FINAL REPORT

Detection of Malicious Scripting Code through Discriminant and Adversary-Aware API Analysis

SUBMITTED BY

ACHINTYA S. RAO	**16BCI0158**
D. VARUN REDDY	**16BCE0040**
AKHIL KOTHARI	**16BCE2232**
D ANURAG VARMA	**16BCE0996**

TO

PROF. VIJAYASHERLY V

CSE4020 : MACHINE LEARNING

VIT – VELLORE CAMPUS

FALL SEMESTER 2018 – 2019

Abstract

Scripting languages play a major role as they can be used to connect databases and also to make the web pages more dynamic and interactive. They also allow one to send and receive the multimedia content in a seamless fashion making it hassle free for the user. This can be easily achieved by using ActionScript and JavaScript. However, it is like a doorway and they open up many vulnerabilities. These vulnerabilities are visible to third party applications. The main motive here is to detect malware based on these scripting languages using machine learning that involves extracting information from API methods, attributes and classes. This method aims to exploit the similarities between the two scripting languages and this is also created by taking into considerations this attacks that are performed to deceive the machine learning classification algorithms used in this. Testing has been done on PDF and SWF data by adding JavaScript and ActionScript codes. So, it was seen that most of the malicious files can be detected with low false positive rates. This methodology is also strong against strong attacks.

Problem Statement

The main motive here is to detect malware based on these scripting languages using machine learning that involves extracting information from API methods, attributes and classes.

1.Introduction

ActionScript and JavaScript are the two most common scripting languages throughout the web which are used to provide the user with the ability to share contents. However, they are extremely flexible, so this enables any user to have a look at the vulnerabilities. This was figured out when there was an ambiguity in Adobe reader such that it was under attack frequently as it contained JavaScript in the PDF documents. In the recent times, 2015 was a particular year where, ActionScript based attacks were widely seen in the ActionScript Virtual machine. This is normally used to read flash files, thus it caused severe damage. Another instance occurred when an Italian security company was hacked, costing them more than 400GB of classified data. Famous web browsing based companies like Mozilla and Google tried to reduce the use of flash-based applications but still a large portion of the web still uses it to deploy multimedia contents. JavaScript is still considered an essential tool to deploy web content and to improve the readability of documents. Due to the above reasons, security reasons on the basis of this need to be addressed and taken care of. Scripting languages allows the user to edit the source code, so the antivirus finds it difficult to detect. To tackle these issues several methods, exist, one way is to perform static and dynamic analysis of the script code carried by PDF files. We use leveraging machine learning guarantees more flexibility. However, the major set back is that either the methods exist for either ActionScript or JavaScript. The main motive is to enable a user to detect ActionScript or JavaScript code embedded in PDF files. We then evaluate it on PDF and SWF data, showing that it can correctly detect a large fraction of malicious files while misclassifying only a small fraction of benign files. We also try to show that it is strong against crafted attacks.

2. Literature Survey

2.1 Hacking teams flash 0-day:

The Adobe Flash zero-day exploit that spyware developer Hacking Team created is accessible to customers who have worked with success against even the most advanced defenses found in Google's Chrome browser. It is accustomed to infect PC users' multiple times before it had been leaked. It permits attackers to sneakily install malware on targets' computers, and there is proof that Hacking Team customers used the Flash zero-day against live targets [1].

2.2 One-and-ahalf-class multiple classifier systems for secure learning against evasion attacks at test time:

In this work, we have a tendency to overcome these limitations by proposing a multiple classifier system capable of rising security against evasion attacks at check time by learning to call a lot of tightly enclosed legitimate samples in feature house, while not considerably compromising accuracy in the absence of attack. Since we combine a set of one-class and two-class classifiers to this end, we name our approach one-and-a-halfclass (1.5C) classification. Our proposal is general and it is often accustomed improve the safety of any classifier against evasion attacks at check time, as shown by the reported experiments on spam and malware detection [2].

