MONTANA STATE UNIVERSITY

Practical Adversarial Malware Example Attacks and Defenses

Fangtian Zhong

Gianforte School of Computing

Email: fangtian.zhong@montana.edu

2024.05.28

Who Am I?

Fangtian Zhong

-------**Assistant professor**

Montana State University

Research interests

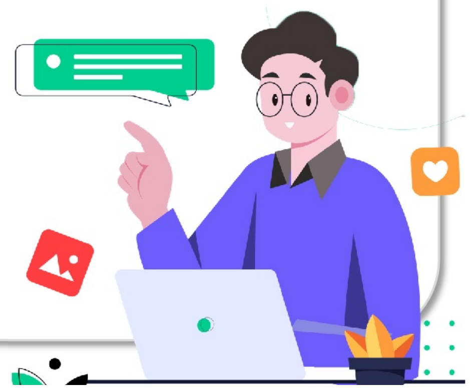
- Software security
- Program analysis
- Machine learning for cybersecurity

Education

- Ph.D. in computer science, George Washington University
- B.E. in Software Engineering, Northeast Normal University

Postdoc Training

- University of Notre Dame
- Pennsylvania State University



Outline

1 Introduction

2 Adversarial Malware Example Attacks

3 Adversarial Malware Example Defenses

Section One

01

Introduction

Who needs to worry about software security?

Researchers

Software engineers

EVERYONE

- Bank records
- Medical records
- Credit card information
- Technical project data
- Etc.

Results of unsecure software

- Identity theft
- Confidential information leakage
- Loss of data
- Slow computer
- Denied service
- Financial loss...

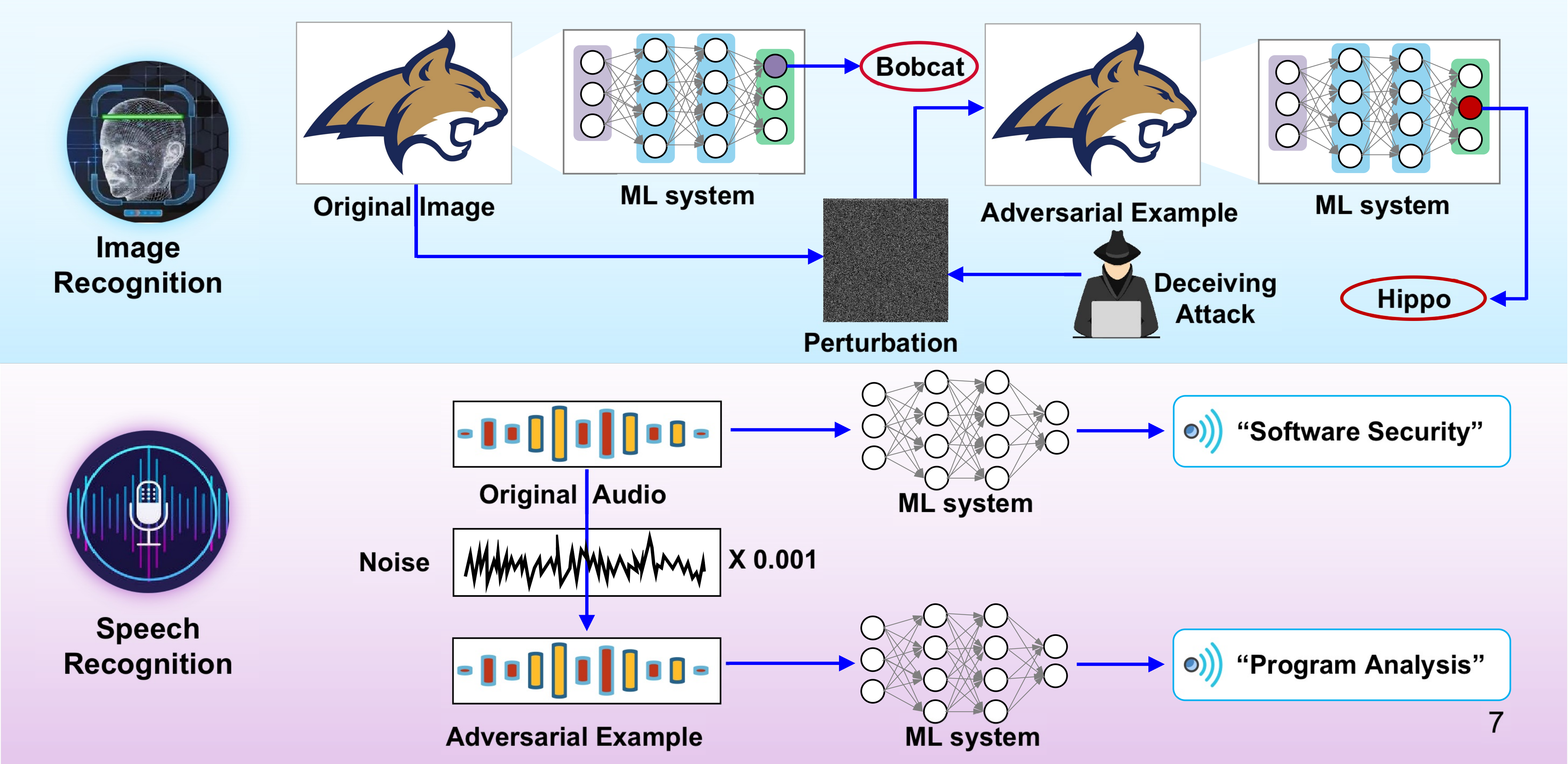
Types of software security issues

Section Two

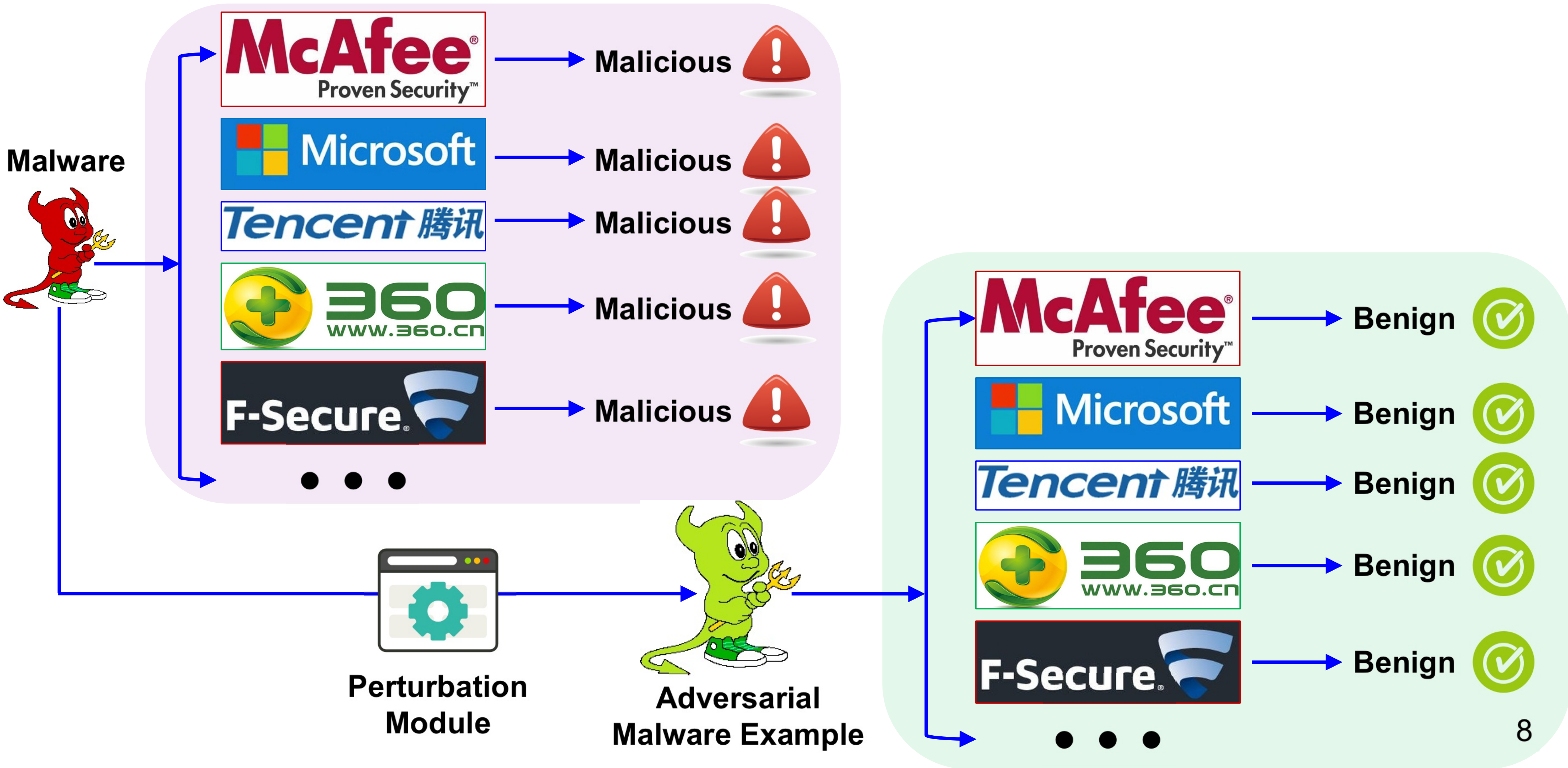
02

Adversarial Malware Example Attacks MalFox

Adversarial Examples



Motivation


Malware
McAfee Proven Security
Microsoft
Tencent 腾讯
360 www.360.cn
F-Secure
Malicious
Malicious
Malicious
Malicious
Malicious
Perturbation Module
Adversarial Malware Example
McAfee Proven Security
Microsoft
Tencent 腾讯
360 www.360.cn
F-Secure
Benign
Benign
Benign
Benign
Benign

