Machine Learning meets iOS Malware: Identifying Malicious Applications on Apple Environment

Aniello Cimitile[1], Fabio Martinelli[2] and Francesco Mercaldo[2]
[1]*Department of Engineering, University of Sannio, Benevento, Italy*
[2]*Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy*
cimitile@unisannio.it, {fabio.martinelli, francesco.mercaldo}@iit.cnr.it

Keywords: malware, iOS, security, machine learning, testing, static analysis

Abstract: The huge diffusion of the so-called smartphone devices is boosting the malware writer community to write more and more aggressive software targeting the mobile platforms. While scientific community has largely studied malware on Android platform, few attention is paid to iOS applications, probably to their closed-source nature. In this paper, in order to fill this gap, we propose a method to identify malicious application on Apple environment. Our method relies on a feature vector extracted by static analysis. Experiments, performed with 20 different machine learning algorithms, demonstrate that malware iOS applications are discriminated by trusted ones with a precision equal to 0.971 and a recall equal to 1.

1 INTRODUCTION AND BACKGROUND

In January of 2007, Apple released the first version of its smartphone: the iPhone. This marked a revolution in the global smartphone market, thanks to the introduction of innovative features such as touch screen interfaces and virtual keyboards. At the time, the iPhone of Apple was the main growth driver for the smartphone market, pushing competitors to develop new products and operating systems to respond to new market demands[1].

This huge diffusion was also decreed by the number of available apps that has been consistently increasing over the years. In March of 2010, there were 150 thousand available apps in the App Store. The number of available apps reached 1.5 million by June 2015, 10 times more than the early 2010 number. The growth in number of apps available is directly related to the number of applications for the release of newly developed apps. In May 2015, the number of applications submitted for release to the App Store surpassed 54,000 for the first time[2].

Regarding availability, the most popular Apple App Store category is gaming with about 23 percent of available apps belonging to this category. Other leading app categories based on terms of availability are business apps, education apps, lifestyle apps and entertainment apps. Gaming leads in terms of downloads as well.

The number of apps downloaded from the Apple Store reached the 100 billion mark for the first time in June 2015. This is a significant increase from the previous year, considering the number of downloads stood at 75 billion a year prior to that. However, in the ephemeral world of apps, downloads do not equal retention. It is estimated that 25 percent apps downloaded by mobile app users worldwide were only used once during the first six months of ownership.

For these reasons the mobile ecosystem, with the increasing number of users that easily download applications from markets, is a very appealing scenario for malware writer. Typically attackers write codes more and more aggressive able to gather personal information from infected devices and to steal banking account.

This trend is confirmed by the fact that there have been hundreds of apps pulled from both Google Play and the Apple App Store for security reasons. McAfee security experts report that the biggest threat in 2016 for iOS came from apps with overly aggressive and invasive adware, whereas Google Play saw

[1]`https://www.statista.com/topics/870/iphone/`
[2]`https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/`

a fair number of apps infected with malware[3]. Both Google and Apple have been very quick to remove malicious apps from their associated app stores, however it is inevitable that some infected apps will still slip through the screening process.

From the defensive side, the signature-based malware detection, which is the most common technique adopted by free and commercial mobile antimalware, is often ineffective. Moreover it is costly, as the process for obtaining and classifying a malware signature is laborious and time-consuming.

While the scientific community has produced a lot of approaches to detect mobile malware on Android environment (Battista et al., 2016; Mercaldo et al., 2016a; Mercaldo et al., 2016b), literature lacks of method related to iOS environment. Our idea is that on Android, due to their open source nature, is easy for researchers retrieve samples to reverse engineering for analysis, while in iOS, that is closed-source, the source code extraction is more laborious and it needs to obtain the unencrypted binary code in order to analyze the application, as we explain in the next section.

To fill this gap in this paper we propose a method to identify malicious software in iOS environment.

We propose a technique for malware detection which uses a features vector, in place of the code signature. The assumption (that will be demonstrated with the evaluation) is that malicious applications show values for this features vector which are different from the values shown by trusted applications.

We consider as feature vector the occurrence of some opcode which form the disassembled code of the application. Our assumption is that malware applications in order the perform their harmful behavior tend to adopt operation codes differently from legitimate applications.

In current literature, several approaches address the malware detection issue using features extraction.

Counting op-codes is a technique used in previous works for the detection of virus: it revealed to be successful with several variants of the W32.Evol metamorphic virus (Choucane and Lakhotia, 2006).

