ELF-Miner: Using Structural Knowledge and Data Mining Methods To Detect New (Linux) Malicious Executables

Farrukh Shahzad and Muddassar Farooq

Next Generation Intelligent Networks Research Center (nexGIN RC)
National University of Computer & Emerging Sciences (FAST-NUCES)
Islamabad, 44000, Pakistan.
{farrukh.shahzad,muddassar.farooq}@nexginrc.org

Abstract. Linux malware can pose a significant threat – its (Linux) penetration is exponentially increasing – because little is known or understood about their vulnerabilities. We believe that now is the right time to devise non-signature based zero-day (previously unknown) malware detection strategies before Linux intruders take us by surprise. Therefore, in this paper, we first do a forensic analysis of Linux executable and linkable format (ELF) files. Our forensic analysis provides insight into different features that have the potential to discriminate malware executables from benign ones. As a result, we can select a features' set of 383 features that are extracted from an ELF header. We quantify the classification potential of features using information gain and then remove redundant features by employing pre-processing filters. Finally, we do an extensive evaluation among classical rule based machine learning classifiers – RIPPER, PART, C4.5 Rules and decision tree J48 – and bio-inspired classifiers – cAnt Miner, UCS, XCS, and GAssist – to select the best classifier for our system. We have evaluated our approach on an available collection of 709 Linux malware samples from *vx heavens* and *offensive computing*. Our experiments show that ELF-Miner provides more than 99% detection accuracy with less than 0.1% false alarm rate.

Keywords: ELF, Data Mining, Information Security, Structural Information, Malicious Executables, Machine Learning, Malware Forensics, Linux Malware, Evolutionary Computing.

1 Introduction

Linux – due to its open source nature – is getting an ever increasing attention both by researchers and developers [2]. Moreover, home users and business enterprizes are preferring Linux based Personal Computers (PCs) and server machines. As a consequence, Linux will definitely become a favorite target for hackers, the moment its market share makes it an attractive proposition to

launch attacks on Linux running hosts. The current scarce availability of Linux malware has also lead Linux security experts to hold a notion that Linux is inherently secure [1]; therefore, malware detection on Linux has never received its due attention. Consequently, Linux based computers – open source nature makes the task of a hacker also easier – are not adequately protected against emerging threats.

In this paper, we first do a forensic analysis – with a particular focus on the structural information present in the header of an ELF file – of available Linux malware. The analysis helped us in identifying a set of structural features that can be used to discriminate a malware file from a benign one. We then apply preprocessor filters to remove redundant features. Finally, we give the remaining features' set as an input to a number of classifiers – classical machine learning and bio-inspired – to finally detect malware. We compare the accuracy of different classifiers on a collection of 709 malware samples available from *vx heavens* [7] and *offensive computing* [3]. Our results show that our system is able to detect Linux malware with more than 99% accuracy. The true strength of our approach is that it does not make a signature from the instructions of a malware; therefore, it is a non-signature based technique and has the ability to detect zero-day (on the day of its launch) Linux malware. Our data mining approach takes only a fraction of a second; therefore, we can deploy it in realtime on Linux systems[1].

The rest of the paper is organized as follows. In the next section, we briefly summarize related work. Section 3 discusses the ELF mined features' set in detail. Section 4 provides an overview of ELF-Miner framework. A brief introduction of malware dataset is given in Section 5. A comprehensive forensic analysis of benign and malware files is provided in Section 6. An overview of our classification methodology has been presented in Section 7. Section 8 presents the experiments, results and evaluation of classifiers on the dataset. Finally, Section 9 presents the conclusion of the paper and future research directions.

2   Related Work

In the related work section, we summarize relevant non-signature based malware detection techniques based on static analysis of executables. All of these techniques are for windows executables. We have recently proposed our technique – PE-Miner [28] – that extracts structural information from the headers of a windows executable and uses it to detect malware. Later, we enhanced it in [26] to overcome the side effects of doing packing (encryption) of executables. As a result, the enhanced version has the capability to discriminate packed malware files from packed benign files and unpacked malware files from unpacked benign files. The other recently proposed malware detection techniques for windows executables are: Perdisci et al. [21], Schultz et al [25], Masud et al. [19] and Kolter et

---

[1] We have already published our initial work on this approach as "PE-Miner" [28] that detects zero-day malicious executables on Windows platform using the structural information in the header of a PE file.

al [18]. We now briefly summarize their detection methodology but an interested reader can refer to [27] for a detailed description.

The technique proposed by Masud et al. [19] utilizes a hybrid features selection scheme to detect malicious PE files. The hybrid set consists of three types of features:(1) n-grams as binary features, (2) n-grams of unique assembly instructions, and (3) n-grams of dynamic link library calls. Decision trees are used to classify the PE files on the basis of the extracted hybrid features' set. The technique uses a disassembler to create the features' set; therefore, the scanning time is significantly large (making it difficult to use it in realtime).

In [21], Perdisci et al., the authors proposed a framework 'McBoost' based on two level classification. The two classifiers – C1 and C2 – have been used to classify the non-packed and packed portable executables (PE) files. They developed a customized unpacker to extract hidden codes from an executable and provide them to the C2 classifier for analysis and classification. In [25], Schultz et al. proposed three different methodologies for detecting malicious executable files on the Windows platform. The first technique is based on the information of dynamic link libraries (DLLs), their function calls and citation counts. In the second technique, they use the strings as binary features (i.e. present or absent). The third technique uses two byte words (instead of a string) as binary features. Later on, Kolter et al. [18] have improved the third technique of Schultz et al. by using 4-grams as binary features. We have already compared the techniques of Schultz et al., Perdisci et al., and Kolter et al. with our PE-Miner in [28]. The results of our experiments show that none of these techniques are realtime deployable because they have: (1) less than 99% detection accuracy, (2) more than 1% false alarm rate, and (3) large file scanning time (order of seconds). In comparison, PE-Miner not only achieves greater than 99% detection accuracy and less than 0.1% false alarm rate but also its scanning time is comparable (200 milliseconds) with that of commercial antivirus software. We do not want to again establish the superiority of ELF-Miner over other approaches because it seems straightforward after the above-mentioned discussion in [28]. Therefore, in this paper, our focus is to explore another dimension of research: whether using the structural information to detect malware could be generalized to executables of other operating systems? This paper proves this thesis.

3   The Features' set of ELF-Miner

In this section, we present an overview of structural features – extracted from the header – of benign and malware ELF files. In order to make the paper self contained, we briefly introduce ELF format.

