深圳大學 Shenzhen University

Effective Malware Detection based on Behavior and Data Features

Zhiwu Xu, Cheng Wen, Shengchao Qin, and Zhong Ming

College of Computer Science and Software Engineering,
Shenzhen University, China

- **Introduction**
- Approach
- Experiments
- Conclusion

Malware

- Malicious software:
  - Computer viruses, worms, Trojan horses, ransomware, spyware, adware, scareware, and other intrusive codes

- Recent report from McAfee:
  - More than 650 million malware samples detected in Q1, 2017, in which more than 30 million ones are new.

Signature-based method

- To compare with the known signatures,
  - Comodo, McAfee, Kaspersky, Kingsoft, and Symantec

- Can be easily evaded by the evasion techniques
  - packing, variable-renaming, and polymorphism.

Heuristic-based method

- To identity malicious patterns though either static analysis or dynamic analysis
- However, heavy-weight or Inefficient

Machine learning approaches

- Most of existing work focus on behaviour features, without data information
  - binary codes, opcodes and API calls

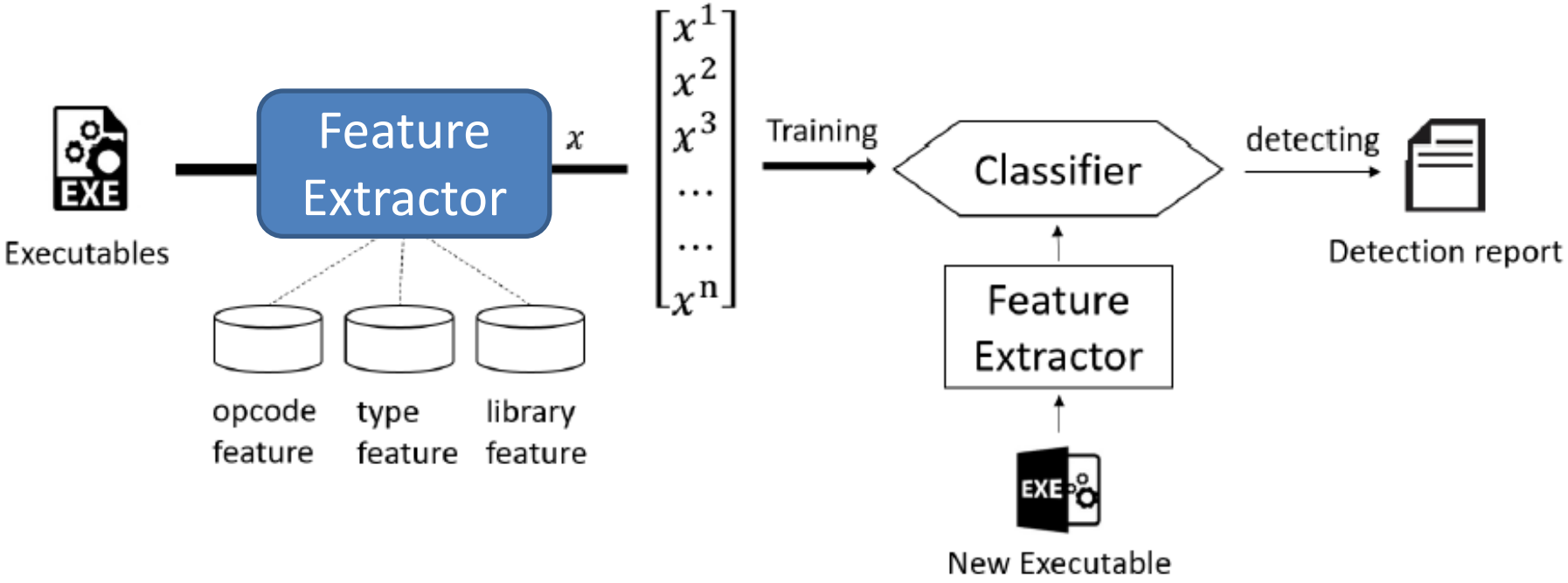
- Can be easily evaded
  - previously-unseen behaviors
  - obfuscate