Bilar (Bilar, 2007) proposes a detection mechanism for malicious code through statistical analysis of op-codes distributions. This work compares the statistical op-codes frequency between malware and trusted samples, concluding that malware opcode frequency distribution seems to deviate significantly from trusted applications. We accomplish a similar analysis, but for iOS malware. In reference (Rad and Masrom, 2010; Rad et al., 2012) the histograms of op-codes are used as a feature to find whether a file is a morphed version of another. Using a threshold of 0.067 authors in reference (Rad and Masrom, 2010) correctly classify different obfuscated versions of metamorphic viruses; while in reference (Rad et al., 2012) the authors obtain a 100% detection rate using a dataset of 40 malware instances of NGCVK family, 40 benign files and 20 samples classified by authors as other virus files.

Researchers in (Mercaldo et al., 2016c; Canfora et al., 2015c; Canfora et al., 2015b; Canfora et al., 2015a) demonstrated that opcode distribution is different between Android malware and trusted application with a precision ranging from 0.94 to 0.97, while authors in (Bernardeschi et al., 2004) studied illegal flow of information in Java bytecode.

Relating to Apple environment, iSAM (Damopoulos et al., 2011) is a prototype of malware developed for research purpose able to wireless infect and self-propagate to iPhone devices. The malware incorporate six malware mechanism and it is able to connect back to a bot master server to update its programming logic.

Researchers in (Garcıa and Rodrıguez, 2016) study the features of iOS malware and classify samples of 36 iOS malware families discovered between 2009 and 2015. Their findings evidence that most of them are distributed out of official markets, target jailbroken iOS devices, and very few exploit any vulnerability.

At the best of authors knowledge, this is the first work with the aim to address the mobile malware issue on Apple environment exploring machine learning techniques.

The paper poses following research question:

- RQ: are the features extracted able to distinguish a malware from a trusted application for iOS platform?

The rest of the paper is organized as follows: the next section illustrates the proposed features and the detection technique; the third section discusses the evaluation; the fourth section explains the performance of our approach and, finally, conclusion and future works are given in the last section.

2 THE APPROACH

We classify malware using a set of features which count the occurrences of a specific group of op-codes extracted from the application under analysis (AUA in the remaining of the paper).

We produce the histograms of a set of opcodes occurring in the AUA: each histogram dimension repre-

[3]`http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf`

Table 1: Most recurrent opcode with relative description

Op-code	Description
MSR	Move to system coprocessor register from ARM register.
MUL	Performs an unsigned multiplication
ADC	Adds two registers
TEQ	instruction to test if two values are equal
LDM	Load and Multiple registers.
ORR	Exclusive OR, and OR operation
SBC	Subtracts two registers
AND	AND operation
MVN	performs a bitwise logical NOT operation on the value.
STC	Sets the carry flag to 1.
STM	Store and Multiple registers.
TST	The instruction performs a bitwise AND on two operands
BX	The instruction causes a branch and exchanges the instruction set
CMN	The instructions compare the value in a register
SUB	The instruction is used for performing subtraction of binary data in byte
CMP	The instruction is used to perform comparison
STR	The single data transfer instructions is used to store single bytes/words of data
MLA	Multiply-Accumulate with signed or unsigned 32-bit operands
LDR	The single data transfer instructions is used to load single bytes/words of data
EOR	It performs the logical EOR between the contents of two registers
B	Branch instruction set
MOV	The instruction is a mnemonic for the copying of data

sents the number of times the opcode corresponding to that dimension appears in the code.

The main problem in retrieving source code by iOS samples, differently to Android environment, is represented by the fact that usually iOS samples are encrypted, for this reason we need to unencrypt them to obtain the disassembled code. For this reason, in the first step (i.e., the *processing*) of our method we need to have the unencrypted binary code to analyze the malicious payload. The iOS application available in the Apple Store can be downloaded as an IPA file, which is ciphered by Apple and thus, we need to decipher it. Otherwise, when the malicious sample is a dylib, package.deb, or an application distributed through Apple Enterprise Provisioning[4], the binary file is not ciphered and thus we can pass over this stage. The otool tool (with option -l) is used to verify whether the sample is encrypted, indicated by the cryptid value of the LC_ENCRYPTION_INFO command (a zero value indicates unencrypted) (Garcıa and Rodrıguez, 2016). When the sample is encrypted, the dumpdecrypted tool is used to obtain the unencrypted code.

Once we obtained the disassembled code we compute the occurrence for each opcode in order to select the most occurring features: this is the *feature selection* step. We compute the occurrency using the NgramTokenizer class provided by Lucene library. It is worth observing that the histogram dissimilarity has been already applied with success in malware detection in (Rad and Masrom, 2010; Rad et al., 2012).