3.1   Executable and linkable format (ELF)

In Linux, the object files play a key role in the process of program linking and execution. The manual of ELF format [29] describes three types of ELF files: (1) relocatable file (it contains the "how to link" information with other object
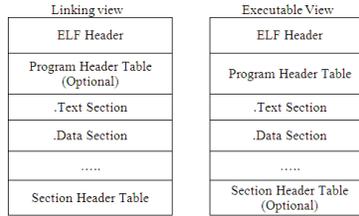
Fig. 1. Executable and linkable format (ELF) structural view [29]

files in order to create a shared library or an executable file), (2) executable file (it contains data and information required by an operating system to create a program image that can be executed by accessing information in the file), and (3) shared object file (it contains all the information required for both static and dynamic linking). We have used both shared and executable ELF files in our dataset.

The structure of an ELF file is as shown in Figure 1. An ELF header, at the beginning of every file, holds a blueprint of a file's organization. In case of a relocatable file, a section header table is mandatory and a program header table is optional. In case of an in-executable object file, a program header table is compulsory and a section header table is optional. Note that a section header table contains the description of all sections that exist in the object file. Every section in the object file has an entry in this table. The section entries provide the attributes of a section: section name, section size etc. The sections primarily hold the object file information required for building and linking ELF programs. The information about program instructions, data, symbols, dynamic linking and relocation is contained in different sections of an object file. We skip further details of ELF format for brevity but an interested reader can find them in [29].

3.2 ELF Structure-based Features' set

We initially extract 383 features from the header of an ELF file (see Table 1). Later we use our forensic analysis to rank these features to select the ones having the best potential to discriminate a benign file from a malware file. We now introduce our features' set in detail.

Table 1. Features extracted from ELF files

No	ELF Structure Name	Features Count
1	ELF Header	16
2	Section Headers	238
3	Program Headers	40
4	Symbols Section	17
5	Dynamic Section	27
6	Dynamic Symbol Section	17
7	Relocation Sections	26
8	Global offset table	1
9	Hash table	1
	Total Fields	383

ELF Header. ELF header consists of data structures that describe the organization of an ELF file. The header structures help in parsing an ELF file. An ELF header structure consists of a number of typical fields: identification, machine type, ELF file version, entry point address, *program header table* file offset in bytes, *section header table* file offset in bytes, processor specific field *flags*, ELF header size in bytes, size and count of individual entries in *program header table*, section header size in bytes, number of entries and index of entry associated with the section name string table in *section header table*. We use 16 fields of ELF header excluding program and section headers offsets in our features' set.

Section Header. The structure of a section header contains the fields: section name, section type, section flags field that represents miscellaneous attributes, address field where section's first byte should reside, section's offset field that holds the first byte of section's offset from the beginning of ELF file, size of section in bytes, *section header table* index link, section info field that depends upon section type, section address alignment field that shows the alignment constraints. We include 238 fields of 34 section header structures in our features' set, excluding the section names, addresses and offsets fields.

Program Header. All executable and linkable files are array of structures, which represent program segments and other information that are mandatory for successfully executing the program. Each executable file segment may contain one or more sections. The segment header structure contains 8 fields: segment type, segment offset that gives the offset from the beginning of the file, virtual address at which first byte resides in the memory, segment's physical address, file size field provides the number of bytes in the segment file image, memory size field contains the number of bytes in memory image, flags' field holds the flags related information of segments, p_align member contains alignment information of segment in file and memory. We select 40 features excluding all addresses and offsets fields of first 8 segments for our features' set.

Symbol Table. It contains the information related to symbols in an ELF file but this information is not mandatory in all types of object files. The symbol table structure has five major fields – name, value, size, info, other and section header index – but the number of entries vary in symbol table of different executable files. Therefore, we make categories on the basis of st_info field – it contains the information of symbol's binding and attributes. The resulting categories are: total number of [*symbols, local symbols, global symbols , weak symbols and stb_lo_proc symbols*]. The objects and functions symbols are further categorized on the basis of their scope i.e. total number of [*local objects, global objects, weak objects, local functions, global functions, weak functions, sections, files , stt_lo_proc and stt_hi_proc*] objects. As a result, we create 17 categorical fields from the symbol table for our features' set.

Dynamic Section. If any ELF file has dynamic linking information, segment PT_DYNAMIC (dynamic section is an integral part of it) becomes part of the program's header table. The structure of dynamic section contains a 4 bytes field d_tag and union d_un. All fields are classified on the basis of the

information available in parameter d_tag. Like symbol table, dynamic section also doesn't contain a fixed number of entries; as a result, all these entries are classified into 27 categories for our features' set.

Dynamic Symbols Section (DSS.) If an object file contains dynamically linked objects, it may have dynamic symbol section also. The details of dynamic symbol section are similar to that of the symbol section as mentioned earlier in this section. 17 features are extracted from the DSS section.

Relocation Section. In ELF, the process of relocation links referred symbols with their definitions. When a program calls a function, the control must be transferred to the function definition at an appropriate address. The structure of *.rel* contains two members – offset and info. The offset field refers the location where the relocation action should be performed. In case of relocatable files, the offset field contains the offset from the beginning of section to the storage unit of relocation. If the files are executable or shared objects, an offset represents the virtual address of the storage unit. The info field maintains information about the type of relocation and the symbol table index that is used for relocation. We extract 26 features from the contents of sections *.rel* and *.rela* for discriminating the benign and malicious files.

Global Offset Table (GOT). This table contains absolute addresses of objects representing private data. It provides address without affecting position independence and sharing capability of a program text. The programs refer to GOT for absolute address values using position independent addressing. For our features' set, we only have one field that represents the size of GOT.

Hash Table. It is a 32-bit object table that facilitates the access to the symbol table. Only one field of the hash table size is included in our features' set.

4 ELF-Miner Framework

We now discuss the architecture of our proposed framework – ELF-Miner – that mines the structural information in ELF executables to detect malicious ones. Our framework consists of three basic components: (1) features extraction, (2) features preprocessing, and (3) classification (see Figure 2). The feature extraction block first does a validity check to confirm the validity of an ELF header. If the file is legitimate ELF executable, it extracts 383 features from its header. Afterwards, the preprocessing block ranks these features to filter redundant features or features with smaller classification potential; as a result, the remaining features' set is given to a number of classifiers that eventually classify the executable as malicious or benign.

