4.1 Data Processing

The sample set of around 2,500 input malware samples discussed previously proved to be infeasible for testing as a whole within the scope of this research. The primary issue with the dataset was that there were too many executables for it to be processed within a reasonable amount of time. This was especially pertinent in the case of attacks adapted from the *MAB-Malware* library, likely owing to the framework's complicated action minimization algorithm discussed previously. For each adversarial example generated by the framework, the action minimizer must iterate over every major action performed on the original executable and attempt to remove it [6]. If after removing the action's effects from the evasive sample, the file still evokes a misclassification from the malware classifier, then that action is considered redundant and can be removed without being assigned reward points. If removing the action makes the sample no longer evasive, then it can be considered essential to the generation of the adversarial attack, and given the points as usual. However, the algorithm must still iterate over all of the minor actions within that larger, essential action and attempt to remove them in a similar fashion to determine which parts of the action are truly essential [6].

These minor actions can include modifications as small as adding or removing one byte at the end of a section within the executable. Since all combinations of these tiny changes need to be generated and passed to the *MalConv* classifier, the attacks implemented in *MAB-Malware* quickly become very costly to process. Although the major actions can be run individually on a dataset, this process of optimizing out unnecessary minor actions still occurs, which is the main bottleneck on performance. Even with access to high-performance computing resources, the largest number of samples that could be processed using the *MAB-Malware* library for periods of sustained testing was 100. Since we wanted to test all of the attacks and modification types in the same environment and with the same input data, the testing for the *secml-malware* actions also had to be limited to input sizes of just 100.

4.2 Confidence Levels

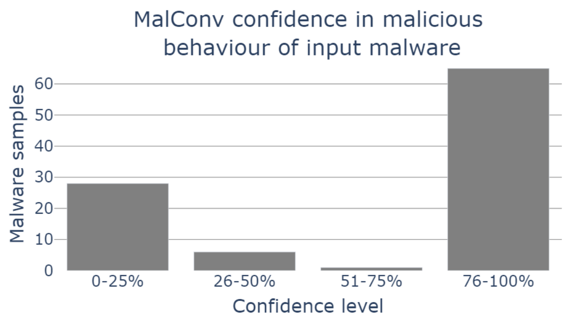

Figure 2: *MalConv* confidence levels in malicious behaviour over a dataset of 100 Windows malware samples

Another essential consideration that needed to be made in conducting this research was ensuring that the input malware samples being fed to the attack scripts are actually being recognized initially as malicious. If they are not, then the samples being considered evasive after actions are applied is an uninteresting outcome, as the classifier already could not determine that it was processing malware. Unlike some antivirus engines that return a simple true or false result as to whether the input is determined to be malicious, the *MalConv* classifier that we are using outputs a confidence rating on how likely the file is to be malware. The confidence levels returned for each executable passed to the classifier from our dataset of 100 executables are shown in figure 2.

To adapt to this system, it is necessary for us to determine a confidence level at which we will consider a sample initially undetected and exclude it from the data set. It would also be useful to determine such a number because it simplifies the process of determining at what point an adversarial attack has made a sample into an evasive one. We ultimately settled on 50% as the baseline of confidence for malicious input, as this rating suggests that the classifier expects that the executable is more likely to be malware than not. If a sample with a confidence level any lower than this is chosen, we will not be able to gather as much useful information from its evasion rate over repeated iterations, so we chose to remove these from the input. As illustrated in figure 3, this resulted in a small portion of our dataset being removed as unsuitable for research.

