No Need to Teach New Tricks to Old Malware: Winning an Evasion Challenge with XOR-based Adversarial Samples - Summary - Sidharth Anil

Presentation

The paper is, in a way, an extension of the paper "Shallow Security: on the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors." It presents the author's efforts in the next edition of the same competition. The new competition had a couple more models compared to the previous edition. Moreover, the contestants had to build their own defensive models as well as adversarial samples. Through this paper, the authors have focused on building adversarial samples that bypass ML models, building their own model with an emphasis on the strategy used while building a model, the performance of the samples in real AV engines as well as publishing the source code for their works.

The 3 major models under focus here are the Ember model provided by the organizers, the Need for Speed model that the authors built, and the Domumpdq model that was built by other contestants. The malware samples provided by the authors spanned more malware families than the last edition of the competition.

Defense

The baseline defense model is a Random Forest with 100 estimators that use categorical, textual, and static features with a training dataset consisting of malware samples collected from Brazilian cyberspace, and goodware files from a pristine Windows system. The categorical features were converted into a format that the model could work with using a One-hot encoder, and textual features using TF-IDF. However, it was noticed that the model performed badly on new samples provided by the competition. The reason for the model's performance was probably because a model trained on dataset specific to a certain region would not perform well with malware samples from other regions. This was confirmed by training the same model on the ember dataset.

Attack

The attack strategy used in the previous year's competition does not work well with the new models in the competition. 36 samples managed to bypass the ember model provided by the hosts, 8 samples over the model created by the authors (Need for Speed) and none managed to bypass the model Domumpqb. The Need for Speed model flags the dropper technique in the previous competition as malicious using the functions such as FindResource among others. The usage of this function can be hidden using packers such as TeLock and PeLock. But this technique works against the way we want it to, as the classifier starts suspecting the file as malicious by detecting the trace of packers.

The next strategy to bypass Need For Speed is to customize the dropper into looking like a completely benign application such as a calculator. The way to achieve this is by inserting the calculator part of the code into dead sections that would never be invoked, with no optimizations. So even though the code would not be invoked (which indicates that the functionality remains the same), it is still visible to the ML models and hence could evade the detection system.

The hypothesis behind the samples failing to bypass Domumpdq is that the model is recognizing the embedded payload which would indicate that we need to hide the payload. This could be done by encoding the binaries as a base64 string or by XORing the binary with a key. This strategy led to most samples bypassing the model.

The adversarial samples generated were submitted to the AV model and the detection rate dropped further down than the previous year's attempts. It could also be observed that most of the adversarial samples were similar to each other because the strategy (custom calculator dropper) was utilized for this purpose. Another important experiment that the authors conducted was that they trained the model using adversarial samples. It was noticed that a model trained in this manner started misclassifying most of the goodware files as malware too, which shows that it is not really wise to train using adversarial samples in this manner.

Disucssion

- The idea behind the base64 and XOR strategy was to hide the bytes that the model is on the lookout for. However, XOR has an issue in that it could introduce the byte '0' to the file which indicates the end of strings and hence breaks the functionality. Base64 on the other hand does not have this issue.
- A question that arose was whether it would be possible to detect bytes that do not translate to anything valid as an indicator of maliciousness. It was concluded that this would not be possible because even legitimate applications might try to obfuscate or encrypt part of their code.
- It was also mentioned how hard it is to catch adversarial samples even if it was trained on them. If the adversarial samples are too similar to the goodware files then training on it would lead to the side effect of the model classifying many goodware files as malware. In the case of the custom dropper used in the competition, the dropper was made to look like a calculator. Training the model on such a file might lead the model to classify calculators as malicious.
- The way to defend against such strategies is to make the model try and identify the payload bytes inside the dropper. Using more robust representational forms also helps since it might not be as affected by the benign packaging as a raw feature-based model or a PE file feature-based model would.
- If a model starts detecting a certain library function as malicious, how does the malware creator still make use of it without it being classified by the malware detection system? The malware creator could either write their own implementation of the function or could dynamically load it in which case it would not be visible during the static analysis of the file.

- It was also noticed that even though the detection rates did go down, the files were still classified into different malware families. This could be because certain AVs are not good at identifying certain families. Another observation was that most of the adversarial samples were being classified under the razy malware family. This might be because the structure of the custom dropper could be similar to the razy malware family.