5 ELF Malware Dataset

In this section, we present an overview of the datasets used in our experiments. In order to remove any bias because of the size of files, we divide ELF files – both malicious and benign – into six bins of different sizes starting from 20 KB to 4
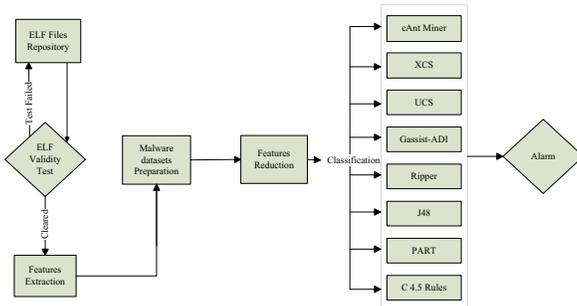
Fig. 2. Block diagram of ELF-Miner framework

MB. In order to have a balanced dataset, we ensure that the percentage of benign and malicious files in a certain category is approximately the same to remove any bias on the basis of size (see Table 2). The benign ELF files are collected from Linux operating system's directories */bin, /sbin*, and */usr/bin*. We use the *'size based filtering'* criteria to select approximately an equal percentage of benign ELF files. As a result, only those files are selected which have size greater than a given threshold that is calculated on the basis of number of files and their size in a specific malware category. In comparison, Linux malware dataset is collected from `vx heavens` [7] and `offensive computing` [3]. We have combined some malware categories because of their similar functionality. For example, we have combined Exploits, Rootkits and Hacktools to create a single Expts + RK + HT category. As a result of the unification process, the number of malware samples per category are increased significantly. We now provide a brief description of individual and combined malware categories – 8 to be precise – to make the paper self-contained.

Backdoor + Sniffer (Bkdrs). A backdoor is a program which allows by-passing of standard authentication methods of an operating system. As a result, remote access to computer systems is possible without explicit consent of the users. Information logging and sniffing activities are possible using the remote access.

Constructor + Virtool (Cnstr). This category of malware mostly includes toolkits for automatically creating new malware by varying a given set of input parameters. Virtool and constructor categories are combined because of their similar functionality.

DoS + Nuker (DoS). Both DoS and nuker based malware allow an attacker to launch malicious activities at a victim's computer that can possibly result in a denial of service attack. These activities can result in slowing down, restarting, crashing or shutting down of a computer system.

Email- + IM- + SMS Flooder (Fldrs). The malware in this category initiate unwanted information floods such as email, instant messaging and SMS

Table 2. Benign and malware file size normalization

File Ranges	Benign Files		Malware Files												
	Total	%	Bkdrs	Spfr	Wrms	Vrs	Expts	RK	Cnstr	DoS	Fldrs	HT	Trjns	Total	%
20K	445	61	116	7	16	71	91	35	14	33	33	16	17	449	63
20-50K	161	22	61	0	5	9	32	23	2	1	1	7	6	147	21
50-100K	41	6	4	0	4	6	1	8	0	0	0	1	2	26	4
100-500K	67	9	11	0	18	10	11	14	0	0	0	5	0	69	10
500K-1MB	15	2	4	0	1	2	1	3	0	0	0	3	0	14	2
1-4MB	5	1	1	0	0	1	0	0	0	0	0	2	0	4	1
Total	734	-	197	7	44	99	136	83	16	34	34	34	25	709	-

floods. Similarly, well known network packets based attacks – TCP SYN and ICMP floods – are also launched by this category of malware.

Exploit + Hacktool + Rootkits (Expts + RK + HT). The malware in this category exploit vulnerabilities in a system's implementation which most commonly results in buffer overflows. These attacks are launched to take administrative permissions remotely or to execute malicious code on systems.

Email- + M- + IRC- + Net Worm (Wrms). The malware in this category spread through instant messaging networks, IRC networks and port scanning.

Trojans (Trjns). A trojan is a broad term that refers to stand alone programs which appear to perform a legitimate function but covertly do possibly harmful activities such as providing remote access, data destruction and corruption.

Virus + Spoofer (Vrs + Spfr). A virus is a program that can replicate and attach itself with other benign programs. It is probably the most well known type of malware. Similarly, a spoofer is a program that successfully masquerades as another host or program and gains an illegitimate access to the system.

We have recently obtained latest malware dataset from `vx heavens` [7] and `offensive computing` [3] which contains more than 900 Linux malware; however, only 709 passed the validity test. The distribution of malware into different categories and the size is shown in Table 2. The top four malware categories in a descending order are Bkdrs, Expts, Vrs and RK respectively. Another important observation is that 63% of malware is less than 20KB which signifies the importance of removing the size bias (as a feature) during classification; therefore, we have ensured that 61% of benign files are also less than 20KB in our dataset.

6 Forensic Analysis of Benign and Malware ELF Executables

We now provide our forensic analysis – by closely studying the pattern of the fields in the headers of different ELF files – to understand the difference in the headers of benign and malware ELF executables. The aim is to establish the thesis that the structural information of malware files is different from that of the benign ones. We utilize well known information-theoretic measures – Resistor-Average (RA) divergence [17] and Frequency Histogram analysis – to analyze the difference between various fields in ELF headers of benign and malware files. (Note that we do not rank the identified features, by using information-theoretic

measures in our forensic analysis, for classification purposes.) The definition of RA divergence is given as:

$$RA(p,q) = 1/(\frac{1}{KL(p||q)} + \frac{1}{KL(q||p)})$$ (1)

In equation (1), KL (Kullback-Leibler) distance [12] is another information-theoretic measure, which can be used to measure the difference between two probability distributions – $p(x)$ and $q(x)$ in equation (2). KL distance is always non-negative and is zero only in case of $p=q$. Moreover, in a special case when $p(x) = 0$ or $q(x) = 0$, the distance becomes zero or infinity; therefore, we simply add $eps = 2^{-52}$ – which is the distance from 1.0 to the next largest double precision number – in both the distributions $p(x)$ and $q(x)$ to avoid this problem. Moreover, KL distance is not symmetric over two distributions. As a result, we use a symmetric distance measure.

$$(KL(p||q)) = \sum_{\forall x} |p(x).\log \frac{p(x)}{q(x)}|$$ (2)

Similarly, the frequency histogram provides the count of occurrence – of field values or headers in benign and malware files separately. In order to abbreviate the large names of fields in an ELF header, we use a convention: the ELF headers' or sections' names are used as a prefix with the field name. We will briefly describe the functionality of fields, but an interested reader is recommended to consult [29] for details.

Fig. 3. ELF header RA divergence graph for benign and malware files

ELF Header Examination. The RA divergence values for 16 fields of ELF header for benign and malware are shown in Figure 3. It is interesting to note that 4 header fields are significantly different in benign and malware files. The fields are: ELFHEENTRY, ELFHEPHNUM, ELFHESHNUM and ELFHESHSTRINDX. Most of these fields contain distinct numeric values – indices and header sizes – which are significantly different in benign and malware executables. Most of other fields contain either zero or constant values; as a result, RA divergence is zero or negligible.

