Shallow Security: on the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors - Summary - SidharthAnil

Presentation

The presentation started off by giving context to the paper and its findings. A company called EndGame hosted a competition where the contestants were tasked with evading the 3 ML-based static malware detection models trained on Ember 2018 dataset that the company set up. Being a white box attack competition, the source code and all details were made available to the contestants through a GitHub repo. Moreover, 50 malware samples from different malware families were made available from which the contestants had to generate adversarial samples.

The 3 models forming the defense where as follows:

1. MalConv - A deep learning model that took in raw data, that is the binary file as a whole. The bytes were fed through an 8-dimensional embedding system, a gated 1D convolution layer, and a fully connected layer of 128 units with softmax output for each class.
2. Non-negative MalConv - MalConv model but with an added restriction that the weights must be non-negative. This meant that the suspicion score can only increase, that is, adding more benign elements wouldn't influence a model to believe a sample to be benign.
3. LightGBM - A gradient-boosting decision tree that takes in as input parsed information out of the PE file structure.

Different strategies were tried out to make adversarial samples and their effects were observed on the models:

1. Appending Random data to the end of a malware file - MalConv was heavily affected after about 1 MB of random data.
2. Appending goodware strings - MalConv was heavily affected after about 500 strings were attached. Non-Negative MalConv was also affected after about 10K of strings attached. This was unexpected as benign strings are not supposed to affect this model. It was inferred that the reason for this happening was because of an implementation error in the Non-negative MalConv.
3. Changing Binary Headers - Many header values of the PE file are not used by the Windows Operating System upon execution and hence can be easily modified without execution being affected. However, only 6 samples managed to bypass the models using this strategy.
4. Packing the sample with UPX - A packer like UPX compresses the executable and attaches a stub code (where execution begins) whose role is to uncompress the executable and hand over the execution to the extracted payload. However, experiments showed that the model has a large bias

towards popular packers like UPX, since packing samples with UPX were increasing the detection rate rather than decreasing it.

5. Packing with a distinct packer like TeLock and appending data - All samples were able to bypass all models using the strategy however it broke execution for some extreme RAT family malware files.
6. Embedding samples with a dropper Dr0p1t and appending data - Dropper is a type of packer that contains the executable inside, and the executable is written to the disk and a process created on it. All samples bypassed the models using this strategy but some generated adversarial samples were above 5 MB which was against the rules

Based on the experiments the authors created a custom dropper that embedded the payload as a resource file. Using this custom dropper along with data appending managed to bypass all the models while staying within the competition rules. These adversarial samples were submitted to VirusTotal too to check their performance on the real-world AV engines. It was observed that the detection rates went down for all the samples, showing that the strategy was affecting the machine-learning model part of the AV engine.

Discussion

- The first point of discussion was about the representation data takes and its effect on the model. The raw bytes data as well as the parsed data were different representations of the same data. However, as its representation changed, so did its robustness and along with it the amount of effort the attackers had to invest to generate adversarial samples. Another example of this is the graph data like the control flow graph, which is an even more robust representation.
- Another type of adversarial sample generally used are downloaders. Downloaders contain no malicious payload within themselves, but instead, download the malware from a specified URL. This makes it harder for the detection system as its maliciousness is now entirely on the URL. Moreover, the malware attackers have methods to obfuscate this URL, splitting it apart, and forming the URL dynamically. The detection model can try to extract the URL and check it against the reputation of the domain and such metrics in order to judge whether the URL is malicious or not. Another method is to observe on runtime the network accessed, the files downloaded, etc. This would however require more computational resources and hence cannot be run on devices like Android.
- One major observation was that the adversarial samples are bigger than the original samples. This would make it more suspicious as well as harder to transmit for an attacker. So the attacker normally would try to fool the detection systems with a minimal amount of added data. Moreover, it is the models that rely on raw data that get affected by such data appending. This is because, with enough data, it would be possible to affect all of the decision-making entities (for example a neuron) for the detection system to consider the sample benign.
- The design choices in the Windows operating system give a lot of leniency in how the PE file should be. This means that, even if there are additional data or if the headers are not exactly the way it should be, Windows would still run it. This is the reason why the attackers are able to make such modifications to fool the malware model without breaking the execution. The reason behind this choice of the operating system is to ensure backward compatibility. But considering the security

issues caused by it, it could be argued that it is time to break this backward compatibility, make the rules stricter, and hence limit the power the attackers would have.

- Building on the previous point, the decisions taken by the operating system influence the entire landscape of malware to a huge extent. Most of these decisions are trade-offs that might increase the security but might affect convenience or backward compatibility etc. For example, the windows operating system ignores the overall checksum but instead re-computes the section checksum. This is exploited by the attackers as a change in the file would not be easily detected. Another area that the attackers utilize is the size mentioned in the header files which is not strictly enforced by the operating system.
- Packers offer a creative solution for the attacker to have new headers without modifying them. Once a file is packed, the new file would have headers related to the packer, and all the headers related to the actual malware payload would be enclosed and compressed within the packer file. Another technique in this domain is the use of crypters that can encrypt the contents making static analysis very difficult.
- The example of how UPX was initially used for benign purposes, but then started to be used by malware creators which led to the detection systems flagging it as a potential indicator of maliciousness, is a good example of concept drift. It shows how different actions by the attackers and the responses by the defenders change the data distribution of malware, emphasizing the importance of the model taking drift into consideration.