Challenges-Adversarial Malware Example Generation

To generate practical adversarial malware examples

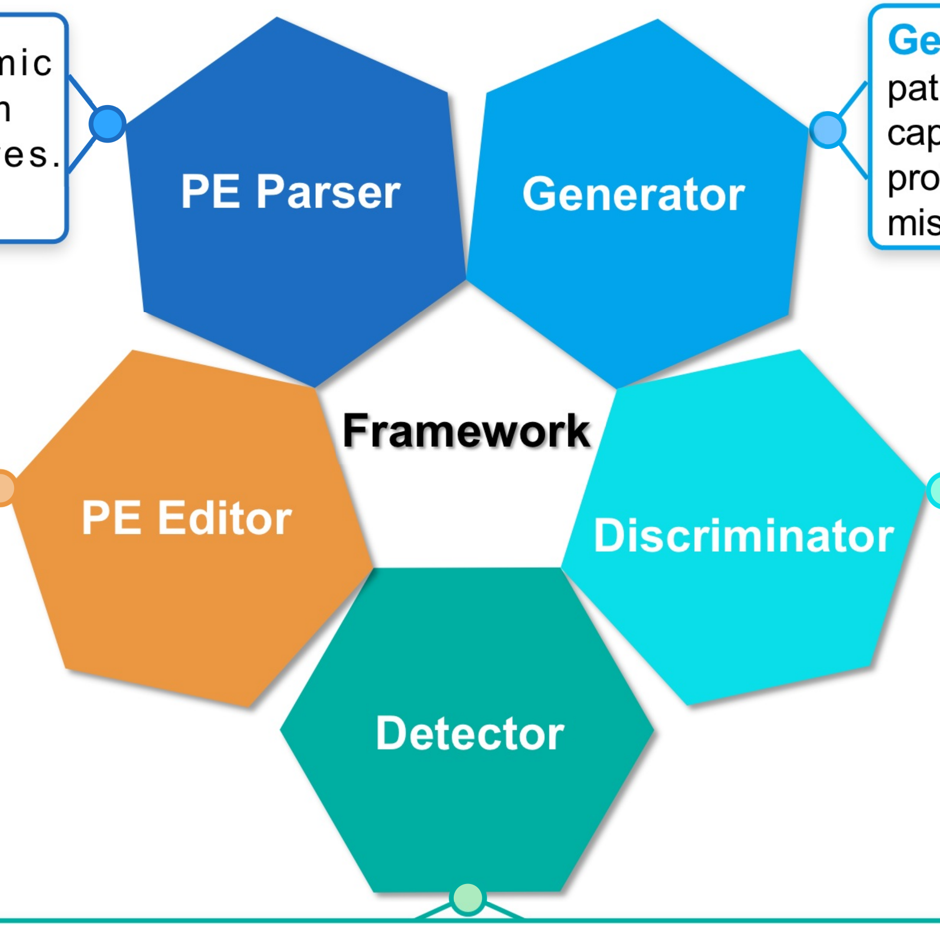
Challenges

1. Instrumenting malware samples to attach perturbation.
2. Remaining the functionality of original malware samples.
3. Evading detection by malware detectors under a black-box setting.

MalFox Framework

PE Parser retrieves the dynamic link libraries (DLLs) and system functions in programs as features.

Generator produces a perturbation path that improves malware's evasive capability while maximizing the probability of Discriminator making a mistake.



PE Editor follows the perturbation path output from Generator to produce an adversarial malware example.

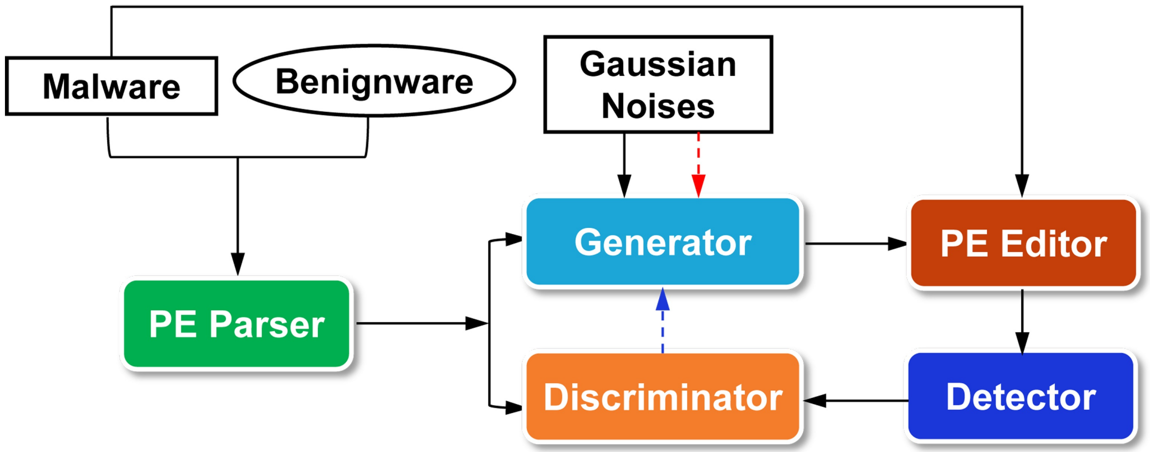
Discriminator estimates the probability of maliciousness for a malware program, and provides gradient information to train Generator.

Detector ensures the reliability of our datasets, provides labels for Discriminator, and validates the performance of MalFox.

MalFox Training and Test Procedure

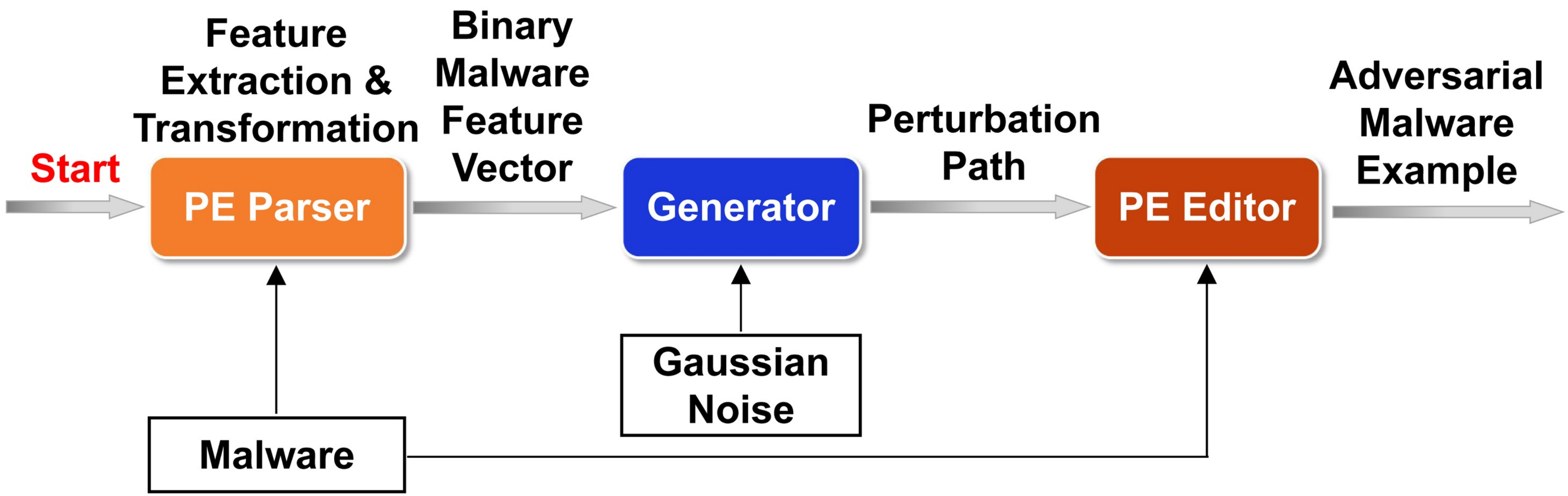
Algorithm 1 MalFox Training Procedure	
1	Convert each malware and benignware program in the training dataset into a binary feature vector by PE Parser;
2	**while** not converging **do**
3	Sample a minibatch of malware feature vectors and three-dimensional Gaussian noises, combine each malware feature vector with a noise sample, and input the results to Generator;
4	Generator generates perturbation paths and inputs them to PE Editor;
5	PE Editor produces adversarial malware examples following the perturbation paths;
6	Sample a minibatch of benignware feature vectors;
7	Update Discriminator's parameters with the adversarial malware examples and benignware programs by descending along the gradient of L_D;
8	Sample three-dimensional Gaussian noises, combine each with a malware feature vector in the minibatch, and input the results to Generator;

Algorithm 1 MalFox Training Procedure	
9	Generator generates perturbation paths and inputs them to PE Editor;
10	PE Editor produces adversarial malware examples following the perturbation paths;
11	Detector labels the adversarial malware examples;
12	Update Generator's parameters with the newly generated adversarial malware examples by descending along the gradient of L_G
13	**end while**



MalFox Training and Test Procedure

How to generate a powerful adversarial malware example?