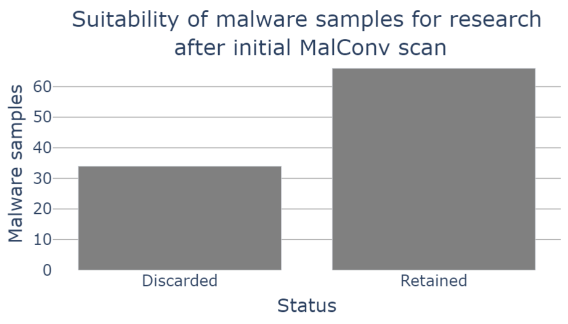

Figure 3: Sample executables removed from the malicious dataset due to low confidence levels

4.3 Evaluation of Algorithm Performance

To make conclusions regarding the effectiveness of the different adversarial example generation techniques, it was necessary for us to define some metrics by which we could determine what constituted better attack performance. We determined that the ideal number of attack iterations over which it was reasonable to assess the effectiveness of an algorithm in creating an evasive sample for a given executable was 20. While this number could realistically be increased, from examining the results of previous research [7], 20 iterations tends to be the point at which an adversarial machine learning-based model begins to hit diminishing returns with further optimizing the action sequence used to evade a classifier [20]. Furthermore, if the number of iterations were any less, the framework may not have sufficient time to learn from its mistakes and generate an adversarial example, making the evasion rate lower than it would be in a practical scenario.

To compare the algorithmic performance of the different action types being tested throughout this research, it will be necessary to also formulate a method of isolating them from other factors and determining their standalone evasion rate. This is a straightforward process with the attacks implemented in the *secml-malware* library, as each action type is provided as a separate module. As such, it is a trivial task to test the modification types individually by importing them and passing them the same input data, and then comparing the subsequent evasion rates. This is also achievable with the *MAB-Malware* framework, as it supports a configuration file that allows for certain features and actions to be enabled and disabled as needed, although this does not allow for the same kind of fine-tuned control available with importing each attack individually.

5 RESULTS

All of the relevant modification types listed in table 1 were tested by taking the implementations of the corresponding attacks from the *secml-malware* and *MAB-Malware* libraries and running them over 20 iterations for each executable in the malware input. The final evasion rate achieved for each sample was recorded using a Python script, along with some general statistics about the entire input like the average number of iterations required for an adversarial example to be generated. This output was then converted into Python dataframes for plotting with the *matplotlib* library, as well as piped to a comma-separated values file for further analysis with Microsoft Excel. Different machine learning parameters like the sampling method were also altered and tested for various values to provide some insight into the optimization of these settings.

5.1 Action Types

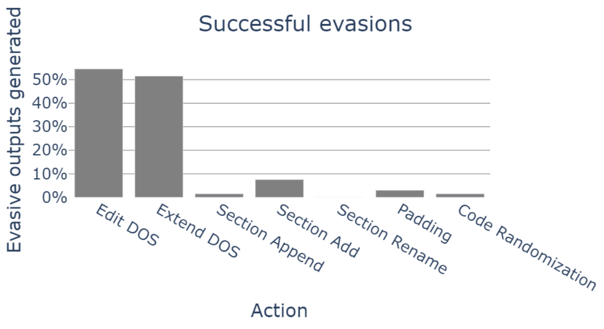

Figure 4: Percent of malware samples for each action for which an adversarial example was generated

Figure 4 illustrates the relative success rates of each of the different action types in generating an adversarial example for a given piece of sample malware in the input. As can be seen in the graph, the most successful types of attacks tend to be those that manipulate the content of the legacy DOS header. This is likely due to the fact that one of the only aspects of the DOS header that cannot be removed without breaking the executable format is a pointer to the real Windows header [21]. By editing the DOS header to modify this pointer's location or destination, the offsets of all of the different sections in the file can effectively be shifted by an arbitrary amount. The fact that this entirely rearranges the layout of the executable, and that the *MalConv* classifier we are using is heavily reliant on structural analysis of files to learn what is and is not malware, offers a possible explanation as to why these DOS attacks cause such a drop in the detection rate