Fig. 4. Sections' frequency histogram for benign and malware files (%)

ELF Section Headers. In object files, various sections contain the control and program information. We initially extract headers of 34 sections and their description is provided in Table 8 of appendix A and in Section 3.

Sections' headers frequency histogram analysis. We have plotted the frequency histogram of different sections – occurring in benign and malware files – in Figure 4. It is interesting to note that 13 out of 34 sections never appear in any benign or malware file. These sections generally contain debug information, initialized values, read-only data, and rarely used pre-existing extensions. Some sections – *.comment*, *.note*, *.strtab*, *.symtab*, and *.sbss*, – are present in malware files but are generally absent in benign files. Similarly other sections – *.rel.dyn* and *.got.plt* – are not used by most malware programs. We have zoomed at the distribution of these 7 sections in Figure 5 to get a better understanding.

Our analysis shows that most malware misuse *.comment*, *.notes* sections (used to hold version control information and file notes respectively) to store their malicious information. It is interesting to see that sections *.symtab* and *.strtab*, which hold the symbols and strings related information of the programs, are missing in most benign files. We investigated this unexpected trend and found that most of these binaries are stripped and these sections are removed from them [6].

Fig. 5. Overall frequency of 7 sections in all malware (left) and benign (right) files

Similarly, *.rel.dyn* holds dynamic relocation information and is used by the majority of the benign files and almost 35% of the malware files. Section *.got.plt* that holds read-only portion of the global offset table is used by almost all benign

files and few malware files. The frequency of all other sections can be observed from the Figure 4.

Fig. 6. Symbol Table RA divergence graph for benign and malware files

Symbol Table. Similarly, the RA divergence in different fields of symbol table is shown in Figure 6. We see that except processor specific symbols – symtabhiproc, symtabloproc, symstthiproc and symsttloproc which are not used by benign and malware programs – a significant difference exists between symbol table entries. It is already mentioned that most of benign programs – mostly stripped binaries – do not contain a symbol table. In comparison, most malware have symbol table entries.

Dynamic Section. The graph of RA divergence for dynamic section fields is shown in Figure 7. We can see that some fields – with high RA divergence – have the potential to discriminate between benign and malware files. Most of these discriminating fields represent different categories of dynamic elements that are used by benign and malware programs in a different manner. For example, the first field – dyntotal – with a RA value of 0.15 shows that total number of dynamic symbols in both types of files are significantly different.

Dynamic Symbol Section. The RA divergence of different fields of this section is plotted in Figure 8. The figure shows that other than the processor specific features only three categories of attributes – sttfile, sttobjstblocal and

Fig. 7. Dynamic section RA divergence graph for benign and malware files

Fig. 8. Dynamic symbol section RA divergence for benign and malware files

STTFUNSTBLOCAL – are missing from both programs. The obvious reason is that local objects and functions are never visible outside the file containing them; therefore, the local objects entry STTOBJSTBLOCAL and local functions entry STTFUNSTBLOCAL are absent. Moreover, it is a well known fact that symbol table's file object STTFILE does not exist in DSS [4]. The other fields represent the counts of elements in local, global and weak object categories. We can see in Figure 8 that these count values are totally different in both types of programs – the reason for high RA divergence.

Relocation Section. The RA divergence values for different relocation categories are shown in Figure 9. It is obvious that only 7 relocation types are used by both malware and benign files – RELASECTIONCOUNT, RELR38632, RELR386PC32, RELR386COPY, RELR386GLOBDAT, RELR386JMPSLOT and RELR386RELATIVE. A detailed analysis has revealed that RELR386PC32 is used only by malware programs. Note that RELR386PC32 relocation type supports the PC-relative addressing while RELR38632 supports absolute addressing. Some malware files use relative addressing while absolute addressing is commonly used by both benign and malware programs. Other *.rel* fields have high RA values that show their classification potential.

In *.rela* relocation section, the field RELASECTIONCOUNT represents the total number of *.rela* relocation types in a program. Three relocation categories – RELAR38632, RELAR386GLOBDAT and RELAR386JMPSLOT – are only used by benign programs. It is inter-
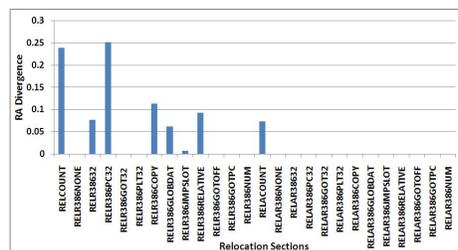
Fig. 9. Relocation section RA divergence graph for benign and malware files

esting to note that no malware writer has used *.rela* sections. As a result, the classification power of the information of *.rela* section is very high.
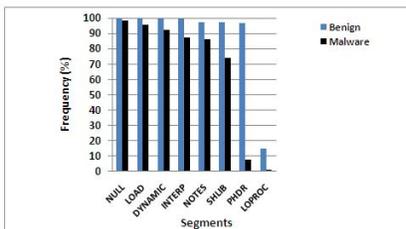
Fig. 10. Segments frequency histogram for benign and malware files

Program Headers. The frequency histogram of segment table is shown in Figure 10. Recall that we have included 8 segment headers in our features' set. We have discussed their details in Section 3. We can easily conclude that most benign and malware programs do not contain processor specific semantic in program headers – the reason for low value of PT_LOPROC. In comparison, Segment PT_PHDR specifies the location and size of the program header table. It's existence shows that the program header table is the part of program's memory image that precedes the loadable segment entry. Most benign programs use this segment while most malware programs do not contain it.

We have proven our thesis that structural information – contained in the header of ELF files – provide good classification potential for discriminating malware programs from benign ones. Now we focus our attention to the classification methodology used in our ELF-Miner framework.