2.3 Evasion attacks against machine learning at test time:

In this work, we have a tendency to gift a straightforward however effective gradient-based approach that may be exploited to consistently assess the safety of many, widely-used classification algorithms against evasion attacks. Following a recently projected framework for security analysis, we simulate attack scenarios that exhibit different risk levels for the classifier by increasing the attacker's knowledge of the system and her ability to manipulate attack samples. This gives the classifier designer a more robust image of the classifier performance below evasion attacks, and allows him to perform a more informed model selection (or parameter setting). We appraise our approach on the relevant security task of malware detection in PDF files, and show that such systems are often simply evaded [3].

2.4 Pattern Recognition and Machine Learning:

This new textbook reflects these recent developments whilst providing a comprehensive introduction to the fields of pattern recognition and machine learning. It is geared toward advanced undergraduates or freshman Ph.D. students and furthermore as researchers and practitioners. No previous data of pattern recognition or machine learning ideas is assumed. Familiarity with variable calculus and basic algebra is needed, and a few expertise within the use of chances would be useful although not essential because the book includes a self-contained introduction to basic probability theory [4].

2.5 Cisco. Annual security report, 2016:

The Cisco 2016 Annual Security Report—which presents analysis, insights, and views from Cisco Security Research—highlights the challenges that defenders face in sleuthing and interference attacks. United Nations agencies use an upscale and ever-changing arsenal of tools. The report also includes analysis from external specialists, like Level three Threat analysis Labs, to assist shed additional light-weight on current threat trends. We take a detailed cross-check information compiled by Cisco researchers to point out changes over time, give insights on what this information means that, and make a case for however security professionals ought to respond to threats [5].

2.6 Detection of malicious pdf-embedded JavaScript code through discriminant analysis of api references:

A novel, light-weight approach to the detection of malicious JavaScript code. Our method is based on the characterization of JavaScript code through its API references, i.e., functions, constants, objects, methods, keywords as well as attributes natively recognized by a JavaScript Application Programming Interface (API). We exploit machine learning techniques to select a subset of API references that characterize malicious code, and then use them to detect JavaScript malware [6].

2.7 Detection and analysis of drive-by-download attacks and malicious JavaScript code:

This paper presents a completely unique approach to the detection and analysis of malicious JavaScript code. Our approach combines anomaly detection with emulation to mechanically determine malicious JavaScript code and to support its analysis. We developed a system that uses variety of options and machine-learning techniques to determine the characteristics of traditional JavaScript code [7].

2.8 ECMAScript language specification:

ECMAScript is associate object-oriented programing language for playing computations and manipulating procedure objects inside a bunch setting. ECMAScript as outlined here isn't supposed to be computationally self-sufficient; so, there are not any provisions during this specification for input of external information or output of computed results. Instead, it's expected that the procedure setting of associate ECMAScript program can give not solely the objects and different facilities delineated during this specification however conjointly sure environment-specific objects, whose description and behavior area unit on the far side the scope of this specification except to point that they'll give sure properties which will be accessed and sure functions which will be called from an ECMAScript program [8].

2.9 Static detection of malicious JavaScript-bearing pdf documents:

In this contribution we have a tendency to gift a method for detection of JavaScript-bearing malicious PDF documents supported static analysis of extracted JavaScript code. Compared to previous work, principally supported dynamic analysis, our method incurs an order of magnitude lower run-time overhead and does not require special instrumentation. Due to its potency we have a tendency to were able to measure it on a particularly massive real-life dataset obtained from the VirusTotal malware transfer portal [9].

2.10 Malicious pdf detection using metadata and structural features:

In this paper, we have a tendency to gift a framework for strong detection of malicious documents through machine learning. Our approach relies on options extracted from document information and structure. Using real-world datasets, we demonstrate the the adequacy of these document properties for malware detection and the durability of these features across new malware variants. Our analysis

shows that the Random Forests classification methodology, associate ensemble classifier that arbitrarily selects options for every individual classification tree, yields the simplest detection rates,even on previously unseen malware [10].