MalFox Component: PE Parser

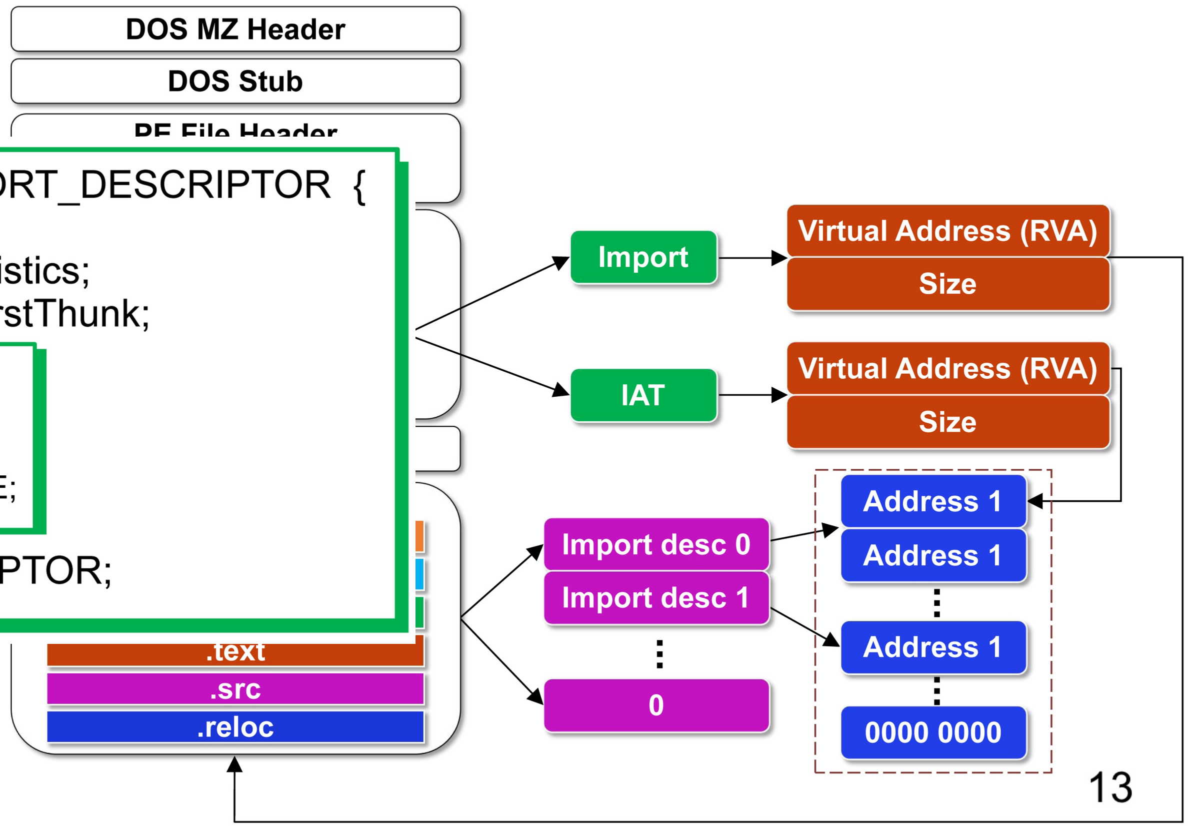
DataDirectories → Import(RVA to FOA) → Import desc. → Name (RVA to FOA) → Dll Name

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
      union {
            DWORD  Characteristics;
            DWORD  OriginalFirstThunk;
      } DUMMYUNIONN
      DWORD  TimeData
      DWORD  Forwarder
      DWORD  Name;
}
```

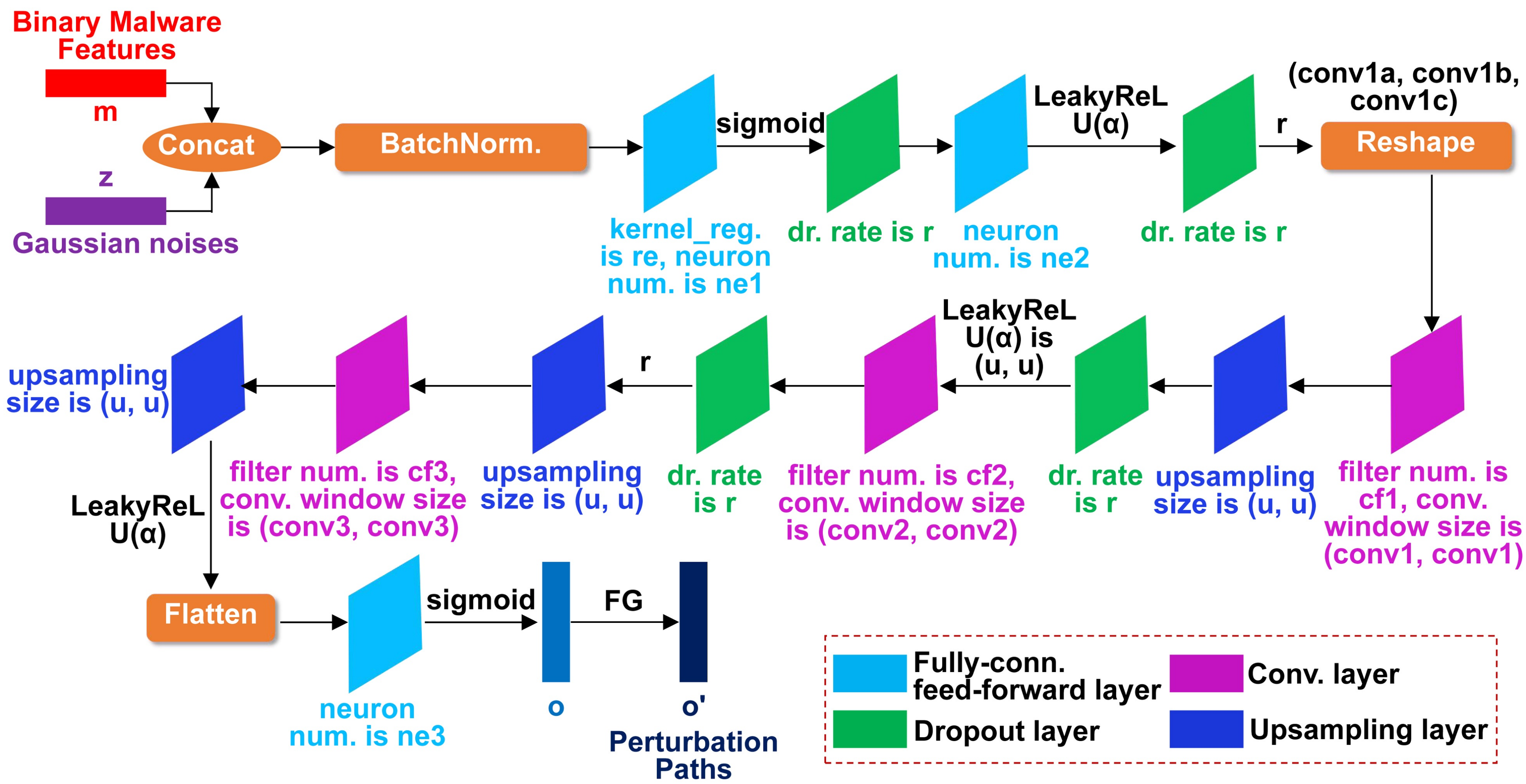
```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
      union {
            DWORD  Characteristics;
            DWORD  OriginalFirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
```

```
typedef struct _IMAGE_IMPORT_BY_NAME {
      WORD  Hint;
      CHAR  Name[1];
} IMAGE_TMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

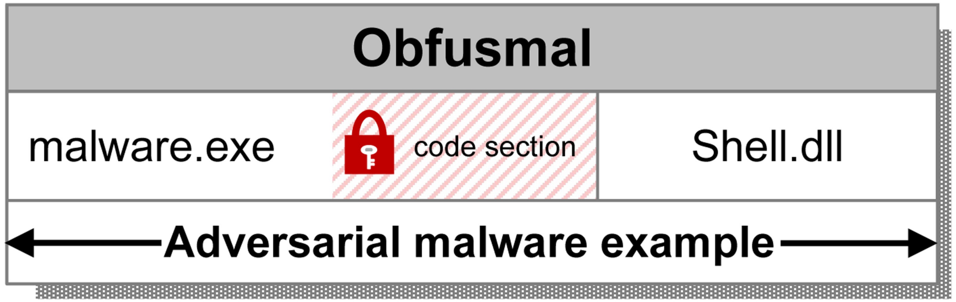
OriginalFirst Thunk (RVA to FOA) → **INT (RVA to FOA)**

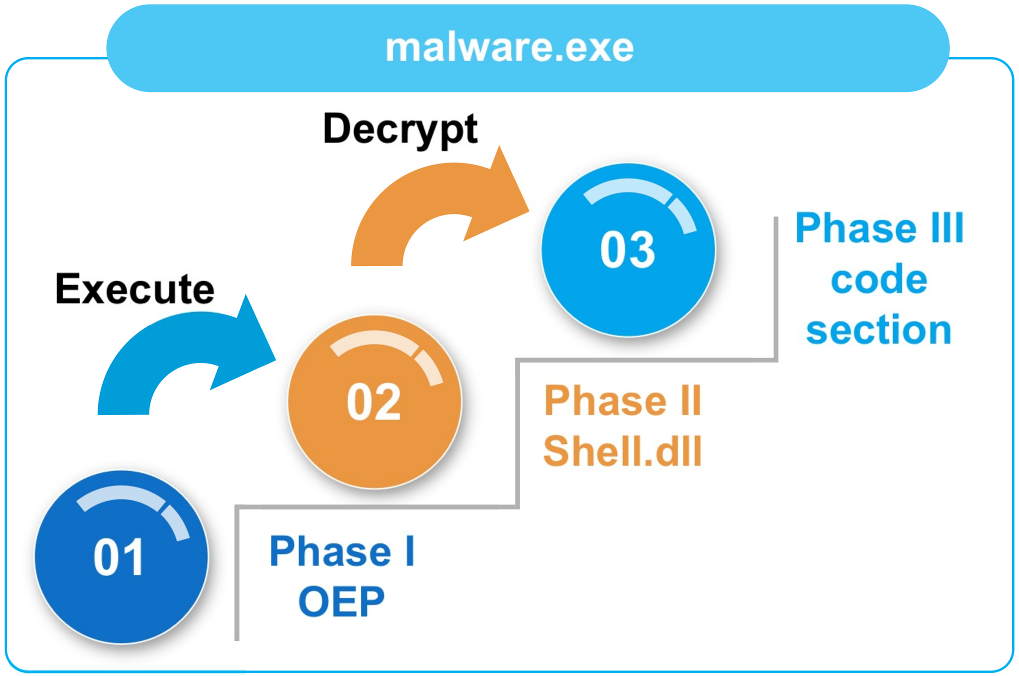


MalFox Component: Generator



MalFox Component: PE Editor (Obfusmal)