7   CLASSIFICATION SCHEME

7.1   Quantification

In Section 6, we have proven that the structural information of ELF files can be used to discriminate between malware and benign executables. We now use an information-theoretic measure – information gain – to rank different features in our datasets. As a result, we can visualize the patterns that exist in our malware datasets. Information gain measures the reduction in uncertainty if the values of an attribute are known [12][33]. For a given attribute X and a class attribute $Y \in \{$Benign, Malware$\}$, the uncertainty is given by their respective entropies $H(X)$ and $H(Y)$. Then the information gain of X with respect to Y is given by $IG(Y; X)$:

$$IG(Y; X) = H(Y) - H(Y|X)$$

A higher value of information gain represents higher classification potential and vice versa. Note that information gain can vary from 1 to 0. We have created

8 datasets after combining benign files with each malware category – benign-virus, benign-worm, etc. We have used *InfoGainAttributeEval* attribute evaluator with *Ranker* search method in Wakaito Environment for Knowledge Acquisition (WEKA) [32]. Figure 11 shows the normal probability plot for information gain of features for Bkdrs, Cnstr, DoS, Expts + RK + HT, Fldrs, Trjns, Vrs + Spfr and Wrms. In Figures 11(a) and 11(b), two types of features can be observed: One type is composed of features with relatively large probability and the other type consists of features with relatively small existence probability. The majority of the features in all datasets have a very low value of information gain (less than 0.3), but some of them have information gain values as high as 0.82. It is interesting to see that a large number of features have relatively large existence probability but almost zero information gain. The reason behind the fact is that most of these features contain constant values in most of the files so their classification potential is zero. Another relevant observation is that the means of information gain distribution of malware datasets' are 0.10, 0.03, 0.04, 0.11, 0.04, 0.03, 0.08, 0.04 for Bkdrs, Cnstr, DoS, Expts + RK + HT, Fldrs, Trjns, Vrs + Spfr and Wrms respectively. So we can envision – on the basis of information gain values – Cnstr and Trjns are the most difficult categories to classify for the classifiers that use higher information gain parameters to build classification rules. In the next section, this ranked (quantified) features' set is then passed through preprocessing filters.

(a) Plot for the first 4 malware categories   (b) Plot for the Remaining malware categories
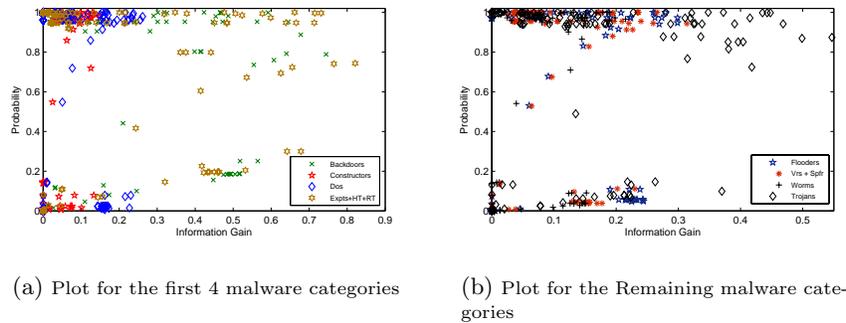
Fig. 11. Probability distribution plot of Information Gain for features extracted from multiple malware datasets'

7.2 Preprocessing Features

Remember in our ELF-Miner framework, we apply preprocessor filters to remove redundant and useless features – the features that have zero or very low classification potential. For classification purpose, we prepare different datasets

Table 3. Dataset analysis for redundant and useless features.

Datasets	Instances	Features	Useless	Remaining	Useless (%)
Bkdrs + Benign	931	383	182	201	47
Cnstr + Benign	750	383	188	195	49
DoS+ Benign	768	383	196	187	51
Expts + RK + HT + Benign	987	383	190	193	49
Fldrs + Benign	768	383	195	188	50
Trjns + Benign	759	383	193	190	50
Vrs + Spfr + Benign	840	383	181	202	47
Wrms + Benign	778	383	196	187	51

by combining benign dataset with all malware datasets: Bkdrs + Benign, Cnstr + Benign, DoS + Benign, Expts + RK + HT + Benign, Fldrs + Benign, Trjns + Benign, Vrs + Spfr + Benign and Wrms + Benign – each of them having 931, 750, 768, 987, 758, 759, 840 and 778 instances respectively (see Table 3). In comparison, as mentioned before, we extract a fixed number of 383 features from an ELF file. Our forensic analysis has shown that ELF files usually do not contain all 34 section headers that are part of our complete features' set (similar is the case of other headers). We have identified with the help of our forensic analysis and features' set quantification that the empty fields that are assigned by default 0 values and fields with constant values have 0 classification potential. It is interesting to note in Table 3 that approximately 50% of features are redundant and useless. We have applied standard useless filter available in (WEKA) [32] to remove these features. It is clear from Table 3 that remaining features are actually given as input to a number of classification algorithms.

7.3 Classification

In the classification phase, our aim is to select a classifier that efficiently mines structural information extracted from ELF executables. The selected classifier must satisfy following requirements: (1) it must mine the structural information in realtime, (2) it must provide high detection accuracy, and (3) it must provide comprehensible and compact rules. In order to meet these requirements, we explore the design space along five dimensions: (1) evaluate different classification paradigms – classical and bio-inspired, (2) do a scalability analysis to rank features as a function of their classification potential, (3) do a robustness analysis of our system by randomly forging features of malware with that of benign files, (4) do a comprehensibility analysis of generated rules, and (5) perform a timing analysis to determine the realtime deployment feasibility of the system.

In our study, we have used supervised learning based classifiers [16][24], four well known classical machine learning classifiers – RIPPER [11], PART [14], C4.5 Rules [23] and J48 [22]. Similarly, we have used four well known bio-inspired genetic machine learning algorithms: (1) cAntMiner [20] is ant colony optimization based classifier, (2) XCS is a Michigan style learning classifier system [31], (3) UCS is optimized for supervised learning environments [10], and (4) GAssist ADI is a Pittsburgh style learning classifier system [9]. We skip details of classifiers in this paper for brevity, but an interested reader may consult [15] for their description.

We have used the standard implementations of classical machine learning algorithms in (WEKA) [32]. Similarly, we have used implementations of bio-inspired evolutionary classifiers in Knowledge Extraction based on Evolutionary Learning (KEEL) [8] for our experiments. The objective of using standard tools is to remove any implementation related bias in our evaluation. Moreover, we have used the best configuration – determined after a number of plot studies – of a classifier in our experiments.

8   EXPERIMENTS AND RESULTS

A stratified $10-$fold cross validation procedure is followed for all experiments reported later in this section. In this procedure, we partition each dataset into 10 folds and 9 of them are used for training and the left over fold is used for testing. This process is repeated for all folds and the reported results are an average of all folds.

In our experiments, we consider malware detection as a two class problem – benign or malware detection. In such problems, the classification decision can possibly lie in one of the following categories: (1) True Positive (TP) is correct classification of a malicious ELF file as malicious, (2) True Negative (TN) is a benign ELF file instance classified as benign, (3) False Positive (FP) is mis-classification of a benign ELF file as malicious, and (4) False Negative (FN) is misclassification of a malicious ELF file as benign. We define detection accuracy as following for our experiments:

$$Detection Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Detection Accuracy. The results of our experiments are tabulated in Table 4. It is interesting to note that classical machine learning algorithms in general outperform bio-inspired evolutionary classifiers for our malware detection problem. Except XCS, other evolutionary classifiers provide comparable accuracy as that of classical algorithms. RIPPER achieves a benchmark of 100% on our dataset that is closely followed by PART and J48.