2.11 Combining static and dynamic analysis for the detection of malicious documents:

In this paper we have a tendency to gift MDScan, a standalone malicious document scanner that mixes static document analysis and dynamic code execution to discover antecedently unknown PDF threats. Our analysis shows that MDScan will discover a broad vary of malicious PDF documents, even when they have been extensively obfuscated [11].

2.12 ActionScript 3 malware detection:

This paper discusses common techniques like heap spraying, JIT spraying, and type confusion exploitation in the context of Flash malware.
Where applicable, these techniques are compared to those used in malicious JavaScript.
Subsequently, FlashDetect is presented, an offline Flash file analyzer that uses both dynamic and static analysis, and that can detect malicious Flash files using ActionScript 3. FlashDetect classifies submitted files using a naive Bayesian classifier based on a set of predefined features. Our experiments show that FlashDetect has high classification accuracy, and that its efficacy is comparable with that of commercial anti-virus products [12].

2.13 Comprehensive analysis and detection of flash-based malware:

As a remedy, we present Gordon, a method for the comprehensive analysis and detection of Flash-based malware. By analyzing Flash animations at completely different levels throughout the interpreter's loading and execution method, our method is able to spot attacks against the Flash Player as well as malicious functionality embedded in ActionScript code.
To achieve this goal, Gordon combines a structural analysis of the container format with guided execution of the contained code, a novel analysis strategy that manipulates the management flow to maximize the coverage of indicative code regions [13].

3. Overview of the Proposed System

3.1. Introduction

The main motive of this project is to detect malware based on these scripting languages using machine learning that involves extracting in-formation from API methods, attributes and classes. This method aims to exploit the similarities between the two scripting languages and this is also created by taking into considerations this attacks that are performed to deceive the machine learning classification algorithms used in this. Testing has been done on PDF and SWF data by adding JavaScript and ActionScript codes. So, it was seen that most of the malicious files can be detected with low false positive rates.

Our proposed system for this project involves two main topics, which are Support vector machines and ResNet.

3.1.1. Support vector machines

Support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

A SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

3.1.2. ResNet

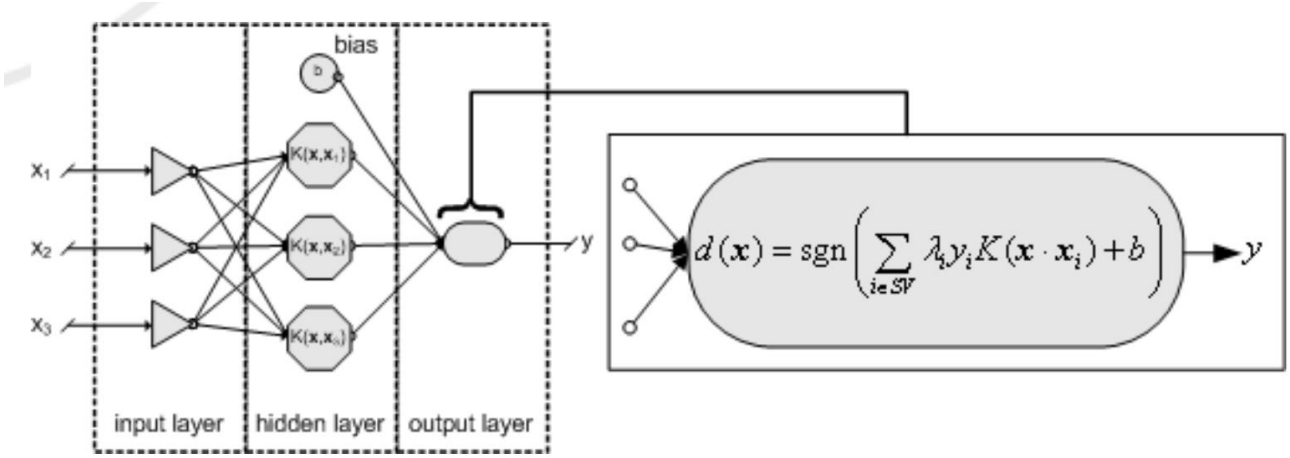
After the celebrated victory of AlexNet at the LSVRC2012 classification contest, deep Residual Network was arguably the most groundbreaking work in the computer vision/deep learning community in the last few years. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance.

