.5MB size file on an i5 core Desktop, which is a quite small overhead.

### 4.1 Data

Test and training files of 6 different types were collected from a few sources, such as a repository of tested benign files of known types, files resided on a well-protected PC and Google. In order to reduce the chance of random guesses, we collected a large set, containing a total of 3920 files. Collected files distribution is listed in table 1.

| **File Type** | **Training Files** | **Test Files** | **Total Files** |
|---|---|---|---|
| EXE | 500 | 200 | **700** |
| DLL | 500 | 699 | **1199** |
| DOCX | 500 | 182 | **682** |
| PDF | 400 | 19 | **419** |
| RTF | 500 | 255 | **755** |
| ZIP | 120 | 45 | **165** |
| Total | 2520 | 1400 | **3920** |

**Table 1: Files Used for Experiments**

### 4.2 Tests Settings

For the first experiment we used 3-Grams, with the same window size. No statistical measures were collected, except for N-Grams Minimum and Maximum distribution information. Although the Framework allows different settings for different file types, the experiment reported here used the same settings for all types for simplicity. The following simple score rule was applied to all types:

```
If an N-Gram was found then
  If it appeared between Min and Max Times
    Grant the score 100
  Else
    Grant the score 80.
Else
  Grant the score 0.
```

In order to maintain strong signatures, we set the training files percentage threshold, which defines the min percentage of files that an N-Gram should appear in (see 3.4.1, 1), to a value that will assure at least 100 N-Grams in a signature. We succeeded in keeping this rule of thumb for all types except zip files, which seems to have less N-Grams in common due to high entropy of bytes sequences in compressed files. Table 2 lists the threshold for different types.

| Type | Threshold |
|---|---|
| DLL | **99%** |
| PDF | **79%** |
| EXE | **94%** |
| ZIP | **55%** |
| RTF | **84%** |
| DOCX | **100%** |

**Table 2: Files Percentage Threshold**

### 4.3 Results for the first experiment

Tests were two-folded. The first part was a K-Fold Cross Validation with K = 5. Training files for each type were split into 5 subsets. For each subset, a signature was made from the other 4 and the subset was used as a test set. This step was done on all types except zip, due to the small number of available zip files. DOCX files received the best scores, all in the range between 95 and 100 out of 100, while 92% of the DOCX files received the perfect score 100.

RTF files received the worst results. In one subset case, about 33% of the files scored in the range 40 and 49 out of 100 and 17% scored in the range between 50 and 59. But nevertheless, in the actual tests, RTF files detection rate reached 100% success with no FP and FN (see Table 3). This can be explained if we understand that the score does not stand by itself and should be compared to other types score for the tested file, i.e., RTF files received low scores for the type RTF, but much lower for other types, so they were recognized as RTF.

In the second part, file scanning was made on test files, based on signatures that were created using all training data. A total of 1400 test files from all 6 different types were used. 1293 files, which are about 92%, were accurately recognized.

When closely inspecting the remaining 8% files, an interesting picture is raised as shown in Figure 6.

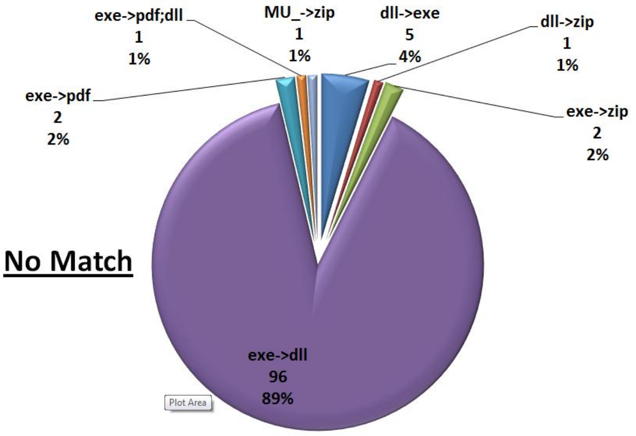


Figure 6: No Match File and their distribution

A total of 97 files, which are about 90% of non-matched files, are EXE files that were mistakenly identified as DLL files (one file was also identified as PDF). 5 DLL files (4%) were identified as EXE files. Also, MU_ file, which is a compressed file, was recognized as ZIP. Since DLL and EXE files are executable files with very similar characteristics, we can consider them as executable files. This changes the picture entirely and leaves us with 6 non-recognized files out of 1440, which is about 0.4% error. We also calculated the F-Measure values for the results in order to get a sense of the Precision and Recall. The results are presented in Table 3.

| Type | Files | TP | FP | FN | Precision | Recall | F-Measure |
|---|---|---|---|---|---|---|---|
| DOCX | 182 | 182 | 0 | 0 | 1 | 1 | 1 |
| RTF | 255 | 255 | 0 | 0 | 1 | 1 | 1 |
| PDF | 19 | 19 | 3 | 0 | 0.864 | 1 | 0.927 |
| DLL | 699 | 693 | 97 | 6 | 0.877 | 0.991 | 0.931 |
| EXE | 200 | 99 | 5 | 101 | 0.952 | 0.495 | 0.651 |
| ZIP | 45 | 45 | 3 | 0 | 0.938 | 1 | 0.968 |
| **Total** | 1400 | 1293 | 108 | 107 | 0.923 | 0.924 | 0.923 |
| | | | | | | | |
| Exec. | 899 | 894 | 0 | 6 | 1 | 0.993 | 0.997 |
| **Total Exec.** | 1400 | 1394 | 6 | 6 | 0.996 | 0.996 | 0.996 |

Table 3: Results Precision, Recall and F-Measure

As can be seen, DOCX and RTF achieved the best results, scoring perfect F-Measure value of 1. PDF, DLL and ZIP files also did well, scoring 0.931-0.968. EXE files gained a poor Recall value of 0.651, but when taking DLL and EXE files as Executable files, the gained Precision is 1 and Recall is 0.993, resulting in an F-Measure value of 0.997.