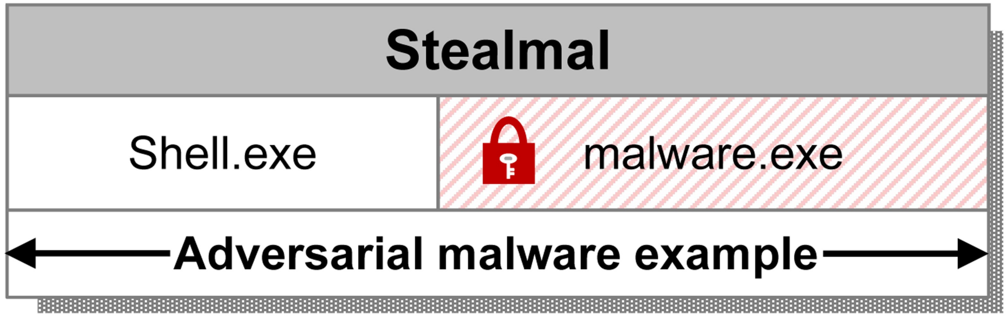


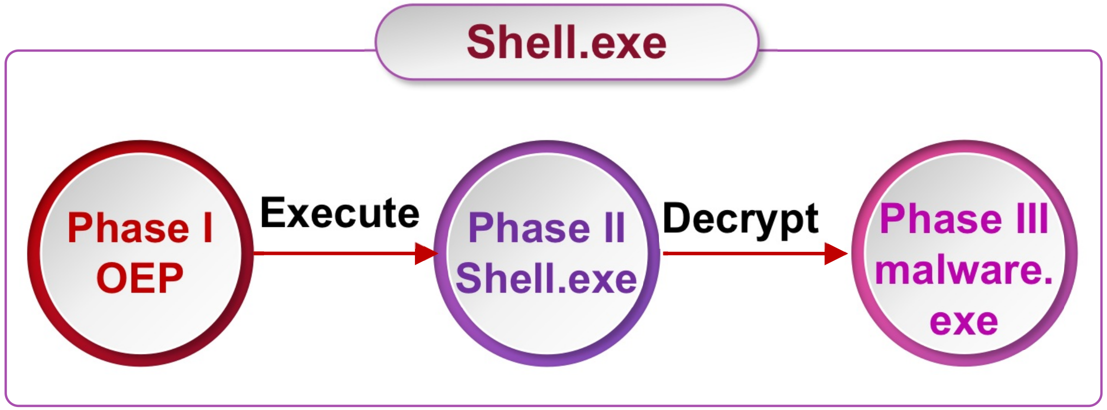
i. Read malware.exe, obtain the address and size of its code section, and encrypt the code section;

ii. Develop Shell.dll with the functionality that can store the crucial information, decrypt the code section, and jump to the start address of program execution (OEP) of malware.exe to execute the code;

iii. Add a section with the length up to Shell.dll in malware.exe, and save Shell.dll in the newly added section of malware.exe.

MalFox Component: PE Editor (Stealmal)

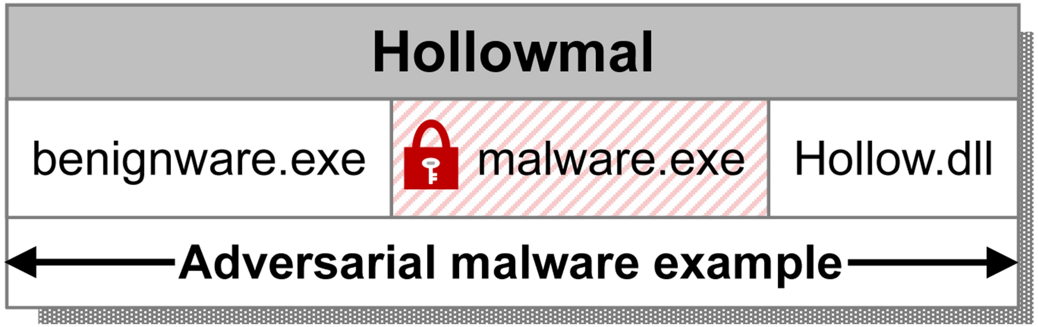


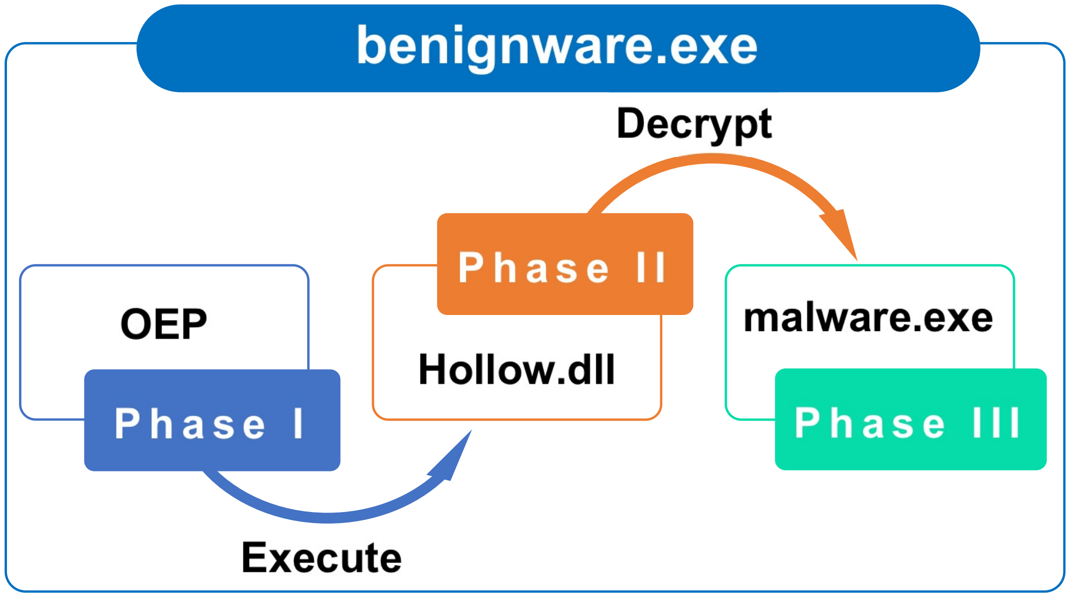


2 Develop a program Shell.exe with the functionality that can decrypt the malware, create a suspended process, obtain the process space, copy the malware into the space, change the context of the process to the entry point of the malware, and resume the process. And add a section in Shell.exe to save malware.exe.

1 Encrypt the entire malware.exe.

MalFox Component: PE Editor (Hollowmal)

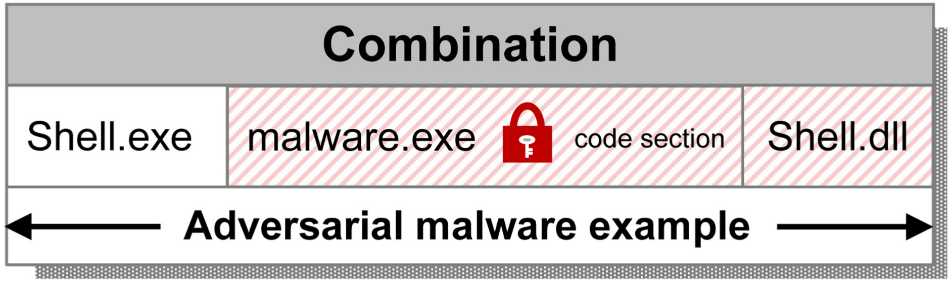


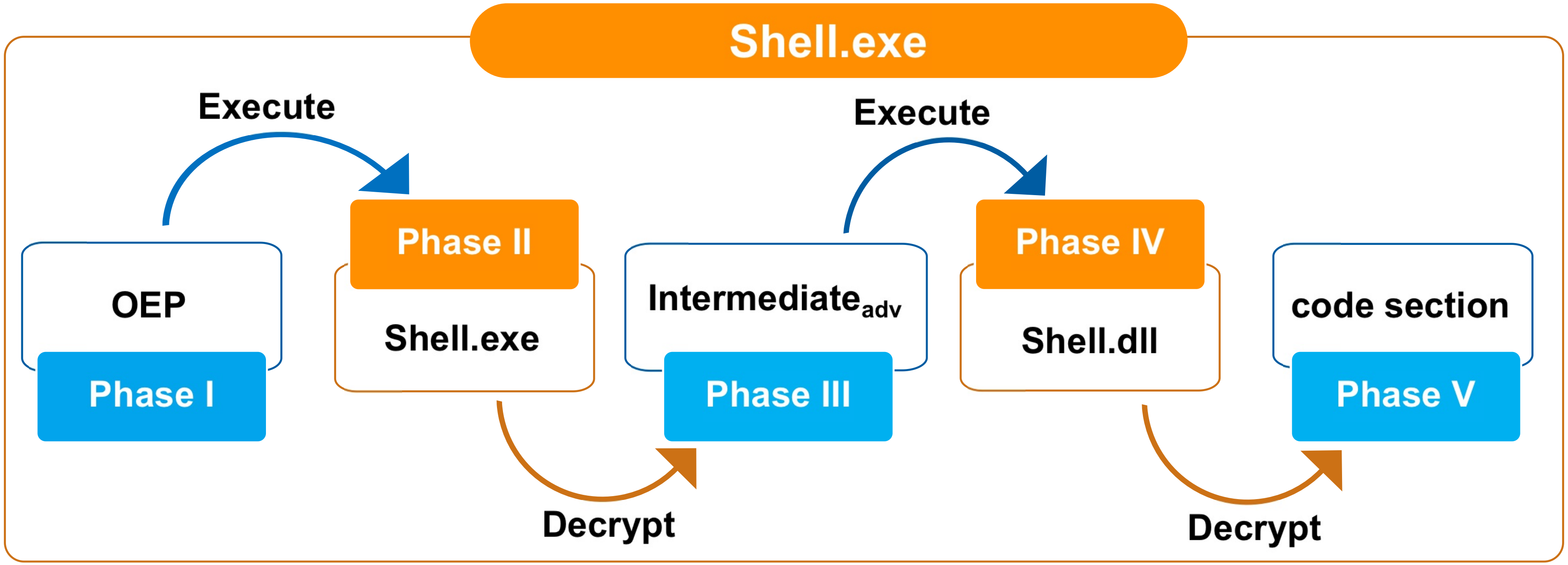


01 Select benignware, encrypt the entire malware.exe, and add a section in the benignware to save the encrypted malware.exe;

02 Develop a DLL named Hollow.dll embracing similar functionality as Shell.exe, and add another section in the benignware to save Hollow.dll following the encrypted malware.exe.

