# Functionality preserving Black Box - Optimization of Adv Windows Malware - Summary - Sidharth Anil

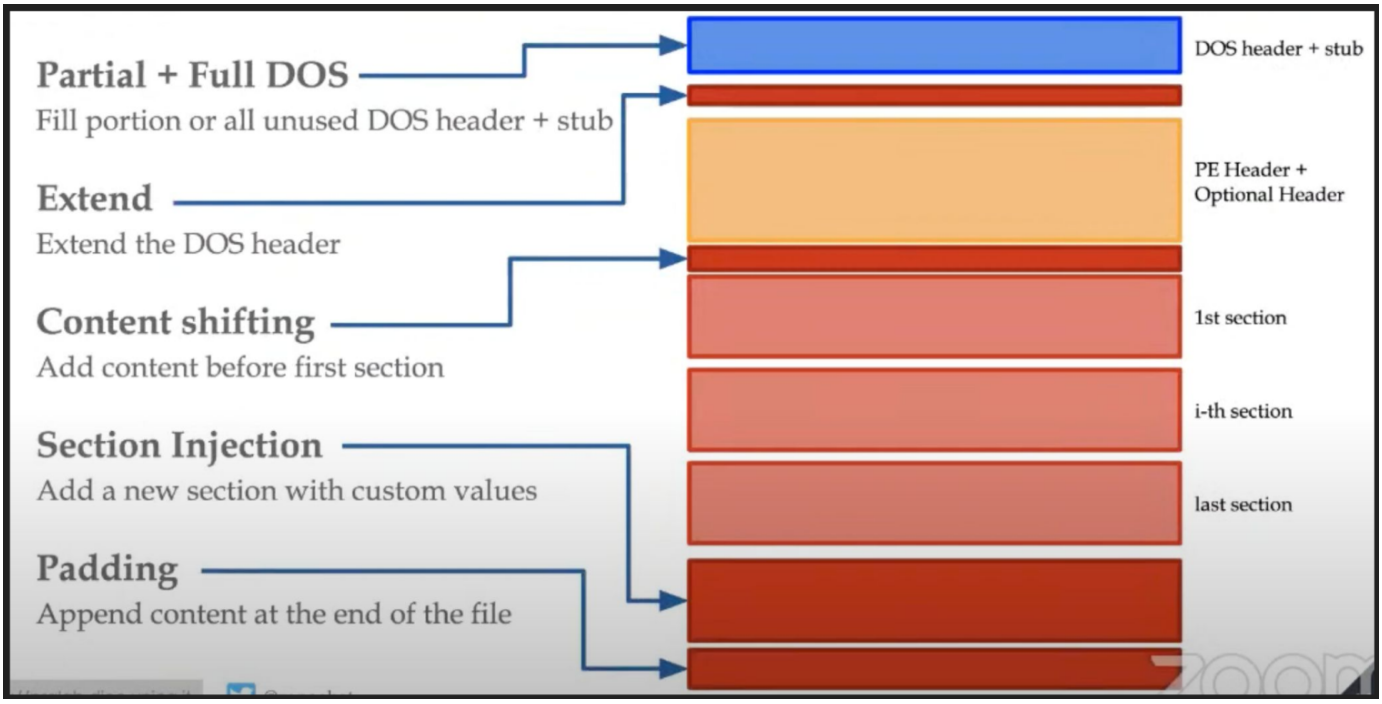
Adversarial attacks on Machine learning-based Malware detection system is a common action by attackers. However, in the case of a black box model where the attacker does not have access to information regarding the internal workings of the model, his/her capabilities get limited. The conventional adversarial sample generation becomes ineffective as it requires a lot of queries to the black box, as well as the requirement of a malware sandbox to ensure that the functionality is preserved after each stage of modification. Therefore the primary concern of the attackers would be to have a strategy that is query-efficient, functionality preserving, and stealthy (the adversarial samples are likely small).

The presentation went over a brief overview of the PE file format as it is necessary to understand the structure and to know what you can modify without compromising the functionality. This includes the DOS Header section which contains a magic number that acts as a signature for the file and an offset value that points to the real header.

The authors focused on two Windows malware static analysis models to build their adversarial technique. They are:

1. MalConv - MalConv models were introduced previously in other discussions and presentations. The key point to note however is that this model that uses CNN only takes in the first 2 MB of the binary as its input and ignores the rest.
2. Gradient Boosting Decision Trees - Unlike MalConv, there is a fixed feature space to which the file information is parsed out using the PE File structure. This feature space is compressed using hashing tricks.

The technique that the authors present is called "Genetic Adversarial Machine Learning Malware Attack"(GAMMA). In this technique, they inject benign content either at the end of the file (padding) or at sections that will never be executed. This is to ensure the functionality-preserving requirement of the strategy. Moreover, this is a genetic algorithm where each batch tries to optimize an optimization function that is reliant on the probability of the malware detector classifying a file as malware as well as the size of the injection.




