EvadeDroid: A Practical Evasion Attack on Machine Learning for Black-Box Android Malware Detection - Summary - SidharthAnil

The paper presents an efficient method of making android malware evasive to a Black-Box malware detection system. The claims about the contributions they make to the paper are that

- The strategy works in the problem space and not the feature space which would mean that there won't be an issue of converting the solution in the feature space back into problem space to get an adversarial malware sample
- The technique doesn't rely on information about the model and hence is a true black-box model attack strategy that can work on both hard and soft labels
- It can select the optimal perturbations in a query effective manner

The attack basically works in different stages. The first stage involves the Disassembly of all the APK files into a higher form of representation called "smali" files using tools like APKTool. Once the smali files are present, we create n-grams out of the opcodes (the instruction to be carried out) in the smali files. Once we have the n-gram feature vector for all the samples both malware and goodware, we use it to compare the similarity among them. For each malware sample, we try and find a goodware sample with the highest number of similar features in a stage called donor selection. This goodware file selected called the donor file would be used as the basis for making modifications to the malware file to make it more evasive.

Once we have the set of donor files selected for the malware files, we extract gadgets out of these goodware donor files. A gadget is a self-contained snippet of API call that forms the subsection of the entire program. And since it is self contained, containing all the objects and references included as part of the gadget, it can easily be transferred into different programs. To extract the gadget, tools like FlowDroid are used that map out a flow-graph for each API call, allowing you to extract out all the dependencies and references, say to other APIs. These gadgets can then be injected into your malware file without breaking its execution in order for it to evade the malware detection system (it might get fooled by the presence of goodware gadgets).

The question then remains as to where these gadgets should be injected without breaking the original functionality of the malware files. The gadgets should be injected where it does not affect or take part in the flow of execution at any point. This includes places like after the return statements in functions, or in a conditional block that would never be executed where the condition that the block is reliant on cannot be analyzed statically.

To experiment with the attack, the authors took a dataset of 10,000 Benign samples and 2,000 Malicious samples for training and 2,000 benign samples, and 1,000 malicious samples collected from

AndroZoo. They tested it against 4 classifiers - ADE_MA, DREBIN, MAMADROID and Sec-SVM. The results show that while EvadeDroid does not perform as well as an attack with Perfect Knowledge of the system would do, it was much better than random perturbations of the malware samples. It was also shown that while a perfect knowledge attack was able to achieve a 100% evasion rate by adding 5-30 features, EvadeDroid took 50-85 features. But this result is expected, as the novelty of EvadeDroid is in the fact that it is a zero-knowledge system. EvadeDroid also excels in performing against hard labels, with the results showing comparable results between hard and soft labels. Moreover, the samples crafted from EvadeDroid were tested against real world anti-virus systems in order to check their real world applicability, and the samples evaded 5 real world AVs with an Evasion Rate average of 82%. The transferability of the model was also tested with the best results being produced by EvadeDroid on Sec-SVM which transferred well into the other malware detection models too.

The presenter then went over his practical implementation of the EvadeDroid, the challenges he faced, and the technical details regarding each stage in the EvadeDroid process. Gadget Extraction was the most challenging aspect of the task, which had the issue of a reference explosion that had to be dealt with by including a reference depth. Wrapping tools like APKTool into Python was also difficult, as was finding APK samples for conducting experiments.

Discussion

- Gadgets are, explained in a sample manner, pieces of the application that can operate independently. This is why it is easy to transfer this from one application to another, and hence acts as blocks that can be added in easily for the purpose of EvadeDroid. The concept of gadgets are there in other areas of security like offensive security, where the payloads being delivered, say through a buffer offerflow attack, are independent self contained pieces of code and hence a gadget.
- While all the other techniques presented and discussed till now in the domain of PE file adversarial sample creation are applicable in the context of Android, the key point of difference that makes EvadeDroid possible is that it is feasible to decompile APK files which was not the case for PE files. This is because an APK file is at its core a zip file that can be unzipped to an extent. The Smali code, therefore, acts like the assembly code of JAVA.
- The professor also disagreed with the claim that it is hard to get APK files, as there are a lot of APK samples available on the internet. In fact, a counterpart APK file could be found on the internet for most of the applications available in the Play Store. Furthermore, malicious APK files are available easily through datasets like AndroZoo.
- There were also comments regarding the challenges the presenter faced in making tools like APKTool usable in the context he was working in. It was stated that making security technologies more accessible and user friendly is a direction that a lot of progress and research work could be focused towards.
- Even though the authors claim that it is an attack against true black box models, this is not entirely true as the attackers did have some knowledge about the models that they utilized in creating the

strategy for EvadeDroid. For example, they knew that the models give high priority to API calls which is the reason why the whole concept of EvadeDroid is based on gadgets that are just self-contained chain of API calls.

- Using gadgets from APK files that were identified as goodware is similar to the concept of adding benign strings to a malware file as opposed to random strings. There is a chance that the model might have learned certain patterns as benign and hence replicating those patterns would increase the chances of your evasive malware sample being detected as goodware.
- There was also a discussion regarding the different models that EvadeDroid was tested against. MAMADROID is based on markov models that work using probability where the model uses the information that is present to make probabilistic assumptions on the future. Markov models were used heavily in the domain of Natural Language Processing before the introduction of transformers and more recent advancements.
- Sec-SVM was also under focus as the results show that adversarial samples that work well with Sec-SVM have a good chance of working well with other malware detection systems too. The reason for this is that Sec-SVM is trained on adversarial samples and hence evading it is much harder compared to the other models. This also leads to the conclusion that a sample that is capable of evading Sec-SVM would therefore be an efficient adversarial samples and hence would work well for other models too.
- The feature space between Android files and PE files was compared. Even though the restrictions and the file structure are very different between Android and PE, the fact remains that the diversity possible within the feature space is completely dependent on the creativity of people.
- Another suggestion was made to use the concept discussed in the previous class, GAN to decide which permissions to focus on, in order to get a low detection rate.
- Another area of discussion was the difference between source code, assembly code, and byte code. It is a misconception that byte code is at a lower level than assembly code. Both are an intermediary level of code, where byte code works with the interpreter and assembly code works with a compiler. So in the case of Andorid, byte code is run by the Java Virtual Machine. However, byte code is easier to decompile than assembly code which is why a technique like EvadeDroid is possible in the landscape of Android. A point of interest was that while Java creates an explicit intermediate language, python follows the same process behind the hoods where you don't see the intermediate language.