secSVM attack model

Eynav Ben Shlomo 209328970, Elon Ezra 313534133

January 19, 2023

Abstract

The classifier of a machine-learning algorithm called SecSVM, which is intended to strengthen the security of another algorithm called Drebin by making it more resistant to evasion attacks, is the target of an attack presented in this article. In order to train the SecSVM algorithm, we use a dataset of 75,000 apps, both malicious and innocent, and extract different features, such as hardware components and network addresses, etc. we propose an adversary methodology for abusing the system security by replicating evasion attacks and assessing SecSVM's efficacy in fending them off. The methodology of the attack is to challenge the system by combining several feature attacks one after the other, which allows for the study of the classifier, the identification of its vulnerabilities and the testing of its boundaries. The attacks used in this experimental method include adding noise, deepfool and the CW attack. All these attacks aim to manipulate the features and test the resilience of the classifier. Our main insights are that performing the attacks together is better than separately combined with the separation between malicious and innocent apps. Because for every attack that failed to change the classification of some applications to innocence, we dress up another attack so that it succeeds.

1 Introduction

SecSVM is a classifier in which to improve the SVM classifier and be more resistant to attacks and correctly identify applications as malicious or innocent. SecSVM's creators suggest an adversary-aware strategy in which the machine-learning algorithm is built to be more resilient against evasion. SecSVM uses an adversary-aware methodology to increase system security. They use the efficient detection of Android malware using static analysis as a case study and concentrate on enhancing the security of the Drebin machine-learning approach. It aims to strengthen Drebin's defenses against sneakier attacks, i.e., expertly designed malware samples that avoid detection without displaying a lot of manipulation. They replicate evasion attacks and assess SecSVM's efficacy in fending them off using an adversarial framework built on earlier research on adversarial machine learning. Our motivation is to explore the classifier, find the existing holes in the classifier, test the limits of secSVM and perform attacks that will be able to bypass detection of secSVM, and check how much an attack lowered the protection of the classifier. The contribution of the article is to carry out an innovative attack on the system because in their article they do not talk about several feature attacks but about specific feature manipulations and we decided to challenge the system by combining several feature attacks one after the other.

2 Related Work

Many of the articles that dealt with strengthening and securing learning machines are classified applications or classified in general. The most classic app classification machine is Drebin. The authors in the article present the machine as a lightweight method for detecting malware in Android operating systems even while running the operating system from within the smartphone. Drebin performs static analysis on each application and collects as many features as possible on the application. In our article, as will be mentioned, we used Durbin's functionality for the purpose of extracting the necessary features for machine learning [1]. Another machine that is interesting to look at is MaMaDroid. Similar to Drebin, this machine also extracts features and reads API calls, but unlike Drebin, it advocates a behavioral approach as stated in the article, in which the machine follows calls made in the application and creates a graph describing the process of calls, and according to this, it draws conclusions by learning combinations of calls [2] . Another approach to classifying app malware is runtime diagnostics. As shown in A. Shabati's article where they present

a machine called Andromaly unlike the other machines this machine as we mentioned diagnoses while running, the machine runs the application extracts features from its behavior and then learns. The advantage of this method is the identification of malicious applications that are visible through static analysis safe but while running they perform malicious injections or any other attack that may not necessarily appear malicious at first glance[3]. The machine we attacked is SecSVM which is an improvement of Drebin and is based on SVM. Both of these machines work like Drebin they are classified by static analysis. They solve an optimization problem that depends on the values of the features and define an appropriate weight for each feature (as explained in detail in the Methodology section) [6] [4] [5]. In the training phase of the machine, we used H. Berger's article in that we used the 5-fold cross method for the purpose of learning the machine, in order for the attack tests to be as accurate as possible in relation to the attacks we launched. Something that helps to get a more accurate result and overcome overfitting. [6] And regarding the attacks, also in other articles we mentioned various attacks on the SecSVM machine were mentioned. We decided to focus on two known attacks Deepfool and C&W. The articles do not directly concern masking attacks on Android systems and these two attacks including the examples shown in previous articles on the subject are shown on images [7] [8]. We decided to interpret these attacks for cloaking malicious apps as if we treated each feature of an app as a pixel on an image.