Table 1 shows the most occurrency opcode in malware applications.

The output of the feature selection step is represented by a series of histograms, a histogram for each AUA; each histogram has 22 dimensions, where each dimension corresponds to one among the 22 opcodes included in the model divided the total number of opcode in the application.

For the sake of clarity, we compute each histogram according with following formula:

$$\#F_x = \frac{\sum_{i=1}^{N} X_i}{\sum_{i=1}^{N} O_i}$$

where F be one of the 22 features extracted, let X be one of the occurrence of the 22 opcode (i.e., O) from the i-th function and N is the total number of the functions forming the AUA.

For a single application we compute the 22 histograms i.e., the feature vector for each application is composed by the value of the histograms.

3 THE EVALUATION

We designed an experiment in order to evaluate the effectiveness of the proposed technique, expressed through the research question RQ, stated in the introduction.

The evaluation dataset includes 50 iOS trusted applications and 50 real-world iOS malware applications: the trusted samples were retrieved from App Store[5], while the malicious ones from Contagio Mobile[6]. The dataset includes different types of malware categorized by installation methods and activation mechanisms, as well as the nature of carried malicious payloads and they are appeared between July 2013 and February 2016, while trusted application were the free most downloaded in September, 2016. Table 2 shows the malware families we considered in the study with the upload date on the Contagio Mobile website.

Table 2: iOS malware families involved in the study with respective upload date on Contagio Mobile website.

Families	Description
TRracer - commercial spyware	July 13, 2013
iOS adware using Cydia	March 25, 2014
iOS AppBuyer malware - infostealer	September 15, 2014
Xsser (mRat) for IOS	October 8, 2014
IOS iphone Stealer.A	December 20, 2014
Inception APT iOS sample	December 20, 2014
Cloud Atlas / Inception iOS - WhatsAppUpdate.deb	December 20, 2014
iPhone / IOS clickfraud	June 5, 2015
KeyRaider: iOS infostealer	September 1, 2015
YiSpecter iOS	October 4, 2015
Wirelurker for iOS	November 5, 2014
ZergHelper	February 22, 2016

We submitted the malicious and trusted application to the 57 antimalware provided by VirusTotal[7],

[4]`https://developer.apple.com/`

[5]`http://www.apple.com/iphone/appstore`

[6]`http://contagiominidump.blogspot.it/`

[7]`https://www.virustotal.com/`

in order to test, respectively, the maliciousness and the trustworthiness of the dataset.

The classification analysis was aimed at assessing whether the features where able to correctly classify malware and trusted applications: we apply the classification algorithm shown in Table 3 to the feature vectors.

We evaluated the effectiveness of the classification method with the following procedure:

1. build a training set $T \subset D$;
2. build a testing set T' = D÷T;
3. run the training phase on *T*;
4. apply the learned classifier to each element of *T'*.

We performed a 10-fold cross validation: we repeated the four steps 10 times varying the composition of *T* (and hence of *T'*). The analysis was accomplished with the Weka tool[8], a well-known collection of machine learning algorithms for data mining tasks.

The results that we obtained with this procedure are shown in Table 4. Five metrics were used to evaluate the classification results: recall, precision, f-measure, RocArea and MCC (i.e., Matthews correlation coefficient).

The precision has been computed as the proportion of the examples that truly belong to class X among all those which were assigned to the class. It is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved:

$$Precision = \frac{tp}{tp+fp}$$

where *tp* indicates the number of true positives and *fp* indicates the number of false positives.

The recall has been computed as the proportion of examples that were assigned to class X, among all the examples that truly belong to the class, i.e. how much part of the class was captured. It is the ratio of the number of relevant records retrieved to the total number of relevant records:

$$Recall = \frac{tp}{tp+fn}$$

where *fn* is the number of false negatives. Precision and recall are inversely related.

The F-Measure is a measure of a test's accuracy. This score can be interpreted as a weighted average of the precision and recall:

$$F\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

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

The Roc Area is defined as the probability that a positive instance randomly chosen is classified above a negative randomly chosen.

MCC takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes:

$$MCC = \frac{tp*tn - fp*fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}}$$

where *tn* is the number of true negatives.

Table 4 shows the classification results.

We compute both the value of the metrics related to malware and trusted identification; relating to precision the more accurate algorithms to discriminate iOS malware samples is the *OneR* algorith, with a precision equal to 0,971 and a recall equal to 1.

