Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

On the Automatic Detection of Embedded Malicious Binary Code using Signal Processing Techniques
Project Report

Benjamin Jochheim

October 17, 2012

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Contents

1 Introduction

This project report examines a lightweight method to detect binary instruction code, possibly embedded within regular data, as suggested in [1] and demonstrated at SIGCOMM 2012. We will present details of this scheme and explore opportunities for a fairly reliable detection method that can be implemented with a low computational overhead.

1.1 Motivation

Today portable communication devices have become common and can host a wide range of applications. Many of those applications handle personal data that must be well protected from unauthorized access. The recently increasing attacks on mobiles [2] show, that there is a great demand for effective protection.

Protection of a mobile device is especially hard because of the mobility and its many communication interfaces that exhibit a direct access from the „outside world“. In many mobility scenarios, the users depend on the connection to those untrusted networks and access business data via the Internet.

Each new interface brings its own communication stack that can be prone to attacks thus generates possible attack vectors. For securing connections, the mobiles can use encryption functions, but the devices must cope with limits in terms of memory-space and CPU-power. Attackers may have much more computational power than the mobiles. Finally, the users of mobiles are in most cases no computer experts. The users expect to be protected in a non obtrusive way that shields them from unnecessary details.

1.1.1 Why are mobile Devices attacked?

Valuable information stored by users, such as calendars and business contacts, is one reason for the interest of attackers to seek ways to access the data. Besides theft of valuable data, mobile phones expose a greater incentive to attackers by constituting an indirect access to a bank account [2]. If an attacker can trick a phone to use services such as premium rate texting or voice-calls, a money transfer to a third party can be initiated. Mobiles are also used to access electronic wallets to conduct micropayment transactions. To perform micropayment transactions the mobiles often use near field communication (NFC) [3]. NFC is a technology providing short-range wireless communication channels for mobile devices that is prone to attacks due to the shared medium [4].

Like any computer, a mobile device needs regular software updates to fix disclosing software bugs. Customers of mobiles depend on the responsibility of vendors to supply patches in short term. For a mobile the time from introduction to the market until new models become available, and thus the support for older models cease to exist, are very short. Those short cycles lead to a situation, where devices are no longer supplied with security updates by the vendors, but are still actively used by customers. As the software is no longer updated, chances begin to grow, that there are undiscovered security holes.

There is always a gap between the detection of a security related software bug and the distribution of the patch. Those unfixed bugs can be used to craft exploits. The so called *Zero-day-exploits* are a great threat to devices with huge software libraries and with constantly extended functionality such as the current mobiles.

Mobile devices can often be *identified easily* within the Internet, because there are IP-ranges that are exclusively assigned to mobile devices by the telecommunication-providers. Attacks are common in those networks and have been analyzed using mobile honeypots [5]. In some installations of IP-Networks for mobiles, the networks use a basic protection implemented by the telecommunication-providers. The protection is done either by using NAT or a firewall to protect the mobiles from direct access from the Internet. This leads to a false sense of security. Devices can still be reached with the help of the user, e.g., by opening an email containing malicious code. If one device behind a firewall is successfully infected, the attackers have a starting point for further attacks behind the first security barrier. Once this barrier has been overcome, network based attacks can spread very quickly within the IP-ranges of the mobiles. We have shown that mobile phones have a potentially high risk of being attacked and misused. In the next section we will take a look at proposed countermeasures.

1.1.2 Countermeasures against Attacks

Many solutions for the different threats to mobiles have been proposed. Some techniques are used that have been established for desktop computers. Among those methods are signature based schemes used by desktop virus-scanners and trusted computing mechanisms.

Signature based schemes store a signature, such as a hash-value of a piece of code, that is known to be malicious. Signature based schemes have various shortcomings that are intrinsic to these methods. They require that an attacking piece of code has been identified and its signature is distributed to the mobile phone. The time from detection to the distribution has to be as short as possible. The creation of a signature is a manual process, requiring an engineer to take a close look at the attacking piece of code. Attacks that are known to be working and are not identified are so called *zero-day-exploits*. Signature based schemes cannot detect or prevent zero-day-exploits. Limiting factors for the application of signature based methods on mobiles are storage space, processing power and network usage for the transfer of new signatures. On mobiles these factors have a harder limit than on desktops. Small changes to known malware can restrain a signature based scheme from correct detection.

In contrast *Statistical malware detection schemes* work with statistics features that describe the structure of malware. These schemes can be lightweight but are often not as accurate as signature based schemes. The Trusted Computing Group proposed an Architecture [6] that implements various methods to attest that a computer system is not being tampered with. The Attestation-functionality is one of the proposed measurements to proof that a system only runs the software that it is allowed to run. An implementation of this attestation, in the case of mobiles, is that of a central, trusted authority that can cryptographically sign every piece of software that is executed on the mobile. A piece of software that is not signed cannot be executed. This implementation of attestation requires the exchange of cryptographic public keys with a central authority that the device trusts. The central authority has to sign every piece of software that can be executed on the mobile. An implementation of this scheme can be very lightweight. To apply those methods, special hardware is required that can store cryptographic keys in a tamper-proof manner. A specification for such a trust anchor is the Mobile Trusted Module (MTM) [7]. Attestation methods cannot help against attacks where the regularly installed software is used by an attacker to start actions that were not intended. An example would be a regular running implementation of JavaScript containing a bug that exposes data to an attacker.

1.2 Problem Statement

Like any communicating system, a mobile device faces the problem of receiving malicious data from a communication channel that was either established by the mobile itself or by a correspondent node. Typically, unwanted instruction code is received from an attacking site that exploits weaknesses of the processing software and may be embedded in regular data. Common exploits target at the operating system, or - more frequently - at application programs like Web browsers or games. In contrast to stationary devices, mobile nodes are always less powerful. In particular, they are battery-powered and thus vulnerable to power exhaustion attacks. Mobile nodes have many communication interfaces, that can operate in parallel. Each communication interface has its own software stack. Software stacks such as Bluetooth and GSM, have been attacked in the past [8, 9, 10].

The interaction of multiple subsystems pose a risk as an entry point for malicious software. Software that is commonly used on mobile phones is branched from other projects. Software is modified to meet the special needs, for instance memory and CPU power requirements, of mobile devices. When branches diverge and new attack vectors are found in the original branch, the fix in the mobile branch typically takes time until applied. One of the major competitors in the mobile phone market has a poor history of supplying patches appropriately. DeGusta [11] visualized the update problematic. This shows that many mobile phones are up to three major releases behind schedule. There is no regular update cycle for many phones on the market.

The end user applications on common phones are written by thousands of developers worldwide. Different security models are established for monitoring the software [12]. Even in centralized and monitored software market environments, such as Apples iPhone store, the installation of malicious software could not be stopped [13].

To prevent an attack of embedded shell code, it needs to be detected prior to processing by the vulnerable software. This bears the problem that attacks need to be identified even if they are unknown and applied for the first time. As malware creation for the mobile regime is a new field of growing activity [14], generic detection mechanisms are needed that work on zero-day exploits. In addition, any protection scheme should be able to process data in real-time. Protection schemes should be able to work on data streams, thus allowing to send warnings as early as possible. Protective actions must comply to user requirements, and may not interfere with regular usability. It must not exhaust the mobile resources itself. It has to shield the user from unnecessary detail while informing him of possible threats with a high level of certainty.

2 Related Work: Malware Code Detection and Machine Learning

More than one decade ago, malware has been described as a growing problem [15]. With the pervasive use of computers in the everyday live, supported by tablet-PCs and smartphones, the problem is still growing. Many approaches have been proposed to detect malicious software. Not every method is applicable to the mobile realm, thus there is a demand for suitable routines. The algorithmic decision, whether a software is a malware, is closely tied to the halting problem [16]. The bad news is that many methods can aid in the detection, but there will never be a universal solution that can detect all malware [17]. Predicting behavior of programs can be reduced to the halting problem and is thus undecidable. Although in many cases estimations of behavior predictions can be accurate.

Work related to our study of statistical malware code detection cover papers from multiple fields. We discuss various approaches to malware detection including statistical methods, often applied in digital forensic analysis.

2.1 File Scanners

The scanning technique is used in every anti-virus(AV) software on the market. The scanners search for known patterns (signatures) in files. Szor examined scanning techniques prevalent in current AV-software [18]. He names different methods of scanning. Among them are string scanning techniques that match for a simple string, wildcard scanning that uses wildcards to cover small changes in the malware and smart scanning that can overcome simple mutations in the malware code.

With the rise of polymorphic viruses[1] simple scanning methods are not sufficient. Symantec uses a hybrid approach that combines scanning with code emulation [19]. The scanner can examine code of a running program and find virus-like behavior by combining a static analysis with properties gathered during run-time.

2.2 Statistical Approaches

To circumvent a comparing of known patterns with unknown samples, methods have been proposed that use statistical methods, often applied in data-mining applications. Data-mining methods use schemes from various fields of computing, such as statistics and machine learning. The most important components of such a detection are the classification algorithm and the selection of features. To aid the feature composition, statistics are applied on input data to gain a more robust and compact form. A typical basic scheme of data-mining applications, presented in this section, is shown in figure 1. The raw data to be analyzed is preprocessed in a way that aids the feature extraction. An example is the split of the input data in overlapping windows for further processing. Feature extraction typically normalizes the length of the input via statistical measurements. The choice of the right features is often the most important task. In the final step, the classifier assigns a class to the extracted features. This scheme can be applied to data streams.

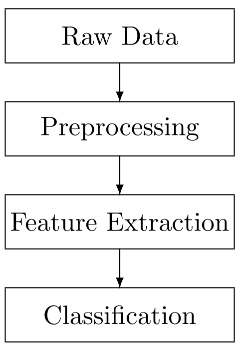


Figure 1: Typical data-mining scheme

Data mining has long been used in malware detection. Recent papers show that data mining is still a viable option. In 1996, IBM researchers applied neural networks to the

[1] Polymorphic viruses automatically change the content of their execution code sequences, without altering their malicious behaviour.

problem of finding boot sector viruses [20]. They were able to identify up to 85% of the viruses.

In a recent paper [21] Adobe researcher Raman used selected header fields of PE-Windows files to classify them as malware or benign using a tree algorithm. With a large test set he confirmed that tree learning algorithms are sufficient to find critical patterns in malware. In his paper he used the decision tree Learning Algorithm C4.5 [22]. The work focuses on PE-EXE files prevalent on Windows machines. He selected various features only concerned with EXE-files, thus this method is not a general scheme. The C4.5 Algorithm achieved a true positive rate of 0.98.

Like Raman, Siddiqui et al. [23] also tried to detect Windows based malware. They extracted sequences of op-codes from binaries to use them as a feature, thus stripping header and data sections. Their tests are based on a set of more than 800 malware samples. Their test data did not cover any encrypted or polymorphic viruses, thus only static viruses were tested. Their approach showed a 98.4% detection rate. The result was achieved with previously unknown code, not used in the training process of the classifier. The false positive on unknown malware was at 1.9%.

Instead of using one classifier, the work of Schultz et al. [24] trained multiple classifiers on a set of malicious and benign executables to detect new malicious code samples. The work focuses on Windows PE executables. With naive Bayes classifiers, the accuracy was greater than 90% with a false positive rate of less than 2%.

The approaches of Raman, Siddiqui et al. and Schultz et al. were specific to Windows malware. A more general method that can work on any file-type was proposed by Lyda et al. [25]. They have used (stationary) Shannon-entropy averages to roughly distinguish certain data types. Their approach shows that a pure entropy analysis can support a manual malware search. For an automated detection of high accuracy, entropy averages alone are not satisfactory.

Conti et al. [26] also used the Shannon-entropy. They additionally applied other statistical measures to correlate in their scheme. In contrast to Lyda et al. they applied multiple statistics to cluster data of different file types. They split each file in fragments and applied normalized mean, Shannon Entropy, Chi Square, and Hamming Weight as characteristic feature Classification by the k-nearest-neighbor algorithm achieved a 96.7% accuracy, accumulated for ELF and PE files, in their test-set.

The use of multiple classifiers was already suggested by Schultz et al.. The method of using multiple classifiers for a classification problem is a basic principle applied in boosting. Boosting is an established method in machine learning. The boosting method merges multiple classifiers to gain a single, more efficient classifier. A larger comparison of the effectiveness of different classifiers on malware detection is given by Kolter and Maloof [27]. They applied the classifiers naive Bayes, decision trees, support vector machines, and boosting. For testing, they collected about 1600 malicious code samples for the Windows platform from various sources, one of them being a message board about viruses called VX Heavens [2]. For the pre-processing of the data, text mining methods were applied by selecting relevant n-grams from sample code. The amount of n-grams was then filtered by only using the most relevant n-grams according to the *information gain* calculated by a formula of Yang and Pederson [28]. The information gain helped to select the best features for classification automatically. Their tests were aided by WEKA [3] a data mining software. Boosted decision trees showed the most promising results. For a desired false-positive rate of 0.05 boosted decision trees achieved a true-positive rate of 0.98.

2.3 Activity Monitoring and Behavior Detection

Monitoring the activity of a program is a frequently used method to reveal the intentions of a program. The activity is monitored during run-time and thus called dynamic analysis. One of the most common methods of monitoring program behavior is the monitoring of API-calls. Egele et al. [29] compared 18 general malware analysis tools that use dynamic methods for the malware detection.

Wagner et al. [30] report on a prototype that builds a control flow graph from learning the behavior on a static basis, without executing the code. During run-time on the actual machine, the constructed calling graph was compared to the actual API calls to detect differences and thus abnormal behavior. They showed examples of real malware samples and demonstrated their method. A greater malware set was not used in their tests, thus a success rate was not presented in the paper. The application of control flow graphs was

[2] virus exchange message board `http://vx.netlux.org`

[3] `http://www.cs.waikato.ac.nz/ml/weka/`

tested on metamorphic malware, where it proofed effective [31]. Cesare et al. [32] also used control flow graphs to detect malware. They propose a method to build malware signatures using control flow graphs based on the decompilation technique of structuring. During the signature generation the malware code is emulated in a safe environment. The signatures consist of a small grammar that represents the control flow graph. Similarities between signatures are determined using string edit distances. Their method combines dynamic and static aspects of malware analysis. An essential step of their static analysis is unpacking of packed malware.

While the other works deal with call graphs to a large set of operating system APIs, Bai et al. [33] select a smaller subset of APIs that are critical for most of the known malware. They construct calling graphs (critical API-calling graphs (CAGs)), using only critical APIs (e.g., network access) and discarding non-critical API. This method shrinks the graph, compared to a graph featuring all APIs. With a known CAG Graph signature of a malware they can detect variants of this malware using similarities in the calling graph.

Younghee et al. [34] executed code in a sand-boxed environment. With generated behavior graphs they could find matches in malware. Using sub-graphs they were also able to detect certain polymorphic malware. The tests were performed on a set of 300 malware samples. They classified the samples in multiple malware-groups. Only 5.3% of samples could not be classified in any class.

In contrast to building graphs from static analysis, Rieck et al. [35] execute suspicious software in a sandbox environment. The focus of their approach is the classification and clustering of malware-groups based on the behavior. Sequences of API-calls are mapped to short sequences of observed instructions, representing groups of malicious behavior.

In contrast to the above mentioned papers, Kim et al. [36] use a dynamic method, applied during run-time, to detect malware from energy usage profiles of applications. The work shows that malware can have conspicuous energy usage profiles. The detection is built upon gathering the differences between known usage profiles and the actual profile.

2.4 Network Monitoring

Methods from the realm of intrusion detection overlap with the goals of malware detection. Instead of monitoring activity on the device, the external data sources can also be monitored for suspicious traffic.

Nazario [37] proposes techniques to detect Internet worms in networks. He describes different patterns of data acquisition, among them are packet capture and statistics from switches. He describes the change of traffic patterns of a host as a meaning of detecting infected hosts in a network.

Wang et al. used statistics on incoming and outgoing packets to detect zero-day exploits in networks [38]. They recorded a regular profile of a network sites traffic. Using the network profile, they were able to detect anomalies in traffic flows. They applied statistics to build clusters of suspicious content flows. With a collaborative security system they were able to detect many network worms. The statistical methods applied here were related to methods used in file carving, presented in section2.5

Olivain and Goubault showed that the entropy analysis can be applied to the network layer [39]. Their software *net-entropy* is able to detect attacks on the handshake of encrypted network protocols without accessing the decrypted content. In the training phase they record typical entropy profiles for small chunks of encrypted handshake data. The entropy profile generation uses entropy functions generated from many typical handshakes. In the working phase, the error between the recorded profile and the actual data is compared to detect attacks to network streams in real-time. In the process they used an approximation of the Shannon-entropy to get reports on the entropy before receiving the complete data [40] .

The work of Gu et al. [41] applied the method of maximum entropy estimation on the detection of anomalies in network traffic. The maximum entropy estimation algorithm is applied to extract the baseline distribution of the packet classes from the training data. Their tests yield detection rates above 90%.

The work of Nychis et al. [42] applied time-series of entropy values from network related sources to anomaly detection. The entropy time series were supplied by traffic volume, source addresses, destination addresses, in-degree, out-degree and other network sources. Their approach showed that time-series of entropy values of address and port distributions are strongly correlated and provide a stable detection capability for malicious activity in a network.

2.5 Excursion: Computer Forensics

A common problem in computer forensics is the type identification of a file. While this problem is not similar to malware detection, the methods applied are very much alike the data-mining methods presented in section 2.2. In practice, the question of file type identification occurs, when files have to be reconstructed. Reconstruction is required, whenever directory information is lost or deleted. The reconstruction process is called *file carving*. Another example is the restoration of content from network streams. The methods presented here try to identify the type of a file using statistical analysis of their contents without the use of parsing.

A basic approach is the file type identification by Hickok et al. [43]. They use a combination of extension and magic bytes prevalent in the files. The work proofed that methods that rely on the prevalence of patterns, such as magic bytes, are ineffective with many file formats. The detection of magic bytes is similar to virus scanners presented in section 2.1.

McDaniel et al. propose a method for file type identification driven by statistics based on segments from entire files [44]. The algorithm did not concentrate on prevalent patterns such as magic bytes. They used three different algorithms to generate fingerprints, for file-types, based on a set of known input files. The algorithms are based on using byte-value distributions of the file content and include byte frequency analysis, byte frequency cross-correlation analysis and file header/trailer analysis. They predict the file-type by finding the minimal difference in a histogram for an unknown file type, compared to a *fileprint*. The *fileprint* is a centroid constructed from known files. Their tests show a large variance of results, that depend on the data provided. Their results vary from 27.5% up to 95.83%, depending on the feature selection. They conclude that the results of their approach show that basic statistical methods are not enough to construct a reliable detection method. The success of their method depends strongly on prevalent patterns within the input data.

The *fileprints* method proposed by Li et al. [45] uses a similar approach as McDaniel. They use the byte frequency as a statistical measurement. They extend the method of McDaniel by using a set of centroids, instead of just one, to describe a file type. Clustering is applied to find a minimal set of centroids with a high detection rate. The use of multiple centroids leads to better results than achieved by McDaniels. The underlying problem of the requirement of prevalence of statistically relevant patterns within the input data is not solved. Without regular patterns in the data, the detection rate can lead to a sudden decrease. Compared to McDaniels, the use of multiple centroids leads to a rather complex method, because it requires more resources in terms of processing time and memory usage. Using exemplar files as centroids, the method achieved a 94.1% accuracy on EXE files.

An approach that is similar to Li et al. can be found in the work of Karresand et al. [46] with their *OSCAR*-method. In addition to the byte frequency, the OSCAR-method also uses the rate of change. They define the rate of change as the absolute difference between two consecutive byte values. The rate of change is applied to also take the ordering of bytes into consideration. This improved method shows better results than their predecessors. The method has similar problems than McDaniels approach.

In his master thesis, Harris [47] implemented a file type detection algorithm for image files. The work uses neural networks with up to 30 hidden neurons to learn patterns of 5 different image file types. Small segments of a file were repeatedly fed into the neural network for classification. The intent is the identification of entire files, not small segments as were the goal of the above mentioned methods. This was done to stop unwanted effects when a file contains many null values. As the approach did not use any statistical measurements, the detection rate was never above 50 percent for any file type of the test set. This approach proofed that neural networks can be applied for the pattern detection. Nonetheless the ability of neural networks to detect patterns remains insufficient. An algorithm is needed that extracts features from the input data that can support the neural network.

Hall and Davis [48] use entropy in a sliding window approach to determine the type of files. When calculating the entropy for a sliding window, there are many values that have to be recalculated when the window slides to the next position. Hall and Davis rewrote the entropy formula to prevent recalculation of the entropy values. To identify file types, they collected average entropy functions from a test-set. The method features from file-types. Identification is performed by calculating a distance between known and new file-types. Instead of a distance measure they also tried Pearsons Rank Order Correlation which led to better results. The approach fails to identify file types correctly. It can help to give a rough idea about the file-type. They had a success rate of 97% for ZIP-files.

The work of Erbacher and Mulholland [49] deals with the localization of data types embedded within a file. They applied 13 statistical tests to measure features of the file. The most successful statistics were the average, kurtosis, distribution of averages, standard deviation, and distribution of standard deviations. In their tests, these statistics were sufficient to determine the type of the file. The paper focuses on window sizes and their effects on statistics. A success rate was not stated.

Moody and Erbacher [50] implemented Erbachers work in a test method called Statistical Analysis for Data Type Identification (SÁDI). Their approach tries to identify the type of a file without relying on meta-data. Their tests showed false positive rate of 13.6% for Windows DLL and EXE data.

Veenman [51] applies Fishers Linear Discriminant(FLD) classifier to the entropy based fileprint and a measure based on the Kolmogorov complexity [52], a measurement for code complexity. Unlike the entropy, the Kolmogorov complexity measures substring order. To calculate the Kolmogorov complexity, Veenman uses the formulas by Lempel and Ziv [53]. Compared to other papers in this section, tests were conducted with a large set of 450MB. They achieved a 0.78 positive rate on the test set.

A variation of Veenmans approach has been done by Calhoun and Coles [54]. They also applied the FLD to the classification problem. Additionally, several different statistics and the use of the longest common sub-sequence algorithm were applied which led to better results. They compared different statistics to discern different file-types from each other. The Shannon entropy lead to an 78.5% average detection rate.

3 Related Methods

In this section, we introduce a small set of basic methods that we will use later in our detection scheme. The related methods discussed here are mature methods that have been proved to be successful in a wide range of applications. We start with the Shannon-entropy that extracts information about the order of our input data. The short-term Fourier analysis is then applied to analyze our data further. The final step is the classification that is aided by using an artificial neural network, a method from the field of artificial intelligence.

3.1 Shannon Entropy

The Shannon-entropy [55] is a measure of uncertainty in the information theory. It describes the *information-density* of a data sample. A high information-density of a data sample denotes that it has a low order and thus often a poor *compressibility*. The Shannon-entropy is a lightweight measurement that can be computed with low computational overhead. It can be computed as

$$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i),$$

with X a symbol sequence composed of a finite alphabet.

The X_i represents one character of the alphabet of X, and $p(X_i)$ is the probability of the occurrence of X_i within the measured sample X.

As an example, we use an alphabet of only two characters, $\{A, B\}$. With a two character alphabet, our entropy result will be in the range between 0 and $log_2(2) = 1$. For a first example, let $X = AAAA$, which yields $p(X_1) = p(A) = 1$ and $p(X_2) = p(B) = 0$. This results in the minimum entropy of $H(X) = 0$. The order of data is often confused with randomness. In the next example we show an example that has a high entropy but also a low order. If we change the sample to $X = ABAB$, our formula yields the maximum entropy of 1. This example shows that a low orderliness leads to high entropy, even if the symbols are not random. This sample is not random, because the sequence can be described with simple deterministic „rule“ of generation. Another sample $X = ABBB$ results in an entropy-value between the two extremes of $H(X) = 0.8113$. This illustrates that a low orderliness yields a higher entropy.

On the byte-level, we have an alphabet with 256 possible values X_i that show relative frequencies given by $p(X_i)$. The resulting $H(X)$ yields entropy-values ranging from 0 to $log_2(256) = 8$.