3 Methodology

Our work will be a model of an attack on the classifier as our work was carried out in several main steps.

3.1 data exploration

When we performed data exploration on the classifier we came to some important data

3.1.1 The model formula of the classifier

The SVM model on which the more improved model we worked on works as follows. It works by calculating a loss function on a given set of features. f is a linear function that multiplies the set of features by the vector of weights for the features where b is some offset calculated by calculating the gradient decent. During learning, the machine receives sample labels on each respective application. And it calculates according to the function f the closest value to the appropriate class.

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b\,,$$

$$\min_{\boldsymbol{w},b} \mathcal{L}(\mathcal{D}, f) = \underbrace{\tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}}_{R(f)} + C \underbrace{\textstyle\sum_{i=1}^{\mathsf{n}} \max(0, 1 - y_i f(\boldsymbol{x}_i))}_{L(f,\mathcal{D})}\,,$$

We will note that the solution to the optimization problem of the improved SecSVM model is indeed the same as the simpler model

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C \textstyle\sum_{i=1}^{\mathsf{n}} \max\left(0, 1 - y_i f(\boldsymbol{x}_i)\right)\,, \\ \text{s.t.} \quad & w_k^{\text{lb}} \leq w_k \leq w_k^{\text{ub}}\,,\ k = 1, \ldots, \mathsf{d}\,. \end{aligned}$$

SVM. However, we will note that this time there is a condition on the values of w. There are now limit values for the weights of the features including the lower limit and upper limit. The author of the article on the development of SecSVM expands on the values of the algorithm in which he defines the limits of the weights. using the following expression:

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}} r(f_{\boldsymbol{\mu}}, \mathcal{D}) = A(f_{\boldsymbol{\mu}}, \mathcal{D}) + \lambda S(f_{\boldsymbol{\mu}}, \mathcal{D})\,,$$

3.1.2 features in the secSVM dataset

(S1) Hardware components, which are used to set the hardware permissions required by the software. (S2) Requested permissions, which are granted by the user at the time of installation and allow the application software to access the corresponding resources. (S3) App components, which include four different types of interfaces: activities, services, content providers and broadcast receivers. (S4) Filtered intents, which are used for process communication between different components and applications. (S5) Restricted API (Application Programming Interface) calls, access to a series of key API calls. (S6) Used permissions, a subset of permissions that are actually used and requested in S5. (S7) Suspicious API calls, API calls for allowing access to sensitive data and resources. (S8) Network addresses, the IP addresses accessed by the application, including the hostname and URL.

3.1.3 type of classifier

The type of classifier that analyze is static because Drebin performs a static analysis and this work on improving Drebin's security using a machine learning algorithm.

3.2 Extracting the features

The dataset of 75,000 apps, malicious and innocent. We started the feature extractor and took all the apps (75,000 apps, malicious and innocent) we divided the malicious and innocent apps into 2 separate folders and started the feature extractor which gave us 2 json files and a vector of a label (a label is a classification of the app if it is malicious or innocent) and in addition another file of a matrix of features. The extractor goes through all the applications interactively and opens the application, converts it to a smiley code and scans the static analysis code and according to the features it detects in the application, it adds the data it detected in the application to the table. The code is extracted: Python feature_ext.py {path_app_directory} {0/1} 0 - Benign 1 - Malicious Then you get 2 json files and put them in the dataset and the json file of the train (which contains both maliciousness and innocence) put in train. And the json file of the test (which contains both malicious and innocent) puts in the test 1219 applications, 1 innocent and 1218 malicious. The division is 90% innocence and 10% malice. After the extraction of features was finished, we moved to the next step.