RQ response: The evaluation shows that the features are effective to detect iOS mobile malware, obtaining the best detection capability with the *OneR* algorithm.

4 PERFORMANCE EVALUATION

In this section we discuss the performances of our approach. In order to measure performances, we used the System.currentTimeMillis() Java method that returns the current time in milliseconds. The machine used to run the experiments and to take measurements was an Intel Core i5 desktop with 4 gigabyte RAM, equipped with Linux Mint 15. We consider the overall time to analyse a sample as the sum of two different contributions: the average time required to extract the feature vector from an iOS application (t_{fv}) and the time required to test the extracted feature vector with the model learned by using the *OneR* algorithm (t_m).

Table 5 shows the performance of our method.

The most intensive task from the computational point of view is represented by t_{fv}, while t_m requires 0.0289 seconds to evaluate the feature vector: the proposed approach takes 2,1873 seconds to test a new sample.

5 CONCLUSION AND FUTURE WORK

While research community has largely studied Android malware, literature lacks of approach considering Apple environment. This is the reason why in

Table 3: Classification algorithms used in the evaluation

Alg	Name	Description
BN	BayesNet	Bayes Network learning using various search algorithms and quality measures.
NB	NaiveBayes	Naive Bayes classifier using estimator classes. Numeric estimator precision values are chosen based on training data analysis.
NBU	NaiveBayesUpdateable	This is the updateable version of NaiveBayes.
Log	Logistic	Class for building and using a multinomial logistic regression model with a ridge estimator.
MP	MultilayerPerceptron	A Classifier that uses back propagation to classify instances. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become unthresholded linear units).
SGD	Stochastic Gradient Descent	Implements SGD to learn various linear models (binary class SVM, binary class logistic regression, squared loss, Huber loss and, epsilon-insensitive loss linear regression). Globally replaces all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes, so the coefficients in the output are based on the normalized data.
SL	SimpleLogistic	Classifier for building linear logistic regression models. LogitBoost with simple regression functions as base learners is used for fitting the logistic models. The optimal number of LogitBoost iterations to perform is cross-validated.
SMO	SVM	Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier. This implementation globally replaces all missing values and transforms nominal attributes into binary ones.
IBk	K-nearest neighbours classifier	Can select appropriate value of K based on cross-validation. Can also do distance weighting.
KStar	K*	is an instance-based classifier, that is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. It uses an entropy-based distance function.
LWL	Locally weighted learning	Uses an instance-based algorithm to assign instance weights which are then used by a specified WeightedInstancesHandler.
AB	AdaBoost M1	Class for boosting a nominal class classifier using the Adaboost M1 method.
LB	LogitBoost	Class for performing additive logistic regression. This class performs classification using a regression scheme as the base learner.
JRip	Repeated Incremental Pruning	This class implements a propositional rule learner.
OneR	One Rule	Class for building and using a 1R classifier; it uses the minimum-error attribute for prediction, discretizing numeric attributes.
PART	partial decision tree	Class for generating a PART decision list. Uses separate-and-conquer. Builds a partial C4.5 decision tree in each iteration.
J48	C4.5	It is an algorithm used to generate a pruned or unpruned decision tree.
RF	RandomForest	Class for constructing a forest of random trees.
RnTree	RandomTree	Class for constructing a tree that considers K randomly chosen attributes at each node. Performs no pruning.
RepTree	Reduced Error Pruning Tree	Fast decision tree learner. Builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning. Missing values are dealt with by splitting the corresponding instances into pieces, as in C4.5.

this paper we propose a method to identify iOS malicious application through static analysis and machine learning. We obtain the best results with the *OneR* algorithm. As future works, we plan to extract code n-gram instead of occurrences to try to improve the detection of the method and to explore the usage of dynamic analysis, e.g. extracting system call sequences, to identify the iOS malware family.

Acknowledgements

This work has been partially supported by H2020 EU-funded projects NeCS and C3ISP and EIT-Digital Project HII.

REFERENCES

Battista, P., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. (2016). Identification of android malware families with model checking. In *International Conference on Information Systems Security and Privacy. SCITEPRESS*.

Bernardeschi, C., De Francesco, N., Lettieri, G., and Martini, L. (2004). Checking secure information flow in java bytecode by code transformation and standard bytecode verification. *Software - Practice and Experience*, 34(13):1225–1255.

Bilar, D. (2007). *Opcodes as predictor for malware*. International Journal of Electronic Security and Digital Forensics, Vol. 1, No. 2, pp. 156-168.

Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., and Visaggio, C. A. (2015a). Effectiveness of opcode ngrams for detection of multi family android malware. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, pages 333–340. IEEE.