MalFox Component: PE Editor (Combination)

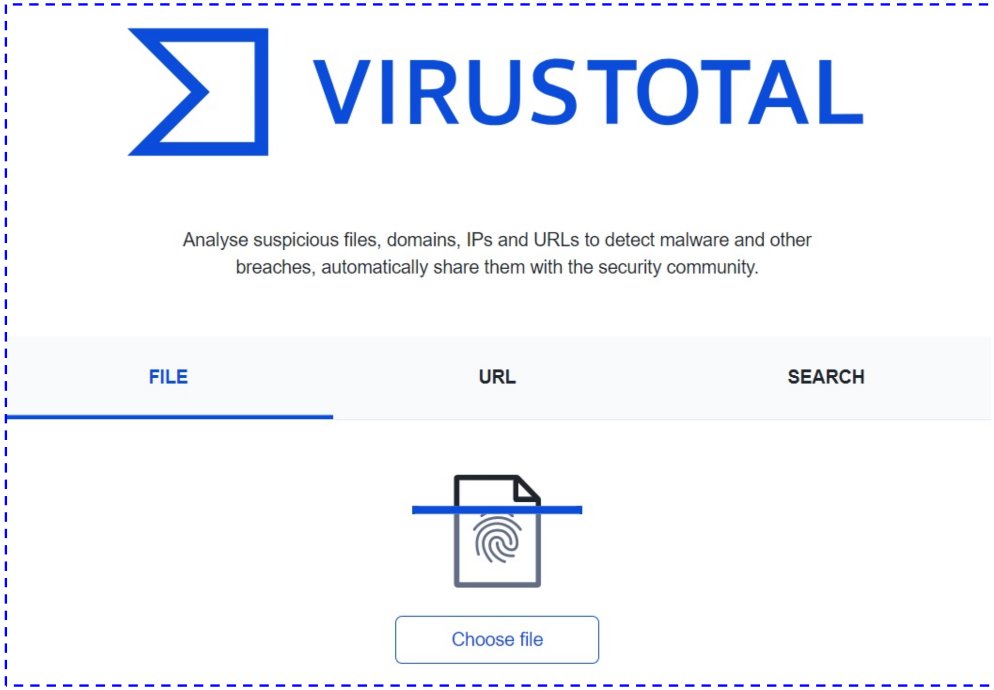




MalFox Component: Detector

VIRUSTOTAL

https://www.virustotal.com/gui/home/upload



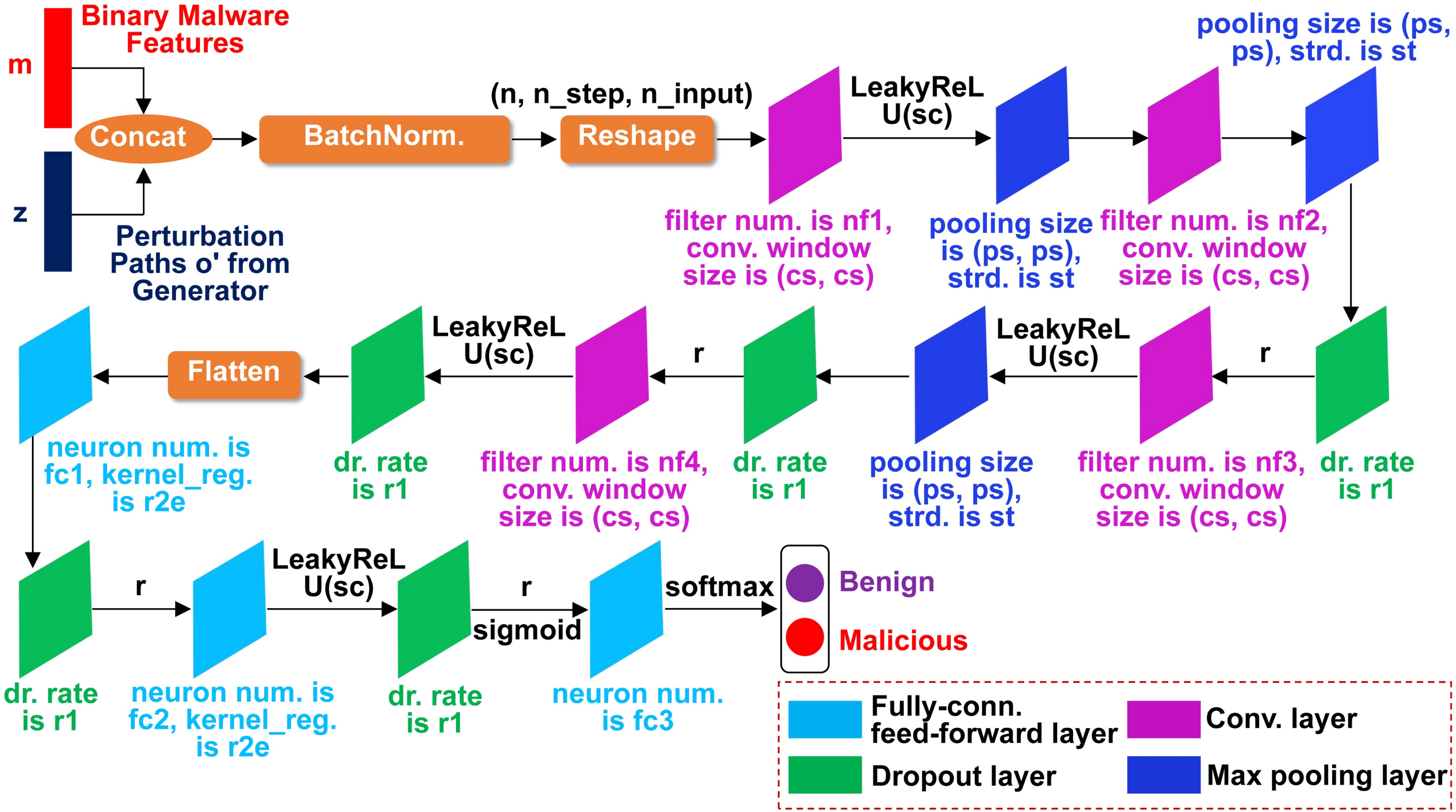
Virus Total

(1) Contains many antivirus products and online scan engines to check for malware.

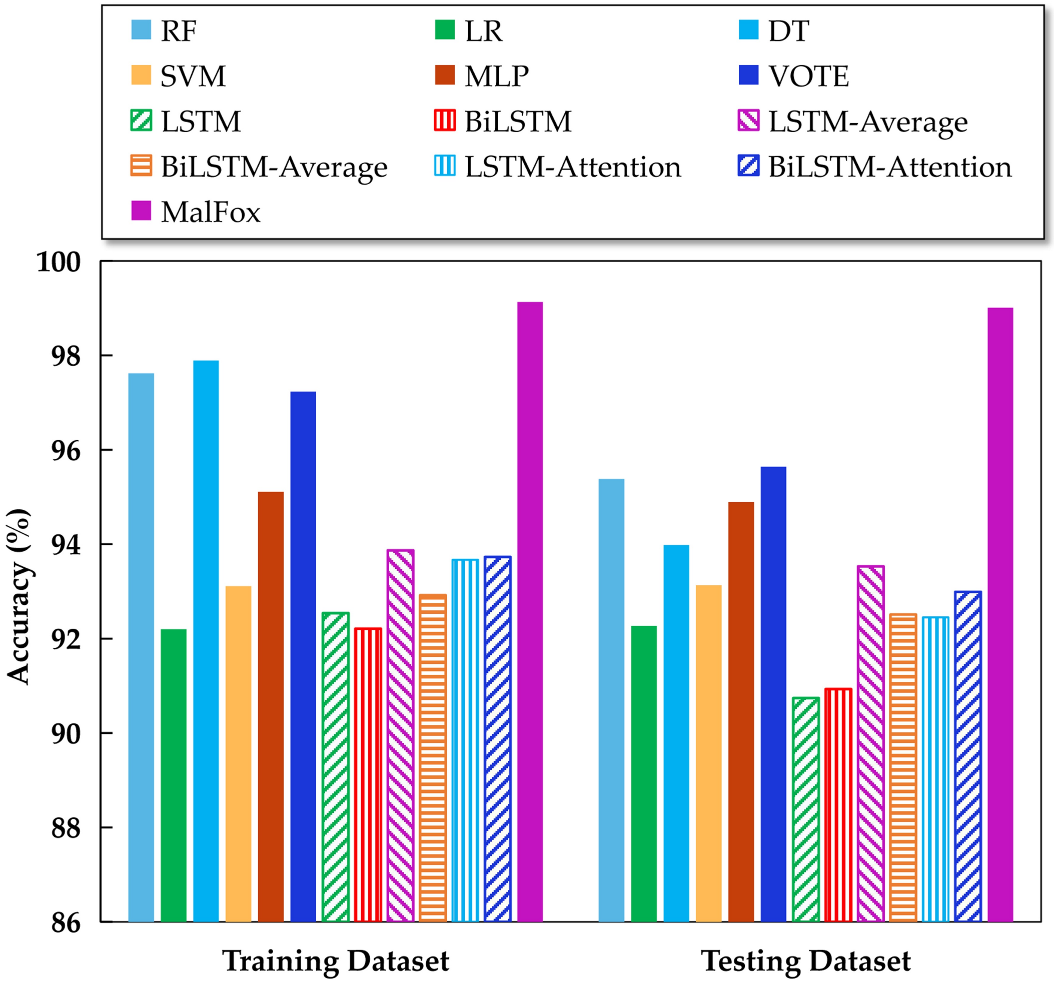
(2) Popular tools such as McAfee, F-Secure, Tencent, 360, and Microsoft in VirusTotal, have been widely adopted on laptop and mobile devices.