3.2 Short-Term Fourier Transform

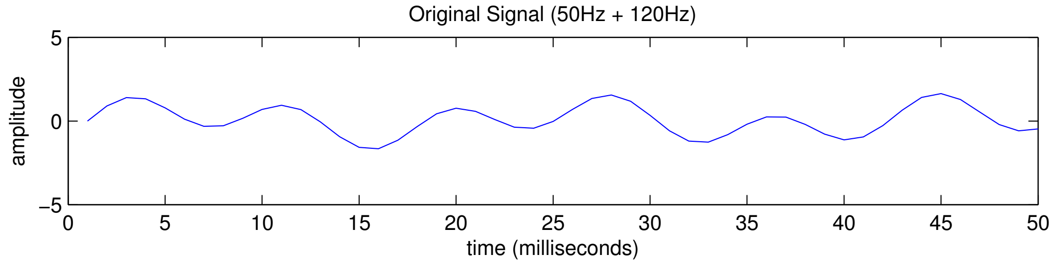
The Fourier transform can convert a signal from the time domain to the frequency domain. Thus the Fourier transform gives a change in the view of a signal that can often help to gain a better understanding of its characteristics. The Fourier transform of a signal is a representation containing a sum of complex exponentials of varying frequencies, magnitudes and phases. The Fourier integral transform is defined by

$$\phi(t) = \int_{\infty}^{\infty} e^{ixt} f(x) dx$$

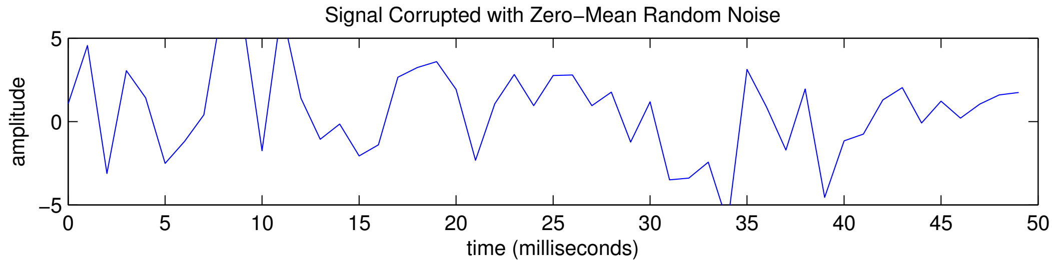
The application of the Fourier transform to identify the frequency composition of noisy signal is called *Fourier analysis*.

The mathematical concept of the Fourier analysis uses the idea that any signal can be approximated by a sum of sinusoidal signals. The approximation improves as more sinusoidal signals are added. As a mathematical concept, the Fourier analysis is only applicable to continuous functions, with the implicit assumption that a function has a periodic character.

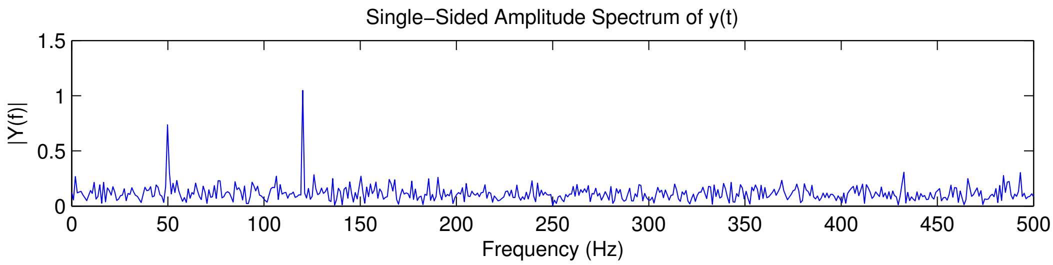
An example of the discrete Fourier transform (DFT) is shown in plot 2. Figure 2(a) is produced by adding two sinusoidal signals of 50 Hz and 120 Hz. The plot uses a sampling-frequency of 1 Hz. The second figure 2(b) added zero-mean random noise to the signal. The last figure 2(c) shows the application of the Fourier analysis to signal 2(b), resulting in two spikes at 50 Hz and 120 Hz. The y-axis shows the magnitude (amplitude) of the initial signal of figure 2(a). The amplitude means the maximum absolute value of the signal (a periodically varying quantity). In the field of signal processing applications, the resulting Fourier-transformed data has to be multiplied with the initial sampling-frequency, resulting in correct frequency scales. For our purposes of pattern detection, this „normalization“ is not important. Note that in the first two diagrams the x-axis is labeled *time*. Whereas the third diagram uses frequency as the x-axis label. The label changed because of the *transform*



(a) Two sinusoidal signals of 50 Hz and 120 Hz.



(b) Figure 2(a) with random noise added.



(c) Fourier transform of figure 2(b), showing spikes at 50 Hz and 120 Hz.

Figure 2: Example of the frequency analysis using the Fourier transform.

from time to frequency using the Fourier transform. The result of the Fourier transform shows a magnitude and a phase (of the sinusoidal signal). For our pattern detection, we are only interested in the magnitude. The phase shift would be important for the correct reconstruction of a signal by using the inverse Fourier transform.

The mathematical concept of the Fourier analysis can only be applied to stationary-signals. To extend the Fourier transform to *non-stationary* signals, a window based variant can be applied [56], which is called the discrete short term Fourier transform (STFT). The STFT can be computed by

$$STFT\{m, \omega\} = \sum_{n=-\infty}^{\infty} X(n)W(n-m)e^{-i\omega n},$$

where the function $X(n)$ is the signal-function that is to be transformed to the frequency domain. The STFT multiplies a window function, denoted as $W(n-m)$ with the input data in $X(n)$, to protect the result from so called „leakage effects“. We will talk about leakage effects in section 3.2.1, where we discuss the effects of different window functions. In the discrete case, the window function has a finite length m with n as the time index. The resulting $STFT(m, \omega)$ contains the magnitude and the phase.

There are two problems with the Fourier analysis using the STFT. First it can only measure the signal for a limited amount of time. The second problem is that the STFT only calculates results for certain frequency ranges, the so called *bins*. These bins cummulate the magnitude of frequencies within intervals. A limit on measurement time is fundamental to any frequency analysis. The frequency sampling problem is especially prevalent in numerical methods like the STFT.

3.2.1 Window Function

The application of a window technique delivers results that are not completely accurate. The windowing measures the signal only for a limited amount of time and thus can cut out parts of the signal, leading to numerical errors. This effect is called the *leakage effect*. The *leakage effect* impedes the result from being accurate. There are frequencies for which the magnitude is not represented correctly by the STFT. This *leakage effect* is provoked by the windowing that separates a signal on unfavorable positions. The leakage effect can be minimized, by amplifying parts of the signal before the Fourier transform is applied. In the STFT, the amplification is done by multiplying the signal of each window with a special function, called the window function.

There are many window functions with varying impact on the frequency domain. A window function has to be selected carefully, depending on the application needs. In [57], Harris gave an overview of the effect of many different window functions for the discrete Fourier transform. From Harris table we selected the Hann window [58] (called Hanning window in Harris table) for our application. This window was chosen due to the low impact on leakage and the time-dependent, non-repetitive type of signals that we are processing. The Hann window is a window function that is defined by

$$w(n) = \frac{1}{2}\left(1 - cos\left(2\pi\frac{n}{N}\right)\right), 0 \leq n \leq N,$$

where N is the width of the window that can be selected according to application needs. The parameter n is the position within this window. Figure 3 shows a Hann window with a width of $N = 64$. If the Hann window is multiplied with a signal window $X(n)$ within the STFT, the magnitude of the edges of the window will be decreased. In the STFT scheme, this window function is multiplied with every STFT-window that is processed.

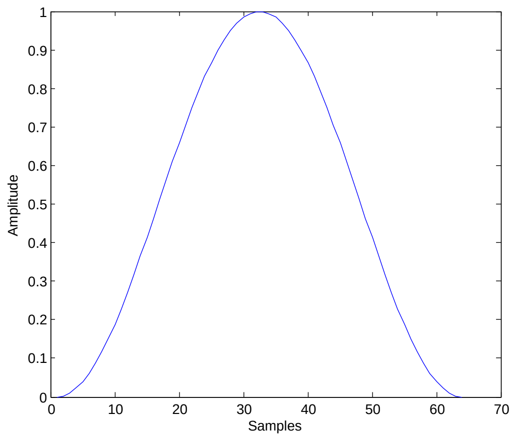


Figure 3: Hann Window with a width of $N = 64$ samples

3.3 Artificial Neural Networks

A classifier is the implementation of an algorithm that can separate a set of input data to different output classes. In the field of machine learning, there are numerous algorithms for the classification of data. In this context, a **class** is data from a set where each class is discriminable from each other class with respect to some observables.

A well known classifier is „Artificial Neural Networks“ (ANN) (see [59]). We will use ANNs in this work to discriminate executable- from non-executable code.

Artificial Neuronal Networks are inspired by neurons found in biological brains. Artificial neurons are small units with a very limited functionality. Many of those neurons can be put together in a network to deliver complex results.

3.3.1 Artificial Neurons

A single neuron can be seen as some mathematical line that divides a two dimensional space in two regions. Given such an intersection, a single neuron can be used as a classifier for a problem with two input parameters. On a plot, the X and Y axis would represent the numerical input parameters of the neuron. The formula of the neuron could then be used to determine which output class is assigned to which input.

The output of the neuron with two inputs o_i is determined by

$$f_{act}(o_1 \cdot w_1 + o_2 \cdot w_2 - \theta),$$

with the two input parameters o_i , the internal weights of the neuron w_i and the neuron bias θ. The weights and the bias are numbers that are determined during the training phase of the neuron. The result of this formula is fed to an activation function f_{act}, such as a sigmoid function ($f_{log} = \frac{1}{1+e^{-x}}$). The activation function is used for two reasons. First it will give smooth transitions between classes. For example, if one input-set is on the side of class 0 and near the border of class 1, the activation might deliver values close to 0.5. The second reason for an activation-function in a neuron is that it delivers an upper bound for the output of the neuron. Let us consider a network of neurons, where the output of one neuron is the input of another. In such a typical setup, the results would be growing numerically in every new layer of neurons. To limit these growing outputs, the activation function is used. For this reason, most activation functions have the limits

$$\lim_{n \to -\infty} = 0$$

and

$$\lim_{n \to +\infty} = 1.$$

For practical applications, differentiable functions such as the sigmoid function are used. For theoretical observation (for easier mathematical handling) the non differentiable Heaviside step function is used. If the Heaviside step function is applied as f_{act}, defined as

$$f_{act}(X) = \begin{cases} 1 & if X \geq 0 \\ 0 & otherwise \end{cases},$$

we can transform the output neuron function to show that it is a simple line function. The equation of the dividing line would be

$$o_1 \cdot w_1 + o_2 \cdot w_2 = \theta$$

If this equation is solved for o_2, we get

$$o_2 = -\frac{w_1}{w_2} \cdot o_1 + \frac{1}{w_2} \cdot \theta,$$

which for the argument o_1 is the equation of a straight line $y = mx + b$.

If we combine multiple layers of neurons, we can describe more complex classes of problems. These layers of neurons raise the simple two dimensional representation of a single neuron to a hyperspace with neurons describing the class boundaries within the hyperspace. In the hyperspace, very complex problems can be represented. The „knowledge“ of a neural network consists of its weights and the bias of every neuron.

3.3.2 Training of Artificial Neural Networks

Training of the neural net consists of adjusting the *weights* and the *bias* of all neurons. The activation function is predefined and the same for all neurons. For most of the practical applications, a type of sigmoid function is used.

The training of an ANN requires a set of training data that has to be prepared manually. Often not the raw data is used for classification, but a specific set of features that support the class finding problem. By using features of the problem domain, the learning of a feature-to-class relationship can be supported.

One practical example of using features are in image processing and in the classification of objects in two dimensional images. In the domain of image object classification, we often encounter the problem that objects are rotated to a certain degree. The goal is that classification should deliver the same results, no matter what the rotation angle was. If the raw data is used, then the neural net would need to „learn“ all of the rotation variants. Thus the neural network would have to store more information, and would grow bigger than necessary. A more subtle approach is to select features that are rotation-invariant. Carefully selecting the right features can substantially support the ANN training process.

For the classification scheme, we do not use the data stream directly as an input to our ANN. Instead we change the view on the data by decomposing a signal into its constituent frequencies. Thus a „higher“ level of information of features, using the frequency distribution is used. In practice, the problem of finding the right feature set is most challenging.

Training data consists of typical input samples together with the desired class output of the neural net. For our problem, we need to find samples of our desired classes and label them accordingly. As the network topology, we use a *feed forward network*[59], where every neuron layer is connected with the next layer. After constructing the training data, the training of the net can be initialized.

The training method is a computationally intense process that does not need manual intervention. In a training method called *Backpropagation*, the weights are adjusted in small steps. Backpropagation needs to know what rate of error is acceptable for the user. Then Backpropagation iterates until the desired error rate has been reached. For every iteration, a sample is picked and its neural net output is computed. The difference between the desired output of the neural net and the actual output is used to change the weights and bias. The change starts at the output neurons and is propagated backwards until the input neurons are reached. The Backpropagation can be seen as a gradient-descent-method, because it descends in small steps within the „space“ of potential settings to get nearer to the desired output.

4 A proposed Binary Detection Method for Executable Code Fragments

With the basic methods shown in the last section we will now take a closer look at the details of our binary code detection method. In our approach, we will first use the Shannon entropy to generate an entropy-function of the input data. Then we will use the a frequency analysis to the entropy-function. The frequency analysis is conducted by an application of the short term fourier transform on the entropy-function. The data obtained through the frequency analysis supports the binary classification in code and non-code classes. The classification is conducted by artificial neural networks.

4.1 Extraction of a Statistical Function

We use the Shannon-entropy described in 3.1 to extract a statistical signal function from the input data that we call the „entropy function“ . The signal function is generated through the application of the Shannon-entropy of small, overlapping windows of the size w_e of input data.

The adjustable parameters of an entropy-function are the *window-size* and the *step-size.* The step-size is the amount of bytes that the entropy window is shifted on every iteration. This method was already suggested in [60] with a window-size of 256 bytes and a step-size of 128.

The windowing is done with a simple scheme shown in figure 5. Formula 2 defines windowing with the start position i, the end position j of the n-th window. The *step size* s, and the *window size* w are constant.

$$i_n = i_{n-1} + s \qquad (1)$$
$$j_n = i_n + w \qquad (2)$$

Figure 4: Calculating the window indices of start i and end j.

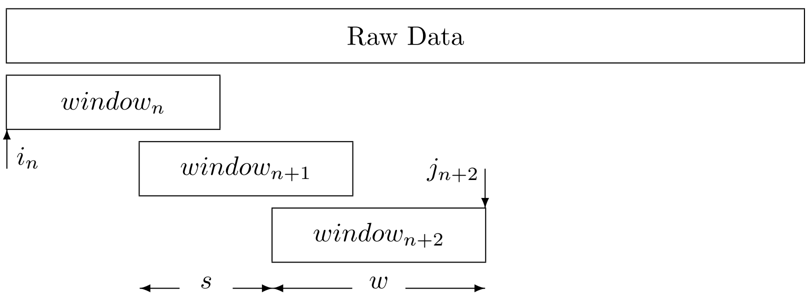
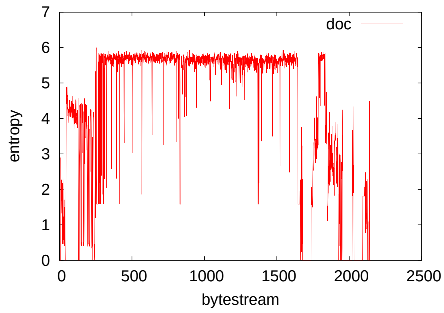


Figure 5: Generalized windowing scheme

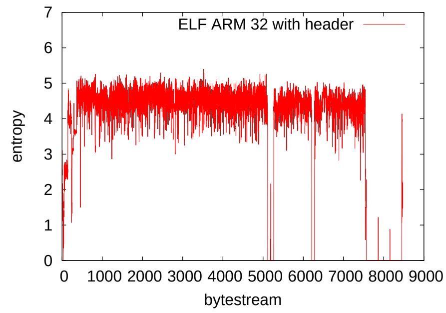
Plotting the resulting values shows an entropy-function of the raw data stream, which denotes sections of different levels of entropy. The plot 6 shows characteristic areas of high and low entropy for different file-types. Sub-figure 6(b) shows a ELF-ARM 32 file, which contains code and data. Such an entropy plot can disclose the overall file-structure with areas of different entropy levels. In file types that contain different types of data the resulting entropy-function can show information about the file at a different perspective, without knowledge of the exact type of the content. One example are PDF files shown in figure 6(g). The PDF-file contains multiple data-types such as text and images. The different entropy levels represent the position within the file.

The entropy function allows a coarse overview of file-contents. Different entropy levels can give hints about the content of the data while still containing noise. Generating an entropy function requires 2 parameters. The window size w_e and the overlap with the previous window o_e. The figures 7-13 show the effect of selected parameters for w_e and o_e on the entropy function of a PDF file. Figure 14 shows the effects on different settings for the Fourier window size.

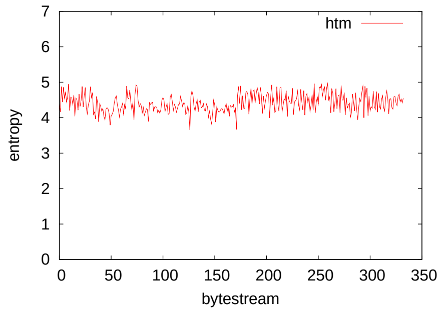
In [61] the authors have used the average and standard deviations to cope with that problem of noise. That approach seems to work if a rough file-type identification is required. The



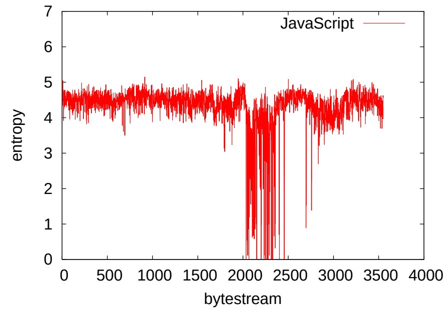
(a) Filetype: doc



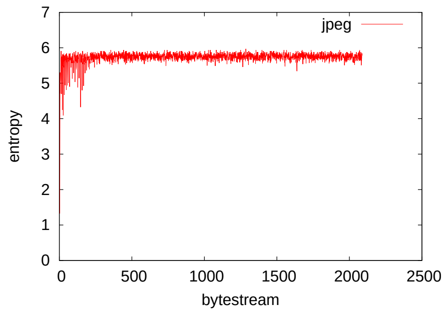
(b) ELF ARM 32 with headers



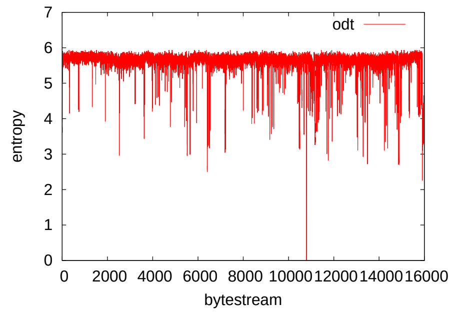
(c) Filetype: htm



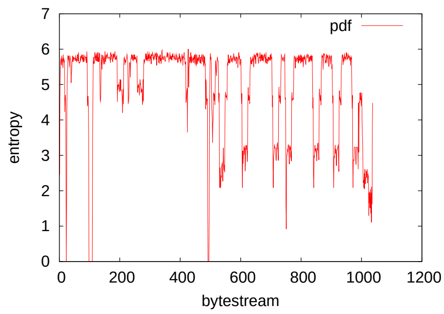
(d) Filetype: javascript



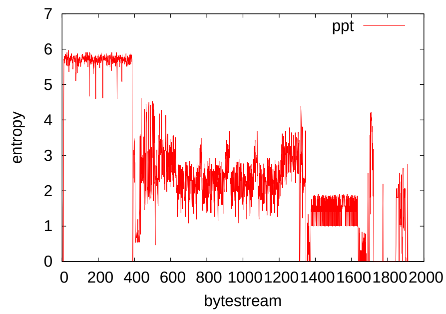
(e) Filetype: jpeg



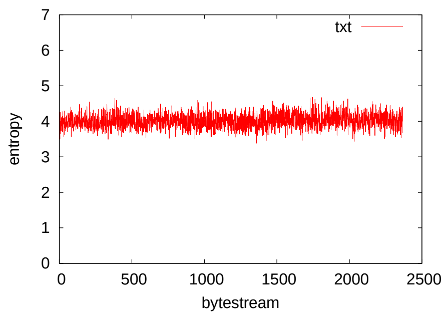
(f) Filetype: odt



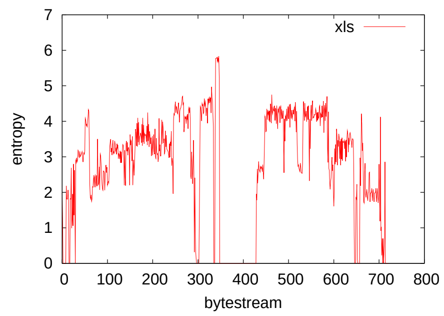
(g) Filetype: pdf



(h) Filetype: ppt

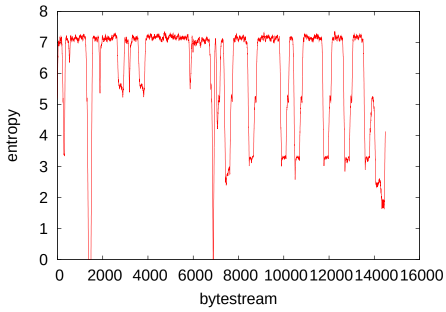


(i) Filetype: ASCII text

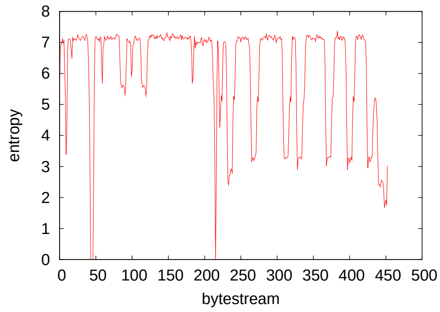


(j) Filetype: xls

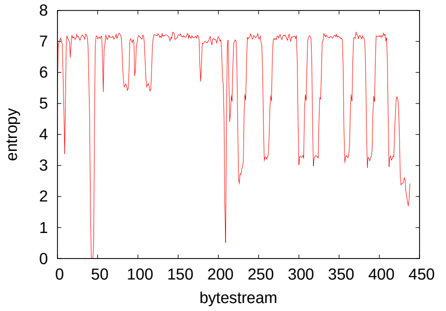
Figure 6: Typical Entropy Functions for different filetypes (plotted with $w_e = 64$ and $o_e = 8$)