3.3 Running the classifier

We activated the secSVM classifier, svm can also be activated in the classifier, but this is not relevant to the attack. The classifier receives a dataset extracted by the feature extractor and performs verification in the learning process 5-fold cross validation which takes the data matrix and divides them into 5 groups, 4 of the groups he performs training on and the fifth he performs a test, the above process repeats itself 5 times and in this way the machine tests itself that it does not have overfitting and that it is as accurate as possible. The metrics we calculate to evaluate a learning machine in particular secSVM: it's Accuracy, recall, precision, and F1 Accuracy, recall, precision, and F1 are all metrics used to evaluate the performance of a machine learning model, specifically for classification problems. Accuracy is a simple metric that measures the proportion of correct predictions made by the model. It is the ratio of correct predictions to total predictions. Precision is a metric that measures the proportion of true positive predictions (correctly predicted positive examples) out of all positive predictions made by the model. It is the ratio of true positives to true positives plus false positives. Recall, also called Sensitivity or TPR (True Positive Rate) is a metric that measures the proportion of positive examples that were correctly predicted by the model. It is the ratio of true positives to true positives plus false negatives. F1 score is a metric that combines precision and recall, it is the harmonic mean of the precision and recall. It gives a balance between precision and recall, it is a single value that represents both the precision and recall. It is useful when you want to seek a balance between precision and recall The next step is to read the json files of the test and train and put the data in the classifier's learning machine.

3.4 Search classified hacks

We looked for holes (hacks) in the classifier, we came to the conclusion that attacks of the type Obfuscate Or Encrypt Attacks, Mimicry Attacks, Limited Knowledge Attacks may break through the protection of the classifier and cause applications identified as malicious by the classifier to cause the classifier to recognize them as innocent. Explanation of these attacks: Obfuscation or Encryption Attacks: Durbin and sec-SVM uses static code analysis to detect malicious code before it is executed. A way to exploit this vulnerability is to obfuscate or encrypt the malicious code so that the feature extraction process will not detect it. Impersonation attacks: secSVM's parsing method is static and features that are truly innocent can be used by attackers. Malware samples can almost exactly replicate benign data, and this is possible due to an internal feature representation vulnerability. Limited knowledge attacks: Machine learning models used in the classification process are given a vector of features and trained to classify them based on their maliciousness. By manipulating such features such that they match feature vectors that pass the classifier test, an attacker can successfully pass the test. This can be done by adding or subtracting features to a malicious app that makes the feature vector appear benign, even though it may affect the app's functionality.

We decided to direct our research to machine resilience against feature attacks since there is no concrete mention of feature attacks in the SecSVM developers' article. In feature attacks, we take a set of features () from a dataset and obfuscate them by changing the feature values in such a way that moves the classification of the application as much as possible in the direction of maliciousness and away from classifying the application as innocent. This is done by chang-

ing the feature values (usually adding values) in such a way that an optimization problem will define the changed feature values to the malicious class.

3.5 Creating the attack algorithm

The feature attacks we will perform on the classifier are noise addition attack, deepFool attack, CW attack. The implementation of each attack was done by us and no library was used.

3.5.1 The attack combination algorithm

When the combination of the various attacks into one attack is carried out as follows, when first of all it separates the applications that are classified as innocent and the applications that are classified as malicious, then only the malicious attacks are transferred in the first attack and the classification of the applications is obtained after the attack, the applications are separated again according to their classification and the applications that are classified as malicious are transferred in the attack The second and the classification of the applications is obtained after the attack, the applications are separated again according to their classification and the applications classified as malicious are transferred in the third attack and the classification of the applications is obtained after the attack. Connect all the applications that were accepted as innocent in the previous steps with the applications that remained malicious after all the attacks and then transfer to get the accuracy prediction and get a new dataset file. That is, the attack is carried out in such a way that the test feature will go through a series of attacks one after the other in such a way that each attack will cover the attack that preceded it. There will be a list of apps to test so that the feature set will go through attack A_1 which will eventually give malicious and innocent apps. We will keep the innocent apps in a list aside and the malicious apps will go through attack A_2. This attack will also give a forecast of malicious and innocent apps. We will clean up the innocence and transfer the maliciousness to the next attack A_3. In the end we will be left with a list of applications with a maximum classification for innocence. The experiment will be conducted so that between each attack A_i the type of attack will alternate. In this way we will know what is the optimal order of attacks. The order selection of A_i is determined by scoring the forecast results of the model.