(3) It is well received by security professionalists and researchers.

MalFox Component: Discriminator



Experiment Results-Discriminator as The Attack Target



Types	Training Dataset	Test Dataset
RF	97.62	95.38
LR	92.2	92.27
DT	**97.89**	93.98
SVM	93.11	93.13
MLP	95.11	94.89
VOTE	[illegible]	**4**
LSTM	[illegible]	4
BiLSTM	[illegible]	3
LSTM-Avera	[illegible]	**3**
BiLSTM-Aver	[illegible]	1
LSTM-Attent	[illegible]	5
BiLSTM-Atter	[illegible]	9
MalFox	[illegible]	[illegible]

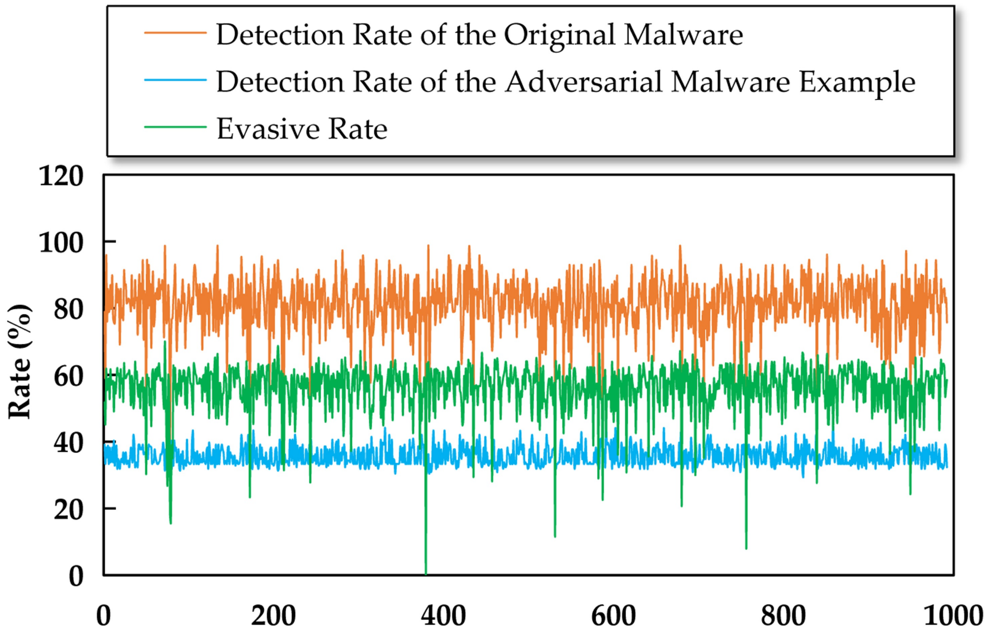
Accuracy is the ratio of incorrectly predicted adversarial malware examples (a), and A is all adversarial malware examples (A) by Discriminator.

$$accuracy = \frac{a}{A}$$

Experiment Results-VirusTotal as The Attack Target

Comparison Results (%)

Evaluation Metrics	Average	Max	Min
Dectection Rate (Malware)	68.8	85.4	26.8
Detection Rate (Foxy Malware)	29.7	43.9	18.3
Evasive Rate (Foxy Malware)	56.2	74.6	9.1



$$detection\quad rate = \frac{n}{N}$$

$$evasive\quad rate = \frac{N_{orig} - N_{adv}}{N_{orig}}$$

N_{orig} is the number of entities that detect the malware, and N_{adv} is the number of entities that detect the corresponding adversarial malware example.

entities (N) in VirusTotal.

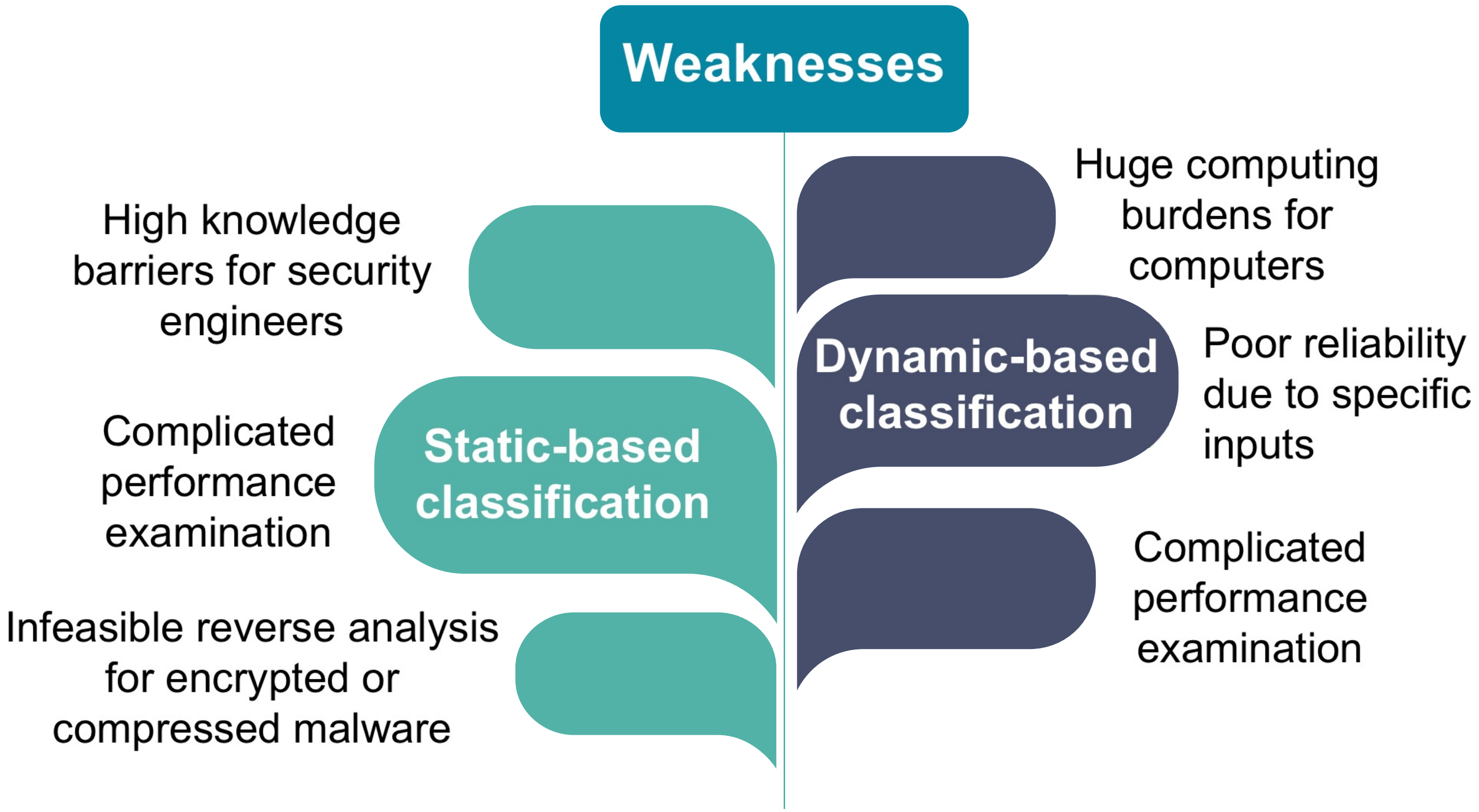
Section Three

03

Adversarial Malware Example Defenses

Weakness

After the generation of adversarial malware examples, existing malware detectors based on classification, either static-based or dynamic-based, should be improved.



Goal

Provide an efficient but simple classifier to distinguish different types of adversarial malware examples as well as other types of malware samples.

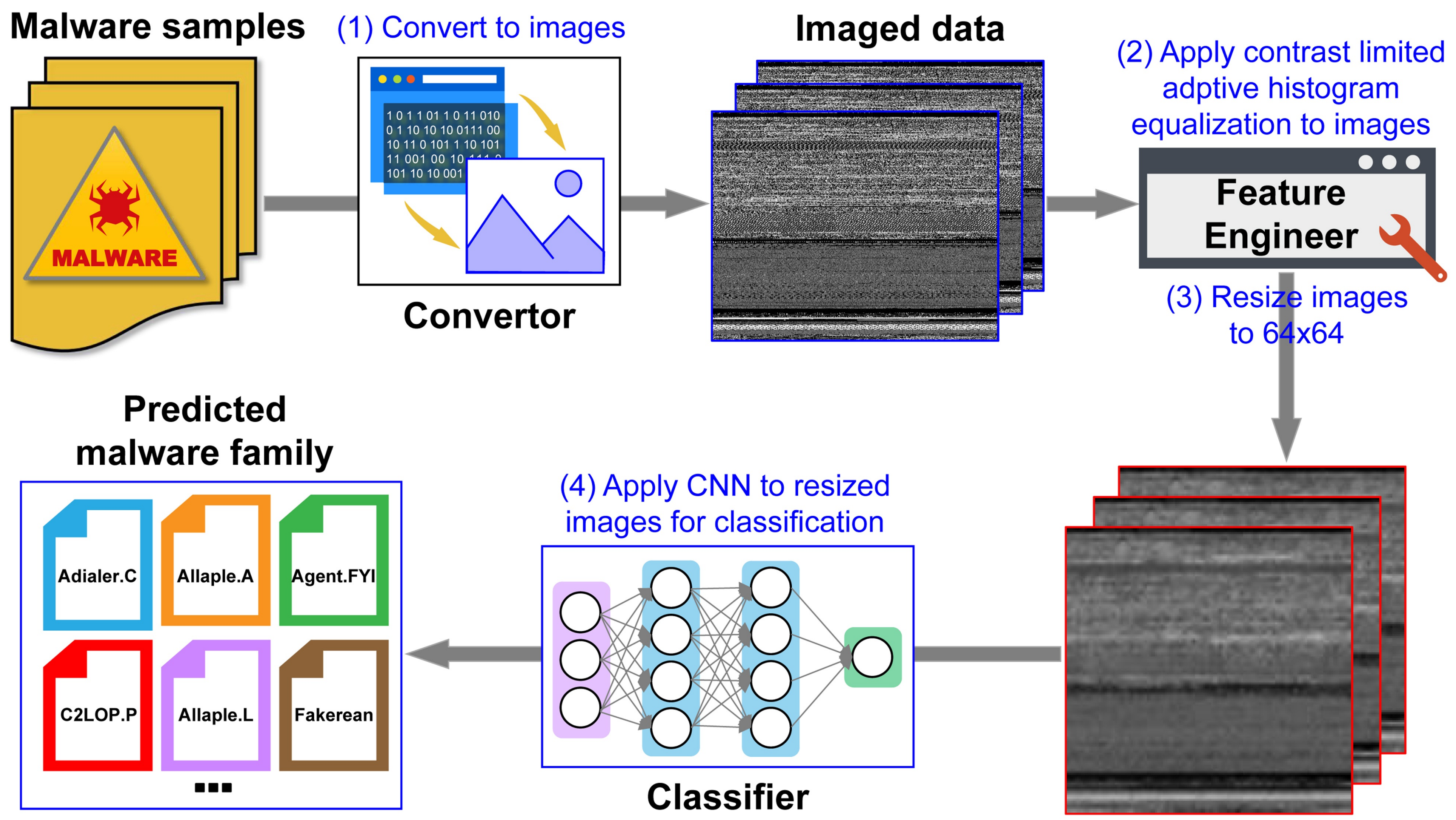
Challenges-The Efficient and Simple Defense