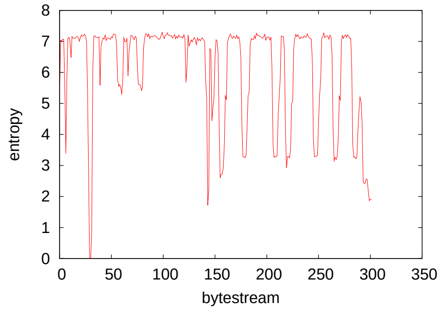
(a) o_e:252 w_e:256



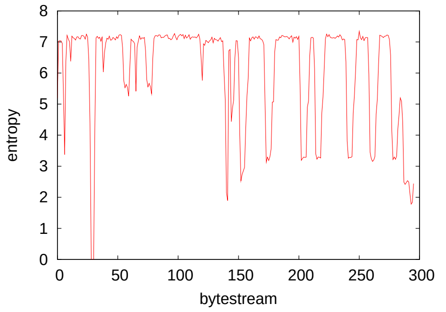
(b) o_e:128, w_e:256



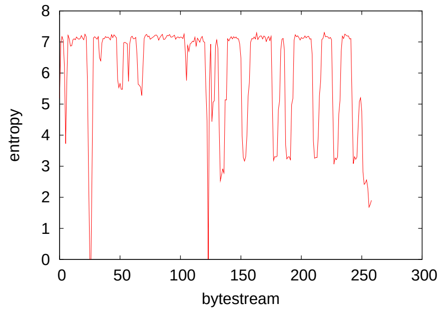
(c) o_e:124, w_e:256



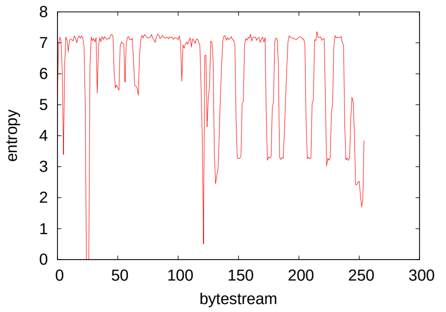
(d) o_e:64, w_e:256



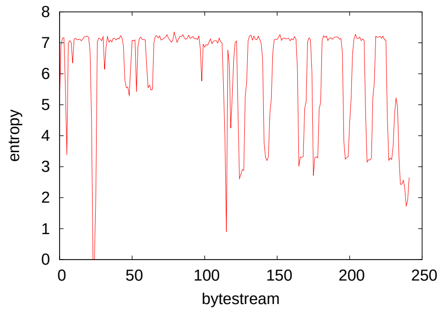
(e) o_e:60, w_e:256



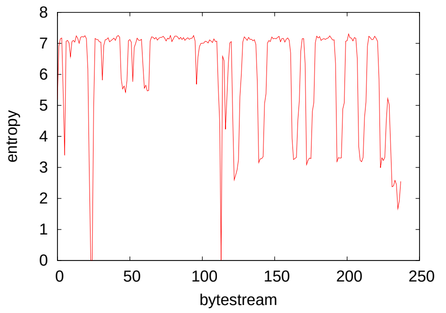
(f) o_e:32, w_e:256



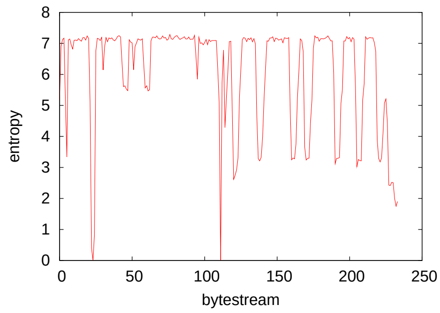
(g) o_e:28, w_e:256



(h) o_e:16, w_e:256

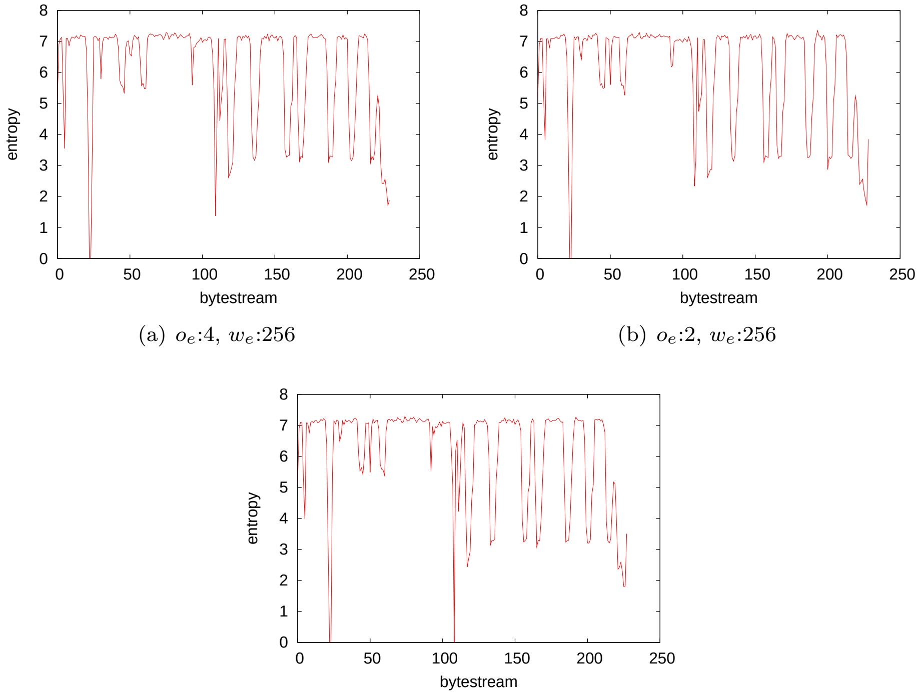


(i) o_e:12, w_e:256



(j) o_e:8, w_e:256

Figure 7: Entropy function of a PDF file with selected parameters for $w_e = 256$ and various o_e.



(a) o_e:4, w_e:256

(b) o_e:2, w_e:256

(c) o_e:1, w_e:256

Figure 8: Entropy Function of a PDF File with selected parameters for $w_e = 256$ and various o_e.

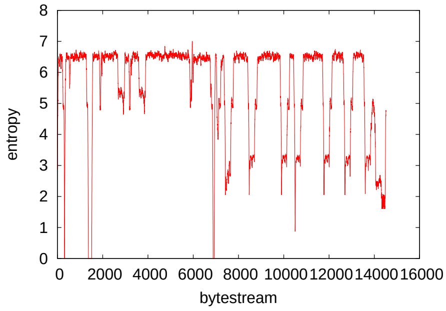
detection of embedded malware can be more difficult because the malware sections can be very small compared to the rest of the file. Thus we need further algorithms to extract more information out of the entropy function to detect embedded binary code.

4.2 Signal Analysis

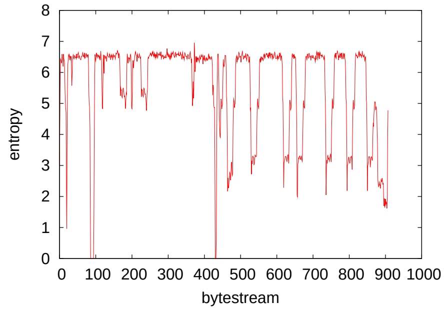
The entropy function that we built in the previous section can be regarded as a discrete signal that can be analyzed with signal processing methods. In this section we will apply a frequency analysis to the entropy-function. In the last section, we showed that the entropy-function delivers a noisy, non-stationary signal, that needs further examination.
We use the short term Fourier-Transform to convert the entropy-function from the time into the frequency space. The regular fourier-transform is only applicable to stationary-signals. With non-stationary signals, the overlapping is important to prevent the missing of lower frequencies that are larger than one window.
The resulting transform yields complex values that are not required for this purpose of signal analysis, thus we use the absolute of the transformation. The complex values of the transform would be required for a back transformation from frequency to the time space. This method focuses on the detection of small chunks of embedded malware code. To fulfill the requirements of an accurate detection of small units, this method has to work with small windows.
The result of the Fourier transform shows the magnitude of the high and low frequencies, that were present within the entropy function at that specific byte position.
The windowing method is the same as the entropy-function generation, shown in formula 2, but this time applied on the stream of entropy values. We call the result of this windowing operation an *entropy spectrum*.

4.3 Classifier

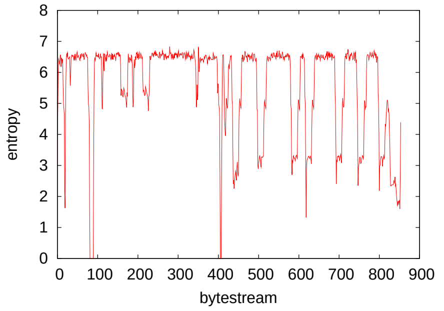
The *entropy spectrum*, shown in the last section still leads to *noisy* signals. The automatic detection of noisy binary-instruction code is not trivial, thus we apply our Artificial Neural Networks (ANNs), introduced in section 3.3 to avoid the problems of noise. We want to use ANNs to sort small windows of Byte streams in the instruction-code-class or the non-instruction-code class, which are our predefined classes for this particular problem. To detect binary instruction code, we processed the data according to the scheme described in sections 4.1 and 4.2. Now we describe the steps that are required to send the preprocessed data to



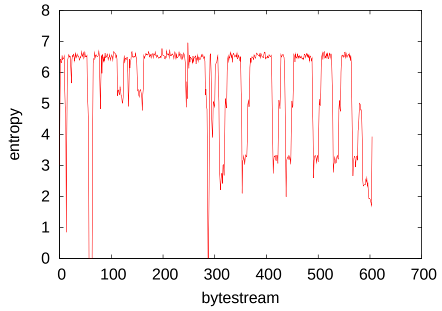
(a) o_e:124, w_e:128



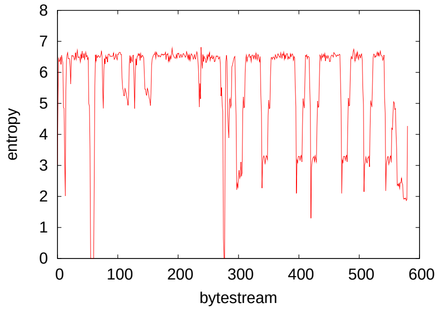
(b) o_e:64, w_e:128



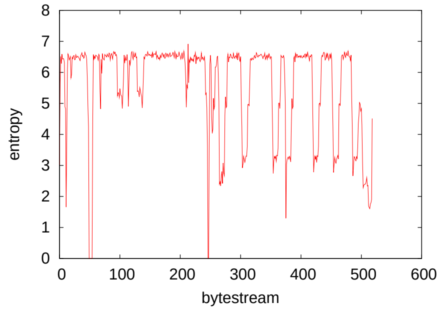
(c) o_e:60 w_e:128



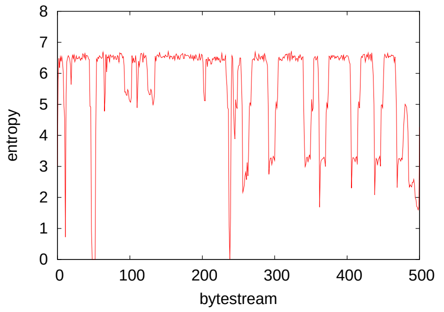
(d) o_e:32 w_e:128



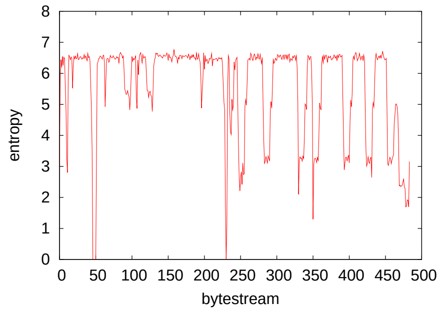
(e) o_e:28 w_e:128



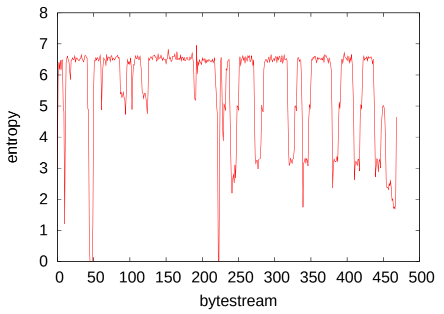
(f) o_e:16 w_e:128



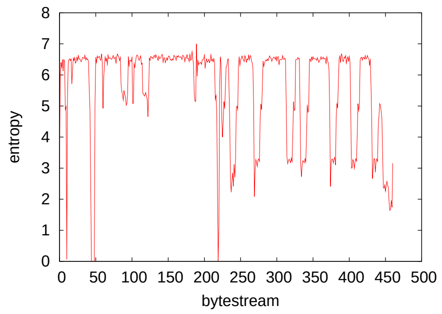
(g) o_e:12 w_e:128



(h) o_e:8 w_e:128

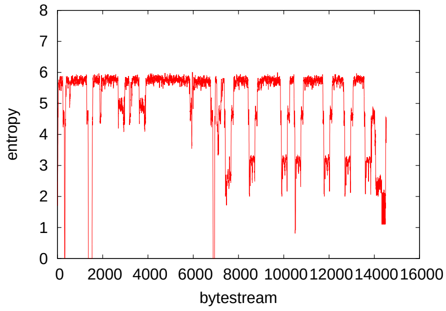


(i) o_e:4 w_e:128

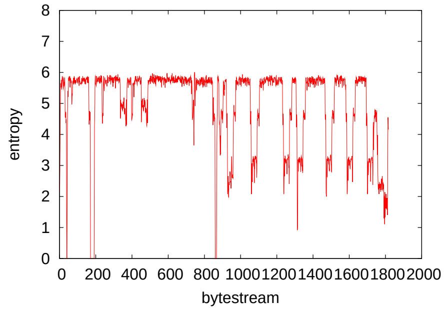


(j) o_e:2 w_e:128

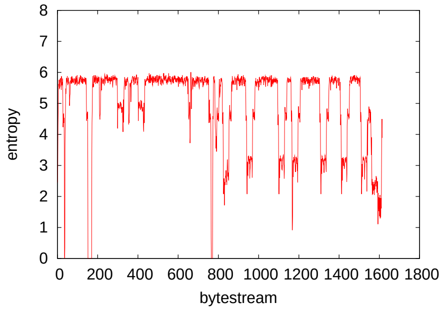
Figure 9: Entropy Function of a PDF File with selected parameters for $w_e = 128$ and o_e.



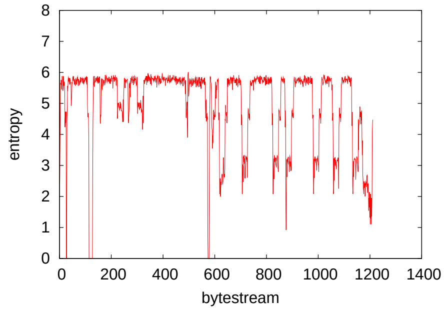
(a) o_e:60 w_e:64



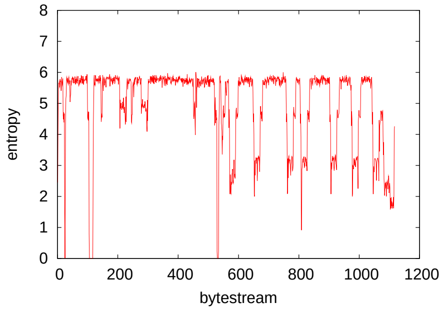
(b) o_e:32, w_e:64



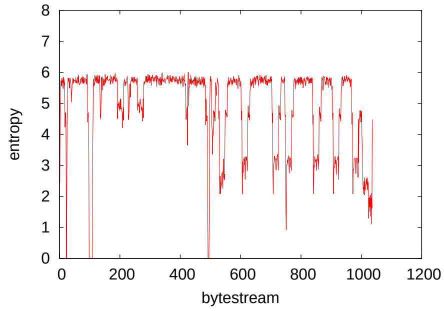
(c) o_e:28, w_e:64



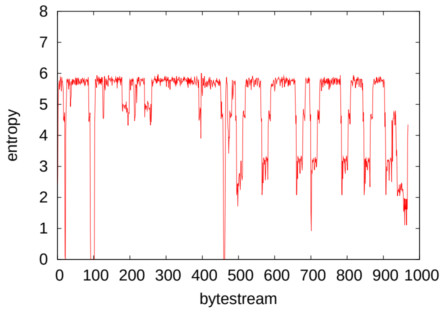
(d) o_e:16, w_e:64



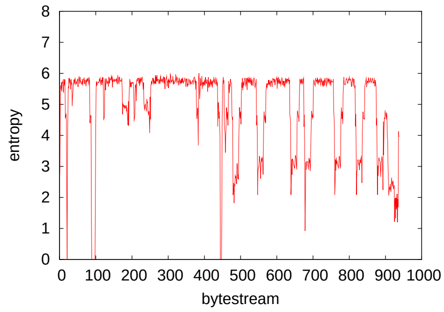
(e) o_e:12, w_e:64



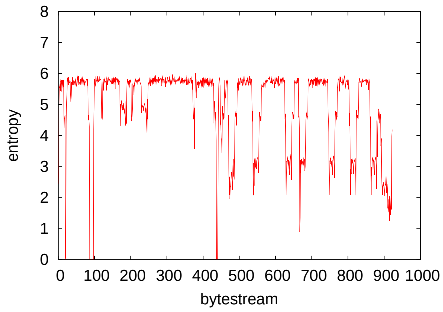
(f) o_e:8, w_e:64



(g) o_e:4, w_e:64

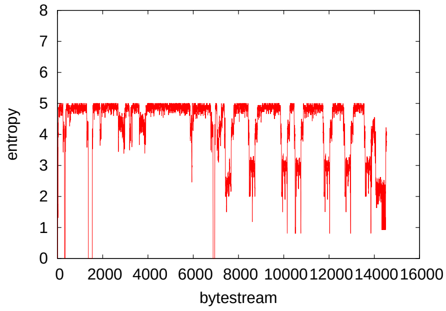


(h) o_e:2, w_e:64

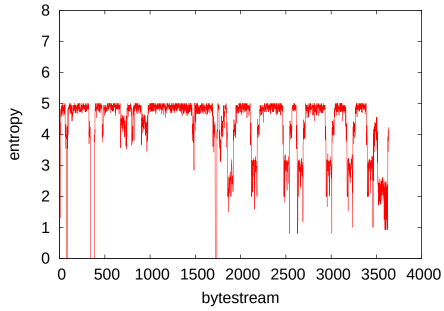


(i) o_e:1, w_e:64

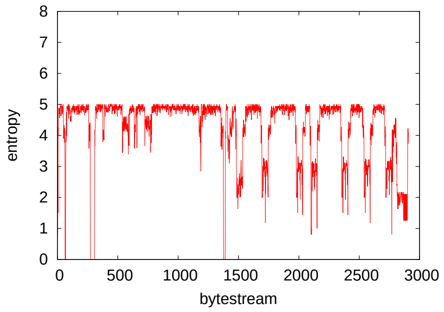
Figure 10: Entropy Function of a PDF File with selected parameters for $w_e = 64$ and o_e.



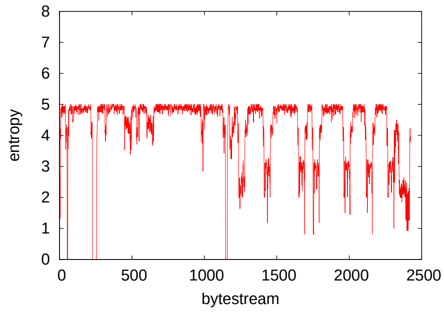
(a) o_e:28, w_e:32



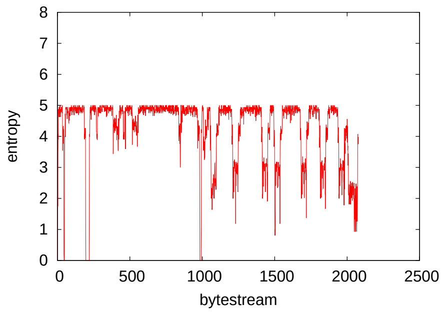
(b) o_e:16, w_e:32



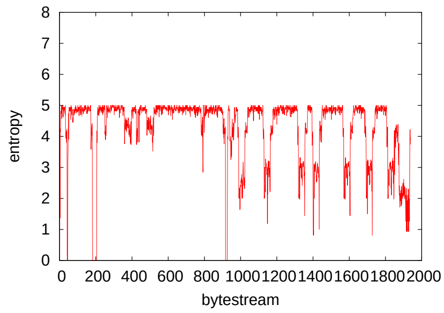
(c) o_e:12 w_e:32



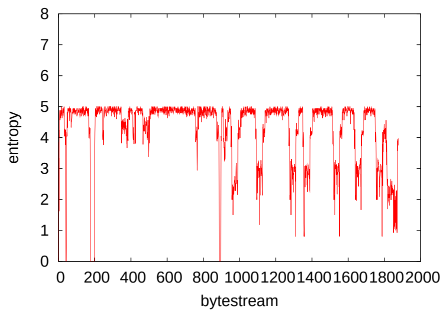
(d) o_e:8, w_e:32



(e) o_e:4 w_e:32

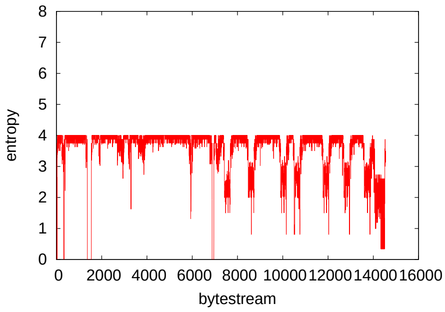


(f) o_e:2 w_e:32

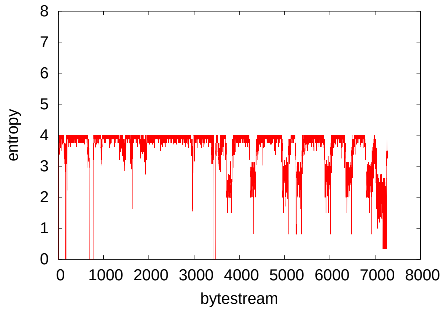


(g) o_e:1 w_e:32

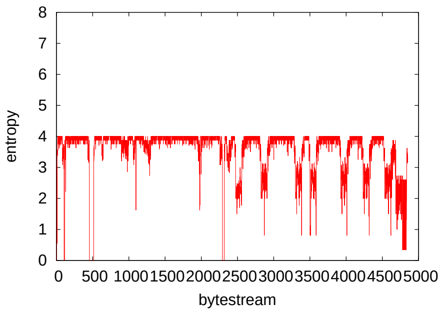
Figure 11: Entropy Function of a PDF File with selected parameters for $w_e = 32$ and o_e.



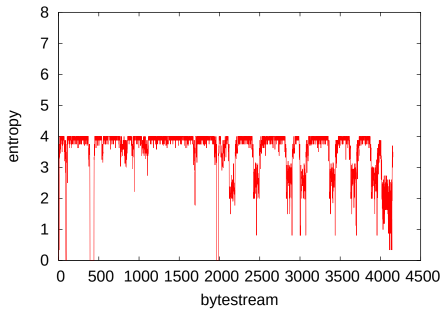
(a) o_e:12 w_e:16



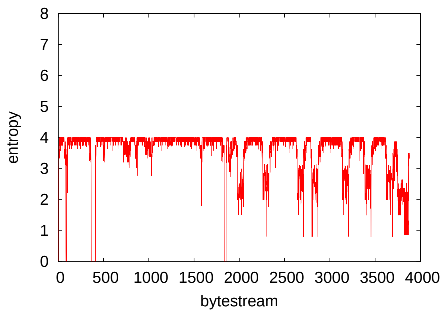
(b) o_e:8 w_e:16



(c) o_e:4 w_e:16

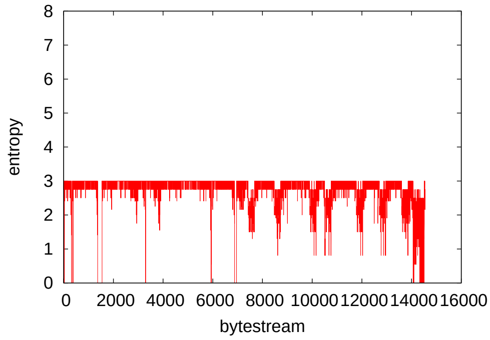


(d) o_e:2, w_e:16

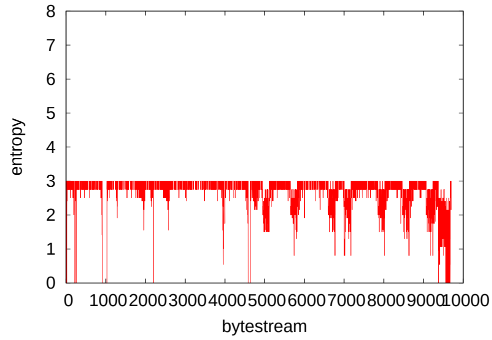


(e) o_e:1 w_e:16

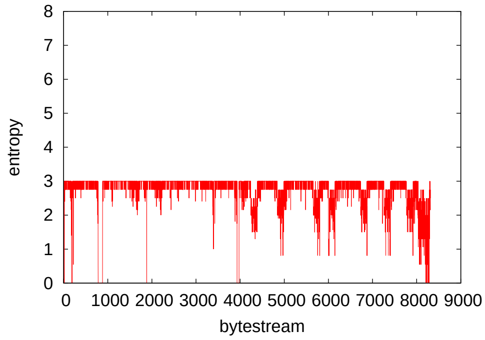
Figure 12: Entropy Function of a PDF File with selected parameters for $w_e = 16$ and o_e.