- Introduction
- **Approach**
- Experiments
- Conclusion

Our approaches

- Based on machine learning
- Consider both the behaviour information and the data information.
- Consider the time-split samples and obfuscated samples

Framework

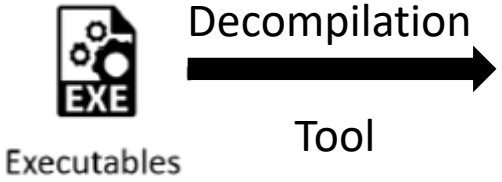

EXE
Executables
Feature Extractor
opcode feature
type feature
library feature
x
x^1
x^2
x^3
...
...
x^n
Training
Classifier
detecting
Detection report
Feature Extractor
EXE
New Executable

Feature Extractor

- Decompilation
- Information Extraction
- Feature Selection and representation

Feature Extractor

- **Decompilation**
- Information Extraction
- Feature Selection and representation




```
.text:00401000 ?Bubble_Sort@@YAXPAHH@Z proc near       ; CODE XREF: _main+47↓p
.text:00401000
.text:00401000 temp            = dword ptr -0Ch
.text:00401000 j               = dword ptr -8
.text:00401000 i               = dword ptr -4
.text:00401000 num             = dword ptr  8
.text:00401000 n               = dword ptr  0Ch
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 4Ch
.text:00401006                 push    ebx
.text:00401007                 push    esi
.text:00401008                 push    edi
.text:00401009                 mov     [ebp+i], 0
.text:00401010                 jmp     short loc_40101B
.text:00401012 ; ---------------------------------------------------------------------------
.text:00401012
.text:00401012 loc_401012:                             ; CODE XREF: Bubble_Sort(int *,int):loc_401098↓j
.text:00401012                 mov     eax, [ebp+i]
.text:00401015                 add     eax, 1
.text:00401018                 mov     [ebp+i], eax
.text:0040101B
.text:0040101B loc_40101B:                             ; CODE XREF: Bubble_Sort(int *,int)+10↑j
.text:0040101B                 mov     eax, [ebp+i]
.text:0040101E                 cmp     eax, [ebp+n]
.text:00401021                 jge     short loc_40109D
.text:00401023                 mov     [ebp+j], 0
.text:0040102A                 jmp     short loc_401035
.text:0040102C ; ---------------------------------------------------------------------------
```

ASM codes

Feature Extractor

- Decompilation
- **Information Extraction**
- Feature Selection and representation

```
.idata:00402008 ; DWORD __stdcall GetCurrentThreadId()
.idata:00402008                 extrn __imp__GetCurrentThreadId@0:dword
.idata:00402008                                         ; CODE XREF: ___security_init_cookie+43↑p
.idata:00402008                                         ; DATA XREF: ___security_init_cookie+43↑r
.idata:0040200C ; BOOL __stdcall QueryPerformanceCounter(LARGE_INTEGER *lpPerformanceCount)
.idata:0040200C                 extrn __imp__QueryPerformanceCounter@4:dword
.text:00401012
.text:00401012 loc_401012:                              ; CODE XREF: Bubble_Sort(int *,int):loc_401098↓j
.text:00401012                  mov     eax, [ebp+i]
.text:00401015                  add     eax, 1
.text:00401018                  mov     [ebp+i], eax
.text:0040101B
.text:0040101B loc_40101B:                              ; CODE XREF: Bubble_Sort(int *,int)+10↑j
.text:0040101B                  mov     eax, [ebp+i]
.text:0040101E                  cmp     eax, [ebp+n]
.text:00401021                  jge     short loc_40109D
.text:00401023                  mov     [ebp+j], 0
.text:0040102A                  jmp     short loc_401035
```