3.5.2 Noise addition attack

is an innovative attack that takes the weight of the features of the neural network and multiplies for each application the value of the feature F_i by the weight W_i in this way the attack adds random noise that is not directly related to the classification of the classes for each application.

3.5.3 deepFool attack

The DeepFool attack algorithm is a method for generating adversarial examples, which are inputs to a machine learning model that have been deliberately modified to cause the model to make a mistake. The goal of the DeepFool algorithm is to find the smallest possible perturbation to a given input image that will cause the model to misclassify it. The algorithm works by iteratively finding the direction in which the model's decision boundary is most sensitive, and then adding a small perturbation in that direction. The process is repeated until the model's output is changed to the desired incorrect class. Here is the general process of the algorithm: Initialize the input as the original image and the target class as the desired incorrect class. Compute the gradient of the model's output with respect to the input. Find the direction in which the gradient is the highest, this direction is called r_i. Move the input in the direction of r_i by a small step size, and evaluate the model's output. Repeat steps 2-4 until the model's output is the desired target class. This algorithm is efficient because it only requires the computation of gradients of the model's output with respect to the input, rather than computing the gradients with respect to all the parameters of the model. It is important to note that the DeepFool algorithm is a white-box attack, meaning it assumes knowledge of the model's architecture and parameters. Also, it is computationally expensive for large models and images.

3.5.4 CW attack

As is known for evasion attacks, formulas for optimization problems fall into the following template: $\min\{D(x,x+\delta)|\, C(x+\delta)=t, x+\delta\in[0,1]^n\}$ D is the distance function between the original feature vector x and the one with the noise $x+\delta$ We want the minimum distance between them, to of course make the attack as quiet as possible. And of course under the following conditions: first condition, $C(x+\delta)=t$ C

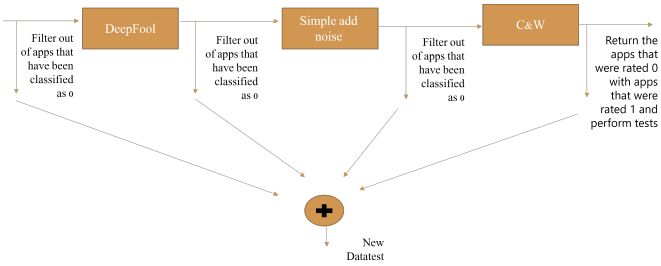
This is a function that brings me the class to which the application is associated with a given feature vector. In this case we want the class to be classified into the noisy vector to be the target class t for example malicious classification. second condition. $x + \delta \in [0,1]^n$ The dimension of the features vector will be preserved, and the new values of the features will be between 0 and 1. In the CW attack, the first condition changes, since the function C is not linear, which makes it difficult to solve the optimization problem. They use an objective function f that behaves in a way that reflects C. like for example $C(x + \delta) = t$ We can express with a function f like this, $f(x + \delta) \leq 0$ In his article, the inventor of the method, Carlini points to 7 objective functions, the best of which is the one we use in the attack $f(x') = \max\{\max\{Z(x')_i | i \neq t\} - Z(x')_t, -k\}$ x' is a sample of vector features. Function Z(x')¬I is a function that gives the probability of receiving class i. The function f returns a constant -k, or the difference between the probability of the class closest to the set of features and the probability of the set of features to the target class.

We will present the course of action of a CW attack. · We will define the cost function c.f that expresses the · We will define an x_test sample on which we want to run noise · As long as the prediction of x_test is not equal to the target class do: · Calculate the derivative (gradient) of the cost function c.f on x_test · Updated x_test by adding a small step in the gradient direction · Repeat until the model prediction gives t.