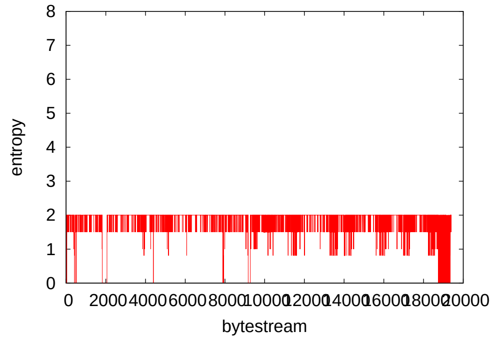
(a) o_e:4-we-8



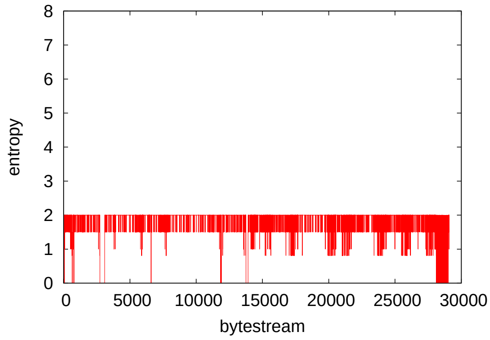
(b) o_e:2-we-8



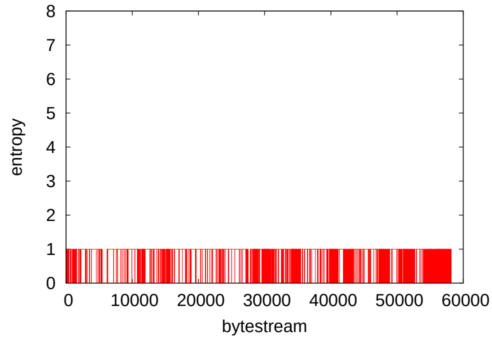
(c) o_e:1-we-8



(d) o_e:1-we-4

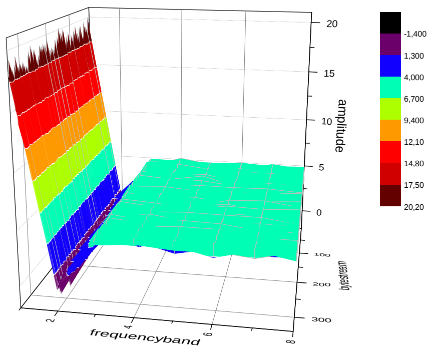


(e) o_e:2-we-4

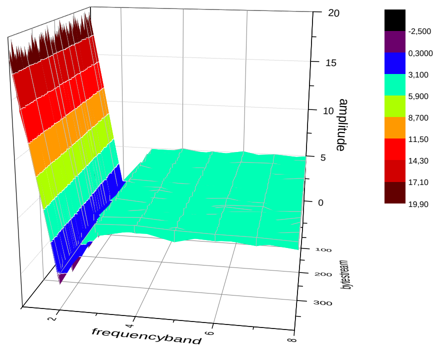


(f) o_e:1-we-2

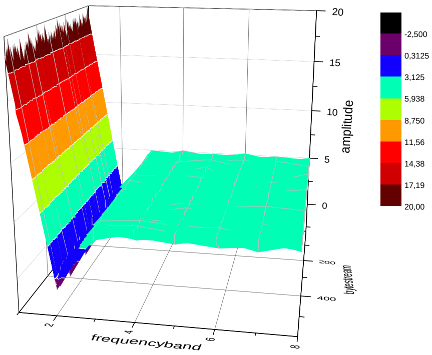
Figure 13: Entropy Function of a PDF file with selected parameters for $w_e = \{8, 4, 2\}$ and o_e.



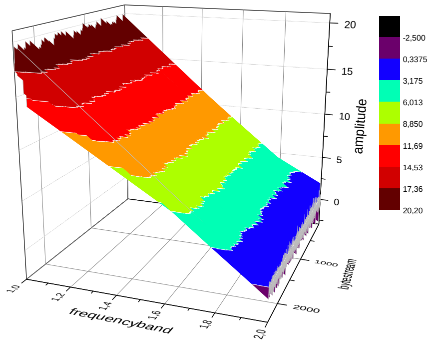
(a) w_f:16,o_f:2



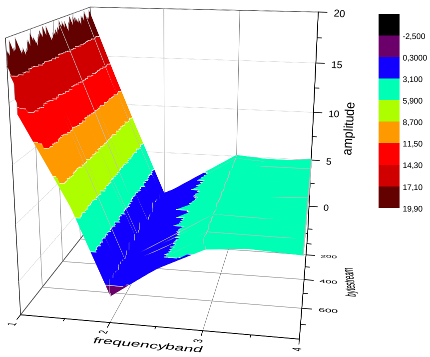
(b) w_f:16,o_f:4



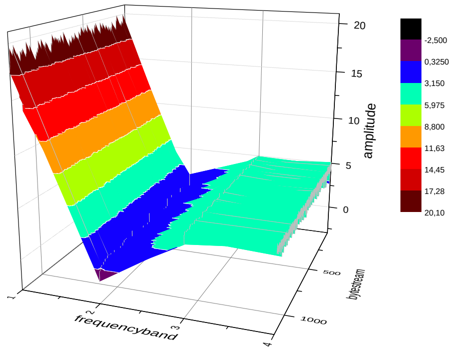
(c) w_f:16,o_f:8



(d) w_f:4,o_f:2



(e) w_f:8,o_f:2



(f) w_f:8,o_f:4

Figure 14: Entropy Spectras of a ELF-ARM-32 file with variations on w_f and o_f, for $w_e = \{64\}$, $o_e = \{8\}$.

an ANN classification algorithm.
We base our input on w_n consecutive entropy-spectra. We calculate three different statistical properties for every band of the spectras, separately for the real and imaginary values. The number of spectral bands is determined by half the size of the Fourier transform window w_f. The statistical properties that were used on every band-vector X are:

- arithmetic mean, $mean(X)$
- median, $median(X)$
- mean absolute deviation, $mad(X) = mean(|X - mean(X)|)$

We use these statistics as our input for the ANN.

w_e	o_e	w_f	o_f	w_n	overhead (%)	minimal input size (byte)
256	252	4	2	10	600	80
32	16	4	2	10	150	320
64	4	16	2	10	28.57	8400
64	8	16	4	10	33.33	6720
32	16	4	2	1	150	32
64	16	8	2	1	38.89	288

Table 1: Example of parameters and their implications on the system

parameter	description	unit
w_e	window size of the entropy calculation	byte
o_e	amount overlap of the entropy calculation	byte
w_f	window size of the STFT	entropy values
o_f	overlap of the STFT	entropy values
w_n	number of STFT units (entropy-spectra) fed to the ANN	FFT windows

Table 2: Parameters of the classification scheme

4.4 Overhead and Minimal Malware Size

Before testing our method on real world data in section 5.2, let us consider the implication of changing the systems parameters (see table 2). The system parameters affect the detection accuracy and overhead of the system. The obvious constraints are $w_e > o_e$ and $w_f > o_f$, because the overlap with the previous window must be smaller than the window itself. The overhead generated by our method should be as small as possible while delivering a high detection accuracy. We define the overhead in the percentage of bytes generated in the processing steps compared to the input data (= 100%). With

$$entropy_{overhead} = \frac{input_{size}}{w_e - o_e},$$

we calculate the overhead generated in the entropy step. With

$$fourier_{overhead} = \frac{entropy_{overhead}}{w_f - o_f} \cdot w_f,$$

we calculate the overhead generated in the Fourier transform step. The size of the basic type *double* in Java is 64-Bits (8 Byte). Thus the total overhead in bytes is given by

$$overhead_{total} = (entropy_{overhead} + fourier_{overhead}) \cdot 8.$$

The minimum amount of data that is required for the scheme prior to a classification is given by

$$minimum_{data} = (w_e - o_e) \cdot (w_f - o_f) \cdot w_n.$$

In Table 1, we show some examples of the overhead and their minimal input size. The table shows that a very high entropy overlap of $o_e = 252$ leads to a small detection size but it also leads to six times the amount of data during processing. The parameters in line 3 lead to a small overhead but the scheme requires about 8000 bytes of data before classification can take place. In Section 5, we test our scheme on different settings and for parameters shown in Table 2.

5 Evaluation of Entropy based Malware Detection

In this section we describe how we evaluate the proposed method. For a thorough test of the method, files that are typically used on a mobile phone are required. We choose to include the typical filetypes shown in table 3 in column one. We assume that typical operation of a mobile includes data transfer of those filetypes. The binary filetype selected for these tests include ELF-ARM-32. The *ELF*-format is a container format for executable code and can contain executable code of different processor architectures. Most of the current Android mobiles use an ARM-Processor. Thus the ELF-ARM-32-Format is used for native executables on those platforms.
We limit the testing to one processor architecture, to show the feasibility of our approach. Nevertheless the method is also applicable to other processor architectures.

5.1 Collecting Test-Data

Testing a statistical malware detection method can lead to a bias on the properties of the test-set, when the test-data is not carefully selected. One reason for such a bias can be the focus on non-obvious properties that are prevalent in the test-set. A typical filetype can be generated by different programs. These programs could leave certain patterns that statistical methods can focus on, while learning binary and non-binary code patterns. One example are various tools that all can produce PDF-data. In our tests we want to avoid a bias on a particular software. To circumvent these biases the test-set has to be large enough and the contents of one filetype have to stem from different programs. To obtain a representative amount of files for each type we use the Google search engine. When downloading a certain amount of files for each filetype, we assume that we get files generated by different programs for each filetype. The files from the ELF-ARM-32 were generated by the GCC-compiler[4] which is the most commonly used compiler for the ELF-ARM-32 format. We do not know how much data is needed from each filetype to avoid a bias. Testing a large parameter space is a computationally intense task. Thus we limit the set of test-data to 1MB per filetype.

5.1.1 Test-Data from the Internet

To gather data from the Internet, we conducted scripted searches. The following steps were performed for every file type in table 3.
We started with search words extracted from a large Word list of an English-German dictionary [5]. From this word list, we selected (German and English) words at random. The random selection used the Python „random.shuffle()" method [6]. These random words were used to search on Google, using the Google-option „filetype:" (e.g., „filetype:pdf") to narrow down our search results to certain file types. From the Google search results, we obtained all links from the first three page results that linked to our desired document type.
In this way we gathered 1000 links per file type. The next step was to download the files, which again was randomized (using Python „random.shuffle") in the order of links and downloaded test-data until we reached a 1 MB size limit of data per file-type.
Some sets contained large files that were larger than the 1MB limit. To make sure that we have a larger variety of files, we decided to drop files that were larger than 100 KB, which is approximately $\frac{1}{10}$ of our test-set size. In this way we made sure that the test-set for each file type contained at least 10 files. The contents of the test-data is shown in table 3. The samples of ELF-code stems from two different Debian-Linux installations.

5.1.2 Preparing and Labeling

To start the tests we have to seperate the filetype in two classes. One class includes files which contain only executable content the other another class contains only non-executable content. The ELF-File-Format contains, by definition, both types. Thus we had to strip the non-executable part from the ELF-files, keeping only the ARM-32 executable machine code. The different ELF-formatted-sections, can be extracted using the Linux tool objcopy [7]. We removed the non-executable sections of our ELF-Files and used only the executable sections (*Text-sections*) for training. While stripping down the content of the files, the size of the test-set size declined because the executable section is often only a small part of the whole ELF-file. Thus we had to add more files to keep the 1 MB per file-format limit.

[4] `http://gcc.gnu.org/`
[5] `http://www-user.tu-chemnitz.de/~fri/ding/`
[6] The Python „random.shuffle()" is based on a pseudo-random number generator.
[7] `http://www.gnu.org/software/binutils`

file type	number of files	source
doc	21	data from the Internet as described in 5.1.1
htm	90	
odt	8	
pdf	35	
ppt	15	
xls	39	
text	10	
JavaScript	14	
JPEG	13	
ELF-ARM-32	41	Debian Linux

Table 3: contents of the 1MB per file-type test-data set

parameter	value
w_e	$\{256, 32, 64\}$
o_e	$\{4, 8, 16, 32, 252\}$
w_f	$\{4, 8, 16\}$
o_f	$\{2, 4, 8\}$
w_n	$\{1, 4, 10\}$

Table 4: parameters of our scheme for the conducted tests

By definition the non-binary formats do not contain binary executable code, thus we did not have to change them.

5.2 Testing with Various Parameters

We conduct tests of our method with different parameters of $p = \{w_e, o_e, w_f, o_f, w_n\}$. Searching the best parameters leads to a multi-dimensional optimization problem. The variations of parameters are shown in table 4. To focus our tests on interesting parameters, we have to observe some practical constraints on the parameters discussed in section 4.4. The Results of our tests are shown in the tables 5,6,7 and 8.

we	oe	wf	of	wn	Is nonBinary classified as nonBinary	Is nonBinary classified as binary	Is binary classified as nonBinary	Is binary classified as binary
32	4	16	2	1	73,0243612597	26,9756387403	1,3974082951	98,6025917049
32	16	16	8	1	73,9813736903	26,0186263097	1,4776495556	98,5223504444
64	32	4	2	1	80,9505334627	19,0494665373	0,8807417331	99,1192582669
64	4	8	2	1	88,5917030568	11,4082969432	0,6827084499	99,3172915501
64	16	16	4	1	91,520979021	8,479020979	0,5610935355	99,4389064645
32	4	16	4	1	74,9872643912	25,0127356088	1,4380222841	98,5619777159
64	32	16	4	1	88,5198135198	11,4801864802	0,5968961401	99,4031038599
64	8	8	4	1	88,9983022071	11,0016977929	0,5803156917	99,4196843083
64	16	8	2	1	87,5600174596	12,4399825404	0,5461545349	99,4538454651
32	4	4	2	1	60,1323828921	39,8676171079	1,4745565015	98,5254434985
64	16	4	2	1	80,4743198021	19,5256801979	0,861511525	99,138488475
64	16	16	8	1	92,5407925408	7,4592074592	0,5093716423	99,4906283577
64	4	16	4	1	90,8196721311	9,1803278689	0,5894642591	99,4105357409
64	32	16	2	1	89,3949694086	10,6050305914	0,6545961003	99,3454038997
64	8	16	2	1	87,380952381	12,619047619	0,6174845629	99,3825154371
32	8	16	2	1	73,4725050916	26,5274949084	1,3718662953	98,6281337047
64	32	16	8	1	89,8212898213	10,1787101787	0,5518119595	99,4481880405
64	8	4	2	1	82,3968765914	17,6031234086	0,8762665243	99,1237334757
32	4	8	4	1	73,8075029706	26,1924970294	1,5111245227	98,4888754773
64	4	8	4	1	88,9737991266	11,0262008734	0,5372065261	99,4627934739
64	4	16	2	1	89,2857142857	10,7142857143	0,5396935933	99,4603064067
64	4	4	2	1	81,8115678429	18,1884321571	0,7784423692	99,2215576308
32	4	16	8	1	76,8421052632	23,1578947368	1,3788300836	98,6211699164
64	8	16	8	1	89,8639455782	10,1360544218	0,5478434468	99,4521565532
64	32	8	2	1	88,0384167637	11,9615832363	0,6904098687	99,3095901313
32	16	4	2	1	55,8141790321	44,1858209679	1,7084154049	98,2915845951
32	16	8	2	1	67,7771312191	32,2228687809	1,5200652593	98,4799347407
64	8	16	4	1	90,7142857143	9,2857142857	0,5362116992	99,4637883008
64	32	8	4	1	87,970508343	12,029491657	0,6327099085	99,3672900915
64	16	16	2	1	91,7431192661	8,2568807339	0,3621169916	99,6378830084
32	4	8	2	1	73,0651731161	26,9348268839	1,5041520865	98,4958479135

Table 5: Test results number 1

we	oe	wf	of	wn	Is nonBinary classified as nonBinary	Is nonBinary classified as binary	Is binary classified as nonBinary	Is binary classified as binary
32	16	16	4	1	74,7962747381	25,2037252619	1,6713091922	98,3286908078
32	16	16	2	1	73,9646978955	26,0353021045	1,613277623	98,386722377
64	8	8	2	1	89,1492613347	10,8507386653	0,6232590529	99,3767409471
32	16	8	4	1	68,1959262852	31,8040737148	1,561194307	98,438805693
64	16	8	4	1	89,5518044237	10,4481955763	0,5988857939	99,4011142061
64	4	16	8	1	91,3265306122	8,6734693878	0,5770283042	99,4229716958
64	16	8	2	10	96,943231441	3,056768559	0,3880597015	99,6119402985
64	8	16	4	10	100	0	0	100
32	16	4	2	10	78,5160038797	21,4839961203	1,0147234381	98,9852765619
64	16	8	4	10	97,084548105	2,915451895	0,338241146	99,661758854
64	8	8	4	10	94,8979591837	5,1020408163	0,278551532	99,721448468
64	8	4	2	10	88,9643463497	11,0356536503	0,6267409471	99,3732590529
32	16	8	4	10	84,5780795344	15,4219204656	0,7096431888	99,2903568112
32	4	8	4	10	85,5687606112	14,4312393888	0,9285051068	99,0714948932
64	32	8	2	10	96,7930029155	3,2069970845	0,19896538	99,80103462
64	32	8	4	10	94,1747572816	5,8252427184	0,3846664014	99,6153335986
64	4	8	2	10	98,9071038251	1,0928961749	0,223880597	99,776119403
32	8	8	2	10	90,1746724891	9,8253275109	0,596925832	99,403074168
64	8	16	2	10	100	0	0	100
32	8	8	4	10	85,2983988355	14,7016011645	1,0246717071	98,9753282929
64	4	16	4	10	98,9010989011	1,0989010989	0,0746268657	99,9253731343
32	4	16	4	10	93,3673469388	6,6326530612	0,348189415	99,651810585
32	8	16	8	10	88,3381924198	11,6618075802	0,6764822921	99,3235177079
32	4	16	8	10	87,074829932	12,925170068	0,8124419684	99,1875580316
32	8	16	4	10	94,3231441048	5,6768558952	0,3582089552	99,6417910448
64	4	16	2	10	100	0	0,1742160279	99,8257839721
32	16	8	2	10	87,3362445415	12,6637554585	0,7262236371	99,2737763629
64	4	16	8	10	100	0	0,0497512438	99,9502487562
64	4	8	4	10	96,7153284672	3,2846715328	0,2487562189	99,7512437811

Table 6: Test results number 2

we	oe	wf	of	wn	Is nonBinary classified as nonBinary	Is nonBinary classified as binary	Is binary classified as nonBinary	Is binary classified as binary
32	16	16	2	10	93,1972789116	6,8027210884	0,4874651811	99,5125348189
32	8	4	2	10	80,2037845706	19,7962154294	1,0545165141	98,9454834859
32	16	16	4	10	93,0029154519	6,9970845481	0,4576203741	99,5423796259
64	4	4	2	10	89,6174863388	10,3825136612	0,683994528	99,316005472
32	16	16	8	10	89,3203883495	10,6796116505	0,7825971614	99,2174028386
32	4	8	2	10	86,9897959184	13,0102040816	0,6789693593	99,3210306407
256	252	4	2	10	91,176113606	8,823886394	0,492443398	99,507556602
32	8	16	2	10	94,387755102	5,612244898	0,348189415	99,651810585
64	8	8	2	10	97,4489795918	2,5510204082	0,139275766	99,860724234
64	16	16	2	10	100	0	0,069637883	99,930362117
64	16	16	4	10	99,1228070175	0,8771929825	0,2388059701	99,7611940299
32	4	4	2	10	79,4567062818	20,5432937182	1,0213556175	98,9786443825
64	32	4	2	10	88,7487875849	11,2512124151	0,7627006234	99,2372993766
64	32	16	8	10	93,7743190661	6,2256809339	0,3714513133	99,6285486867
64	32	16	2	10	100	0	0,1857010214	99,8142989786
64	8	16	8	10	100	0	0,1393404552	99,8606595448
64	32	16	4	10	99,4152046784	0,5847953216	0,358137684	99,641862316
32	4	16	2	10	95,8333333333	4,1666666667	0,365704998	99,634295002
64	16	4	2	10	89,9563318777	10,0436681223	0,5670513331	99,4329486669
64	16	16	8	10	98,2456140351	1,7543859649	0,3184713376	99,6815286624
64	32	4	2	4	84,5168800931	15,4831199069	0,7640270593	99,2359729407
64	32	16	8	4	88,4914463453	11,5085536547	0,6685768863	99,3314231137
256	252	4	2	4	90,70828681	9,29171319	0,5667216034	99,4332783966
64	16	4	2	4	85,9138533178	14,0861466822	0,7043374453	99,2956625547
64	8	16	2	4	96,6666666667	3,3333333333	0,4874878128	99,5125121872
32	16	16	2	4	85,597826087	14,402173913	0,6778087279	99,3221912721
64	32	8	2	4	88,4749708964	11,5250291036	0,6048547553	99,3951452447
64	32	8	4	4	88,8198757764	11,1801242236	0,5995649175	99,4004350825
64	16	16	4	4	95,8041958042	4,1958041958	0,3342884432	99,6657115568
32	4	16	4	4	80,2040816327	19,7959183673	1,1699164345	98,8300835655
64	4	16	4	4	95,6140350877	4,3859649123	0,3582089552	99,6417910448

Table 7: Test results number 3

we	oe	wf	of	wn	Is nonBinary classified as nonBinary	Is nonBinary classified as binary	Is binary classified as nonBinary	Is binary classified as binary
64	16	8	2	4	92,3076923077	7,6923076923	0,4417382999	99,5582617001
32	8	8	2	4	79,8253275109	20,1746724891	1,2236614338	98,7763385662
64	4	4	2	4	85,807860262	14,192139738	0,8257063271	99,1742936729
32	16	4	2	4	68,0116391853	31,9883608147	1,2826464697	98,7173535303
64	32	16	2	4	91,553133515	8,446866485	0,6685236769	99,3314763231
64	4	8	2	4	94,3231441048	5,6768558952	0,4924638114	99,5075361886
64	8	8	2	4	90,612244898	9,387755102	0,3621169916	99,6378830084
64	4	16	2	4	96,4285714286	3,5714285714	0,3133704735	99,6866295265
64	16	16	2	4	95,9183673469	4,0816326531	0,5292479109	99,4707520891
32	8	16	4	4	79,1958041958	20,8041958042	1,2893982808	98,7106017192
32	4	16	8	4	81,1141304348	18,8858695652	1,3463324048	98,6536675952
64	8	16	4	4	96,3265306122	3,6734693878	0,6128133705	99,3871866295
64	4	16	8	4	91,8367346939	8,1632653061	0,6965174129	99,3034825871
32	4	4	2	4	72,3693143245	27,6306856755	1,4322191272	98,5677808728
32	16	16	4	4	86,3795110594	13,6204889406	1,1460405889	98,8539594111
64	32	16	4	4	89,9766899767	10,0233100233	0,4457179242	99,5542820758
64	16	8	4	4	89,7555296857	10,2444703143	0,4536410665	99,5463589335
64	8	4	2	4	85,3940217391	14,6059782609	0,756731662	99,243268338
64	8	8	4	4	89,2663043478	10,7336956522	0,5663881151	99,4336118849
64	8	16	8	4	92,0980926431	7,9019073569	0,3714710253	99,6285289747
32	16	8	2	4	77,9394644936	22,0605355064	0,9669717469	99,0330282531
32	4	8	2	4	76,883910387	23,116089613	1,3997214485	98,6002785515
32	4	8	4	4	76,6983695652	23,3016304348	1,3138347261	98,6861652739
32	4	16	2	4	76,4285714286	23,5714285714	1,2837179071	98,7162820929
32	16	8	4	4	74,9320915794	25,0679084206	1,016049874	98,983950126
32	16	16	8	4	81,7546583851	18,2453416149	0,9815885817	99,0184114183
64	16	16	8	4	95,5710955711	4,4289044289	0,3820439351	99,6179560649
32	8	8	4	4	78,812572759	21,187427241	1,2813370474	98,7186629526
32	8	16	2	4	80,6517311609	19,3482688391	1,0445682451	98,9554317549
32	8	4	2	4	72,4469013675	27,5530986325	1,2912596248	98,7087403752
64	4	8	4	4	89,8107714702	10,1892285298	0,7262236371	99,2737763629
32	8	16	8	4	79,8603026775	20,1396973225	1,2495025865	98,7504974135

Table 8: Test results number 4

5.2.1 Interpretation

The results of the tests are shown in the last section the tables 5,6,7 and 8, starting at page 29.
The tables show the classification results with accumulated results for binary and non-binary classes. There are 4 possible outcomes for a classification, which is represented by the 4 result columns. Some data can be either

- non-binary and correctly classified as non-binary or
- non-binary but classified as binary (**false positive**) or
- binary but classified as non-binary (**false negative**) or
- binary classified as binary.