Taking advantage of its powerful representational ability, the performance of many computer vision applications other than image classification have been boosted, such as object detection and face recognition.

Since ResNet blew people's mind in 2015, many in the research community have dived into the secrets of its success, many refinements have been made in the architecture. This article is divided into two parts, in the first part I am going to give a little bit of background knowledge for those who are unfamiliar with ResNet, in the second I will review some of the papers I read recently regarding different variants and interpretations of the ResNet architecture.

3.2. Framework, Architecture or Modules of the Proposed System

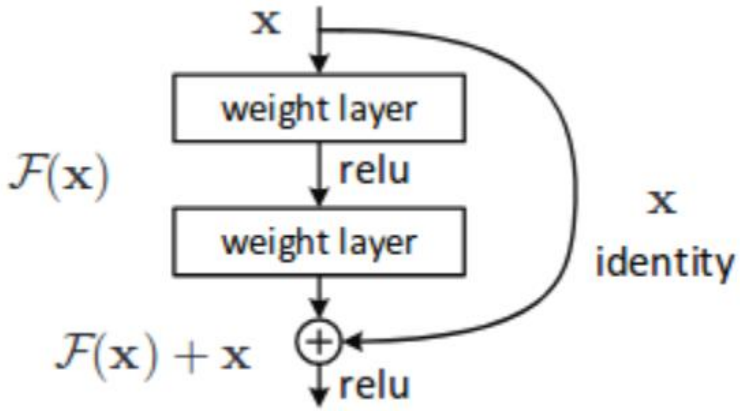
3.2.1 SVM Architecture



Architecture of SVM with linear or non-linear Kernel function

The main idea of support vector classification is to separate examples with a linear decision surface and maximize the margin between the different classes. This leads to the convex quadratic programming problem

3.2.2 ResNet Architecture



Residual block

There are two kinds of residual connections:

1. dentity shortcuts (x) can be directly used when the input and output are of the same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \qquad (1)$$