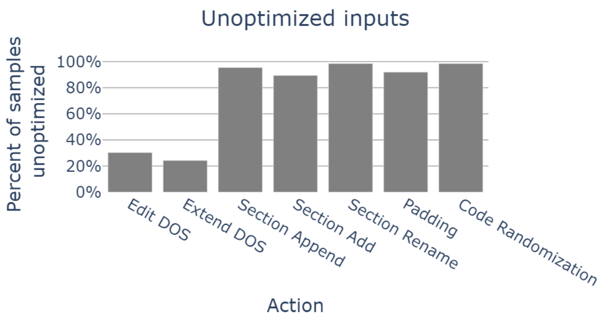

Figure 5: Percent of malware samples for each action type for which the MalConv confidence level could not be decreased at all

The graph in figure 5 shows the portion of samples fed to each attack type that the model could not optimize the *MalConv* malware confidence rate for at all, even after completing 20 iterations of training. Almost all of the samples run with the Section Append and Code Randomization actions fell under this category, demonstrating that the names of the sections within the file and the order of the instruction sequence are not particularly relevant to assessing whether or not a given program is malicious.

Interestingly, the Extend DOS attack was the least likely to not be able to make any optimizations to an input malware file, failing to do so in only around 25% of instances. However, this action type was flawed in other respects. Along with the Section Add attack, it would break the structure of the malicious file in around one in five cases, rendering the executable unusable. From investigating into the issue, however, this can likely be attributed to problems with the detection of header sizes in the *secml-malware* implementation of these attacks, as opposed to a fundamental flaw with how they work.

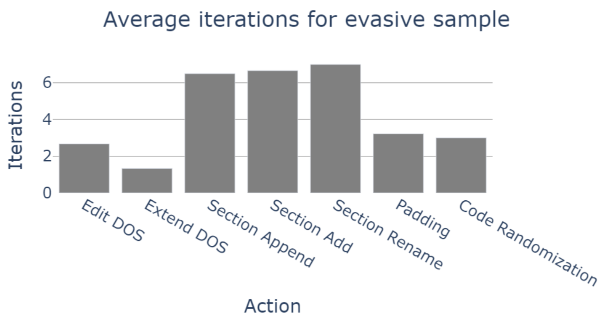

Figure 6: Average iterations for each action type required to create an evasive sample, for inputs where an adversarial example was successfully generated

Figure 6 illustrates the average iterations required by each attack type to generate an adversarial example, where it did successfully create such an example. Unsurprisingly, the attacks that tended to be more likely to do so in general also managed to achieve this in less iterations of the machine learning model, whereas less efficient action types such as Section Append and Section Rename needed more iterations to learn how to evade the classifier.

Figure 7: Percent of malware samples for each action type for which an adversarial example was created within a single iteration of the model

Figure 7 shows the percent of malware samples for which an evasive sample was generated within a single iteration of the model, without any need for further training. The Extend DOS attack was able to achieve this with around half of all malware samples, likely owing to the fact that it is a simple action that just shifts the main header by a custom number of bytes, so there is not much room for optimization. The Edit DOS attack, despite being more successful overall, was less likely to create an evasive sample on the first iteration, probably because the model needs to learn what kind of bytes are appropriate to insert in the DOS header and where exactly to put them.

5.2 Iterations

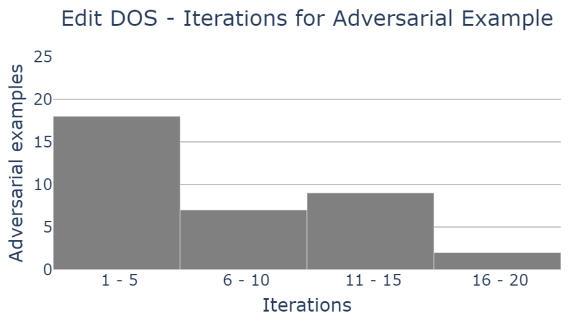

Figure 8: Iterations required to create an adversarial example using the Edit DOS attack type, in cases where an evasive sample was successfully generated

Pictured in figures 8, 9, 10, and 11 are histograms showing the number of iterations of the machine learning algorithm that was required to create an adversarial example for the Edit DOS, Extend DOS, Padding, and Section Add attacks respectively. The same data could not be gathered for the Section Append, Section Rename, and Code Randomization action types because they are only implemented in the *MAB-Malware* library, which does not offer the same level of detailed information about the status of each individual malware sample being processed as *secml-malware*.