The most dangerous are the false negatives, where a real-world system could not detect an attack with malware code. Wrong classification in the form of false positives can also be problematic, depending on the actions a real-world implementation chooses. A real-world system might choose to invest processing time on a false positive, draining system resources. Because this method is intended to scan any incoming data in real-time, either a high false positive or a false positive rate should be avoided.
The test-results show many parameter settings with a high detection rate. Choosing a best setting,based on the detection rates, would be easy but the detection rates should not be the only result considered. Because the results show many parameter settings with nearly equal (high) detection rates, we also have to observe the implications on a real-world system. We should not only consider the detection rates for binary and non-binary data, but also the overhead and the minimal detection size. The overhead and the minimal detection size were discussed in section 4.4. With the overhead and the minimal detection size in mind, we can not choose an absolute winner of the test. Instead we can show favourable and less favourable settings for a detection system.
In tables 6 and 7 we can see extreme results, where the settings led to a unrealistic high detection rate of 100%. Such high detection rates can not be expected outside the testing environment. These extreme results can be explained with the small amount of data to be checked with those settings. With some of the settings, the minimum amount that the method can classify is very large. This leads to a situation where larger but less chunks of the testset-data have to be classified, thus increasing the chance for a correct classification. For one of the extreme results from table 6 ($w_e = 64$, $o_e = 4$, $w_f = 16$, $o_f = 2$ and $w_n = 10$), the minimal detection size is 8400 byte. It means that the scheme has to load 8400 bytes before a decision is made if malware is prevalent. In such large chunks a decision on the binary or non-binary class is easier than deciding it on a very small chunk. In a real-world application,where data is scanned in real-time, a warning after more than 8kb of data has passed our scheme, might be too late. We assume that the classifier has memorized the exact content of some files, leading to such extreme results. The influence of system settings on the minimum detection size was explained in section 4.4. Examples for this problem are given in table 1 on page 26.
Results show that large entropy windows of $w_e = 256$ do not lead to significantly higher detection rates than smaller entropy window sizes. Instead the extremely large entropy windows dramatically increase the processing overhead. For example the setting ($w_e = 256$, $o_e = 252$, $w_f = 4$, $o_f = 2$ and $w_n = 10$) leads to an overhead of 600%.
Let us take a closer look at the implications of individual parameters on the detection rate. The selection of one parameter might have side effects on the performance of others. We discuss the effects of the parameters individually, because we want to show the effects of each individual parameter on the detection rate. The entropy window size w_e changes the amount of bytes that are considered during the creation of the entropy function. As explained above, very large entropy window settings of $w_e = 256$ lead to a huge overhead while having little effect on the detection rate. Let us take a closer look at the results of the remaining w_e settings $\{32, 64\}$. In table 9 we show the average, median and standard deviation of selected parameters for the results in the tables 5,6,7 and 8. We see that $w_e = 32$ has a higher average false positive rate with 20.08%, compared to a rate of 7.94% for the $w_e = 64$ setting. The performance for the true positives is slightly better with 99.52% average rate, when $w_e = 64$ windows are used. The overall performance seems to be better with $w_e = 64$ windows.
The parameter o_e changes the overlap with the last entropy window and thus has a direct influence on the overhead. As explained above, a large entropy window of $w_e = 256$ does not help the detection process. Thus huge overlaps, such as $o_e = 256$, can also be ruled out as ideal candidates. Table 10 shows the performance for the remaining $o_e = \{4, 8, 16, 32\}$

	classified as	is	
		binary	non-binary
$w_e = 32$	binary	[98.91 / 98.83 / 0.38]	[20.08 / 20.54 / 8.64]
	non-binary	[1.09 / 1.17 / 0.38]	[79.92 / 79.46 / 8.64]
		binary	non-binary
$w_e = 64$	binary	[99.52 / 99.46 / 0.22]	[7.94 / 8.93 / 5.03]
	non-binary	[0.48 / 0,54 / 0.22]	[92.06 / 91.07 / 5.03]

Table 9: Test-results for w_e settings $\{32, 64\}$ in the form of [average / median / standard deviation]

settings. We can see that the detection rate for binaries does not differ much, between the parameter settings. With non-binaries we can see that the $o_e = 32$ setting leads to the best average detection rate of 90.52%. When $o_e = 32$ is applied in our tests, we test this setting with $w_e = 64$, leading to an overlap of 50%. The overlap leads to a higher overhead in general. For our tests with $o_e = 32$, we have an overhead of at least 53.57 % ($w_e = 64$, $o_e = 32$, $w_f = 16$, $o_f = 2$, $w_n = 1$) and a maximum overhead of 75% with ($w_e = 64$, $o_e = 32$, $w_f = 16$, $o_f = 2$, $w_n = 10$). When looking at the result-tables, we can see that the average detection rate is slightly lower for $o_e = 16$, but the best results of $o_e = 16$ can outperform the detection rate of most of the $o_e = 32$ settings.

	classified as	is	
		binary	non-binary
$o_e = 4$	binary	[99.18 / 99.29 / 0.46]	[14.09 / 12.17 / 9.61]
	non-binary	[0.82 / 0.71 / 0.46]	[85.91 / 87.83 / 9.61]
		binary	non-binary
$o_e = 8$	binary	[99.32 / 99.39 / 0.40]	[11.70 / 10.85 / 7.50]
	non-binary	[0.68 / 0.61 / 0.40]	[88.30 / 89.15 / 7.50]
		binary	non-binary
$o_e = 16$	binary	[99.22 / 99.36 / 0.45]	[14.30 / 11.56 / 10.46]
	non-binary	[0.78 / 0.64 / 0.45]	[85.70 / 88.44 / 10.46]
		binary	non-binary
$o_e = 32$	binary	[99.44 / 99.40 / 0.19]	[9.48 / 10.89 / 4.68]
	non-binary	[0.56 / 0.60 / 0.19]	[90.52 / 89.11 / 4.68]

Table 10: Test-results for o_e settings $\{4, 8, 16, 32\}$ in the form of [average / median / standard deviation]

The parameter w_f sets the window size of the Fourier window. A higher setting leads to a better frequency resolution, while lowering the time resolution of the frequency-analysis. In table 11, we can see statistics on the test-results for w_f. We can see a decline in false positives, when the fourier window grows. The average false positive rate shrinks from 19.39% (with $w_f = 4$) down to 10.10% (with $w_f = 16$). When looking at the true positive rate, we can see only a slight increase in the detection rate, when the window becomes larger. The true-negative rate declines with growing windows, while the biggest increase is between $w_f = 4(80.61\%)$ to $w_f = 8(86.83\%)$. The results strenghten the assumption that either $w_f = 8$ or $w_f = 16$ is a good seeting for the parameter.

	classified as	is	
		binary	non-binary
$w_f = 4$	binary	[99.07 / 99.16 / 0.32]	[19.39 / 17.90 / 9.41]
	non-binary	[0.93 / 0.84 / 0.32]	[80.61 / 82.10 / 9.41]
		binary	non-binary
$w_f = 8$	binary	[99.27 / 99.39 / 0.40]	[13.17 / 11.29 / 8.01]
	non-binary	[0.73 / 0.61 / 0.40]	[86.83 / 88.71 / 8.01]
		binary	non-binary
$w_f = 16$	binary	[99.35 / 99.45 / 0.43]	[10.17 / 8.48 / 8.01]
	non-binary	[0.65 / 0.55 / 0.43]	[89.83 / 91.52 / 8.01]

Table 11: Test-results for w_f settings $\{4, 8, 16\}$ in the form of [average / median / standard deviation]

The overlap with the last Fourier window is set by the parameter o_f. Statistics on the test-results for o_f are shown in table 12. The average true-positive rate shows only a slight increase (99.24% with $o_f = 2$ to 99.31% with $o_f = 8$), when the overlap gets larger. Even the true-negative rate shows only an increase of about 3% when changing $o_f = 2$ to $o_f = 8$. Considering the increase in overhead, with larger overlaps, we assume that $o_f = 2$ is a reasonable setting.

	classified as	is	
		binary	non-binary
$o_f = 2$	binary	[99.24 / 99.32 / 0.41]	[14.17 / 11.96 / 9.75]
	non-binary	[0.76 / 0.68 / 0.41]	[85.83 / 88.04 / 9.75]
		binary	non-binary
$o_f = 4$	binary	[99.30 / 99.42 / 0.43]	[11.58 / 10.35 / 8.01]
	non-binary	[0.70 / 0.58 / 0.43]	[88.42 / 89.65 / 8.01]
		binary	non-binary
$o_f = 8$	binary	[99.31 / 99.38 / 0.40]	[10.91 / 10.16 / 7.09]
	non-binary	[0.69 / 0.62 / 0.40]	[89.09 / 89.84 / 7.09]

Table 12: Test-results for o_f settings $\{2, 4, 8\}$ in the form of [average / median / standard deviation]

Let us consider the effects of parameter w_n. When the parameter w_n is greater than 1, the entropy spectras are first accumulated and then given to the neural network for classification. The w_n parameter has direct implications on the minimal detection size, as it is a factor of the minimal-size shown in the formulas in section 4.4. The test-results show that there is no significant increase in the detection accuracy with settings larger than $w_n = 1$. Nevertheless there might be differences when classification results for individual filetypes are observed.
We continue with the assumption that the parameters ($w_e = 64$, $o_e = 16$, $w_f = 8$, $o_f = 2$ and $w_n = 1$) are a good setting to archieve a good accuracy, while having a reasonable overhead. Table 5 displays the results of the test with the parameters $w_e = 64$, $o_e = 16$, $w_f = 8$, $o_f = 2$ and $w_n = 1$. Results show a tradeoff between a low minimal detection size of only 288 bytes, a 99.45% detection rate on binaries and a 87.56% detection rate on non-binaries.
The results look promising, but we have to go into more detail and look at the classification performance for each individual filetype. In section 7 we employ our scheme on a larger set of test-data, with 10MB per filetype. The results in that section are shown on a per filetype basis. The more detailed results of the above favoured test-setting for the parameters $w_e = 64$, $o_e = 16$, $w_f = 8$, $o_f = 2$ and $w_n = 1$ can be found in table 53 on page 46.
In section 6 we apply our scheme on actual malware.

6 Application to Real World Malware samples

In this section we test our scheme on two actual malware samples. We show that our scheme is effective on real-world malware samples.

6.1 Android.RootSmart Malware

The Android Malware RootSmart uses a vulnerability[8] in the android volume manager daemon, which is widely exploited to jailbreak or root Android 2.2 and 2.3 devices. The exploit code itself is not included in the initial malware application. During execution time of the initial application, the malware loads the exploit code file *shell.zip* from a web page to avoid initial detection by AV-Programs. An in depth explanation of the RootSmart malware can be found on `http://resources.infosecinstitute.com/rootsmart-android-malware/`.
We retrieved the shellcode *shell.zip*, extracted the zip-file and tested our scanner on the malware. One of the files called **exploit** contains the GingerBreak exploit. In figure 15 we can see the entropy spectra. The spectrum shows a great amount of changes within a small range of the bytestream. This can be a hint for executable code. The detection process itself is executed by the neural-network, which can not be shown graphically. The malware was successfully detected by our method. In a real-world scenario we had prevented the installation of the malware.

[8]GingerBreak exploit `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1823`

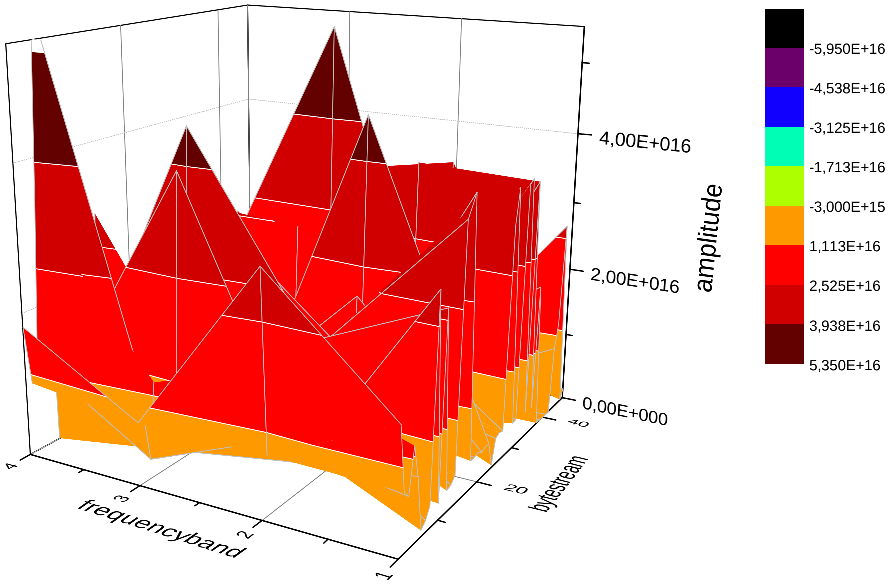


Figure 15: Entropy-Spectrum of the RootSmart exploit

6.2 Webkit Vulnerability

Webkit is an Android 2.0-2.1 based reverse Shell Exploit. It was published in 2010[9].
Part of the exploit consists of 11Kb Arm-Opcodes encoded in JavaScript as shown in the shortened Listing 1. The exploit uses a bug in the JavaScript interpreter to load and execute binary code. The binary code itself is not just embedded in JavaScript, but also encoded. The recoding of executable code can make a detection harder. The peak in the spectra marks the beginning of the codeblock, market with *„scode“* in the JavaScript code. We show in figure 16 the entropy spectra. The code was successfully detected as binary code. This attack uses binary code which is encoded in escaped letters within JavaScript. Even this encoded version of malware could be successfully detected.

Listing 1: Webkit Vulnerability exploited in JavaScript.

```
<html>
<head>
<script>
// source: http://www.exploit-db.com/exploits/15423/
// bug    =  webkit code execution CVE-2010-1807 http://cve.mitre.org/cgi-bin/cven
//            listed as a safari bug but also works on android :)
//tested =  moto droid 2.0.1 , moto droid 2.1 , emulater 2.0 - 2.1
//patched=  android 2.2
//author =   mj
// hardcoded to return a shell to 10.0.2.2 port 2222
//
function sploit(pop)
        {
        var span = document.createElement("div");
        document.getElementById("pwn").appendChild(span);
        span.innerHTML = pop;
        }
function heap()
        {
        var scode = unescape("\u3c84\u0057\u3c80\u0057\u3c7c
\u0057\u3c78\u0057\u3c74\u0057\u3c70\u0057
\u3c6c\u0057\u3c68\u0057\u3c64\u0057\u3c60
\u0057\u3c5c\u0057\u3c58\u0057\u3c54\u0057␣...␣");
        // ....
        // Listing shortened, 11kb list of encoded opcodes follows here...
```

[9] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1807

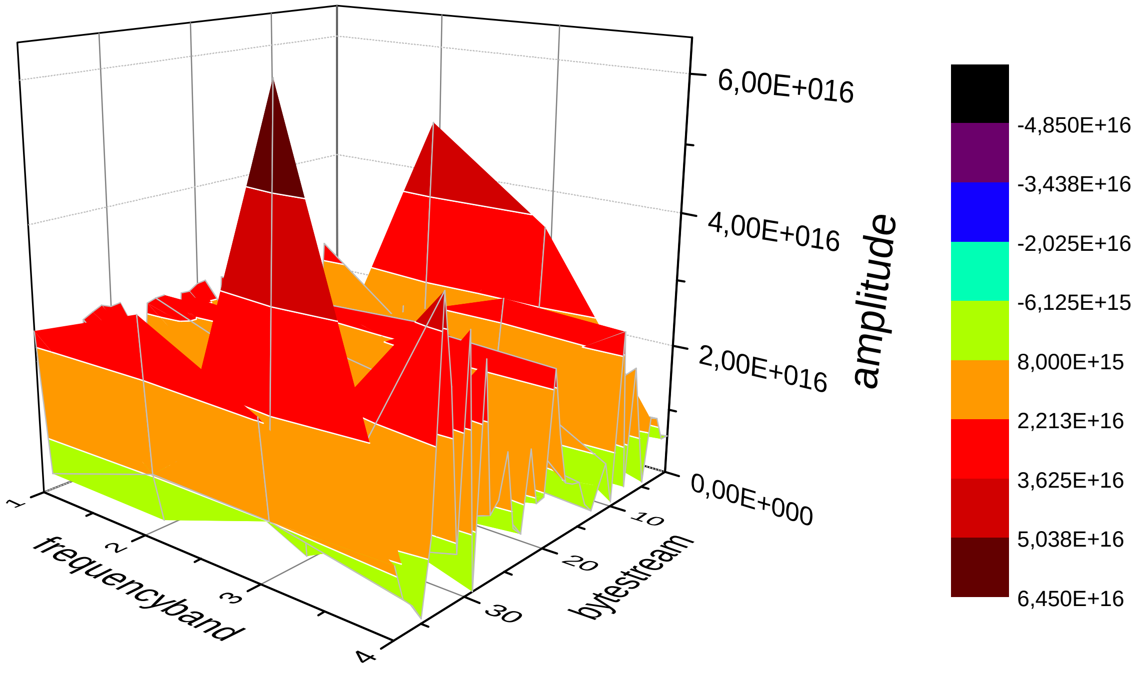


Figure 16: Entropy-Spectrum of the Webkit exploit

```
          //
           do {
             scode += scode;
          } while(scode.length < 0x1000);
          target = new Array();
          for(i = 0; i < 1000; i++)
              target[i] = scode;
          for (i = 0; i <= 1000; i++)
          {
                    if (i>999)
          {
          sploit(-parseFloat("NAN(ffffe00572c60)"));
          }
          document.write("The targets!! " + target[i]);
          document.write("<br />");
          }
}
</script>
</head>
<body id="pwn">
woot
<script>
heap();
</script>
</body>
</html>
```

7 Systematical Exploration of the Parameter Space

In this section, we show the result of 126 classification-tests. In addition to the results of section 5.2 we show performance per filetype and we use a larger testset of 10MB per filetype. A larger testset is required because we want to have a larger variance in the files included in the tests. When we have found out more about favourable parameters, future work should look at larger testsets. The conducted tests use variations on the parameters shown in table 13. We show the detection performance for each tested data type.

parameter	settings used
w_e	$\{32, 64\}$
o_e	$\{4, 8, 16\}$
w_f	$\{4, 8, 16\}$
o_f	$\{2, 4, 8\}$
w_n	$\{1, 4, 10\}$

Table 13: Parameters used during test of the parameter space

The first group of tests begins in section 7.1, with the parameter setting $w_e = 64$ and alterations on the other parameters. In section 7.2 on page 53 we use the setting $w_e = 32$. Section 7.3 on page 64 gives a discussion of the test results.

7.1 Tests with $w_e = 64$

File type	% classified as binary	% classified as non-binary
elf-arm-32	81.5981039445	18.4018960555
doc	3.05228407261	96.9477159274
htm	2.84347231716	97.1565276828
javascript	1.87573462191	98.1242653781
jpeg	0.322646611075	99.6773533889
pdf	3.43357856196	96.566421438
ppt	2.07811607574	97.9218839243
txt	0.0	100.0
xls	1.29689041049	98.7031095895

Table 14: Parameter:w_e:64, o_e:4, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	90.2784048157	9.72159518435
doc	2.66849440256	97.3315055974
htm	3.08954203691	96.9104579631
javascript	1.68473292412	98.3152670759
jpeg	0.245398773006	99.754601227
pdf	3.02892899247	96.9710710075
ppt	1.88919643249	98.1108035675
txt	0.0	100.0
xls	1.25158715763	98.7484128424

Table 15: Parameter:w_e:64, o_e:4, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.9458239278	7.05417607223
doc	2.13616632397	97.863833676
htm	3.28092959672	96.7190704033
javascript	1.02857142857	98.9714285714
jpeg	0.102249488753	99.8977505112
pdf	2.64142387484	97.3585761252
ppt	1.41695957821	98.5830404218
txt	0.0	100.0
xls	1.20167781431	98.7983221857

Table 16: Parameter:w_e:64, o_e:4, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	84.6670428894	15.3329571106
doc	2.52236197265	97.4776380273
htm	2.58040261116	97.4195973888
javascript	1.28318150651	98.7168184935
jpeg	0.207907293797	99.7920927062
pdf	4.36320353681	95.6367964632
ppt	1.8156654694	98.1843345306
txt	0.0	100.0
xls	1.1699486447	98.8300513553

Table 17: Parameter:w_e:64, o_e:4, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.7088036117	7.29119638826
doc	2.89278859336	97.1072114066
htm	8.43587640142	91.5641235986
javascript	4.13401253918	95.8659874608
jpeg	0.163599182004	99.836400818
pdf	3.76819480343	96.2318051966
ppt	2.10914843132	97.8908515687
txt	0.0	100.0
xls	2.42176870748	97.5782312925

Table 18: Parameter:w_e:64, o_e:4, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.4695259594	5.53047404063
doc	3.25677065478	96.7432293452
htm	11.3846153846	88.6153846154
javascript	2.49877511024	97.5012248898
jpeg	0.13633265167	99.8636673483
pdf	4.45578231293	95.5442176871
ppt	1.91166776533	98.0883322347
txt	0.0	100.0
xls	2.34693877551	97.6530612245

Table 19: Parameter:w_e:64, o_e:4, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.5537998495	13.4462001505
doc	2.66398592611	97.3360140739
htm	3.69568684636	96.3043131536
javascript	1.88069350573	98.1193064943
jpeg	0.220404453533	99.7795955465
pdf	4.60924569796	95.390754302
ppt	2.0539970563	97.9460029437
txt	0.0	100.0
xls	1.45111554508	98.5488844549

Table 20: Parameter:w_e:64, o_e:4, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.7012791573	7.29872084274
doc	2.82398099068	97.1760190093
htm	2.93474298214	97.0652570179
javascript	1.16248693835	98.8375130617
jpeg	0.163606616979	99.836393383
pdf	3.28314892073	96.7168510793
ppt	1.62565905097	98.374340949
txt	0.0	100.0
xls	1.16089243606	98.8391075639

Table 21: Parameter:w_e:64, o_e:4, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	95.1091045899	4.89089541008
doc	3.17641681901	96.823583181
htm	8.77392889699	91.226071103
javascript	2.44937949053	97.5506205095
jpeg	0.204498977505	99.7955010225
pdf	3.33333333333	96.6666666667
ppt	1.53778558875	98.4622144112
txt	0.0	100.0
xls	2.04081632653	97.9591836735