System call

Opcode

Data Type： int *

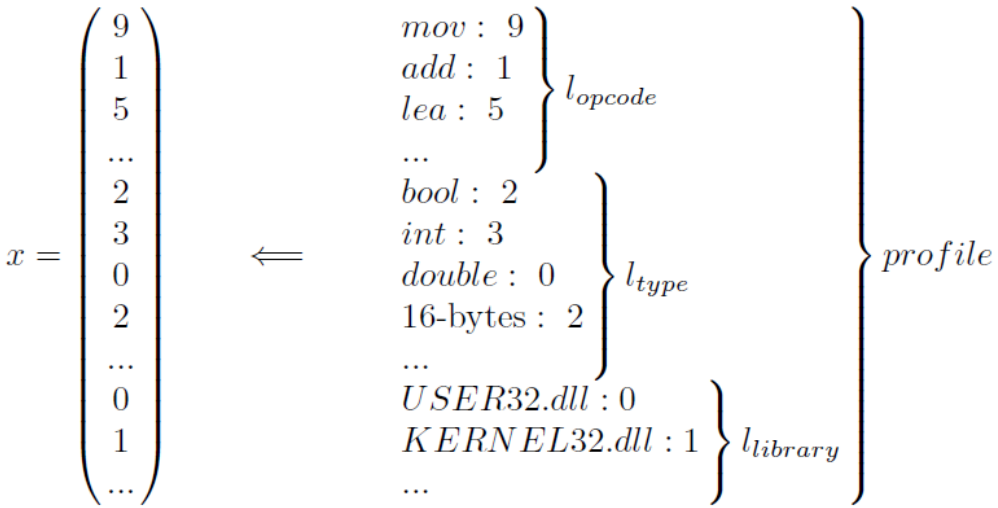
Feature Extractor

- Decompilation
- Information Extraction
- **Feature Selection and representation**

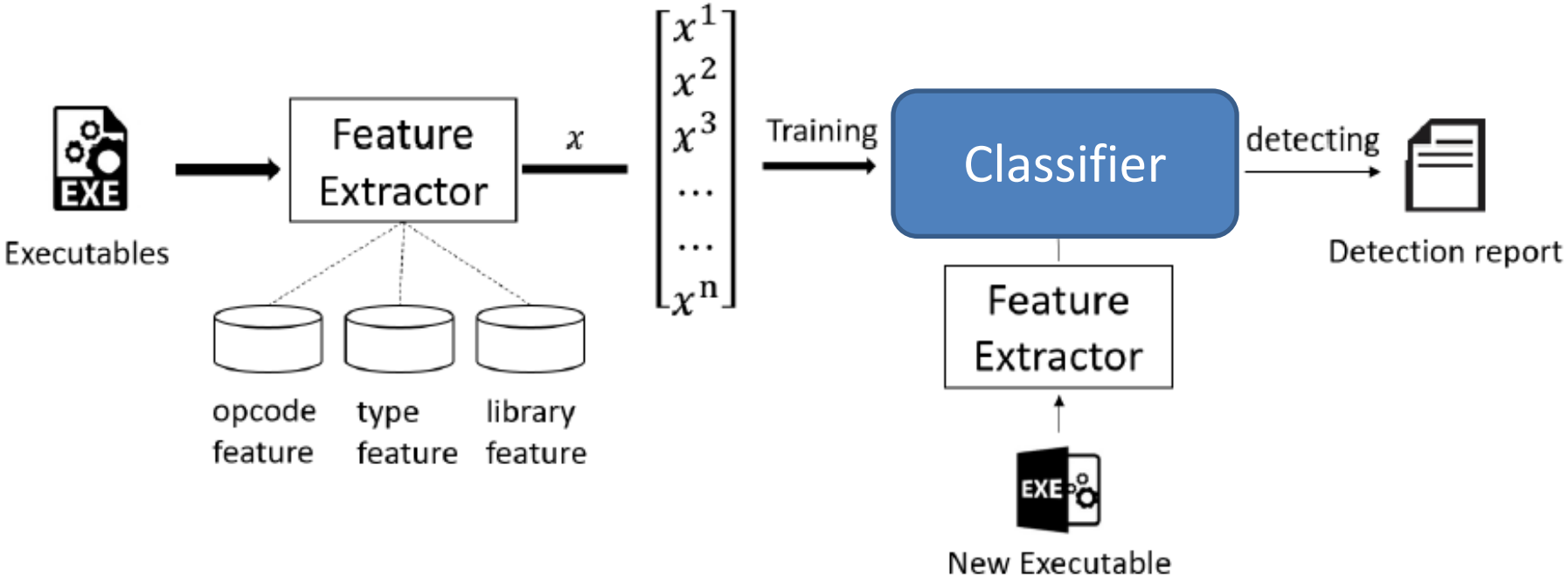
Selection:

Term Frequency and Inverse Document Frequency (TF-IDF)

Representation:



Framework


EXE
Executables
Feature Extractor
opcode feature
type feature
library feature
x
x^1
x^2
x^3
$\cdots$
$\cdots$
x^n
Training
Classifier
Feature Extractor
EXE
New Executable
detecting
Detection report

Classifier

- Classifier Training
  - An executable e can be represented as a vector x. D_0 represent the available dataset with known categories. Our training problem is to find a classifier $C: X \rightarrow [0,1]$ such that

$$min \sum_{(x,c) \in D_0} d(C(x) - c)$$

- Malware Detection
  - Given an executable e and its vector representation, the goal of the detection is to find c such that

$$\min d(C(x) - c)$$

- Introduction
- Approach
- **Experiments**
- Conclusion

Experiments

Malware dataset (11376 samples)

- BIG 2015 Challenge
- theZoo aka Malware DB

Benign dataset (8003 samples)

- QIHU 360 software

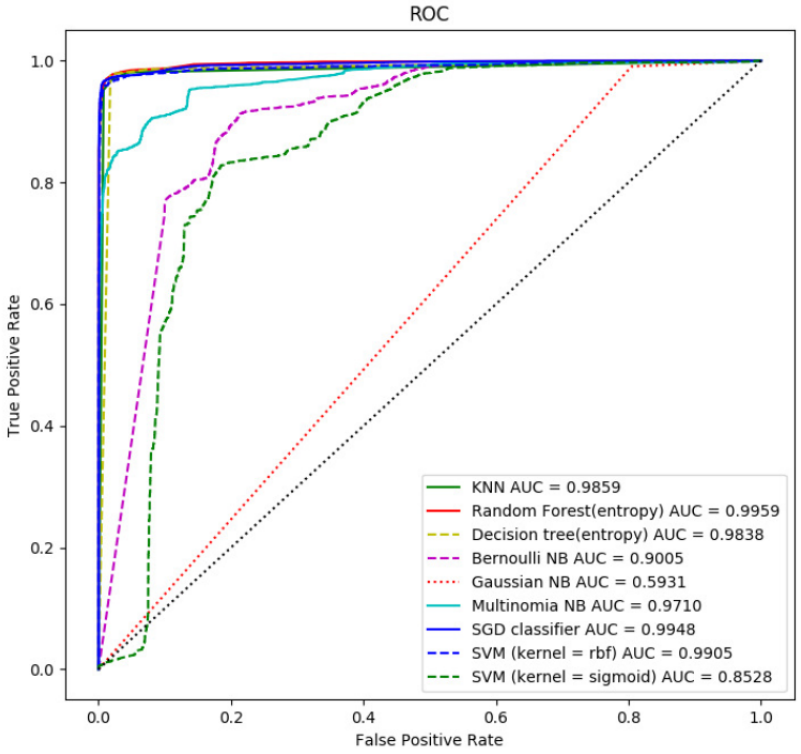
(with the total size of **250 GB**)