Average Detection Difficulty (ADD). Another important question that we want to investigate: which malware category is the most difficult to detect? To answer this question, we take an average of detection accuracies of different classifiers and report average detection difficulty for each category at the end of each row in Table 4. The average detection difficulty of Bkdrs, Cnstr, DoS, Expts + HT + RK, Fldrs, Trjns, Vrs + Spfr and Wrms datasets' are 96.49%, 96.80%, 97.34%, 97.17%, 96.56%, 96.55%, 97.0% and 96.54% respectively. The analysis of the results shows that Bkdrs, Wrms, Trjns and Fldrs are the most difficult malware categories to detect both for evolutionary and non-evolutionary classifiers. For our subsequent studies, we have short listed top 2 classifiers from Table 4 – RIPPER and J48.

Scalability Analysis of Features' Set. Recall from Table 2 that our dataset consists of 734 instances of benign executables and 709 instances of

Table 4. ELF malware detection Avg. Accuracy and ADD comparison of evolutionary and non-evolutionary algorithms

	Evolutionary Classifiers				Non-Evolutionary Classifiers				
Datasets	cAnt Miner	XCS	UCS	GAss-ADI	C4.5 R	Ripper	PART	J48	ADD
Bkdrs	100 ±0	76.82 ±2.54	99.03 ±0.09	99.57 ±0.05	100 ±0	100 ±0	100 ±0	100 ±0	96.49
Cnstr	100 ±0	77.77 ±6.31	99.86 ±0.04	100 ±0	100 ±0	100 ±0	100 ±0	100 ±0	96.80
DoS	100 ±0	81.41 ±4.36	100 ±0	100 ±0	100 ±0	100 ±0	100 ±0	100 ±0	97.34
Expts RK+HT	99.79 ±0.004	81.33 ±6.74	99.28 ±0.07	99.59 ±0.04	100 ±0	100 ±0	100 ±0	100 ±0	97.17
Fldrs	100 ±0	76.60 ±4.34	99.61 ±0.6	99.87 ±0.03	99.86 ±0.03	100 ±0	100 ±0	100 ±0	96.56
Trjns	99.96 ±0.01	76.28 ±5.38	99.73 ±0.5	100 ±0	100 ±0	100 ±0	99.86 ±0.03	99.86 ±0.03	96.55
Vrs+Spfr	100 ±0	80.21 ±5.25	99.40 ±0.1	99.40 ±0.07	100 ±0	100 ±0	100 ±0	100 ±0	97.0
Wrms	100 ±0	76.31 ±3.97	99.74 ±0.5	99.74 ±0.05	100 ±0	100 ±0	100 ±0	100 ±0	96.54
Avg. Accuracy	99.96	78.34	99.58	99.77	99.98	100	99.98	99.98	

different categories of malware executables. For scalability analysis of our features' set, we have combined all benign and malware datasets' and prepared a comprehensive dataset (i.e. 709+734=1443 instances). Afterwards, we rank all attributes using information gain measure. Finally, we select only those features that have an information gain of 0.1 or above; as a result, we get 88 top ranking features (see Table 5). Subsequently, we have created 9 different datasets by gradually removing the features that have information gain values greater than 0.89, 0.79, 0.69, 0.59, 0.49, 0.39, 0.29 and 0.19 respectively. The number of attributes in each category are shown in Table 5. The ROC curves in Figure 12 have been plotted by varying the threshold on output class probability [13], [30].

It is interesting to see that our idea of using large number of features – extracted from different portions of ELF headers – has shown remarkable resilience, using any classifier, even when we just use 42 top ranked features. The system classifiers on the average merely show a 1 to 2% deterioration in true positive rate. However, the moment we remove the attributes having information gain more than 0.40, the true positive rate of classifiers significantly deteriorate by 1% to 11%.

Robustness against Forged/Spoofed ELF Headers. Now as a next step, we analyze the "robustness" of our features' set in a scenario when malware writers forge/spoof different sections of ELF header of their malware with

Table 5. Scalability analysis of ELF mined features' set

Info-Gain	Removed Attr.	Remaining Attr.	J48 Results (%)			RIPPER Results (%)		
			Accuracy	TP Rate	FP Rate	Accuracy	TP Rate	FP Rate
0.99 - 0.10	0	88	100	100	0.0	100	100	0.0
0.89 - 0.10	3	85	99.58	99.9	0.1	99.45	99.6	0.4
0.79 - 0.10	8	77	99.58	99.7	0.3	99.38	99.4	0.6
0.69 - 0.10	11	66	99.58	99.7	0.3	99.51	99.4	0.6
0.59 - 0.10	3	63	98.68	99.3	0.7	98.82	99.0	1
0.49 - 0.10	21	42	99.02	99.4	0.6	99.09	99.9	0.1
0.39 - 0.10	10	32	98.40	98.6	1.4	98.27	98.2	1.8
0.29 - 0.10	11	21	98.05	97.7	2.3	97.78	98.2	1.8
0.19 - 0.10	8	13	91.40	90.6	9.4	90.57	88.0	12

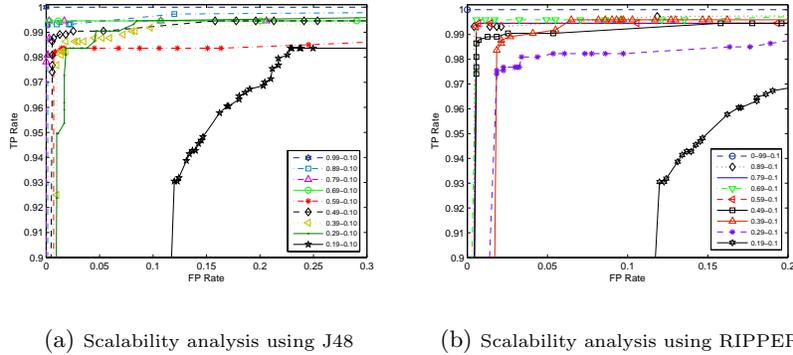(a) Scalability analysis using J48   (b) Scalability analysis using RIPPER