Overall, in the latter case, the total F-Measure for all 1400 files is 0.996.

## 4.4 Improving accuracy

As can be seen in the previous section, when isolating the EXE type, the settings of the first experiment gained poor results, with an F-Measure value of 0.651. Since the result revealed that, in most miss-identifications, EXE files were recognized as DLL and vice versa, we performed two additional experiments, one using 4-Gram and the another using 5-Gram, only for the DLL and EXE types. N-Gram sizes for the rest of the types were left unchanged (3). Also, other settings for the experiment were left unchanged. The precision, recall and F-Measure for the EXE type for the different N-Gram sizes are presented in Table 4.

| N-Gram Size | Files | TP | FP | FN | Precision | Recall | F-Measure |
|---|---|---|---|---|---|---|---|
| 3 | 200 | 99 | 5 | 101 | 0.952 | 0.495 | 0.651 |
| 4 | 200 | 108 | 7 | 92 | 0.939 | 0.54 | 0.686 |
| 5 | 200 | 132 | 8 | 68 | 0.943 | 0.66 | 0.777 |

Table 4: EXE Precision, Recall and F-Measure for different N-Gram sizes

As can be seen clearly, with the increasing of the N-Gram size, the accuracy of the Framework improved when identifying EXE files. This experiment demonstrates the strength of the Framework in finding the best settings for signature creation.

From these experiments we can identify few factors that contribute to the results accuracy:

- The increasing accuracy when improving the score rules.
- Grouping of file types with a similar structure into categories. As shown, categorizing may contribute for increased accuracy.
- Threshold values, which set the ground rules to determine if an N-Gram will be counted for the signature.

### 4.5 Identifying tampered files

One of the challenges, the Framework is facing, is the ability to correctly identify tampered files. For instance, in many file types, the first few bytes (magic numbers) are used to identify the file type. These can be easily manipulated to obfuscate the type of the file. In order to test the ability of the Framework to overcome this challenge, we performed another experiment.

We randomly picked 24 files, which were correctly identified in the first experiment. The file list contained 4 files from each of the 6 different types. The first 10 bytes of each file were set to zero (0). An attempt to open these files, using the relevant application completed with a failure, as expected. The 4-Gram experiment settings were used for the experiment.

All 24 files were correctly identified by the Framework. This result demonstrates the strength of the method used by the Framework to overcome common cases of file content tampering. More complex forms of content tampering, which affect the whole content of the file, will be dealt with in a future work. Furthermore, the tests were performed on six file types and it is possible that the identification accuracy may be reduced when adding more type signatures. This may be overcome by removing mutual N-Grams of many different type signatures and will be investigated in the future.

## 5 Conclusions

This paper presents the CTR framework, which is a general framework for revealing the true type of various files. Using very simple settings, the framework demonstrated promising results, successfully identifying 1394 out of 1400 files and achieving F-Measure value of 0.996, when taking EXE and DLL files into account as executable files. There is no need of prior knowledge or manual process of finding patterns in file structures. The process can be done automatically without any human intervention.

Not all tested types achieved the same signature strength. While DOCX files have many repetitive patterns, strong structure characteristics and similarities, ZIP files, for example, demonstrated very weak signature with small number of descriptive N-Grams. Investigation of additional file types using the CTR framework is planned in future work.

File content tampering is still an option, but it is much harder since the signature usually covers large amount of N-Grams and the scan is made on full content, or at least large portion of the file. This was clearly shown by the results in 4.5.

In future work we intend to use the framework as a basis for identifying anomalies in files, in order to mark them as suspicious or benign. Also, we will explore new directions in text classification, trying to identify content language or identify data leakage.

## References

[1] M. McDaniel and M. H. Heydari, "Content Based File Type Detection Algorithms," in *Proceedings for the 36th Hawaii International Conference on System Sciences*, 2002.

[2] W.-J. Li, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying File Types by n-gram Analysis," in 2005 IEEE Workshop on Information Assurance, West Point, NY, 2005.

[3] M. Karresand and N. Shahmehri, "Oscar – file type identification of binary data in disk clusters and ram pages," in *Proceedings of IFIP International Information Security Conference: Security and Privacy in Dynamic*, 2006, pp. 413-424.

[4] M. Karresand and N. Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," in *Proceedings of the 2006 IEEE Workshop on Information Assurance United States Military Academy*, West Point, NY, 2006.

[5] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," in *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[6] K. S. Dash, S. R. K. Dubba, and K. A. Pujari, "New Malicious Code Detection Using Variable Length n-grams," in *Algorithms, Architectures and Information Systems Security*. World Scientific, 2008, ch. 14, pp. 307-323.

[7] A. Irfan, L. Kyung, S. Hyunjung, and H. ManPyo, "Content-Based File-type Identification Using Cosine Similarity and a Divide-and-Conquer Approach," *IETE Technical Review*, vol. 27, no. 4, Jul. 2010.

[8] R. Moskovitch, et al., "Unknown malcode detection and the imbalance problem," *Journal in Computer Virology*, vol. 5, no. 4, pp. 295-308, 2009.

[9] T. Pedersen, S. Banerjee, A. Purandare, B. T. McInnes, and Y. Liu. (2009) NSP - Ngram Statistics Package.