To provide an efficient, simple, and effective defense strategy

Challenges

1. **The processing time for classification being withstood by users.**
2. **Correctly distinguishing different malware families although encrypted or compressed.**
3. **Providing a simple method of validating defensive performance for users.**

The Overview of The Framework-VisMal



VisMal Component: Convertor

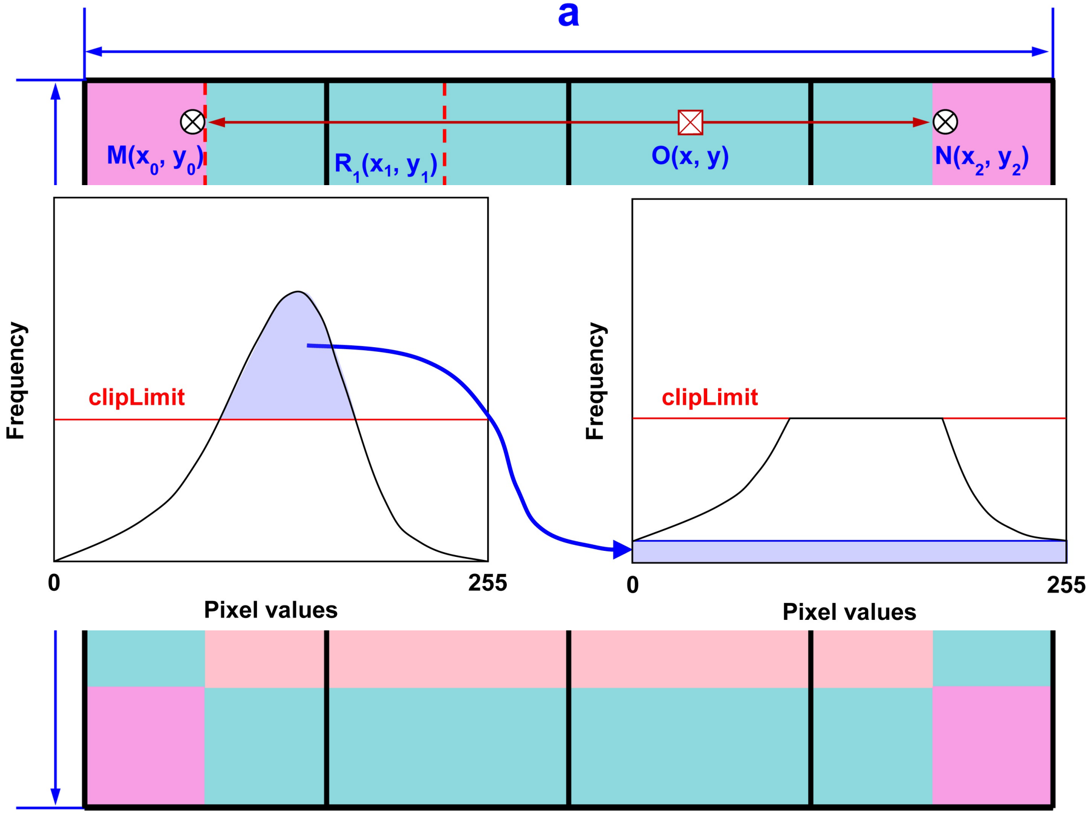
Correspondence between the malware sample file size and the converted imaged width

File Size	Width	Height
≤10KB	32	(0, 312]
10KB-30KB	64	(156, 468]
30KB-60KB	128	(234, 468]
60KB-100KB	256	(234, 390]
100KB-200KB	384	(260, 520]
200KB-500KB	512	(390, 976]
500KB-1000KB	768	(651, 1302]
≥1000KB	1024	(976,∞)

1. Sequentially reads the binary data in bytes and converts each byte into a number ranging in [0-255]

2. Reshape the image data by following a recommended fixed width with a variable height

VisMal Component: Feature Engineer



$$cdf(i) = \sum_{j=0}^{i} n_j, 0 \le i < L \quad \textbf{(1)}$$

$$y = h_R \quad \textbf{(2)}$$

(1) where L is the total number of gray levels (typically 256), and n_j is the total

$$y = \quad \textbf{(3)}$$

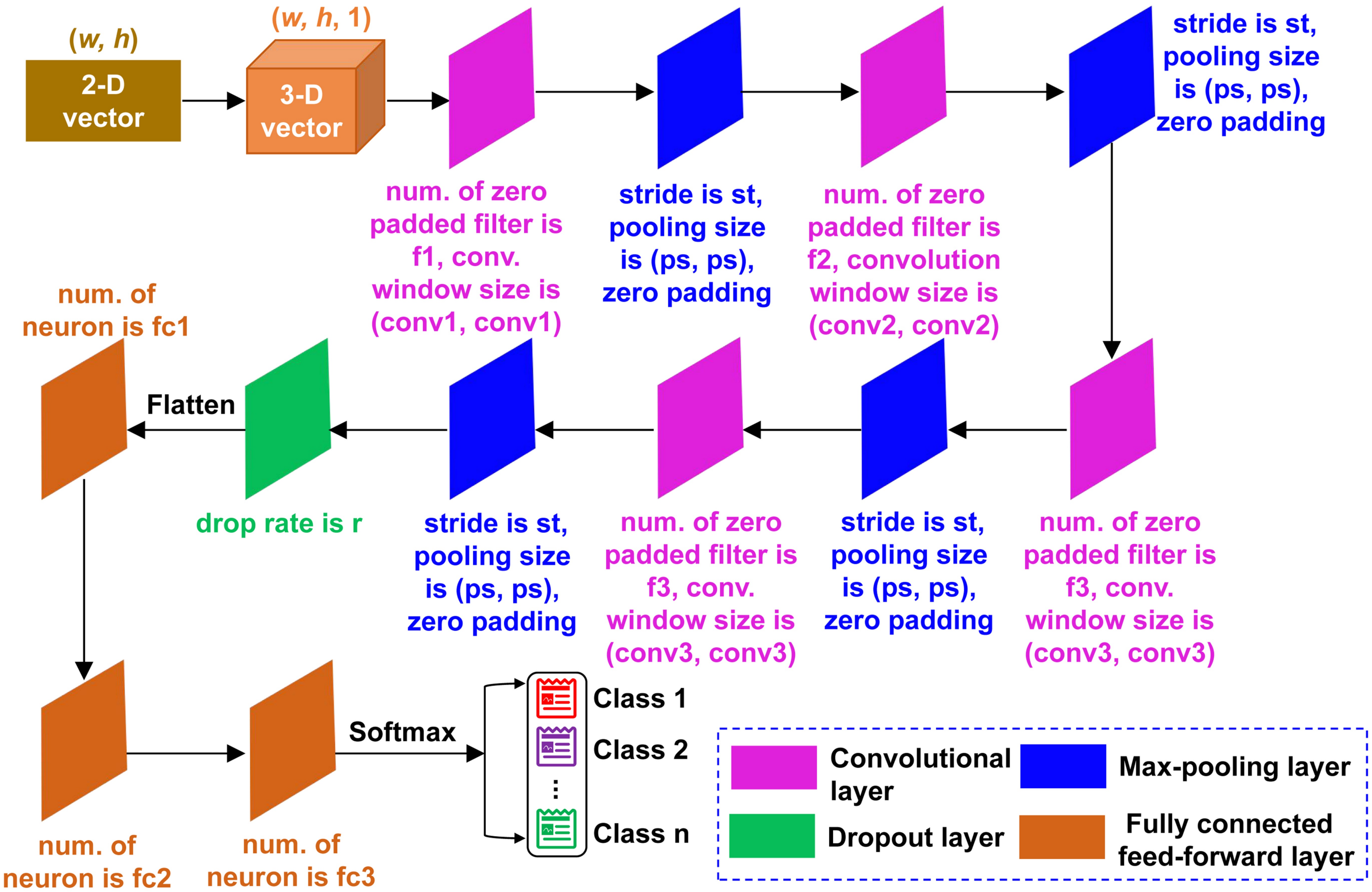
(2) cdf_{min} is the minimum non-zero value of the cumulative distribution function calculated in in Eq (1), while cdf_{max} gives the maximum value.