Fig. 12. The magnified ROC plots for scalability analysis of ELF-Miner features' set

that of benign files. We have gradually replaced malware headers with benign file headers and the corresponding true positive rate is given in Table 6. It is surprising to see that both classifiers are able to maintain 99% and 98% true positive rate, even when 77 of 88 features are forged/spoofed. It's because, we have forged/spoofed randomly selected headers (mentioned in the first column of Table 6) and not necessarily the features that have high information gain values. This experiment validates that our features are "robust" against crafty attacks. **Moreover, our experience is that most of executables with forged headers are unable to successfully load into the memory. This makes the task of a "crafty attacker" even more difficult.**

Comprehensibility Analysis of Rules. We now do the comprehensibility analysis of the generated rules by different classifiers. We have given some snapshots of generated rules by different classifiers in Table 7. It is interesting to see that in case of XCS and UCS, both classifiers have generated rules in the form of IF-THEN constructs by specifying intervals for all parameters in the dataset. As a result, the number of rules and their size becomes significantly large that makes it very difficult – if altogether not impossible – to comprehend and interpret them. In comparison, cAntMiner generates rules in IF-THEN form and

Table 6. Malware detection capability of ELF features' set with spoofed headers

Headers Removed	Removed Attr.	Remaining Attr.	J48 Results (%)			RIPPER Results (%)		
			Accuracy	TP Rate	FP Rate	Accuracy	TP Rate	FP Rate
None	0	88	100	100	0.0	100	100	0.0
[Reloc, DynSym]Sec,GOT	18	70	99.45	99.6	0.4	99.23	99.3	0.7
ELF, [Cmnt, BSS, Dyn]SecH	10	60	99.24	99.6	0.4	99.03	99.2	0.8
SymTab Sec	12	47	99.45	99.6	0.4	99.37	99.6	0.4
[DynStr, DynSym, Fini, Hash, Init]SecH	7	40	99.51	99.7	0.3	99.70	99.7	0.3
Segments	11	29	99.51	99.7	0.3	99.24	99.4	0.6
[StrTab, SymTab]SecH	10	19	99.51	99.7	0.3	99.44	99.7	0.3
[GOTPLT, RELDyn]SecH	11	10	99.58	99.6	0.4	98.74	99.0	1
[Note, GOT, SBSS, PLT, RData]SecH, Dyn Sec	0	11	99.37	99.6	0.4	98.88	99.3	0.7

Table 7. Comprehensibility analysis of rules generated by all evolutionary and non-evolutionary classifiers

```
(1) cAnt Miner
IF GOTPLTSHENTSIZE [< 4.0] AND RELDYNSHSIZE [< 48.0] : malware
IF RELDYNSHENTSIZE [>= 8.0 AND STRTABSHTYPE [< 3.0] : benign
IF RELAR386JMPSLOT [< 5.0] : malware
(2) XCS
IF ELFHEININDENT [0.0 0.5] AND ELFHETYPE [0.0 0.52] .... HASHTABLESIZE [0.0 0.06]: malware
IF ELFHEININDENT [0.0 0.8] AND ELFHETYPE [0.0 0.7] .... HASHTABLESIZE [0.0 0.35]: malware
(3) UCS
IF ELFHEININDENT [0.0 1.0] AND ELFHETYPE [0.0 1.0] ... HASHTABLESIZE [0.0 1.0]: benign
IF ELFHEININDENT [0.0 1.0] AND ELFHETYPE [0.0 1.0] ... HASHTABLESIZE [0.0 0.9] : malware
(4) GAssist-ADI
IF RELAR386JMPSLOT [> 0.0] [661/661] : benign
ELSE IF RELAR386JMPSLOT [<= 0.0] [177/177] : malware
ELSE benign
(5) C4.5 Rules
IF BSSSHTYPE [<= 1.0] [465/465] : benign
ELSE IF RELAR38632 [> 11.0] [659/659] : benign
ELSE IF RELAR38632 [<= 11.0] AND BSSSHTYPE [> 1.0] [23/23] : malware
ELSE benign
(6) RIPPER
IF RELAR386JMPSLOT [> 0.0] [661/661] : benign
ELSE malware
IF RELR386COPY [> 10.0] [650/650] : benign
ELSE IF COMENTSHTYPE [> 0.0] [15/54]: malware
ELSE benign
(7) PART
IF RELAR38632 [> 11.0] [658/658] : benign
ELSE IF BSSSHTYPE [> 1.0] [24/225] : malware
ELSE benign
IF RELAR386JMPSLOT [> 0.0] [660/660] : benign
ELSE malware
(8) J48
IF GOTPLTSHTYPE [<= 0]
. . IF RELR386GLOBDAT [<= 247] : malware
. . ELSE IF RELR386GLOBDAT [> 247]
. . . . IF COMENTSHTYPE [<= 0] : benign
. . . . ELSE IF COMENTSHTYPE [> 0] : malware
ELSE IF GOTPLTSHTYPE [> 0]
. . IF SYMSTTHIPROC [<= 19]
. . . . IF RELR386RELATIVE [<= 3] : malware
. . . . ELSE IF RELR386RELATIVE [> 3] : benign
. . . . ELSE IF SYMSTTHIPROC [> 19] : benign
```

GAssist-ADI generates rules in IF-THEN-ELSE form. It is interesting to note that both classifiers do not add all attributes in the antecedent part of the rules; as a result, the number and size of rules is significantly smaller compared with XCS and UCS that increases the comprehensibility and interpretability of their rules.

In case of classical machine learning algorithms, RIPPER, PART and C4.5 Rules generate compact and simple rules in the form of IF-THEN-ELSE. They also highlight the number of data samples covered by a specific rule. These classifiers do not add all input parameters in the antecedent part of the rules. Consequently, their comprehensibility is high. J48 is the only classifier that builds complex hierarchical rules. Its rules are accurate but complex as compared with other classifiers. A closer look at the rules confirm our forensic analysis in Section 6: the classification power of features of relocation section (*.rela*) is very high followed by the information from different section headers – *.comment*, *.note*, *.strtab*, *.symtab*, and *.sbss*.

Timing Analysis. We now analyze the processing overhead of ELF-Miner that includes time for features extraction, preprocessing, classifier training and testing. On an average, features extraction and preprocessing times per instance are 15.391 and 0.96 milliseconds respectively. The training times of J48 and

RIPPER are 0.173 and 0.616 milliseconds per instance respectively. Similarly, the testing time of both J48 and RIPPER is 6.9 microseconds. So overall processing overhead of ELF-Miner is approximately 16 to 17 milliseconds per instance (file) using any of the two best classifiers. Such a small processing overhead makes our ELF-Miner framework feasible for realtime deployment.