Table 22: Parameter:w_e:64, o_e:4, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	87.3040958778	12.6959041222
doc	2.98280687725	97.0171931228
htm	2.07353058458	97.9264694154
javascript	1.05142857143	98.9485714286
jpeg	0.230652986558	99.7693470134
pdf	4.90437266884	95.0956273312
ppt	1.86851211073	98.1314878893
txt	0.0	100.0
xls	1.07936507937	98.9206349206

Table 23: Parameter:w_e:64, o_e:4, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.0979978925	6.90200210748
doc	3.07101727447	96.9289827255
htm	10.5934907466	89.4065092534
javascript	3.84087791495	96.159122085
jpeg	0.190900413618	99.8090995864
pdf	3.61904761905	96.380952381
ppt	2.09166410335	97.9083358966
txt	0.0	100.0
xls	2.34920634921	97.6507936508

Table 24: Parameter:w_e:64, o_e:4, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	59.9472990777	40.0527009223
doc	9.04	90.96
htm	4.7885075818	95.2114924182
javascript	1.37142857143	98.6285714286
jpeg	0.397772474145	99.6022275259
pdf	7.22222222222	92.7777777778
ppt	5.76923076923	94.2307692308
txt	0.0	100.0
xls	2.14285714286	97.8571428571

Table 25: Parameter:w_e:64, o_e:4, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.294389886	11.705610114
doc	3.53691137158	96.4630886284
htm	2.9735456969	97.0264543031
javascript	1.49882445141	98.5011755486
jpeg	0.204512918399	99.7954870816
pdf	5.18332086253	94.8166791375
ppt	2.08264680683	97.9173531932
txt	0.0	100.0
xls	1.48299319728	98.5170068027

Table 26: Parameter:w_e:64, o_e:4, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	95.1219512195	4.87804878049
doc	3.12585686866	96.8741431313
htm	3.82827454197	96.171725458
javascript	2.46865203762	97.5313479624
jpeg	0.190891737115	99.8091082629
pdf	3.83673469388	96.1632653061
ppt	2.03005536515	97.9699446349
txt	0.0	100.0
xls	1.68707482993	98.3129251701

Table 27: Parameter:w_e:64, o_e:4, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	79.8870056497	20.1129943503
doc	5.9670781893	94.0329218107
htm	17.0314637483	82.9685362517
javascript	11.2745098039	88.7254901961
jpeg	0.204638472033	99.795361528
pdf	6.05442176871	93.9455782313
ppt	4.021094265	95.978905735
txt	0.341064120055	99.6589358799
xls	5.6462585034	94.3537414966

Table 28: Parameter:w_e:64, o_e:4, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.0577921589	11.9422078411
doc	3.19426038477	96.8057396152
htm	2.71157088821	97.2884291118
javascript	1.35188087774	98.6481191223
jpeg	0.254499181967	99.745500818
pdf	5.16052965717	94.8394703428
ppt	2.01678456874	97.9832154313
txt	0.0	100.0
xls	1.29699333364	98.7030066664

Table 29: Parameter:w_e:64, o_e:4, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.7453341361	7.25466586394
doc	3.19926873857	96.8007312614
htm	3.37283500456	96.6271649954
javascript	1.48902821317	98.5109717868
jpeg	0.199963642974	99.800036357
pdf	3.51895519681	96.4810448032
ppt	1.66988925998	98.33011074
txt	0.0	100.0
xls	1.45137880987	98.5486211901

Table 30: Parameter:w_e:64, o_e:4, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	75.6024096386	24.3975903614
doc	8.77513711152	91.2248628885
htm	12.3518687329	87.6481312671
javascript	6.59699542782	93.4030045722
jpeg	0.863636363636	99.1363636364
pdf	8.66213151927	91.3378684807
ppt	7.16483516484	92.8351648352
txt	0.0909504320146	99.909049568
xls	5.26077097506	94.7392290249

Table 31: Parameter:w_e:64, o_e:4, w_f:16,o_f:8, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	82.4262640449	17.5737359551
doc	3.20169733672	96.7983026633
htm	3.65975544923	96.3402445508
javascript	2.42107508532	97.5789249147
jpeg	0.327642879864	99.6723571201
pdf	3.68289637953	96.3171036205
ppt	2.14563236184	97.8543676382
txt	0.00212136318798	99.9978786368
xls	1.56064838327	98.4393516167

Table 32: Parameter:w_e:64, o_e:8, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.8876404494	10.1123595506
doc	2.37546912317	97.6245308768
htm	3.44079618918	96.5592038108
javascript	1.5602145295	98.4397854705
jpeg	0.186622555881	99.8133774441
pdf	3.31795674806	96.6820432519
ppt	1.5951775609	98.4048224391
txt	0.0	100.0
xls	1.36702217708	98.6329778229

Table 33: Parameter:w_e:64, o_e:8, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.4691011236	6.5308988764
doc	1.64196609447	98.3580339055
htm	6.36895268474	93.6310473153
javascript	2.11793387171	97.8820661283
jpeg	0.159049941682	99.8409500583
pdf	2.72986985504	97.270130145
ppt	1.4148041829	98.5851958171
txt	0.0	100.0
xls	1.4390011639	98.5609988361

Table 34: Parameter:w_e:64, o_e:8, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	87.0694195723	12.9305804277
doc	2.60043500512	97.3995649949
htm	5.24737631184	94.7526236882
javascript	2.6284512708	97.3715487292
jpeg	0.25130423718	99.7486957628
pdf	4.39930169814	95.6006983019
ppt	2.21128709826	97.7887129017
txt	0.0	100.0
xls	1.67920515506	98.3207948449

Table 35: Parameter:w_e:64, o_e:8, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.5427728614	5.45722713864
doc	2.87871033777	97.1212896622
htm	8.99578920505	91.0042107949
javascript	3.18156884257	96.8184311574
jpeg	0.114518386563	99.8854816134
pdf	2.78059928898	97.219400711
ppt	1.91929133858	98.0807086614
txt	0.0	100.0
xls	1.85396825397	98.146031746

Table 36: Parameter:w_e:64, o_e:8, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.5110642782	14.4889357218
doc	5.91810620601	94.081893794
htm	16.1455009572	83.8544990428
javascript	9.0077732053	90.9922267947
jpeg	0.190900413618	99.8090995864
pdf	5.77777777778	94.2222222222
ppt	3.56813288219	96.4318671178
txt	0.0	100.0
xls	4.60317460317	95.3968253968

Table 37: Parameter:w_e:64, o_e:8, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.6533235015	13.3466764985
doc	2.82326850904	97.176731491
htm	4.05324940456	95.9467505954
javascript	2.25811366753	97.7418863325
jpeg	0.2417610383	99.7582389617
pdf	4.54949426552	95.4505057345
ppt	2.28614778972	97.7138522103
txt	0.0	100.0
xls	1.62734102211	98.3726589779

Table 38: Parameter:w_e:64, o_e:8, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.7658378986	7.23416210142
doc	2.46502900034	97.5349709997
htm	5.31643416128	94.6835658387
javascript	1.99926856028	98.0007314397
jpeg	0.169664065151	99.8303359348
pdf	3.70746571864	96.2925342814
ppt	1.50906257689	98.4909374231
txt	0.0	100.0
xls	1.5830017777	98.4169982223

Table 39: Parameter:w_e:64, o_e:8, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.9936775553	6.00632244468
doc	2.62316058861	97.3768394114
htm	8.65589111017	91.3441088898
javascript	1.98110332216	98.0188966778
jpeg	0.16967126193	99.8303287381
pdf	2.6455026455	97.3544973545
ppt	1.53783063359	98.4621693664
txt	0.0	100.0
xls	1.43915343915	98.5608465608

Table 40: Parameter:w_e:64, o_e:8, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.1882989184	11.8117010816
doc	3.25421704732	96.7457829527
htm	4.18341521513	95.8165847849
javascript	1.65333333333	98.3466666667
jpeg	0.163301662708	99.8366983373
pdf	5.57736463966	94.4226353603
ppt	2.16018372327	97.8398162767
txt	0.0	100.0
xls	1.46666666667	98.5333333333

Table 41: Parameter:w_e:64, o_e:8, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.3628318584	6.63716814159
doc	6.77814272917	93.2218572708
htm	17.0637284098	82.9362715902
javascript	9.09090909091	90.9090909091
jpeg	0.41567695962	99.5843230404
pdf	7.58518518519	92.4148148148
ppt	5.02440424921	94.9755957508
txt	0.0	100.0
xls	4.35555555556	95.6444444444

Table 42: Parameter:w_e:64, o_e:8, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	22.8782287823	77.1217712177
doc	9.70873786408	90.2912621359
htm	12.2114668652	87.7885331348
javascript	4.2689434365	95.7310565635
jpeg	0.14847809948	99.8515219005
pdf	7.85185185185	92.1481481481
ppt	5.09691313711	94.9030868629
txt	0.0	100.0
xls	4.44444444444	95.5555555556

Table 43: Parameter:w_e:64, o_e:8, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.2014327855	10.7985672145
doc	3.55063655556	96.4493634444
htm	5.66543320148	94.3345667985
javascript	2.04809362714	97.9519063729
jpeg	0.254501495196	99.7454985048
pdf	5.37709497207	94.6229050279
ppt	2.26993110236	97.7300688976
txt	0.0	100.0
xls	1.71417687766	98.2858231223

Table 44: Parameter:w_e:64, o_e:8, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.395280236	5.60471976401
doc	4.04402354748	95.9559764525
htm	5.48749361919	94.5125063808
javascript	1.90197512802	98.098024872
jpeg	0.229065920081	99.7709340799
pdf	3.96140172676	96.0385982732
ppt	1.9438976378	98.0561023622
txt	0.0	100.0
xls	1.2192024384	98.7807975616

Table 45: Parameter:w_e:64, o_e:8, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	21.1801896733	78.8198103267
doc	16.122840691	83.877159309
htm	9.18953414167	90.8104658583
javascript	3.47666971638	96.5233302836
jpeg	0.636537237428	99.3634627626
pdf	10.1587301587	89.8412698413
ppt	11.1384615385	88.8615384615
txt	0.254614894971	99.745385105
xls	8.57142857143	91.4285714286

Table 46: Parameter:w_e:64, o_e:8, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.7062789718	11.2937210282
doc	3.33091653516	96.6690834648
htm	4.10871506954	95.8912849305
javascript	1.47507009631	98.5249299037
jpeg	0.224804886325	99.7751951137
pdf	5.66700524801	94.332994752
ppt	2.16936641378	97.8306335862
txt	0.0	100.0
xls	1.36290527385	98.6370947261

Table 47: Parameter:w_e:64, o_e:8, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	91.8235459399	8.17645406013
doc	2.66166183245	97.3383381675
htm	3.99863876127	96.0013612387
javascript	1.60936356986	98.3906364301
jpeg	0.118764845606	99.8812351544
pdf	2.65786355172	97.3421364483
ppt	1.39435695538	98.6056430446
txt	0.0	100.0
xls	1.28682695564	98.7131730444

Table 48: Parameter:w_e:64, o_e:8, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	81.307097681	18.692902319
doc	6.61262798635	93.3873720137
htm	11.2717992344	88.7282007656
javascript	6.89024390244	93.1097560976
jpeg	0.212134068731	99.7878659313
pdf	4.27603725656	95.7239627434
ppt	4.92206726825	95.0779327317
txt	0.0	100.0
xls	4.61473327688	95.3852667231

Table 49: Parameter:w_e:64, o_e:8, w_f:16,o_f:8, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	82.1666114907	17.8333885093
doc	3.19025469034	96.8097453097
htm	3.72658920027	96.2734107997
javascript	2.35337138081	97.6466286192
jpeg	0.280837604973	99.719162395
pdf	3.69994196169	96.3000580383
ppt	2.19849742981	97.8015025702
txt	0.0	100.0
xls	1.45922513241	98.5407748676

Table 50: Parameter:w_e:64, o_e:16, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.3697706615	10.6302293385
doc	2.33221231174	97.6677876883
htm	3.33564215668	96.6643578433
javascript	1.6194754989	98.3805245011
jpeg	0.178136474352	99.8218635256
pdf	3.03975623912	96.9602437609
ppt	1.59221117008	98.4077888299
txt	0.0	100.0
xls	1.26968004063	98.7303199594

Table 51: Parameter:w_e:64, o_e:16, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.4085778781	6.5914221219
doc	2.38530433193	97.6146956681
htm	6.20670798396	93.793292016
javascript	1.95899177223	98.0410082278
jpeg	0.190874386475	99.8091256135
pdf	3.17402738732	96.8259726127
ppt	1.57293497364	98.4270650264
txt	0.0	100.0
xls	1.46925448939	98.5307455106

Table 52: Parameter:w_e:64, o_e:16, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.0248306998	14.9751693002
doc	2.27278958191	97.7272104181
htm	3.40132334445	96.5986766555
javascript	1.92767307918	98.0723269208
jpeg	0.166325835037	99.833674165
pdf	3.84971161171	96.1502883883
ppt	1.80313175515	98.1968682448
txt	0.0	100.0
xls	1.34679218588	98.6532078141

Table 53: Parameter:w_e:64, o_e:16, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.9564746252	7.04352537475
doc	2.67602544418	97.3239745558
htm	7.53581975282	92.4641802472
javascript	2.08463949843	97.9153605016
jpeg	0.152705061082	99.8472949389
pdf	3.34095113723	96.6590488628
ppt	1.60286829063	98.3971317094
txt	0.0	100.0
xls	1.46930779277	98.5306922072

Table 54: Parameter:w_e:64, o_e:16, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	95.39503386	4.60496613995
doc	2.933918289	97.066081711
htm	11.1019961717	88.8980038283
javascript	3.05642633229	96.9435736677
jpeg	0.136351240796	99.8636487592
pdf	3.2380952381	96.7619047619
ppt	1.74004745584	98.2599525442
txt	0.0	100.0
xls	2.31292517007	97.6870748299

Table 55: Parameter:w_e:64, o_e:16, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	82.9591283934	17.0408716066
doc	2.80194472876	97.1980552712
htm	3.1552468967	96.8447531033
javascript	1.661268415	98.338731585
jpeg	0.192681729773	99.8073182702
pdf	4.28040264805	95.719597352
ppt	1.97711815258	98.0228818474
txt	0.0	100.0
xls	1.31867733217	98.6813226678

Table 56: Parameter:w_e:64, o_e:16, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.6077534313	7.39224656875
doc	2.90247112151	97.0975288785
htm	4.564345607	95.435654393
javascript	1.64037195695	98.359628043
jpeg	0.159965098524	99.8400349015
pdf	3.32293404919	96.6770659508
ppt	1.64499121265	98.3550087873
txt	0.0	100.0
xls	1.2842838485	98.7157161515

Table 57: Parameter:w_e:64, o_e:16, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.8892233594	6.11077664058
doc	2.86967647596	97.130323524
htm	8.45789281808	91.5421071819
javascript	1.93312434692	98.0668756531
jpeg	0.127249590983	99.872750409
pdf	3.19245419917	96.8075458008
ppt	1.37082601054	98.6291739895
txt	0.0	100.0
xls	1.63250498821	98.3674950118

Table 58: Parameter:w_e:64, o_e:16, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.0952380952	11.9047619048
doc	2.85970187448	97.1402981255
htm	3.95559525329	96.0444047467
javascript	1.46292401938	98.5370759806
jpeg	0.273589107336	99.7264108927
pdf	5.42153377349	94.5784662265
ppt	2.26364027803	97.736359722
txt	0.0	100.0
xls	1.36499269888	98.6350073011

Table 59: Parameter:w_e:64, o_e:16, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.1731984829	6.82680151707
doc	3.27617097517	96.7238290248
htm	7.35068912711	92.6493108729
javascript	2.9992684711	97.0007315289
jpeg	0.12725884449	99.8727411555
pdf	3.50431691214	96.4956830879
ppt	1.89468503937	98.1053149606
txt	0.0	100.0
xls	1.65100330201	98.348996698

Table 60: Parameter:w_e:64, o_e:16, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	70.2845100105	29.7154899895
doc	10.4286628279	89.5713371721
htm	13.7843012125	86.2156987875
javascript	4.20860018298	95.791399817
jpeg	0.4455760662	99.5544239338
pdf	8.57142857143	91.4285714286
ppt	4.73846153846	95.2615384615
txt	0.0	100.0
xls	2.92063492063	97.0793650794

Table 61: Parameter:w_e:64, o_e:16, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	87.4029257721	12.5970742279
doc	3.25729326607	96.7427067339
htm	1.84293995406	98.1570600459
javascript	0.987460815047	99.012539185
jpeg	0.152696733381	99.8473032666
pdf	5.77352124939	94.2264787506
ppt	1.78213645471	98.2178635453
txt	0.0	100.0
xls	0.90335219852	99.0966478015

Table 62: Parameter:w_e:64, o_e:16, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.2355491329	13.7644508671
doc	3.46567229656	96.5343277034
htm	12.7105666156	87.2894333844
javascript	9.05956112853	90.9404388715
jpeg	0.043630017452	99.9563699825
pdf	2.63387026556	97.3661297344
ppt	1.81396329888	98.1860367011
txt	0.0	100.0
xls	3.09098824554	96.9090117545

Table 63: Parameter:w_e:64, o_e:16, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	18.1571815718	81.8428184282
doc	10.5320899616	89.4679100384
htm	8.15098468271	91.8490153173
javascript	3.13479623824	96.8652037618
jpeg	0.38188761593	99.6181123841
pdf	5.38922155689	94.6107784431
ppt	5.16877637131	94.8312236287
txt	0.0545553737043	99.9454446263
xls	2.8307022319	97.1692977681

Table 64: Parameter:w_e:64, o_e:16, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.7204430532	13.2795569468
doc	3.21696216414	96.7830378359
htm	3.14265922928	96.8573407707
javascript	1.40536022151	98.5946397785
jpeg	0.199963642974	99.800036357
pdf	5.57570920699	94.424290793
ppt	1.9016485641	98.0983514359
txt	0.0	100.0
xls	1.17544623422	98.8245537658

Table 65: Parameter:w_e:64, o_e:16, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.2692029858	5.73079701421
doc	3.42205323194	96.5779467681
htm	7.07306402217	92.9269359778
javascript	2.52873563218	97.4712643678
jpeg	0.363583478767	99.6364165212
pdf	3.86010738645	96.1398926136
ppt	2.02474690664	97.9752530934
txt	0.0	100.0
xls	2.24931069511	97.7506893049

Table 66: Parameter:w_e:64, o_e:16, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.7621914509	5.23780854907
doc	5.4113345521	94.5886654479
htm	11.6338439096	88.3661560904
javascript	4.17972831766	95.8202716823
jpeg	0.327272727273	99.6727272727
pdf	5.80551523948	94.1944847605
ppt	3.69198312236	96.3080168776
txt	0.0	100.0
xls	3.15674891147	96.8432510885

Table 67: Parameter:w_e:64, o_e:16, w_f:16,o_f:8, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	80.0405288818	19.9594711182
doc	2.78419164011	97.2158083599
htm	3.81191938708	96.1880806129
javascript	2.49440608757	97.5055939124
jpeg	0.27629163309	99.7237083669
pdf	3.12619023149	96.8738097685
ppt	2.32500922623	97.6749907738
txt	0.000606097339233	99.9993939027
xls	1.49699512691	98.5030048731

Table 68: Parameter:w_e:64, o_e:32, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.430176565	10.569823435
doc	2.53442183502	97.465578165
htm	3.63574501179	96.3642549882
javascript	2.0930556523	97.9069443477
jpeg	0.220552593311	99.7794474067
pdf	3.47746179145	96.5225382086
ppt	1.74098460529	98.2590153947
txt	0.0	100.0
xls	1.49701322886	98.5029867711

Table 69: Parameter:w_e:64, o_e:32, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.9574638844	7.04253611557
doc	2.79639332277	97.2036066772
htm	5.94847490582	94.0515250942
javascript	2.7949499347	97.2050500653
jpeg	0.206010664081	99.7939893359
pdf	3.88149939541	96.1185006046
ppt	1.78090216755	98.2190978325
txt	0.00606097339233	99.9939390266
xls	1.75343128363	98.2465687164

Table 70: Parameter:w_e:64, o_e:32, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	84.0125203154	15.9874796846
doc	2.37969038437	97.6203096156
htm	3.07498815209	96.9250118479
javascript	1.99555961865	98.0044403814
jpeg	0.192681729773	99.8073182702
pdf	3.56035186361	96.4396481364
ppt	1.74334821272	98.2566517873
txt	0.0	100.0
xls	1.40571718784	98.5942828122

Table 71: Parameter:w_e:64, o_e:32, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.7303154346	10.2696845654
doc	1.74733148121	98.2526685188
htm	4.16332482683	95.8366751732
javascript	1.62992372793	98.3700762721
jpeg	0.065440267578	99.9345597324
pdf	2.42327504897	97.576724951
ppt	1.37785588752	98.6221441125
txt	0.0	100.0
xls	1.14633969383	98.8536603062

Table 72: Parameter:w_e:64, o_e:32, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.7989163155	6.20108368453
doc	2.32133065253	97.6786693475
htm	8.69485964273	91.3051403573
javascript	2.45559038662	97.5444096134
jpeg	0.0908925649882	99.909107435
pdf	3.13803736623	96.8619626338
ppt	1.63444639719	98.3655536028
txt	0.0	100.0
xls	1.77761654272	98.2223834573

Table 73: Parameter:w_e:64, o_e:32, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.3972712681	14.6027287319
doc	2.8293265423	97.1706734577
htm	3.76459723184	96.2354027682
javascript	2.27419158236	97.7258084176
jpeg	0.220552593311	99.7794474067
pdf	3.92367870573	96.0763212943
ppt	1.92611858986	98.0738814101
txt	0.0	100.0
xls	1.53208058236	98.4679194176

Table 74: Parameter:w_e:64, o_e:32, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	90.4414125201	9.55858747994
doc	2.47112150899	97.528878491
htm	4.11218587469	95.8878141253
javascript	1.40707718027	98.5929228197
jpeg	0.184197770238	99.8158022298
pdf	3.48730350665	96.5126964933
ppt	1.48092604743	98.5190739526
txt	0.0	100.0
xls	1.20924833124	98.7907516688

Table 75: Parameter:w_e:64, o_e:32, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	92.3956661316	7.60433386838
doc	2.77845478918	97.2215452108
htm	10.5237574432	89.4762425568
javascript	3.83141762452	96.1685823755
jpeg	0.169655841008	99.830344159
pdf	3.94195888755	96.0580411125
ppt	1.62858816637	98.3714118336
txt	0.0	100.0
xls	2.16471157335	97.8352884267

Table 76: Parameter:w_e:64, o_e:32, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.8206334715	14.1793665285
doc	2.73382522284	97.2661747772
htm	2.77317000553	97.2268299945
javascript	1.23727677211	98.7627232279
jpeg	0.190872073295	99.8091279267
pdf	5.49324982014	94.5067501799
ppt	1.80841466415	98.1915853358
txt	0.0	100.0
xls	1.15969018496	98.840309815

Table 77: Parameter:w_e:64, o_e:32, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	91.5425681371	8.45743186288
doc	2.32042313598	97.679576864
htm	6.61902331121	93.3809766888
javascript	1.38990490124	98.6100950988
jpeg	0.0848320325755	99.9151679674
pdf	2.6240054173	97.3759945827
ppt	1.23031496063	98.7696850394
txt	0.0	100.0
xls	1.11750761937	98.8824923806

Table 78: Parameter:w_e:64, o_e:32, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	13.6331693605	86.3668306395
doc	8.31911262799	91.680887372
htm	8.12420246704	91.875797533
javascript	1.28048780488	98.7195121951
jpeg	0.212134068731	99.7878659313
pdf	5.63082133785	94.3691786622
ppt	4.38884331419	95.6111566858
txt	0.0	100.0
xls	1.82049110923	98.1795088908

Table 79: Parameter:w_e:64, o_e:32, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.1365278112	13.8634721888
doc	2.76000731128	97.2399926887
htm	3.0040102078	96.9959897922
javascript	1.31654563502	98.683454365
jpeg	0.159965098524	99.8400349015
pdf	5.60089962636	94.3991003736
ppt	1.77855887522	98.2214411248
txt	0.0	100.0
xls	1.18265916924	98.8173408308