Table 4: Precision, Recall, F-Measure, MCC and RocArea for classifying Malware and Trusted applications.

Algorithm	**Precision**		**Recall**		**F-Measure**		**MCC**		**RocArea**	
	M	T	M	T	M	T	M	T	M	T
BN	0,970	0,962	0,970	0.962	0,970	0.962	0,931	0,931	0.963	0,963
NB	0,931	0,800	0,818	0,923	0,871	0,857	0,736	0,736	0,855	0,875
NBU	0,931	0,800	0,818	0,923	0,871	0,857	0,736	0,736	0,885	0,875
Log	0,931	0,800	0,818	0,923	0,871	0,857	0,736	0,736	0,797	0,792
MP	0,935	0,857	0,879	0,923	0,906	0,889	0,797	0,797	0,832	0,832
SGD	0,929	0,774	0,788	0,923	0,852	0,842	0,707	0,707	0,855	0,855
SL	0,920	0,706	0,697	0,923	0,793	0,800	0,623	0,623	0,804	0,804
SMO	0,793	0,667	0,667	0,769	0,742	0,717	0,463	0,463	0,733	0,733
IBk	0,966	0,833	0,848	0,962	0,903	0,893	0,804	0,804	0,871	0,871
KStar	0,969	0,926	0,939	0,962	0,954	0,943	0,898	0,898	0,990	0,990
LWL	0,962	0,758	0,758	0,962	0,847	0,847	0,719	0,719	0,861	0,861
AB	0,935	0,857	0,879	0,923	0,906	0,889	0,797	0,797	0,956	0,956
LB	0,939	0,923	0,939	0,923	0,939	0,923	0,862	0,862	0,952	0,952
DT	0,943	1,000	1,000	0,923	0,971	0,960	0,933	0,933	0,950	0,950
JRip	0,933	0,828	0,848	0,923	0,889	0,873	0,766	0,766	0,898	0,898
OneR	0,971	1,000	1,000	0,962	0,985	0,980	0,966	0,966	0,981	0,981
PART	0,966	0,833	0,848	0,962	0,903	0,893	0,804	0,804	0,909	0,909
J48	0,960	0,735	0,727	0,962	0,828	0,833	0,692	0,692	0,839	0,839
RF	0,970	0,962	0,970	0,962	0,970	0,962	0,931	0,931	0,969	0,969
RnTree	0,941	0,960	0,970	0,923	0,955	0,941	0,897	0,987	0,946	0,946
RepTree	0,917	0,686	0,667	0,923	0,772	0,787	0,596	0,596	0,786	0,786

Table 5: The performance evaluation (values are expressed in seconds)

t_{fv}	t_m	total time
2.1584 s	0.0289 s	2,1873 s

Canfora, G., Mercaldo, F., and Visaggio, C. A. (2015b). Evaluating op-code frequency histograms in malware and third-party mobile applications. In *International Conference on E-Business and Telecommunications*, pages 201–222. Springer.

Canfora, G., Mercaldo, F., and Visaggio, C. A. (2015c). Mobile malware detection using op-code frequency histograms. In *SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015.*, pages 27–38.

Choucane, M. and Lakhotia, A. (2006). Using engine signature to detect metamorphic malware. In *WORM'06, 4th ACM workshop on Recurring malcode, pp.73-78.* ACM.

Damopoulos, D., Kambourakis, G., and Gritzalis, S. (2011). isam: an iphone stealth airborne malware. In *IFIP International Information Security Conference*, pages 17–28. Springer.

Garcıa, L. and Rodrıguez, R. J. (2016). A peek under the hood of ios malware. In *Availability, Reliability and Security (ARES), 2016 10th International Conference on.*

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016a). Download malware? No, thanks. How formal methods can block update attacks. In *Proceedings of the 4th FME Workshop on Formal Methods in Software Engineering*, pages 22–28. ACM.

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016b). Ransomware steals your phone. formal methods rescue it. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 212–221. Springer.

Mercaldo, F., Visaggio, C. A., Canfora, G., and Cimitile, A. (2016c). Mobile malware detection in the real world. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 744–746.

Rad, B., Masrom, M., and Ibrahim, S. (2012). Opcodes histogram for classifying metamorphic portable executables malware. In *ICEEE'12, International Conference on e-Learning and e-Technologies in Education, pp. 209-213.*

Rad, B. B. and Masrom, M. (2010). *Metamorphic Virus Variants Classification Using Opcode Frequency Histogram.* Latest Trends on Computers (Volume I).