9 Conclusions

In this paper, we have introduced ELF-Miner, a malware detection framework that mines structural features – extracted from different sections of ELF headers – of Linux executables. We have done detailed forensic analysis to investigate the fields that have high classification potential to detect malicious executables. Our system achieves more than 99.9% accuracy depending on the selected classifier. We have also done scalability experiments to show that ELF-Miner is able to provide high detection accuracy even with a small number of features. We have also shown that forging the features does not result in significantly degrading the detection accuracy. It will be an interesting future work to put efforts to establish the generality of our approach on executables of other operating systems – Mac OS X, Symbian etc.

Acknowledgments

References

1. Fedora Project: Security features of Linux. `http://fedoraproject.org/wiki/Security/Features`(viewed-on-June-18-2010).
2. Linux market share sees increase. `http://www.gamegrep.com/news/8342-linux\_market\_share\_sees\_increase/`(viewed-on-July-07-2010).
3. Offensive Computing: Online malware database. `http://www.offensivecomputing.net/`.
4. Symbol Table Layout and Conventions. `http://docs.sun.com/app/docs/doc/819-0690/stlac?a=view`(viewed-on-July-7-2010).
5. The Section Header Table (ELF). `http://www.cs.ucdavis.edu`(viewed-on-July-07-2010).
6. User Commands, Linux Man Pages Section 1. `http://docs.sun.com/app/docs/doc/819-2239/strip-1?a=view`(viewed-on-July-07-2010).
7. VX Heavens: Free malware collection website. `http://www.vxheaven.com`.
8. J. Alcala-Fdez, S. Garcia, F. J. Berlanga, A. Fernandez, L. Sanchez, M. J. del Jesus, and F. Herrera. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 13(3):307–318, 2009.

9. J. Bacardit and J. Garrell. Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. *Learning Classifier Systems*, 4399:59–79, 2007.

10. E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

11. W. W. Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

12. T.M. Cover and J.A. Thomas. *Elements of information theory.* John Wiley & Sons, 2006.

13. T. Fawcett. ROC graphs: Notes and practical considerations for researchers. *Machine Learning*, 31:1–38, 2004.

14. E. Frank and I.H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the International Conference on Machine Learning*, pages 144–151. Citeseer, 1998.

15. J. Holland, L. Booker, M. Colombetti, M. Dorigo, D. Goldberg, S. Forrest, R. Riolo, R. Smith, P. Lanzi, W. Stolzmann, et al. What is a learning classifier system? In *Internatinoal Workshop on Learning Classifier Systems*, volume 1813 of *Lecture Notes in Artificial Intelligence*, pages 3–32. Springer, 2000.

16. K. Hovsepian, P. Anselmo, and S. Mazumdar. Supervised inductive learning with lotkavolterra derived models. *Knowledge and Information Systems*, 2010.

17. D. H. Johnson and S. Sinanovic. Symmetrizing the kullback-leibler distance. *IEEE Transactions on Information Theory*, 2001.

18. J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 470–478. ACM, 2004.

19. M.M. Masud, L. Khan, and B.M. Thuraisingham. A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers*, 10(1):33–45, 2008.

20. F .Otero, A. Freitas, and C. Johnson. cAnt-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes. In *Ant Colony Optimization and Swarm Intelligence*, volume 5217 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2008.

21. R. Perdisci, A. Lanzi, and W. Lee. McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Proceedings of the Computer Security Applications Conference*, pages 301–310. Citeseer, 2008.

22. J. R. Quinlan. *C4. 5: programs for machine learning.* Morgan Kaufmann, 1993.

23. J. R. Quinlan. MDL and categorical theories (continued). In *Proceedings of the International Conference on Machine Learning*, pages 464–470. Citeseer, 1995.

24. A. Y. Rodriguez-Gonzalez, J. F. Martinez-Trinidad, J. A. Carrasco-Ochoa, and J. Ruiz-Shulcloper. Rp-miner: a relaxed prune algorithm for frequent similar pattern mining. *Knowledge and Information Systems*, 2010.

25. M.G. Schultz, E. Eskin, E. Zadok, and S.J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49. Citeseer, 2001.

26. M. Z. Shafiq, S. Momina Tabish, and M. Farooq. Pe-probe: Leveraging packer detection and structural information to detect malicious portable executables. Proceedings of the International Virus Bulletin Conference, 2009.

27. M. Z. Shafiq, S. Momina Tabish, F. Mirza, and M. Farooq. A framework for efficient mining of structural information to detect zero-daymalicious portable executables. Technical report, TRnexGINRC- 2009-21, nexGIN RC, 2009.

28. M. Z. Shafiq, S. Momina Tabish, F. Mirza, and M. Farooq. PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In *Proceedings of the Recent Advances in Intrusion Detection*, volume 5758 of *Lecture Notes in Computer Science*, pages 121–141. Springer, 2009.
29. T. I. Standard. Executable and Linking Format (ELF) Specification Version 1.1. *TIS Committee*, 1993.
30. S. D. Walter. The Partial Area Under The Summary ROC Curve. *Statistics in medicine*, 24(13):2025–2040, 2005.
31. S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
32. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.
33. Richong Zhang and Thomas T. Tran. An information gain-based approach for recommending useful product reviews. *Knowledge and Information Systems*, 2010.

Appendix A: Description of ELF Sections

Table 8. ELF sections & description [29] [5]

No	Section	Description
1	.text	Executable instructions
2	.bss	Uninitialized data in program image
3	.comment	Version control information
4	.data	Initialized data variables in image
5	.data1	Initialized data variables in image
6	.debug	Program debug symbolic information
7	.dynamic	Dynamic linking information
8	.dynstr	Dynamic string section
9	.dynsym	Dynamic symbol information
10	.fini	Process termination code
11	.hash	Hash table
12	.init	Process initialization code
13	.got	Global offset table
14	.interp	Path name for a program interpreter
15	.line	Line number information of symbolic debug
16	.note	File notes
17	.plt	Procedure link table
18	.rodata	Read only data
19	.rodata1	Read only data
20	.shstrtab	Section header string table
21	.strtab	String table
22	.symtab	Symbol table
23	.sdata	Initialized non-const global and static data
24	.sbss	Static better save space
25	.lit8	8-byte literal pool
26	.gptab	Size criteria info for placement of data items in the .sdata
27	.conflict	Additional dynamic linking information
28	.tdesc	Targets description
29	.lit4	4-byte literal pool
30	.reginfo	Information about general purpose registers for assembly file
31	.liblist	Shared library dependency list
32	.rel.dyn	Runtime relocation information
33	.rel.plt	Relocation information for PLT
34	.got.plt	Holds read-only portion of global Offset Table