Cross Validation Experiments

10-*fold cross validation*

Table 2. Results of Different Methods

Classifier	PPV	TPR	FPR	ACY
KNN K=1	0.9609	0.9830	0.0281	0.9764
KNN K=3	0.9572	0.9798	0.0307	0.9736
KNN K=5	0.9556	0.9763	0.0319	0.9715
KNN K=7	0.9556	0.9763	0.0319	0.9715
DT criterion='gini'	0.9658	0.9797	0.0243	0.9773
DT criterion='entropy'	0.9685	0.9787	0.0223	0.9781
RF (n=10, gini)	0.9678	0.9787	0.0228	0.9778
RF (n=10, entropy)	0.9692	0.9800	0.0218	0.9788
Gaussian Naive Bayes	0.9381	0.1990	0.0092	0.6638
Multinomial Naive Bayes	0.9494	0.6590	0.0247	0.8446
Bernoulli Naive Bayes	0.8726	0.6831	0.0701	0.8279
SVM kernel='linear'	0.9581	0.9896	0.0304	0.9778
SVM kernel='rbf'	0.9686	0.9119	0.0207	0.9514
SVM kernel='sigmoid'	0.9671	0.7308	0.0232	0.8580
SGD Classifier	0.9582	0.9862	0.0302	0.9765