Residual block function when input and output dimensions are same

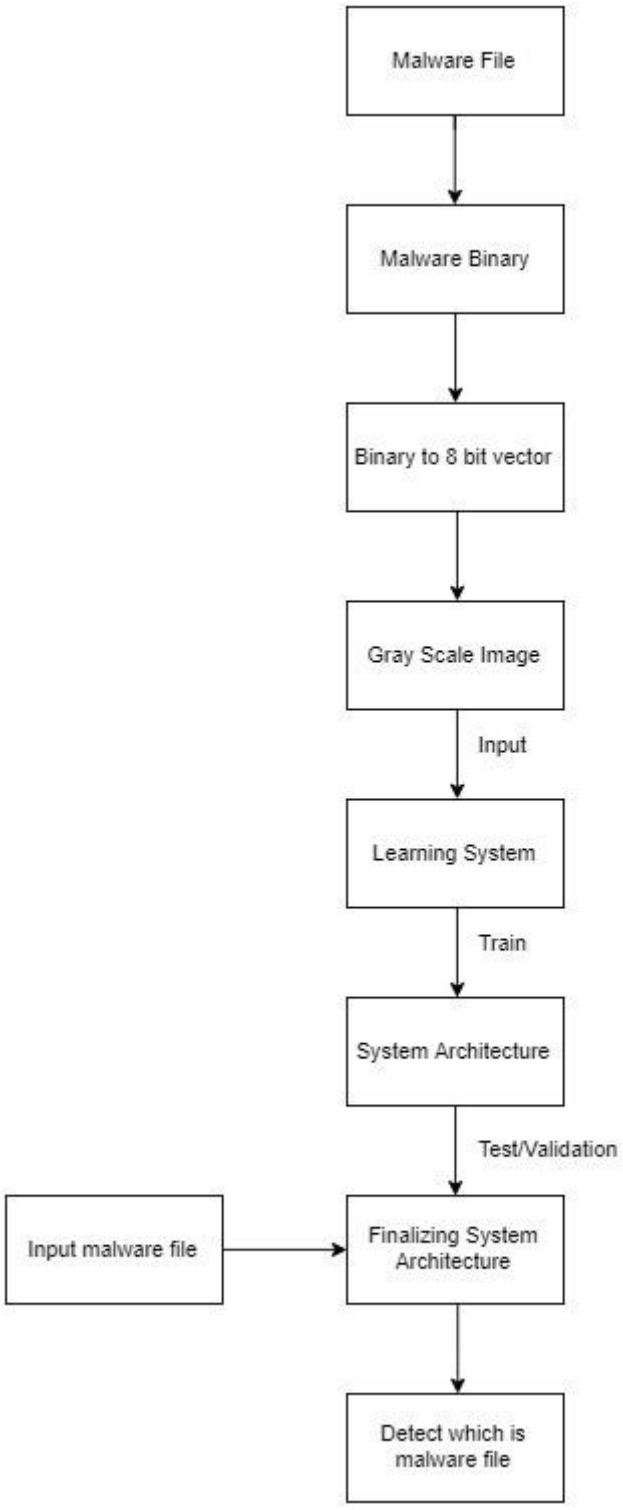
2. When the dimensions change, A) The shortcut still performs identity mapping, with extra zero entries padded with the increased dimension. B) The projection shortcut is used to match the dimension (done by 1*1 conv) using the following formula:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \qquad (2)$$

Residual block function when the input and output dimensions are not same.

The first case adds no extra parameters, the second one adds in the form of W_{s}

3.3. Proposed System Model (Mathematical Modeling /ER Diagram/UML Diagram)



UML Diagram for the proposed system

4. Proposed system analysis and design

In system analysis, the images obtained to us from the pre-processing stage are converted into vectors for the input feed to a learner system. Different machine-learning algorithms can be exploited for the classification task.

We divided our experimental and analysis protocol into two parts:

1. a standard evaluation, in which we assessed the performance of our method on two datasets respectively including PDF and SWF files.
2. an adversarial evaluation, in which we assessed the performance of our method against the mimicry attacks.

We use certain packages like os, pandas, numpy. The os module allows a user to interact with the operating system, it provides operating system dependent functionalities. Similarly, pandas allow one to easily deal with tabular and heterogeneous data. It helps in performing real word data analysis.

Numpy is another fundamental package which allows us to maintain N-dimensional data, it also has inbuild tolls to perform useful algebraic operations and for integrating C, C++ and Fortan code. Using these several packages we import functions that allow us to convert a malware to an image and henceforth import that image. Keras is an important package that provides image-based operations which include zero padding normalization, etc.

One major function in our code is to convert the image to an array. We than apply convolution and zero-padding along with the resNet function. Finally, at the end we perform the normal train, test and split.

5. Implementation Procedure

The API can be used for detection of both JavaScript and ActionScript based attacks. However, the primary aim is to design a system that can successfully tell whether a given Javascript or ActionScript file is malicious or not.
The stages of methodology for the objective are -

5.1 Pre-processing

The process of extraction of detected JavaScript and ActionScript files for the purpose of analysis is called Preprocessing. The operation itself may vary according to the analyzed file. File analysis of -

a. PDF Files: The scripting code is located by analyzing the internal structure of the PDF file.

b. SWF Files: We analyze these files by locating the equivalent ActionScript Bytecode contained in the file by searching for a data structure called DoABC Tag.