Of these attacks, the Edit DOS and Extend DOS actions were the only two that were able to achieve consistent, meaningful success in generating adversarial examples on the given malware dataset. As seen in figure 9, the Extend DOS attack was able to optimize each input executable to pass a confidence test by the *MalConv* classifier within 1-5 iterations in the vast majority of cases, likely owing to the simplicity of this action type overall. The Edit DOS

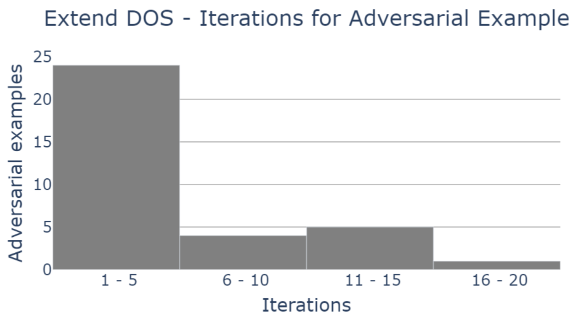

Figure 9: Iterations required to create an adversarial example using the Extend DOS attack type, in cases where an evasive sample was successfully generated

attack had more variance in the number of iterations required to create an evasive sample, but still only required up to five iterations in around half of all cases. What is most interesting about figures 8 and 9 is that both attacks very rarely required any more than 15 iterations to generate an adversarial example, suggesting that the maximum number of iterations could be reduced without having a significant impact on the framework's success rate.

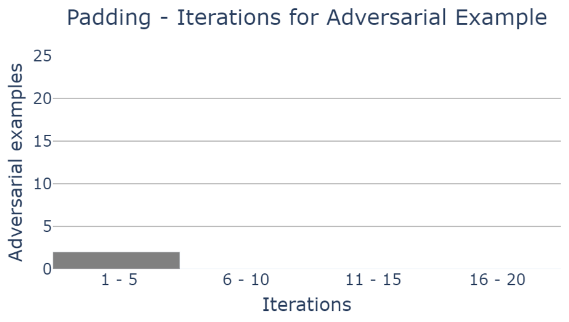

Figure 10: Iterations required to create an adversarial example using the Padding attack type, in cases where an evasive sample was successfully generated

Data is sparse for the Padding and Section Add attacks in figures 10 and 11, since these action types achieved very little success in generating adversarial examples in general. However, it is worth noting that the same conclusions hold for these attacks. The vast majority of evasive samples were created within the first five iterations, and none of them needed more than 15 iterations to create.

5.3 Sampling Method

As discussed previously, the *MAB-Malware* framework makes use of a complicated machine learning model where evasive samples are created by applying combinations of actions to input malware, and the modifications that contributed to adversarial attacks are assigned points [6]. These reward points are then considered and used in deciding which action combinations should be applied to future malware samples to create an adversarial example.

In testing the *MAB-Malware* machine learning model, two options are available for the sampling method used to select these

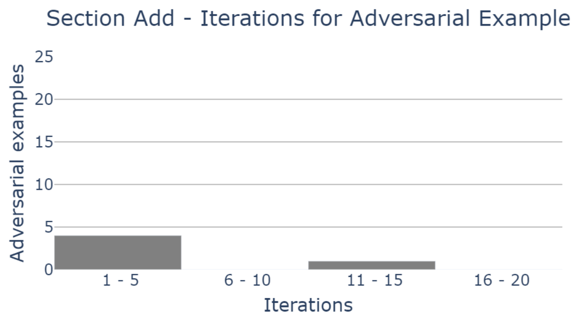

Figure 11: Iterations required to create an adversarial example using the Section Add attack type, in cases where an evasive sample was successfully generated