Table 80: Parameter:w_e:64, o_e:32, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	94.8711774621	5.12882253792
doc	4.51886516525	95.4811348347
htm	11.7106606388	88.2893393612
javascript	5.01567398119	94.9843260188
jpeg	0.305410122164	99.6945898778
pdf	4.30996952547	95.6900304745
ppt	2.65748031496	97.342519685
txt	0.0	100.0
xls	2.93135974459	97.0686402554

Table 81: Parameter:w_e:64, o_e:32, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	91.5713425647	8.42865743528
doc	6.21572212066	93.7842778793
htm	15.020051039	84.979948961
javascript	6.06060606061	93.9393939394
jpeg	0.327272727273	99.6727272727
pdf	5.11611030479	94.8838896952
ppt	3.05799648506	96.9420035149
txt	0.0	100.0
xls	4.20899854862	95.7910014514

Table 82: Parameter:w_e:64, o_e:32, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.115012641	13.884987359
doc	2.80749640533	97.1925035947
htm	3.90083849799	96.099161502
javascript	1.4662858735	98.5337141265
jpeg	0.186625948278	99.8133740517
pdf	5.44135429262	94.5586457074
ppt	1.82073813708	98.1792618629
txt	0.0	100.0
xls	1.29873270775	98.7012672923

Table 83: Parameter:w_e:64, o_e:32, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.0325895007	6.96741049928
doc	3.5776954572	96.4223045428
htm	8.51643009916	91.4835699008
javascript	3.24602953469	96.7539704653
jpeg	0.261780104712	99.7382198953
pdf	4.35329399245	95.6467060075
ppt	2.13723284589	97.8627671541
txt	0.0	100.0
xls	2.29273483603	97.707265164

Table 84: Parameter:w_e:64, o_e:32, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	93.095142513	6.90485748695
doc	7.0679990251	92.9320009749
htm	12.4696159456	87.5303840544
javascript	5.01567398119	94.9843260188
jpeg	0.339393939394	99.6606060606
pdf	6.14268440145	93.8573155985
ppt	3.32786501055	96.6721349895
txt	0.0	100.0
xls	4.86211901306	95.1378809869

Table 85: Parameter:w_e:64, o_e:32, w_f:16,o_f:8, w_n:10

7.2 Tests with $w_e = 32$

File type	% classified as binary	% classified as non-binary
elf-arm-32	61.3581103026	38.6418896974
doc	2.54756358247	97.4524364175
htm	2.80059329197	97.199406708
javascript	5.17955906328	94.8204409367
jpeg	0.241753347966	99.758246652
pdf	3.06920794122	96.9307920588
ppt	1.98516635315	98.0148336469
txt	0.00053033235929	99.9994696676
xls	1.92406375809	98.0759362419

Table 86: Parameter:w_e:32, o_e:4, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	75.941011236	24.058988764
doc	2.49482909355	97.5051709065
htm	2.06273258905	97.937267411
javascript	4.24488054608	95.7551194539
jpeg	0.203583925353	99.7964160746
pdf	3.20150659134	96.7984934087
ppt	1.23016361176	98.7698363882
txt	0.0	100.0
xls	1.99128153039	98.0087184696

Table 87: Parameter:w_e:32, o_e:4, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.0077247191	13.9922752809
doc	2.54811024042	97.4518897596
htm	3.81711855396	96.182881446
javascript	4.06064299863	95.9393570014
jpeg	0.180256600573	99.8197433994
pdf	3.35925514469	96.6407448553
ppt	1.39935414424	98.6006458558
txt	0.0	100.0
xls	2.23256798222	97.7674320178

Table 88: Parameter:w_e:32, o_e:4, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	69.7258578464	30.2741421536
doc	2.82269027171	97.1773097283
htm	1.78787878788	98.2121212121
javascript	3.87393152626	96.1260684737
jpeg	0.181317894804	99.8186821052
pdf	3.43109249032	96.5689075097
ppt	1.35781396565	98.6421860344
txt	0.0	100.0
xls	1.82357795835	98.1764220416

Table 89: Parameter:w_e:32, o_e:4, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	84.3568945539	15.6431054461
doc	2.5460593654	97.4539406346
htm	2.51355661882	97.4864433812
javascript	3.58383616749	96.4161638325
jpeg	0.152691182084	99.8473088179
pdf	2.7677267822	97.2322732178
ppt	1.13182013902	98.868179861
txt	0.0	100.0
xls	2.07592686643	97.9240731336

Table 90: Parameter:w_e:32, o_e:4, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.8843824072	13.1156175928
doc	2.54318618042	97.4568138196
htm	10.350877193	89.649122807
javascript	6.56	93.44
jpeg	0.190870049308	99.8091299507
pdf	3.11061736232	96.8893826377
ppt	1.64539443334	98.3546055667
txt	0.0	100.0
xls	3.71428571429	96.2857142857

Table 91: Parameter:w_e:32, o_e:4, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	72.5245786517	27.4754213483
doc	2.94155276457	97.0584472354
htm	2.20733652313	97.7926634769
javascript	4.39724524622	95.6027547538
jpeg	0.197221927685	99.8027780723
pdf	3.90190228316	96.0980977168
ppt	1.34806811076	98.6519318892
txt	0.0	100.0
xls	1.95001692907	98.0499830709

Table 92: Parameter:w_e:32, o_e:4, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	82.3525280899	17.6474719101
doc	2.63988399864	97.3601160014
htm	3.1941136441	96.8058863559
javascript	3.57752315943	96.4224768406
jpeg	0.173898290707	99.8261017093
pdf	3.11904862669	96.8809513733
ppt	1.36143689002	98.63856311
txt	0.0	100.0
xls	2.11613340105	97.883866599

Table 93: Parameter:w_e:32, o_e:4, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	89.220505618	10.779494382
doc	2.53758396418	97.4624160358
htm	8.01701222754	91.9829877725
javascript	4.76916044492	95.2308395551
jpeg	0.190859930018	99.80914007
pdf	3.34356152788	96.6564384721
ppt	1.92741439409	98.0725856059
txt	0.0	100.0
xls	3.04729658237	96.9527034176

Table 94: Parameter:w_e:32, o_e:4, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	77.7299821791	22.2700178209
doc	3.31753554502	96.682464455
htm	2.28871273864	97.7112872614
javascript	3.26382592928	96.7361740707
jpeg	0.215253293747	99.7847467063
pdf	4.32528514294	95.6747148571
ppt	1.25224255472	98.7477574453
txt	0.0	100.0
xls	2.10346998482	97.8965300152

Table 95: Parameter:w_e:32, o_e:4, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	87.3156342183	12.6843657817
doc	2.64218540081	97.3578145992
htm	8.70924519875	91.2907548013
javascript	4.48047791764	95.5195220824
jpeg	0.192992874109	99.8070071259
pdf	3.46615316249	96.5338468375
ppt	2.2534806947	97.7465193053
txt	0.0	100.0
xls	2.81481481481	97.1851851852

Table 96: Parameter:w_e:32, o_e:4, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	37.6152427781	62.3847572219
doc	9.51847704367	90.4815229563
htm	12.058057313	87.941942687
javascript	4.74666666667	95.2533333333
jpeg	0.519673348181	99.4803266518
pdf	4.92592592593	95.0740740741
ppt	5.382131324	94.617868676
txt	0.0371333085778	99.9628666914
xls	3.59259259259	96.4074074074

Table 97: Parameter:w_e:32, o_e:4, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	79.584957337	20.415042663
doc	3.38088536336	96.6191146366
htm	2.39242081087	97.6075791891
javascript	3.43755714025	96.5624428598
jpeg	0.209950375366	99.7900496246
pdf	5.15473734328	94.8452626567
ppt	1.57158234661	98.4284176534
txt	0.0	100.0
xls	2.00298384281	97.9970161572

Table 98: Parameter:w_e:32, o_e:4, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.1567635904	13.8432364096
doc	2.55885363357	97.4411463664
htm	8.71506954192	91.2849304581
javascript	4.51636496617	95.4836350338
jpeg	0.0763455910421	99.923654409
pdf	2.78059928898	97.219400711
ppt	1.45177165354	98.5482283465
txt	0.0	100.0
xls	2.64126984127	97.3587301587

Table 99: Parameter:w_e:32, o_e:4, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	51.1591148577	48.8408851423
doc	4.19065898912	95.8093410109
htm	9.50861518826	90.4913848117
javascript	2.9263831733	97.0736168267
jpeg	0.445434298441	99.5545657016
pdf	4.31746031746	95.6825396825
ppt	4.64472470009	95.3552752999
txt	0.0	100.0
xls	2.4126984127	97.5873015873

Table 100: Parameter:w_e:32, o_e:4, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	76.7021313951	23.2978686049
doc	3.25614124872	96.7438587513
htm	1.56515821708	98.4348417829
javascript	2.97120219412	97.0287978059
jpeg	0.139966916911	99.8600330831
pdf	4.49870921325	95.5012907867
ppt	1.27942261954	98.7205773805
txt	0.0	100.0
xls	1.88551476034	98.1144852397

Table 101: Parameter:w_e:32, o_e:4, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.4052535468	14.5947464532
doc	2.67826680314	97.3217331969
htm	6.63490983328	93.3650901667
javascript	4.08387175424	95.9161282458
jpeg	0.127248048863	99.8727519511
pdf	3.09801929914	96.9019807009
ppt	1.4598540146	98.5401459854
txt	0.0	100.0
xls	2.35333954118	97.6466604588

Table 102: Parameter:w_e:32, o_e:4, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	88.7600983491	11.2399016509
doc	5.07570910642	94.9242908936
htm	13.7175669928	86.2824330072
javascript	6.06522401707	93.9347759829
jpeg	0.33934252386	99.6606574761
pdf	6.13756613757	93.8624338624
ppt	4.18289932335	95.8171006766
txt	0.0	100.0
xls	4.25396825397	95.746031746

Table 103: Parameter:w_e:32, o_e:4, w_f:16,o_f:8, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	60.5179405768	39.4820594232
doc	2.65242899441	97.3475710056
htm	2.88353314833	97.1164668517
javascript	5.41647077261	94.5835292274
jpeg	0.238117959275	99.7618820407
pdf	3.09413909116	96.9058609088
ppt	1.94364996727	98.0563500327
txt	0.0	100.0
xls	1.9081131083	98.0918868917

Table 104: Parameter:w_e:32, o_e:8, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	75.3002076624	24.6997923376
doc	2.34309317542	97.6569068246
htm	2.48072471429	97.5192752857
javascript	4.617875986	95.382124014
jpeg	0.210855418621	99.7891445814
pdf	3.18485200232	96.8151479977
ppt	1.30748818164	98.6925118184
txt	0.0	100.0
xls	2.07316851976	97.9268314802

Table 105: Parameter:w_e:32, o_e:8, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	81.5213302235	18.4786697765
doc	1.72264107836	98.2773589216
htm	3.29915698337	96.7008430166
javascript	3.59801488834	96.4019851117
jpeg	0.081799591002	99.918200409
pdf	2.22181917113	97.7781808289
ppt	0.984139536927	99.0158604631
txt	0.0	100.0
xls	1.95891715413	98.0410828459

Table 106: Parameter:w_e:32, o_e:8, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	69.3068189513	30.6931810487
doc	2.49208372743	97.5079162726
htm	1.98496240602	98.015037594
javascript	4.12952769017	95.8704723098
jpeg	0.154051695932	99.8459483041
pdf	3.24155943086	96.7584405691
ppt	1.3602214314	98.6397785686
txt	0.0	100.0
xls	1.8092776493	98.1907223507

Table 107: Parameter:w_e:32, o_e:8, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	83.2866817156	16.7133182844
doc	2.78554586829	97.2144541317
htm	4.12314759118	95.8768524088
javascript	3.65175143014	96.3482485699
jpeg	0.201766822991	99.798233177
pdf	3.17771248232	96.8222875177
ppt	1.80840407023	98.1915959298
txt	0.0	100.0
xls	2.17663383577	97.8233661642

Table 108: Parameter:w_e:32, o_e:8, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	83.0248306998	16.9751693002
doc	3.13913639479	96.8608636052
htm	10.1982228298	89.8017771702
javascript	7.93495297806	92.0650470219
jpeg	0.177232447171	99.8227675528
pdf	2.38062848592	97.6193715141
ppt	1.70027678924	98.2997232108
txt	0.0136388434261	99.9863611566
xls	3.5913481159	96.4086518841

Table 109: Parameter:w_e:32, o_e:8, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	70.298251422	29.701748578
doc	2.88594222632	97.1140577737
htm	2.25655046708	97.7434495329
javascript	4.35935275757	95.6406472424
jpeg	0.18358963173	99.8164103683
pdf	4.20505658735	95.7949434127
ppt	1.36285751944	98.6371424806
txt	0.0227287191003	99.9772712809
xls	1.9344306897	98.0655693103

Table 110: Parameter:w_e:32, o_e:8, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	80.2865226028	19.7134773972
doc	2.65389676853	97.3461032315
htm	2.83256170027	97.1674382997
javascript	3.55762198307	96.4423780169
jpeg	0.134511215327	99.8654887847
pdf	3.12681369704	96.873186303
ppt	1.1563741169	98.8436258831
txt	0.0	100.0
xls	2.02423275049	97.9757672495

Table 111: Parameter:w_e:32, o_e:8, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.3506395786	13.6493604214
doc	2.55894717602	97.441052824
htm	7.66496536639	92.3350346336
javascript	4.16612250229	95.8338774977
jpeg	0.15451736048	99.8454826395
pdf	2.9745170944	97.0254829056
ppt	1.52021089631	98.4797891037
txt	0.0	100.0
xls	2.69363323055	97.3063667695

Table 112: Parameter:w_e:32, o_e:8, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	74.6971452649	25.3028547351
doc	3.12180143296	96.878198567
htm	1.72892277266	98.2710772273
javascript	3.15414152496	96.845858475
jpeg	0.146329049497	99.8536709505
pdf	4.04380257102	95.956197429
ppt	1.28863601415	98.7113639859
txt	0.0	100.0
xls	1.89505761356	98.1049423864

Table 113: Parameter:w_e:32, o_e:8, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	82.3640960809	17.6359039191
doc	2.31576253838	97.6842374616
htm	6.89039173153	93.1096082685
javascript	4.71749862863	95.2825013714
jpeg	0.101794121389	99.8982058786
pdf	3.23768410361	96.7623158964
ppt	1.66092519685	98.3390748031
txt	0.0	100.0
xls	2.52698412698	97.473015873

Table 114: Parameter:w_e:32, o_e:8, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	71.9704952582	28.0295047418
doc	2.68714011516	97.3128598848
htm	15.1563497128	84.8436502872
javascript	7.04160951075	92.9583904893
jpeg	0.286350620426	99.7136493796
pdf	4.69841269841	95.3015873016
ppt	3.87573054445	96.1242694556
txt	0.0	100.0
xls	4.34920634921	95.6507936508

Table 115: Parameter:w_e:32, o_e:8, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	76.532731377	23.467268623
doc	3.22138450994	96.7786154901
htm	1.38076229015	98.6192377098
javascript	3.10700152803	96.892998472
jpeg	0.212678936605	99.7873210634
pdf	4.28501469148	95.7149853085
ppt	1.37871039173	98.6212896083
txt	0.0	100.0
xls	1.93720411384	98.0627958862

Table 116: Parameter:w_e:32, o_e:8, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	86.7256637168	13.2743362832
doc	2.69796007896	97.302039921
htm	7.60144372744	92.3985562726
javascript	4.02821316614	95.9717868339
jpeg	0.141797556719	99.8582024433
pdf	3.26477309827	96.7352269017
ppt	1.52905198777	98.4709480122
txt	0.0	100.0
xls	2.50326512843	97.4967348716

Table 117: Parameter:w_e:32, o_e:8, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	25.2821670429	74.7178329571
doc	3.45489443378	96.5451055662
htm	7.54716981132	92.4528301887
javascript	4.27115987461	95.7288401254
jpeg	0.163621488956	99.836378511
pdf	2.72108843537	97.2789115646
ppt	2.21460585289	97.7853941471
txt	0.0	100.0
xls	1.76870748299	98.231292517

Table 118: Parameter:w_e:32, o_e:8, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	75.1339312587	24.8660687413
doc	3.04320807136	96.9567919286
htm	1.85924427189	98.1407557281
javascript	3.4008985477	96.5991014523
jpeg	0.170868703761	99.8291312962
pdf	4.03010791693	95.9698920831
ppt	1.28644487795	98.7135551221
txt	0.0	100.0
xls	1.92994866772	98.0700513323

Table 119: Parameter:w_e:32, o_e:8, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	87.7076811943	12.2923188057
doc	4.51089340547	95.4891065945
htm	10.1130149471	89.8869850529
javascript	8.21230801379	91.7876919862
jpeg	0.18177852105	99.818221479
pdf	4.90459261409	95.0954073859
ppt	1.54657293497	98.453427065
txt	0.00727378527786	99.9927262147
xls	3.15629081411	96.8437091859

Table 120: Parameter:w_e:32, o_e:8, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	79.9518362432	20.0481637568
doc	5.17272893438	94.8272710656
htm	15.475756471	84.524243529
javascript	8.85579937304	91.144200627
jpeg	0.272677694965	99.727322305
pdf	4.46218030111	95.5378196989
ppt	3.3216168717	96.6783831283
txt	0.0	100.0
xls	3.68220569563	96.3177943044

Table 121: Parameter:w_e:32, o_e:8, w_f:16,o_f:8, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	53.60566192	46.39433808
doc	2.39761183118	97.6023881688
htm	2.68636403681	97.3136359632
javascript	5.05097554393	94.9490244561
jpeg	0.235694009797	99.7643059902
pdf	2.57420175865	97.4257982413
ppt	1.65719156115	98.3428084388
txt	0.000303044996121	99.999696955
xls	1.7076886801	98.2923113199

Table 122: Parameter:w_e:32, o_e:16, w_f:4,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	72.4102646416	27.5897353584
doc	2.16397996856	97.8360200314
htm	2.13378698584	97.8662130142
javascript	4.93304776333	95.0669522367
jpeg	0.189040498291	99.8109595017
pdf	2.98289099813	97.0171090019
ppt	1.37426044168	98.6257395583
txt	0.00121217998448	99.99878782
xls	2.01816225106	97.9818377489

Table 123: Parameter:w_e:32, o_e:16, w_f:4,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	80.2668539326	19.7331460674
doc	1.89472401608	98.1052759839
htm	2.1629503615	97.8370496385
javascript	3.78303077794	96.2169692221
jpeg	0.163596703829	99.8364032962
pdf	2.99860951575	97.0013904843
ppt	1.12471442798	98.875285572
txt	0.0	100.0
xls	1.90453157592	98.0954684241

Table 124: Parameter:w_e:32, o_e:16, w_f:4,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	66.5352494169	33.4647505831
doc	2.78359088708	97.2164091129
htm	2.2100504894	97.7899495106
javascript	4.23261766703	95.767382333
jpeg	0.196311881413	99.8036881186
pdf	3.47319833504	96.526801665
ppt	1.98232079152	98.0176792085
txt	0.0954597523501	99.9045402476
xls	1.88001632431	98.1199836757

Table 125: Parameter:w_e:32, o_e:16, w_f:8,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	80.6236080178	19.3763919822
doc	2.21158064044	97.7884193596
htm	2.66851372535	97.3314862747
javascript	3.49997388079	96.5000261192
jpeg	0.163594721344	99.8364052787
pdf	2.99985490424	97.0001450958
ppt	1.25830375031	98.7416962497
txt	0.0	100.0
xls	2.03148806501	97.968511935

Table 126: Parameter:w_e:32, o_e:16, w_f:8,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	45.7035364936	54.2964635064
doc	3.02504112594	96.9749588741
htm	6.32519139628	93.6748086037
javascript	5.83779548126	94.1622045187
jpeg	0.436284311943	99.5637156881
pdf	2.91103654666	97.0889634533
ppt	2.60105448155	97.3989455185
txt	0.0181834712247	99.9818165288
xls	2.80246689643	97.1975331036

Table 127: Parameter:w_e:32, o_e:16, w_f:8,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	65.6678236793	34.3321763207
doc	2.45216003314	97.5478399669
htm	1.99588064816	98.0041193518
javascript	4.25656027443	95.7434397256
jpeg	0.187829838285	99.8121701617
pdf	3.12316742136	96.8768325786
ppt	1.96298964905	98.0370103509
txt	0.0	100.0
xls	1.85612885283	98.1438711472

Table 128: Parameter:w_e:32, o_e:16, w_f:8,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	78.7800963082	21.2199036918
doc	2.29316437188	97.7068356281
htm	2.02201861618	97.9779813838
javascript	3.80302291565	96.1969770843
jpeg	0.179350460494	99.8206495395
pdf	3.10746759528	96.8925324047
ppt	1.14112988261	98.8588701174
txt	0.0	100.0
xls	1.96377179617	98.0362282038

Table 129: Parameter:w_e:32, o_e:16, w_f:8,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.1123595506	14.8876404494
doc	2.18106494456	97.8189350554
htm	5.28010693887	94.7198930611
javascript	3.71789290379	96.2821070962
jpeg	0.163596703829	99.8364032962
pdf	2.76299879081	97.2370012092
ppt	1.20093731693	98.7990626831
txt	0.0	100.0
xls	2.2008585767	97.7991414233

Table 130: Parameter:w_e:32, o_e:16, w_f:8,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	73.514747191	26.485252809
doc	2.87438428897	97.125615711
htm	1.52046783626	98.4795321637
javascript	3.51048269137	96.4895173086
jpeg	0.199346821054	99.8006531789
pdf	3.49563046192	96.5043695381
ppt	1.19328316897	98.806716831
txt	0.00212138568913	99.9978786143
xls	1.82622312511	98.1737768749

Table 131: Parameter:w_e:32, o_e:16, w_f:16,o_f:2, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	85.2387640449	14.7612359551
doc	2.81473899693	97.1852610031
htm	3.93841442668	96.0615855733
javascript	3.98586055583	96.0141394442
jpeg	0.161180861893	99.8388191381
pdf	2.88640595903	97.113594041
ppt	1.30402690068	98.6959730993
txt	0.0	100.0
xls	2.12459793465	97.8754020653

Table 132: Parameter:w_e:32, o_e:16, w_f:16,o_f:2, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	84.7261235955	15.2738764045
doc	2.75111964171	97.2488803583
htm	10.4210974054	89.5789025946
javascript	6.55288021945	93.4471197806
jpeg	0.318133616119	99.6818663839
pdf	2.83597883598	97.164021164
ppt	2.46052901374	97.5394709863
txt	0.0212179079143	99.9787820921
xls	2.98412698413	97.0158730159

Table 133: Parameter:w_e:32, o_e:16, w_f:16,o_f:2, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	73.924035394	26.075964606
doc	2.92800614114	97.0719938589
htm	1.98498049652	98.0150195035
javascript	3.44260154107	96.5573984589
jpeg	0.169050951593	99.8309490484
pdf	3.98476466854	96.0152353315
ppt	1.22842782328	98.7715721767
txt	0.0	100.0
xls	1.82289777262	98.1771022274

Table 134: Parameter:w_e:32, o_e:16, w_f:16,o_f:4, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	84.3607031062	15.6392968938
doc	2.85129404884	97.1487059512
htm	4.60809332847	95.3919066715
javascript	3.39567443318	96.6043255668
jpeg	0.159965098524	99.8400349015
pdf	2.89487049264	97.1051295074
ppt	1.20913884007	98.7908611599
txt	0.0	100.0
xls	1.97344554886	98.0265544511

Table 135: Parameter:w_e:32, o_e:16, w_f:16,o_f:4, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	25.5568934377	74.4431065623
doc	7.56717236337	92.4328276366
htm	10.5541378053	89.4458621947
javascript	3.91849529781	96.0815047022
jpeg	0.527176876931	99.4728231231
pdf	3.99056774896	96.009432251
ppt	4.60456942004	95.39543058
txt	0.0727404982724	99.9272595017
xls	3.13803736623	96.8619626338