5.2 Feature Extraction

In this part of the process, we need to count the number of occurrences of each system API contained in the scripting file. For the scripts -

a. JavaScript: For JavaScript codes contained in PDF files, we count the occurrences of the methods and attributes belonging to the JavaScript for Acrobat API list.

b. ActionScript: For ActionScript scripts contained in SWF files, we count the occurrences of the classes belonging to the official ActionScript 3 API list. This is done by statically analyzing the ABC bytecode in order to detect all the employed API.

5.3 Analysis and Classification

In this stage the results of the classification of the above mentioned 2 stages is done. This will provide us with experimental results. Different machine-learning algorithms can be exploited for the classification task.

We divided our experimental and analysis protocol into two parts:

a) A standard evaluation, in which we assessed the performance of our method on two datasets respectively including PDF and SWF files
b) An adversarial evaluation, in which we assessed the performance of our method against the mimicry attacks

6. Methodology

JavaScript and ActionScript are both derived from ECMAScript, a standardized programming language maintained by Ecma with the ECMA-262 standard. They are object-oriented, interpreted scripting languages.

These scripting languages however are susceptible to attacks. Attacks are performed by invoking system-based or application-specific APIs. The underlying reason is that some of the APIs themselves are vulnerable to attacks. Similarly, system-based APIs can be used to manipulate memory, and they are often an essential element for performing attacks.

A general methodology that can be applied to both JavaScript and ActionScript to detect the corresponding attacks.

The 3 phases of the methodology are:

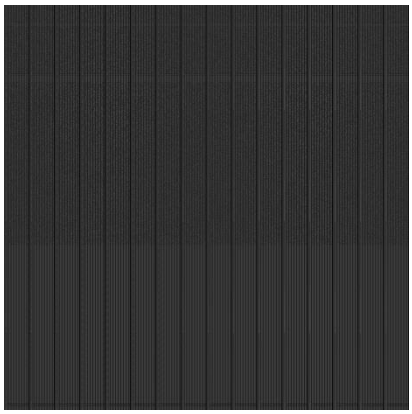
(i) Pre-processing, in this the input file is analyzed (statically or with dynamic instrumentation, depending on the application) to extract the embedded scripting code

(ii) Feature extraction, each sample is represented in terms of a feature vector, whose values correspond to the number of occurrences of each system API found inside the scripting code

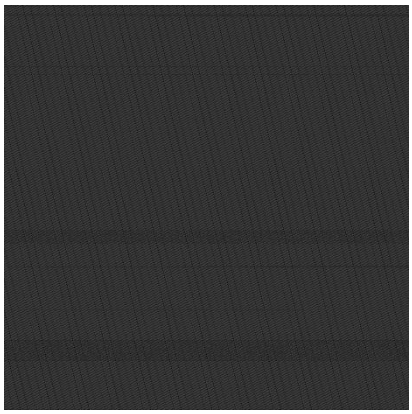
(iii) classification, in this the feature vector of the sample to be classified is provided as input to the machine-learning algorithm, which outputs a decision, i.e., classifies the input file either as benign or malicious.

7. Results and Discussions

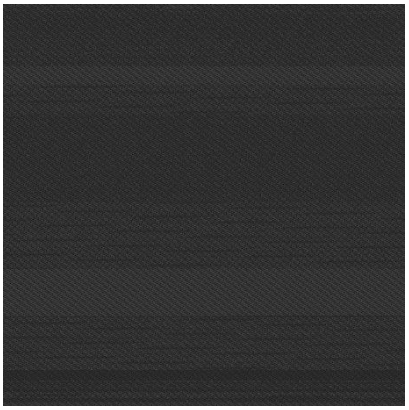
The malware binary files that were converted into jpg format:



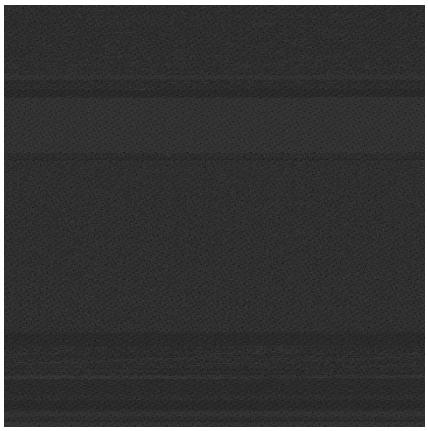
Vundo



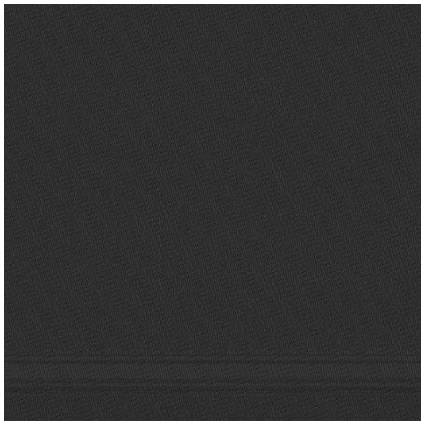
simba



lollipop



Obfuscator



Tracur

8. Conclusion, Limitations and scope for future work

As the work did in this project was converting the malware core into binary format and extracting the features from images of binary files. Using these features, we have trained our models to get an accuracy of 89% and 95% for SVM and ResNet models respectively. With normal classification like Support vector machines we get an accuracy of 74% with minimal data, so it can be said that residual networks are much more powerful. This model can be stated as one of the best models to detect malware. Limitations that are being faced that some malware classes produces almost similar type images which becomes for the leaner system to train on it. This can be solved by GIST of an image where the small features of an image are enhanced and boosted for differentiation.

In future work, we can directly look into the source code of the malware and train based on it instead of converting into images using natural language processing techniques and this would provide more accuracy of the detection of malware compared to image-based detection.

9. References

[1] ArsTechnica. Hacking teams flash 0-day: Potent enough to infect actual chrome user, 2015.

[2] B. Biggio, I. Corona, Z. He, P. P. K. Chan, G. Giacinto, D. S. Yeung, and F. Roli. One-and-ahalf-class multiple classifier systems for secure learning against evasion attacks at test time. In MCS, pages 168–180, 2015.

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. ˇSrndi´c, P. Laskov, G. Giacinto, and F. Roli.Evasion attacks against machine learning at test time. In ECML PKDD, pages 387–402, 2013.

[4] C. M. Bishop. Pattern Recognition and Machine Learning. Springer, 1 edition, 2007.

[5] Cisco. Annual security report, 2016.

[6] I. Corona, D. Maiorca, D. Ariu, and G. Giacinto. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In AISec, pages 47–57, 2014.

[7] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In WWW, pages 281–290, 2010.

[8] Ecma International. Ecmascript language specification (7th edition), 2016.

[9] P. Laskov and N. ˇSrndi´c. Static detection of malicious javascript-bearing pdf documents. In ACSAC, pages 373–382, 2011.

[10] C. Smutz and A. Stavrou. Malicious pdf detection using metadata and structural features. In ACSAC, pages 239–248, 2012.

[11] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. P. Markatos. Combining static and dynamic analysis for the detection of malicious documents. In EUROSEC, pages 4:1–4:6, 2011.

[12] T. Van Overveldt, C. Kruegel, and G. Vigna. Flashdetect: Actionscript 3 malware detection. In RAID, pages 274–293, 2012.

[13] C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck. Comprehensive analysis and detection of flash-based malware. In DIMVA, pages 101–121, 2016.

10. Sample code

```
# In[8]:
def ResNet50(input_shape=(64, 64, 3), classes=9):

 # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)

    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initial-
izer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

 # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2,
block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

# Stage 3 (≈4 lines)
    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3,
block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

# Stage 4 (≈6 lines)
    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4,
block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

 # Stage 5 (≈3 lines)
    X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5,
block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

# AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D((2, 2), name="avg_pool")(X)

# output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc' + str(classes), ker-
nel_initializer=glorot_uniform(seed=0))(X)


# Create model
    model = Model(inputs=X_input, outputs=X, name='ResNet50')

return model
```