This image shows the different sections in which benign content can be potentially injected without modifying the functionality of the program.

$$\begin{aligned} \underset{\boldsymbol{s} \in \mathcal{S}}{\text{minimize}} \quad & F(\boldsymbol{s}) = f(\boldsymbol{x} \oplus \boldsymbol{s}) + \lambda \cdot \mathcal{C}(\boldsymbol{s}), \\ \text{subject to} \quad & q \leq T. \end{aligned}$$

The above function is the optimization function that GAMMA tries to minimize. The first term $f(x \oplus s)$ focuses on the probability of the file x after injection of s being classified as malware. While the second term enforces a penalty on the size of injections. This ensures that GAMMA doesn't try to reduce the detection probability by introducing huge injections. Moreover, the constraint $q \leq T$ limits the number of queries allowed below a specified value.

GAMMA does this under the general framework of a genetic algorithm which includes selection, crossover, and mutations. So there is an element of randomness in the whole process where each batch is mutated and only the highest performing (in this case the ones that minimize the function the most) survives.

The results of this technique on the two malware detection system shows that injecting by introducing new sections provide better results than padding benign content at the end. It is also observed that it is harder to affect the GBDT system in comparison with the MalConv. This is due to the representational power and the robustness of the feature space used by GBDT while MalConv uses the raw data as input. Another straightforward observation is that with a larger number of queries allowed, GAMMA is able to reduce the probability of being detected as a malware by using smaller injections. This technique was also found effective against the models that use a hard label (no confidence value is provided) as well as against a number of commercial anti-virus systems.

The countermeasures were also briefly suggested against GAMMA. Using more robust features is an approach that would definitely make it harder to create effective adversarial samples, as is observed by the difference between GBDT and MalConv. Adversarial retraining is another avenue that could help prevent such attacks against the detection systems, as well as specific measures to detect such an attack against the model in real-time to prevent it. Dynamic analysis is another technique that is always more powerful, but it would be dependent on the amount of resources available.

## Discussion

- The meaning and reasons behind the requirements (query efficiency, functionality preserving, and stealthy) were discussed
    - Query efficiency - A lot of requests would make the AV companies suspicious. They might also consider it a DOS attack. So spread it out over time
    - Functionality preserving - Modifications in the feature space has to translate to the problem space without breaking the original function of the malware.
    - Stealthier - Can't have too big malware files, as it gets suspicious and harder to transmit
- AV companies have a lot of measures in their pipeline system to check if the queries are legit. This includes ML models, heuristic techniques, etc.
- The objective function and its minimization is a more systematic and scientific approach to generating adversarial samples. Solving optimization problems is a common theme across the field of computer science and hence it's a more standardized well researched problem.
- In the absence of precise solutions, or precise ways to get to the solution, you rely on methods like randomness and heuristics that Genetic Algorithm employs to get to a good enough solution
- The modifications and the injections have to happen in parts of the code that wouldn't affect the functionality of the program. These spaces are called slack-space/code-space. These spaces exist in the padded areas that normally take care of aligning sections or the in-memory page sizes to the same size for operating system purposes. And the concern of the paper is on where to place it and to what extent in order to deceive the malware detection systems rather than the actual benign content to be injected. That is what the GAMMA framework is focused on.
- In the case of hard-label, a common approach is to get a goodware, ensure that it is classified as goodware by the model, and then start adding elements from this goodware to the file. If this is found to be not working, start over with a different goodware file. This is because you have no measure available to you to decide if the modifications you are making are causing any progress or are at least moving in the right direction. Also, in the case of hard labels, in the optimization function, the detection term would be binary - either detected or undetected and not a percentage value due to the lack of such results by the model.
- Even though GAMMA might be query efficient out of all the different mutated adversarial samples generated, only the final one is utilized. Which could be inferred as a waste of resources.
- Such formalized and scientific approach to adversarial generation is not found in the real world often as most hackers use existing solutions or services provided by others that are mostly built

using random trial-and-error strategies. More than the attackers, these kinds of tactics would be utilized by AV companies to generate adversarial samples to train their models.

- For an attacker, the better method might be to try and turn the black box model into a white box model by guessing the configuration through queries. Once you have a white box local copy of the model, gradient descent is the best strategy as it would lead mathematically to the optimal solution. However, again most attackers would just follow the trial-and-error appending strategies and not such mathematical or algorithmic techniques.
- An important feature for the malware detection systems to focus on is the entry point. The entry point for most of the goodware files would be the same value, so a deviation from that could be a good indication of malicious intentions.
- The attackers manage to get the program to go to the malicious code without actually changing the entry point by putting a jump at the actual entry point location.
- If techniques like these nor the packers/droppers work, then the option for the attackers is to try and decompile the malicious binary to its source code and make modifications in the source code. However, this is hard as the decompilation process does not always give the right results.
- An antivirus system on your machine would assume that everything in the past to its scan is benign. This might be a wrong assumption to make, and whole scans would be recommended to rectify this. Also, another strategy might be to scan new executions instead of scanning new files added to the system.
- An antivirus trying to look for malware in memory would have to follow similar strategies as in-disk. In order to ensure that the entire memory is malware free, it would have to scan through the entire memory.