Table 136: Parameter:w_e:32, o_e:16, w_f:16,o_f:4, w_n:10

File type	% classified as binary	% classified as non-binary
elf-arm-32	76.6071428571	23.3928571429
doc	3.20218352849	96.7978164715
htm	2.01961284678	97.9803871532
javascript	4.2662116041	95.7337883959
jpeg	0.221764420746	99.7782355793
pdf	4.25740299626	95.7425970037
ppt	1.3801506684	98.6198493316
txt	0.0	100.0
xls	2.1028319911	97.8971680089

Table 137: Parameter:w_e:32, o_e:16, w_f:16,o_f:8, w_n:1

File type	% classified as binary	% classified as non-binary
elf-arm-32	83.0818619583	16.9181380417
doc	2.41263342594	97.5873665741
htm	3.96636368055	96.0336363194
javascript	3.80302291565	96.1969770843
jpeg	0.174503150751	99.8254968492
pdf	3.16324062878	96.8367593712
ppt	1.17161870841	98.8283812916
txt	0.0	100.0
xls	2.13795104963	97.8620489504

Table 138: Parameter:w_e:32, o_e:16, w_f:16,o_f:8, w_n:4

File type	% classified as binary	% classified as non-binary
elf-arm-32	70.8868378812	29.1131621188
doc	3.38776504996	96.61223495
htm	8.36067565925	91.6393243407
javascript	8.06339254615	91.9366074538
jpeg	0.290838584586	99.7091614154
pdf	3.88149939541	96.1185006046
ppt	2.80023432923	97.1997656708
txt	0.0	100.0
xls	2.95078002177	97.0492199782

Table 139: Parameter:w_e:32, o_e:16, w_f:16,o_f:8, w_n:10

7.3 Discussion of the Test Tesults

In this section we discuss the result of 126 tests, shown in the tables 14 to 139, which are shown on pages 38 to 64. We start the discussion with the parameter entropy window-size parameter w_e. The size of the entropy window changes the resolution of the entropy-window. To show the effects of the entropy-window on the classification results, we divide the 126 tests into two subsets with the settings $w_e = 32$ (including 54 tests) and $w_e = 64$ (including 72 tests). The size of the test-sets differs for reasons explained in section 4.
In section 5.2 we started our discussion with the parameter w_e. Let us again consider the effects of the entropy window-size w_e on the results of the correct classification of elf-arm-32 binaries. In our tests, we tested settings for $w_e = 32$ and $w_e = 64$. In our discussion we use average classification rates, calculated over a range of the 126 executed tests. The average values are in some test unreasonably low. Because these values are averages, they include results with classification rates with far more than 90%, while also including much lower detection rates. When considering statistics on the settings using $w_e = 32$, we have an average classification rate of **74.36%** and a median of 78.26% with a standard deviation of 14.61%. When considering $w_e = 64$, we have an average classification rate of **84.85%** and a median of 89.40% with a standard deviation of 17.20%. This shows that the **larger entropy-window increase** the detection rate of the smaller windows of more than 10%.
In the next step we stay with the setting $w_e = 64$ and consider the effects of the entropy-window-overlap o_e. In our tests, we tested settings for $o_e = \{4, 8, 16, 32\}$. Each O_e setting is a subset of the 72 tests with the setting $w_e = 64$. Thus each O_e test-result-set holds the results for 18 tests. The accumulated results are shown in table 140. The mean and median results are much closer to each other than the above w_e comparison. The setting $o_e = 4$ has the lowest standard deviation and thus seems to be the most stable. Nontheless we select our best result based on the median, because there is a higher variance and we do not want to base our selection on peaks within the data. Thus the setting $o_e = 32$ is our selection for a best match.

o_e setting	mean	median	standard deviation
$o_e = 4$	87.28%	89.29%	8.80 %
$o_e = 8$	82.08%	88.95%	22.20 %
$o_e = 16$	84.82%	88.73%	17.75 %
$o_e = 32$	85.22%	90.09%	18.31 %

Table 140: Aggregated test-results with mean, median and standard deviation of the results with $w_e = 64$ and variations on o_e.

With the selection of $w_e = 64$ and $o_e = 32$ we have left 18 test-results. The next step is to determine the best w_f setting. Within the set of 18 test-results we have w_f settings of $\{4, 8, 16\}$. The best results are delivered by a setting of $w_f = 16$,$o_f = 4$ and $w_n = 4$, which shows a elf-arm-32 detection accuracy of 94.87 %. The set of 18 tests shows that a high setting of $w_n = 10$ leads to worse results than smaller settings. An accumulation of many samples seems to blur the results in a way that a binary pattern is harder to detect. The results are blurred, because we are working with averages, which can hide small binary-patterns within larger non-binary patterns.
The best result $\{w_e = 64, o_e = 32, w_f = 16, o_f = 4, w_n = 4\}$ leads to a high overall detection accuracy, shown in table 81 on page 52. Until this point we did not consider the overhead and the minimum data size, which was discussed in section 4.4. The choosen seeting leadsto a **low overhead** in data processing of 58.33 %. The minimal size of malware that can be

detected is 1536 bytes, which can be too large to detect small chunks of real world malware. When the overhead and minimum-size are also considered, the setting $\{w_e = 64, o_e = 16, w_f = 8, o_f = 2, w_n = 1\}$ is more favourable. The overhead and minimum-size for the test samples are shown in the tables 141 and 142.

The test-results are shown in table 53 on page 46. The detection rate of binaries is slightly lower but the detection rate is not lower than 96 % for any non-binary file. The overhead is 38.0 % and the minimum detection size is 288 bytes and thus much more favourable.

In conclusion we found out that the entropy-window w_e may not be to small. Otherwise the entropy curve shows less details and the real existing entropy is underrated. The effects have been shown in the figures starting on on page 18, starting with figure 7 to page 24 with figure 13. An underrated entropy can stop a correct detection. The test-data shows that an overlap of less than $\frac{1}{4}$ of the window size leads to a decline in detection accuracy. This is true for both entropy- and Fourier-overlaps.

w_e	o_e	w_f	o_f	w_n	overhead (%)	minimum data size (byte)
32	4	4	2	1	85.7142857143%	56
32	4	4	2	4	85.7142857143%	224
32	4	4	2	10	85.7142857143%	560
32	4	8	2	1	66.6666666667%	168
32	4	8	2	4	66.6666666667%	672
32	4	8	2	10	66.6666666667%	1680
32	4	8	4	1	85.7142857143%	112
32	4	8	4	4	85.7142857143%	448
32	4	8	4	10	85.7142857143%	1120
32	4	16	2	1	61.2244897959%	392
32	4	16	2	4	61.2244897959%	1568
32	4	16	2	10	61.2244897959%	3920
32	4	16	4	1	66.6666666667%	336
32	4	16	4	4	66.6666666667%	1344
32	4	16	4	10	66.6666666667%	3360
32	4	16	8	1	85.7142857143%	224
32	4	16	8	4	85.7142857143%	896
32	4	16	8	10	85.7142857143%	2240
32	8	4	2	1	100.0%	48
32	8	4	2	4	100.0%	192
32	8	4	2	10	100.0%	480
32	8	8	2	1	77.7777777778%	144
32	8	8	2	4	77.7777777778%	576
32	8	8	2	10	77.7777777778%	1440
32	8	8	4	1	100.0%	96
32	8	8	4	4	100.0%	384
32	8	8	4	10	100.0%	960
32	8	16	2	1	71.4285714286%	336
32	8	16	2	4	71.4285714286%	1344
32	8	16	2	10	71.4285714286%	3360
32	8	16	4	1	77.7777777778%	288
32	8	16	4	4	77.7777777778%	1152
32	8	16	4	10	77.7777777778%	2880
32	8	16	8	1	100.0%	192
32	8	16	8	4	100.0%	768
32	8	16	8	10	100.0%	1920
32	16	4	2	1	150.0%	32
32	16	4	2	4	150.0%	128
32	16	4	2	10	150.0%	320
32	16	8	2	1	116.666666667%	96
32	16	8	2	4	116.666666667%	384
32	16	8	2	10	116.666666667%	960
32	16	8	4	1	150.0%	64
32	16	8	4	4	150.0%	256
32	16	8	4	10	150.0%	640
32	16	16	2	1	107.142857143%	224
32	16	16	2	4	107.142857143%	896
32	16	16	2	10	107.142857143%	2240
32	16	16	4	1	116.666666667%	192
32	16	16	4	4	116.666666667%	768
32	16	16	4	10	116.666666667%	1920
32	16	16	8	1	150.0%	128
32	16	16	8	4	150.0%	512
32	16	16	8	10	150.0%	1280

Table 141: Overhead and minimal size for detection according to formulas in section 4.4 for $w_e = 32$

w_e	o_e	w_f	o_f	w_n	overhead (%)	minimum data size (byte)
64	4	4	2	1	40.0%	120
64	4	4	2	4	40.0%	480
64	4	4	2	10	40.0%	1200
64	4	8	2	1	31.1111111111%	360
64	4	8	2	4	31.1111111111%	1440
64	4	8	2	10	31.1111111111%	3600
64	4	8	4	1	40.0%	240
64	4	8	4	4	40.0%	960
64	4	8	4	10	40.0%	2400
64	4	16	2	1	28.5714285714%	840
64	4	16	2	4	28.5714285714%	3360
64	4	16	2	10	28.5714285714%	8400
64	4	16	4	1	31.1111111111%	720
64	4	16	4	4	31.1111111111%	2880
64	4	16	4	10	31.1111111111%	7200
64	4	16	8	1	40.0%	480
64	4	16	8	4	40.0%	1920
64	4	16	8	10	40.0%	4800
64	8	4	2	1	42.8571428571%	112
64	8	4	2	4	42.8571428571%	448
64	8	4	2	10	42.8571428571%	1120
64	8	8	2	1	33.3333333333%	336
64	8	8	2	4	33.3333333333%	1344
64	8	8	2	10	33.3333333333%	3360
64	8	8	4	1	42.8571428571%	224
64	8	8	4	4	42.8571428571%	896
64	8	8	4	10	42.8571428571%	2240
64	8	16	2	1	30.612244898%	784
64	8	16	2	4	30.612244898%	3136
64	8	16	2	10	30.612244898%	7840
64	8	16	4	1	33.3333333333%	672
64	8	16	4	4	33.3333333333%	2688
64	8	16	4	10	33.3333333333%	6720
64	8	16	8	1	42.8571428571%	448
64	8	16	8	4	42.8571428571%	1792
64	8	16	8	10	42.8571428571%	4480
64	16	4	2	1	50.0%	96
64	16	4	2	4	50.0%	384
64	16	4	2	10	50.0%	960
64	16	8	2	1	38.8888888889%	288
64	16	8	2	4	38.8888888889%	1152
64	16	8	2	10	38.8888888889%	2880
64	16	8	4	1	50.0%	192
64	16	8	4	4	50.0%	768
64	16	8	4	10	50.0%	1920
64	16	16	2	1	35.7142857143%	672
64	16	16	2	4	35.7142857143%	2688
64	16	16	2	10	35.7142857143%	6720
64	16	16	4	1	38.8888888889%	576
64	16	16	4	4	38.8888888889%	2304
64	16	16	4	10	38.8888888889%	5760
64	16	16	8	1	50.0%	384
64	32	16	4	1	58.3333333333%	384
64	32	16	4	4	58.3333333333%	1536
64	32	16	4	10	58.3333333333%	3840
64	32	16	8	1	75.0%	256
64	32	16	8	4	75.0%	1024
64	32	16	8	10	75.0%	2560

Table 142: Overhead and minimal size for detection according to formulas in section 4.4 for $w_e = 64$

8 Conclusions and Outlook

The proposed method can detect a variety of embedded shellcode attacks. Finding embedded malware with a high degree of certainty has become a lightweight process. A proof-of-concept of our method has been demonstrated in 2012 at the CeBit and at the SIGCOMM [1]. The demonstration has shown that a protection with a low system overhead is possible.
We need to consider that the method has its limitations. There are cases when malware is detected, when there is none. For this reason, we suggest that this method is used to scan the vast majority of incoming data, with a low system impact. In cases of uncertainty, another method with a higher complexity can be applied. While recoding executable code, the detection of malware can be avoided in some cases. Detection of malware can be avoided by recoding machine code instructions in a way that the purpose of the data is completely hidden[62]. Those sophisticated recoding methods can only be detected while the code execution is transferred to those sections. Nevertheless some simple forms of recoding can be detected by our method, this has been demonstrated in section 6.2.
There are several directions of improvement that have not been considered in this paper. To get more accurate results, we considered using Wavelet transforms instead of Fourier transforms. Wavelet transforms can achieve a higher frequency-time resolution. The classification algorithm may be improved. As a classification Algorithm the C4.5 Algorithm discussed in section 2.2 seems to perform as good as the ANN-Classifier (in terms of correct results) but with significantly less time during the training phase of the classifier. The performance of alternative classifiers could be tested. Future work could also use larger testsets, to include more variance in the filetypes.

References

[1] M. Wählisch, S. Trapp, J. Schiller, B. Jochheim, T. Nolte, T. C. Schmidt, O. Ugus, D. Westhoff, M. Kutscher, M. Küster, C. Keil, and J. Schönfelder, “Vitamin C for your Smartphone: The SKIMS Approach for Cooperative and Lightweight Security at Mobiles,” in *Proc. of ACM SIGCOMM, Demo Session (SIGCOMM’12)*. New York: ACM, August 2012, pp. 271–272. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p271.pdf

[2] AdaptiveMobile. (2011, February) Global security insight for mobile. [Online]. Available: http://www.adaptivemobile.com/global-security-insight-centre/mobile-report

[3] ECMA, *ECMA-340: Near Field Communication — Interface and Protocol (NFCIP-1)*. Ecma International, Dec. 2004. [Online]. Available: http://www.ecma.ch/ecma1/STAND/ecma-340.htm

[4] C. Mulliner, “Vulnerability Analysis and Attacks on NFC-enabled Mobile Phones,” in *International Conference on Availability, Reliability and Security*, 2009. [Online]. Available: http://www.mulliner.org/nfc/

[5] M. Wählisch, S. Trapp, C. Keil, J. Schönfelder, T. C. Schmidt, and J. Schiller, “First Insights from a Mobile Honeypot,” in *Proc. of ACM SIGCOMM, Poster Session (SIGCOMM’12)*. New York: ACM, August 2012, pp. 305–306. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p305.pdf

[6] T. C. Group. (2007, Aug.) Tcg specification architecture overview, revision 1.4. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14

[7] ——. (2007, June) Tcg mpwg mobile trusted module specification, version 1.0„ revision 1. [Online]. Available: https://members.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf

[8] A. Moreno and E. Okamoto, “Bluesnarf revisited: Obex ftp service directory traversal,” in *Proceedings of the IFIP TC 6th international conference on Networking*, ser. NETWORKING’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 155–166. [Online]. Available: http://dl.acm.org/citation.cfm?id=2039912.2039931

[9] D. Spill and A. Bittau, “Bluesniff: Eve meets alice and bluetooth,” in *Proceedings of the first USENIX workshop on Offensive Technologies*, ser. WOOT ’07. Berkeley, CA, USA: USENIX Association, 2007, pp. 5:1–5:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1323276.1323281

[10] C. Mulliner, N. Golde, and J.-P. Seifert, “SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale,” in *Proceedings of the 20th USENIX Security Symposium*, San Francisco, CA, USA, August 2011.

[11] M. DeGusta. (2011, October) Android orphans: Visualizing a sad history of support. [Online]. Available: http://theunderstatement.com/post/11982112928/android-orphans-visualizing-a-sad-history-of-support?45bebcc0

[12] C. Miller, “Mobile attacks and defense,” *IEEE Security and Privacy*, vol. 9, no. 4, pp. 68–70, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1109/MSP.2011.85

[13] A. Greenberg. (2011, July) iphone security bug lets innocent-looking apps go bad. [Online]. Available: http://www.forbes.com/sites/andygreenberg/2011/11/07/iphone-security-bug-lets-innocent-looking-apps-go-bad/

[14] A. Gostev. (2011, August) Monthly malware statistics: August 2011. [Online]. Available: http://www.securelist.com/en/analysis/204792190/Monthly_Malware_Statistics_August_2011

[15] G. McGraw and G. Morrisett, “Attacking malicious code: a report to the infosec research council,” *Software, IEEE*, vol. 17, no. 5, pp. 33 –41, sep/oct 2000.

[16] A. M. Turing, “On computable numbers, with an application to the entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. 42, pp. 230–265, 1936.

[17] F. B. Cohen, *A short course on computer viruses (2nd ed.).* New York, NY, USA: John Wiley & Sons, Inc., 1994.

[18] P. Szor, *The Art of Computer Virus Research and Defense.* Addison-Wesley Professional, 2005.

[19] S. Corporation, "Understanding heuristics: Symantec's bloodhound technology." *Symantec White Paper Series*, vol. Volume XXXIV, 1997.

[20] S. Tesauro, Kephart, "Neural networks for computer virus recognition," *IEEE Expert*, vol. 11, pp. 5–6, 1996.

[21] K. Raman. (2012, April) Selecting features to classify malware. [Online]. Available: http://infosecsouthwest.com/files/speaker_materials/ISSW2012_Selecting_Features_to_Classify_Malware.pdf

[22] J. R. Quinlan, *C4.5: programs for machine learning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[23] M. Siddiqui, M. C. Wang, and J. Lee, "Data mining methods for malware detection using instruction sequences," in *Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications*, ser. AIA '08. Anaheim, CA, USA: ACTA Press, 2008, pp. 358–363. [Online]. Available: http://portal.acm.org/citation.cfm?id=1712759.1712825

[24] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," *IEEE Symposium on Security and Privacy*, pp. pp. 38–49, 2001.

[25] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 40–45, 2007.

[26] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Alemany, "Automated mapping of large binary objects using primitive fragment type classification," *Digital Investigation*, vol. 7, no. Supplement 1, pp. S3–S12, 2010, the Proceedings of the Tenth Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/B7CW4-50NX65H-3/2/0d07f9648ca609718c856afc5ea253c2

[27] M. M. J.Z. Kolter, "Learning to detect malicious executables in the wild," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining.* ACM, 2004, pp. pp. 470–478.

[28] Y. Yang and J. O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 412–420.

[29] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/2089125.2089126

[30] D. Wagner and D. Dean, "Intrusion detection via static analysis," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, ser. SP '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 156–. [Online]. Available: http://dl.acm.org/citation.cfm?id=882495.884434

[31] J. Lee, K. Jeong, and H. Lee, "Detecting metamorphic malwares using code graphs," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 1970–1977. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774505

[32] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*, ser. AusPDC '10. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2010, pp. 61–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=1862294.1862301

[33] L. Bai, J. Pang, Y. Zhang, W. Fu, and J. Zhu, “Detecting malicious behavior using critical api-calling graph matching,” in *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*, ser. ICISE ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1716–1719. [Online]. Available: http://dx.doi.org/10.1109/ICISE.2009.494

[34] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, “Fast malware classification by automated behavioral graph matching,” in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW ’10. New York, NY, USA: ACM, 2010, pp. 45:1–45:4. [Online]. Available: http://doi.acm.org/10.1145/1852666.1852716

[35] K. Rieck, P. Trinius, C. Willems, and T. Holz, “Automatic analysis of malware behavior using machine learning,” *J. Comput. Secur.*, vol. 19, no. 4, pp. 639–668, Dec. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2011216.2011217

[36] H. Kim, J. Smith, and K. G. Shin, “Detecting energy-greedy anomalies and mobile malware variants,” in *Proceeding of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys ’08. New York, NY, USA: ACM, 2008, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378627

[37] J. Nazario, *Defense and Detection Strategies against Internet Worms.* Norwood, MA, USA: Artech House, Inc., 2003.

[38] K. Wang, G. F. Cretu, and S. J. Stolfo, “Anomalous payload-based worm detection and signature generation,” in *RAID*, 2005, pp. 227–246.

[39] J. Olivain and J. Goubault-Larrecq, “Detecting subverted cryptographic protocols by entropy checking,” Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-06-13, Jun. 2006, 19 pages. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2006-13.pdf

[40] L. Paninski, “Estimating entropy on m bins given fewer than m samples,” *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2200–2203, 2004.

[41] Y. Gu, A. McCallum, and D. Towsley, “Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation,” in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement (IMC’05).* Berkeley, CA, USA: USENIX Association, 2005, pp. 345–350.

[42] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, “An Empirical Evaluation of Entropy-based Traffic Anomaly Detection,” in *IMC ’08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 2008, pp. 151–156.

[43] D. J. Hickok, D. R. Lesniak, and M. C. Rowe, “File type detection technology,” in *38th Midwest Instruction and Computing Symposium April 8 - 9, 2005. University of Wisconsin-Eau Claire, Eau Claire, WI*, 2005, pp. 73–76. [Online]. Available: http://www.micsymposium.org/mics_2005/papers/paper7.pdf

[44] M. McDaniel and M. H. Heydari, “Content based file type detection algorithms,” in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS’03) - Track 9 - Volume 9*, ser. HICSS ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 332.1–. [Online]. Available: http://dl.acm.org/citation.cfm?id=820756.821828

[45] W.-J. Li, K. Wang, and S. J. Stolfo, “Fileprints: Identifying file types by n-gram analysis,” in *IEEE Information Assurance Workshop*, 2005.

[46] M. Karresand and N. Shahmehri, “Oscar - file type identification of binary data in disk clusters and ram pages.” in *SEC’06*, 2006, pp. 413–424.

[47] R. M. Harris, “Using artificial neural networks for forensic file type identification,” Master’s thesis, Purdue University, 05 2007. [Online]. Available: http://web.ics.purdue.edu/~rmharris/Thesis.pdf

[48] G. A. Hall and W. P. Davis, “Sliding Window Measurement for File Type Identification,” http://www.mantech.com/cfia2/SlidingWindowMeasurementforFileTypeIndentification.pdf, 2007.

[49] R. F. Erbacher and J. Mulholland, “Identification and localization of data types within large-scale file systems,” *Systematic Approaches to Digital Forensic Engineering, IEEE International Workshop on*, vol. 0, pp. 55–70, 2007.

[50] S. J. Moody and R. F. Erbacher, “Sádi - statistical analysis for data type identification,” in *SADFE*, 2008, pp. 41–54.

[51] C. Veenman, “Statistical disk cluster classification for file carving,” in *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, aug. 2007, pp. 393 –398.

[52] A. N. Kolmogorov, “Three approaches to the quantitative definition of information,” *Problems of Information Transmission*, vol. 1, pp. 1–7, 1965.

[53] A. Lempel and J. Ziv, “On the complexity of finite sequences,” *Information Theory, IEEE Transactions on*, vol. 22, no. 1, pp. 75–81, 1976. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1055501

[54] W. C. Calhoun and D. Coles, “Predicting the types of file fragments,” *Digital Investigation*, vol. 5, no. Supplement 1, pp. S14 – S20, 2008, the Proceedings of the Eighth Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287608000273

[55] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July/Oct. 1948.

[56] E. Jacobsen and R. Lyons, “The sliding dft,” *Signal Processing Magazine, IEEE*, vol. 20, no. 2, pp. 74–80, March 2003.

[57] F. Harris, “On the use of windows for harmonic analysis with the discrete Fourier transform,” *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, Jan. 1978.

[58] J. T. R.B. Blackman, “The Measurement of Power Spectra,” 1958.

[59] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[60] T. C. Schmidt, M. Wählisch, B. Jochheim, and M. Gröning, “WiSec 2011 Poster: Context-adaptive Entropy Analysis as a Lightweight Detector of Zero-day Shellcode Intrusion for Mobiles,” *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, vol. 15, no. 3, pp. 47–48, July 2011.

[61] G. A. Hall, W. P. Davis, C. Forensics, I. A. Group, and M. Security, “Sliding window measurement for file type identification,” 2007.

[62] J. Mason, S. Small, F. Monrose, and G. MacManus, “English shellcode,” in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS ’09. New York, NY, USA: ACM, 2009, pp. 524–533. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653725