3.6 Performing the attacks on the classifier

This is done after writing the algorithm in the Python code. The run is carried out as follows: {conda activate myenv} - assuming that the environment on which modules are installed is in myenv (read by this name) {python attack.py}

3.7 Illustration describing the work of the system



4 Results

	classifier	addNoise	deepFool	CW	all
Accuracy	0.9	0.016	0.787	0.793	0.013
Recall	0.9	0.015	0.999	0.793	0.941
Precision	0.999	1.0	0.787	1.0	0.013
F1	0.947	0.0307	0.88	0.885	0.026

The classifier column is the metrics of the classifier before running attacks. The addNoise column is the results of running this attack only on the classifier. The deepFool column is the results of running this attack only on the classifier. The CW column is the results of running this attack only on the classifier. The all column is results of running the attacks according to the algorithm that was detailed before.

4.1 Before making the attack

Accuracy - accuracy of the identification of the classification - 0.8999 - TPR Recall - identifies how high the sensitivity of malicious detection is - 0.9006 FNR - Precision - the percentage of the miss, that is, the percentage of how many of the innocents were marked as innocent - 0.9991. F1 - harmonic mean performance on the recall (TPR) - 0.9473.

4.2 After performing the noise adding attack only on the classifier

Accuracy: 0.0164 Recall: 0.0156 Precision: 1.0 F1 score: 0.03072

It can be seen that the noise adding attack caused a drastic decrease in the percentage of accuracy, which indicates that all malicious applications have become innocent. This claim can be confirmed by the Precision score, which is 1

4.3 After performing the deepFool attack only on the classifier

Accuracy: 0.7867 Recall: 0.9989 Precision: 0.7873 F1 score: 0.8806

4.4 After performing the CW-only attack on the classifier

Accuracy: 0.7933 Recall: 0.7933 Precision: 1.0 F1 score: 0.8847

4.5 Carrying out the attack according to the algorithm

4.5.1 option 1

After performing the first separation of malicious and innocent apps, 1 innocent app and 1218 malicious apps are obtained. The noise adding attack is performed on 1218 malicious apps and we get: Accuracy: 0.0156 Recall: 1.0 Precision: 0.0156 F1 score: 0.0307 After the separation of malicious and innocent apps, 1199 innocent apps and 19 malicious apps are obtained. Add the innocent apps together and there are 1200 innocent apps and 19 malicious apps. DeepFool's attack is carried out on 19 malicious apps and we get: Accuracy: 0.8947 Recall: 1.0 Precision: 0.8947 F1 score: 0.9444 After the separation of malicious and innocent apps, 2 innocent apps and 17 malicious apps are obtained. Add the innocent apps together and there are 1202 innocent apps and 17 malicious apps. We check the total attack performed by the algorithm with the 2 attacks and it comes out: Accuracy: 0.0131 Recall: 0.9412 Precision: 0.0131 F1 score: 0.0259

The algorithm received 1 innocent app and 1218 malicious apps, according to this order of the attacks the classification of the apps will be 1202 innocent apps and 17 malicious apps.

4.5.2 Option 2

After performing the first separation of malicious and innocent apps, 1 innocent app and 1218 malicious apps are obtained. The deepFool attack is carried out on 1218 malicious apps and we get: Accuracy: 0.8046 Recall: 1.0 Precision: 0.8046 F1 score: 0.8917 After the separation of malicious and innocent apps, 238 innocent apps and 980 malicious apps are obtained. Add the innocent apps together and there are 239 innocent apps and 980 malicious apps. The noise adding attack is performed on 980 malicious apps and we get: Accuracy: 0.01837 Recall: 1.0 Precision: 0.01837 F1 score: 0.03607 After the separation of malicious and innocent apps, 962 innocent apps and 18 malicious apps are obtained. Add the innocent apps together and there are 1201 innocent apps and 18 malicious apps. We check the total attack performed by the algorithm with the 2 attacks and it comes out: Accuracy: 0.01395 Recall: 0.9444 Precision: 0.014 F1 score: 0.0275

The algorithm received 1 innocent app and 1218 malicious apps, according to this order of the attacks the classification of the apps will be 1201 innocent apps and 18 malicious apps. Therefore there is not much difference whether the noise adding attack is performed first or deepFool first.