$$y_1 = \quad \textbf{(4)}$$

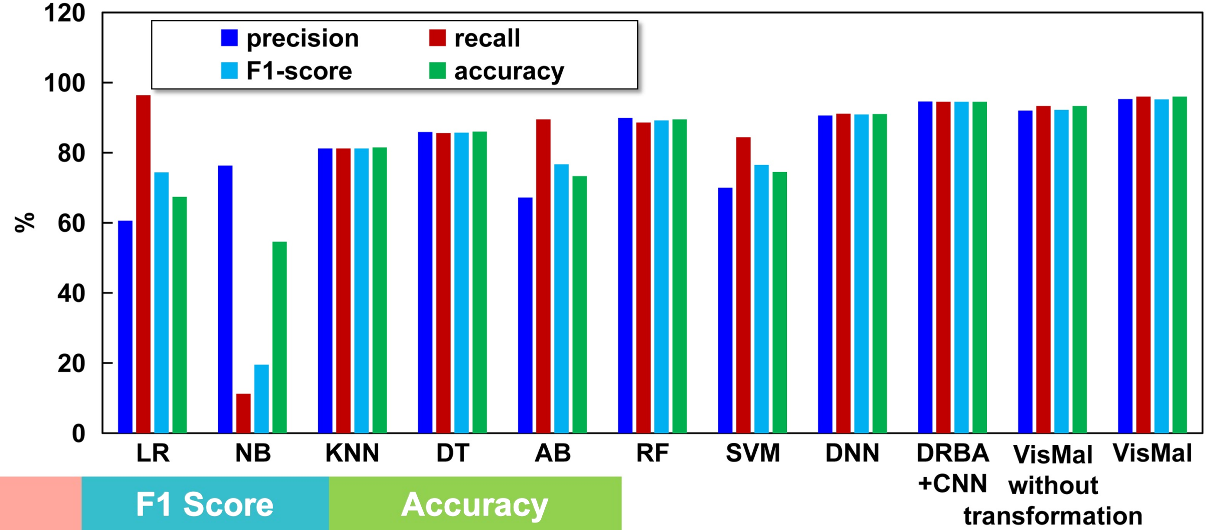
$$y_2 = \frac{x_{22} - x_2}{x_{22} - x_{12}} h_{R_{Q12}}(x_{12}) + \frac{x_2 - x_{12}}{x_{22} - x_{12}} h_{R_{Q22}}(x_{22}) \quad \textbf{(5)}$$

$$y' = \frac{y_2 - h_{R_P}(x')}{y_2 - y_1} y_1 + \frac{h_{R_P}(x') - y_1}{y_2 - y_1} y_2 \quad \textbf{(6)}$$

Classifier


(w, h)
2-D vector
(w, h, 1)
3-D vector
num. of zero padded filter is f1, conv. window size is (conv1, conv1)
stride is st, pooling size is (ps, ps), zero padding
num. of zero padded filter is f2, convolution window size is (conv2, conv2)
stride is st, pooling size is (ps, ps), zero padding
num. of neuron is fc1
Flatten
drop rate is r
stride is st, pooling size is (ps, ps), zero padding
num. of zero padded filter is f3, conv. window size is (conv3, conv3)
stride is st, pooling size is (ps, ps), zero padding
num. of zero padded filter is f3, conv. window size is (conv3, conv3)
Softmax
Class 1
Class 2
Class n
num. of neuron is fc2
num. of neuron is fc3
Convolutional layer
Max-pooling layer
Dropout layer
Fully connected feed-forward layer

Evaluation Results (Accuracy)



Classification Method	Precision	Recall	F1 Score	Accuracy
LR	60.6	96.4	74.4	67.4
NB	76.3	11.2	19.5	54.6
KNN	81.2	81.2	81.2	81.5
DT	85.9	85.6	85.7	86.0
AB	67.2	89.5	76.7	73.3
RF	89.9	88.6	89.2	89.5
SVM	70.0	84.4	76.5	74.5
DNN	90.6	91.1	90.9	91.0
DRBA+CNN	94.6	94.5	94.5	94.5
VisMal without transformation	92.0	93.3	92.2	93.3
VisMal	95.3	96.0	95.2	96.0

Evaluation Results (Efficiency)

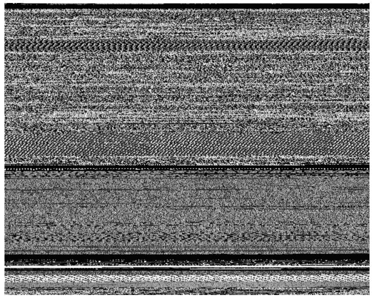
Comparison of The Classification Method

	Extraction Time	Classification Time	Total Time
Nataraj *et al.* [34]	32.7 ms	2.1 ms	34.8 s
Cui *et al.* [44]	-	-	20 ms
Naeem *et al.* [45]	-	4.27s	-
Yuan *et al.* [36]	144.3 ms	191.5 ms	335.8 ms
Vasan *et al.* [46]	-	-	1.18 s
Verma *et al.* [35]	37 ms	10 ms	47 ms
VisMal	**0.3 ms**	**3.7 ms**	**4.0 ms**

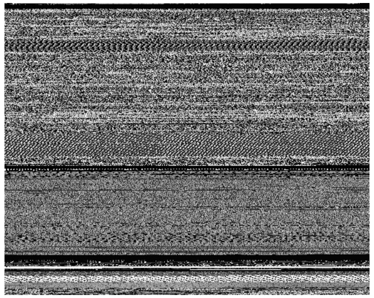
IEEE Transactions on Computers, pp. 1-14, 2022.

Visualization

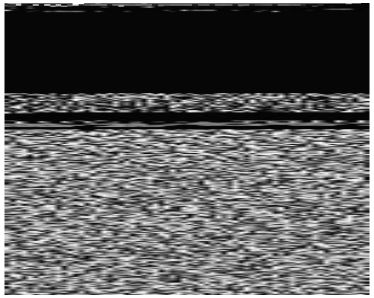
(a) Adialer.C sample1



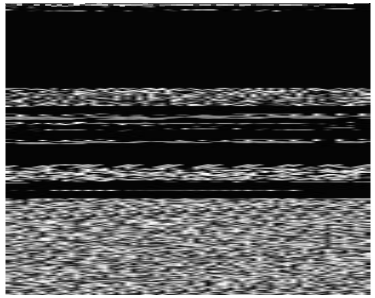
(b) Adialer.C sample2



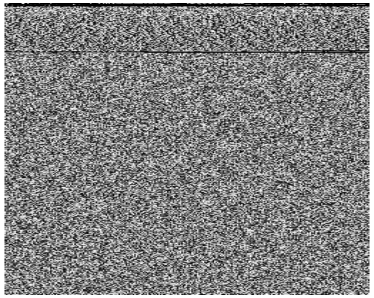
(c) Agent.FYI sample1



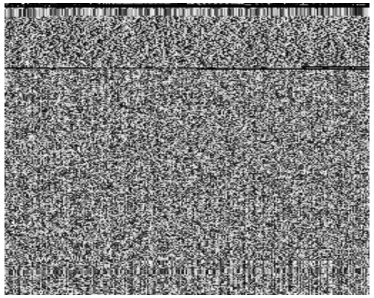
(d) Agent.FYI sample2



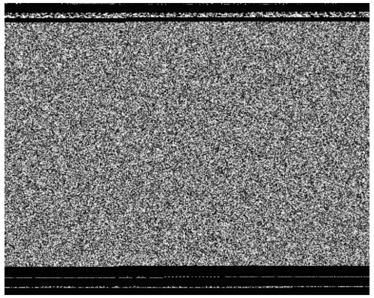
(e) Allaple.A sample1



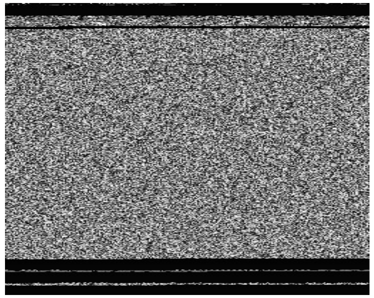
(f) Allaple.A sample2



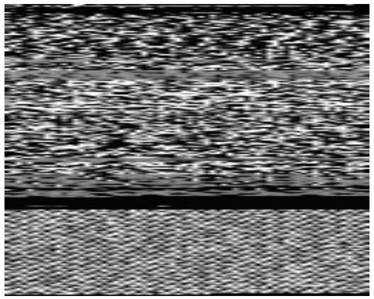
(g) Alueron.gen!J sample1



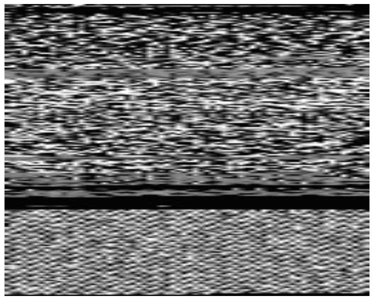
(h) Alueron.gen!J sample2



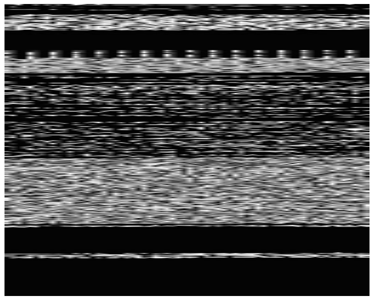
(i) Dialplatform.B sample1



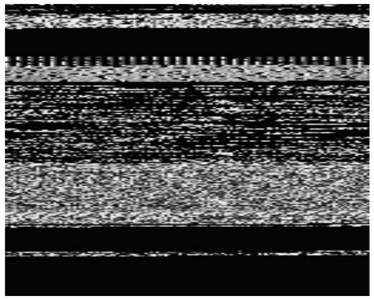
(j) Dialplatform.B sample2



(k) Dontovo.A sample1



(l) Dontovo.A sample2



MONTANA STATE UNIVERSITY

Thanks for your listening!

Fangtian Zhong

Gianforte School of Computing

Email: fangtian.zhong@montana.edu

2024.05.28