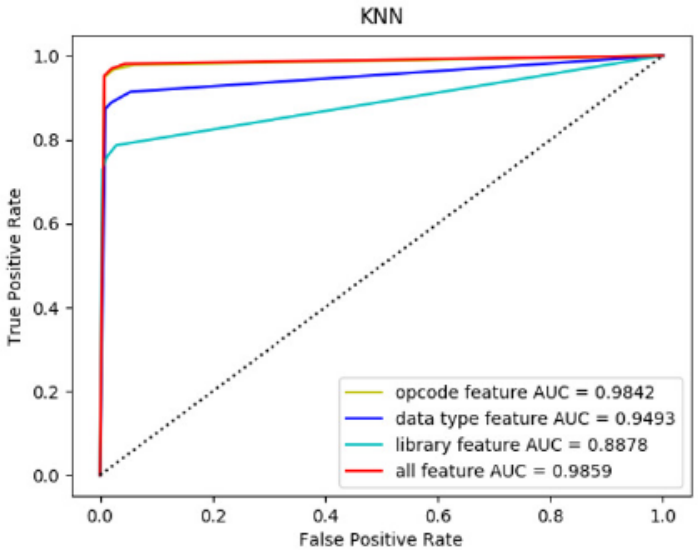


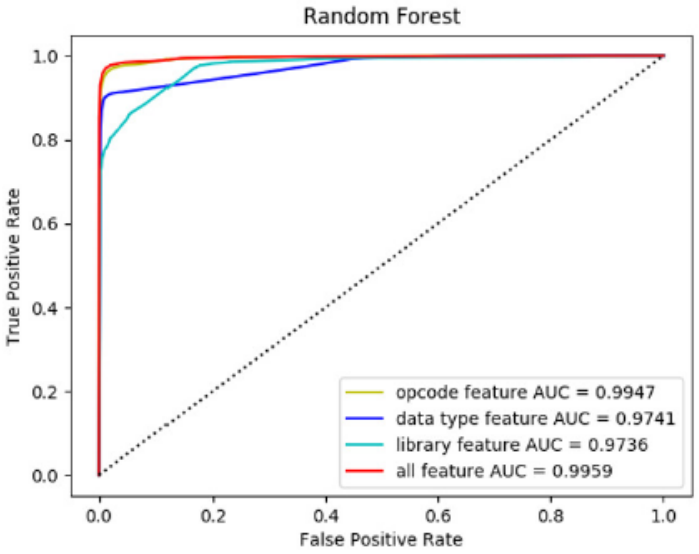
Runtime performance

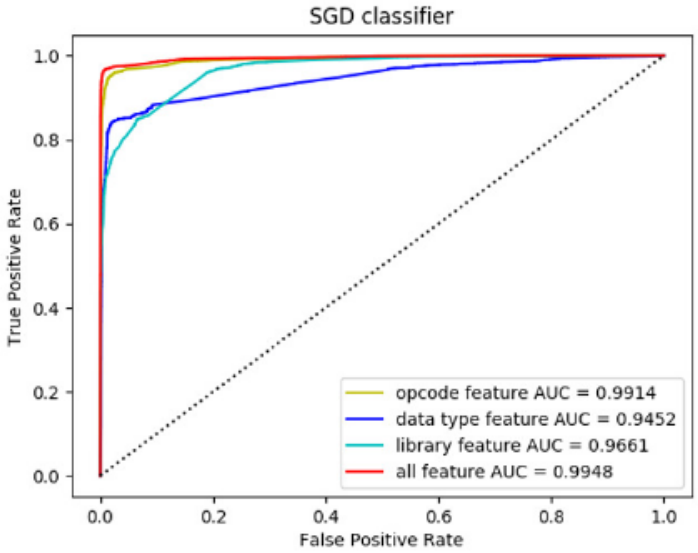
Classifier	Training Time (s)	Testing Time (s)
KNN (k = 1)	0 + (16.477)	178.789
KNN (k = 3)	0 + (16.369)	199.474
KNN (k = 5)	0 + (16.517)	207.052
KNN (k = 7)	0 + (16.238)	210.557
DT (criterion = ‘gini’)	23.442	0.067
DT (criterion = ‘entropy’)	13.485	0.066
RF (n = 10, gini)	4.115	0.086
RF (n = 10, entropy)	3.791	0.077
Gaussian Naïve Bayes	3.093	0.480
Multinomial Naïve Bayes	1.535	0.035
Bernouli Naïve Bayes	1.826	0.828
SVM (kernel = ‘linear’)	150.022	14.494
SVM (kernel = ‘rbf’)	799.310	50.196
SVM (kernel = ‘sigmoid’)	1303.607	130.178
SGD Classifier	22.569	0.048

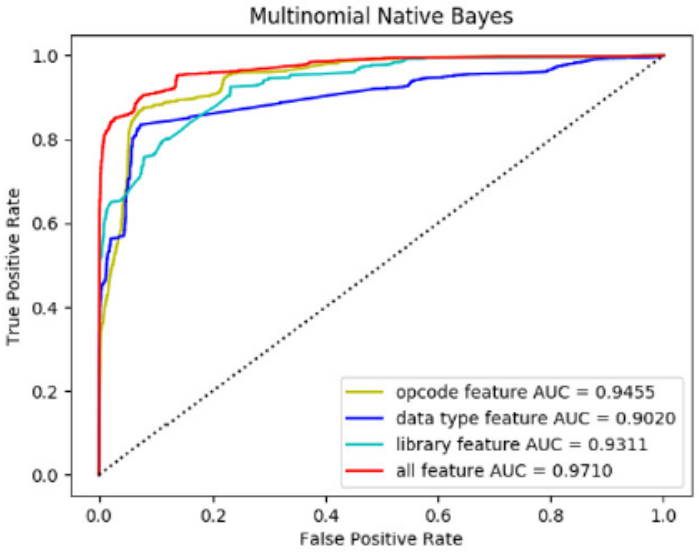
250GB, 15.6 hours, Decompile 0.22s/MB
182GB, 10.5 hours, Extract features 0.20s/MB

Feature Experiment









Time-Split Experiment

We use some fresh malware samples, which were collected dated from January 2017 to July 2017, from the DAS MALWERK website.

Date	Num	RF Classifier	Accuracy
July, 2017	12	6	0.500
June, 2017	61	29	0.475
May, 2017	87	76	0.874
April, 2017	31	28	0.903
March, 2017	48	42	0.875
February, 2017	69	61	0.884
January, 2017	56	53	0.946
Total	364	295	0.810

Obfuscation Experiments

Obfuscation tools：**Obfuscator**

- Change code execution flow

Obfuscation tools：**Unest**

- rewriting digital changes equivalently
- confusing the output string
- pushing the target code segment into the stack and jumping to it to confuse the target code
- obfuscating the static libraries

Table 6. Results of Obfuscated Malware Samples

Tools	Number	RF Classifier	Accuracy
Obfuscator	50	50	100%
Unest	60	60	100%

- Introduction
- Approach
- Experiments
- **Conclusion**

Conclusion

- Machine learning methods based on the opcodes, data types and system libraries.
- Carried out some interesting experiments.
- Capable of detecting some fresh malware
- Has a resistance to some obfuscation techniques

That ’s all.
Thank you very much!