4.5.3 Option 3

After performing the first separation of malicious and innocent apps, 1 innocent app and 1218 malicious apps are obtained. The deepFool attack is carried out on 1218 malicious apps and we get: Accuracy: 0.8046 Recall: 1.0 Precision: 0.8046 F1 score: 0.8917 After the separation of malicious and innocent apps, 238 innocent apps and 980 malicious apps are obtained. Add the innocent apps together and there are 239 innocent apps and 980 malicious apps. The noise adding attack is performed on 980 malicious apps and we get: Accuracy: 0.0184 Recall: 1.0 Precision: 0.0184 F1 score: 0.0361 After the separation of malicious and innocent apps, 962 innocent apps and 18 malicious apps are obtained. Add the innocent apps together and there are 1201 innocent apps and 18 malicious apps. The CW attack is carried out on 18 malicious applications and is obtained: Accuracy: 0.944 Recall: 1.0 Precision: 0.944 F1 score: 0.9714 Add the innocent apps together and there are 1202 innocent apps and 17 malicious apps. We check the total attack carried out by the algorithm with the 3 attacks and it comes out: Accuracy: 0.0131 Recall: 0.9412 Precision: 0.0131 F1 score: 0.0259

Therefore, when we carried out the combined attack according to the algorithm of each of the attacks and combining them together and attacking only the malicious applications, a better result is obtained than when carrying out one of the attacks or in a non-combined manner because the execution of the applications not according to the algorithm can change the classification of the innocent to malicious and also carrying out only one attack gets A worse result than performing several attacks together, each attack attacking in a different way.

5 Conclusion

In this work, we proposed the classifier SecSVM which improves upon traditional SVM by considering the weight of features in the optimization problem. We highlighted that the classifier was specifically designed to handle problem attacks and finding customized attacks. To test the robustness of SecSVM, we proposed a special attack that combines simple noise adding, C&W, and DeepFool methods.

The results of implementing this attack on a dataset showed that using this combination of attacks is a promising strategy to improve adversarial attacks on real-world systems. At the beginning we tested the dataset of separated attacks and than we pass the dataset on all the attacks one by one. Our experiments revealed that by passing the data that was still predicted as malicious through multiple attacks, we were able to improve the success rate of the attack.

As future work, we aim to improve the efficiency of the system by incorporating more advanced attack methods. By using strong attacks such as C&W we believe we can get better results. We also plan to evaluate the attack on a wider range of datasets to further assess the robustness of SecSVM against various types of attacks.

In summary, this work proposed the SecSVM classifier and a specialized attack that combines multiple methods to test its robustness. Our initial results suggest that the proposed classifier and attack have the potential to improve adversarial attacks on real-world systems and further research is needed to fully evaluate its performance.

References

[1] D. Arp, M. Spreitzenbarth, H. Gascon, K. Rieck," DREBIN: Effective and explainable detection of android malware in your pocket", In Proceedings of Symposium Network Distributed System Security, Internet Society, San Diego, USA, 2014.

[2] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building Markov chains of behavioral models (extended version)," 2017, arxiv:1711.07477.

[3] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for android devices," J. Intell. Inf. Syst., vol. 38, no. 1, pp. 161–190, 2012.

[4] B. Rashidi, C. Fung, and E. Bertino, "Android malicious application detection using support vector machine and active learning," in Proc. IEEE Int. Conf. Netw. Serv. Manag., 2017, pp. 1–9.

[5] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection,"IEEE Trans.Dependable Secure Comput., vol. 16, no. 4, pp. 711–724, Jul. 2019.

[6] Harel Berger, Chen Haja, Enrico Mariconti, Amit Dvir " Crystal Ball: From Innovative Attacks to Attack Effectiveness Classifier" IEEE Access, vol. 10, pp. 1317 - 1333, 2021.

[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in Proc. IEEE Symp. Secur. Privacy (SP), May 2017, pp. 39–57.

[8] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 2574–2582.