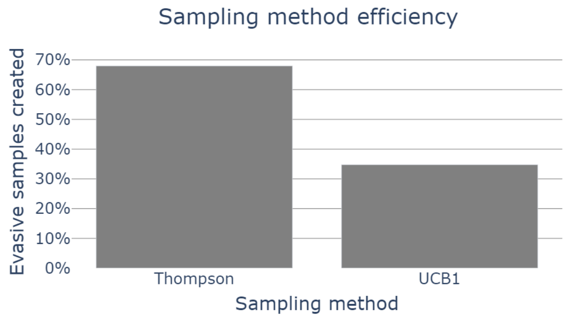

Figure 12: Percent of input malware for which an adversarial example could be created for different sampling methods

action combinations to run on the sample set. UCB1 is a simple sampling algorithm that just takes actions with the highest reward point values to create the action combination that should be applied to the current piece of malware being processed [6]. Thompson sampling, on the other hand, strikes more of a balance between exploration of all of the actions available and exploitation of those known to have been effective on previous samples.

This is achieved through keeping an uncertainty score that begins high and shrinks as more samples are processed by the model. When the uncertainty is high, the framework is more likely to select actions with lower point values, but as it decreases only combinations that have been more successful in the past will tend to be chosen.

These two sampling methods were tested for a period of 4,500 iterations over the given malware sample set. As illustrated in figure 12, Thompson sampling was almost twice as effective in creating evasive samples as the more simple UCB1 algorithm. This demonstrates that some level of exploration is necessary in sampling to allow all of the action types a proper opportunity to generate adversarial examples.

6 CONCLUSION

The primary conclusions drawn from this research related to the types of actions applied to input malware files that were most effective in creating adversarial examples. We showed that when it came to the *MalConv* malware classifier in particular, evasive samples were most commonly created using attack types that edit the legacy DOS header retained in Windows binaries for retro compatibility. This can be attributed to the presence of a pointer in the DOS header to the rest of the file, which can be manipulated by these attacks to effectively rearrange the entire file structure, a modification that *MalConv* has difficulty dealing with. Actions that only manipulated the section names and content of the executable, as well as the instruction sequence of the assembly code, tended to be less effective in generating evasive samples.

The maximum number of iterations allowed for the modifications applied by a particular action on a given sample to be optimized could be reduced to as low as 15, as testing showed that attacks generally experienced diminishing returns beyond this point. The importance of adopting sampling methods that adequately explore all action types available in attempting to create an evasive sample, instead of just picking those that have been the most successful in the past, was also demonstrated. The code of our experiments has been published [1].

Future research in this area could investigate into the possibility of attempting to create evasive samples for commercial antivirus engines, not just *MalConv*. The effectiveness of the *MAB-Malware* action minimizer in optimizing the rewards provided to different actions could also be explored.

REFERENCES

[1] Ruoxi Sun, Wei Wang, Minhui Xue, Gareth Tyson, Seyit Camtepe, and Damith C Ranasinghe. An empirical assessment of global covid-19 contact tracing applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1085–1097. IEEE, 2021.

[2] Xiaotao Feng, Ruoxi Sun, Xiaogang Zhu, Minghui Xue, Sheng Wen, Dongxi Liu, Surya Nepal, and Yang Xiang. Snipuzz: Black-box fuzzing of iot firmware via message snippet inference. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[3] Ruoxi Sun and Minhui Xue. Quality assessment of online automated privacy policy generators: an empirical study. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 270–275. 2020.

[4] Ruoxi Sun, Wei Wang, Minhui Xue, Gareth Tyson, and Damith C Ranasinghe. Venuetrace: a privacy-by-design covid-19 digital contact tracing solution. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 790–791, 2020.

[5] Suibin Sun, Le Yu, Xiaokuan Zhang, Minhui Xue, Ren Zhou, Haojin Zhu, Shuang Hao, and Xiaodong Lin. Understanding and detecting mobile ad fraud through the lens of invalid traffic. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[6] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Mab-malware: A reinforcement learning framework for attacking static malware classifiers. *arXiv preprint arXiv:2003.03100*, pages 1–15, 2020.

[7] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security*, 16:3469–3478, 2021.

[8] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security*, 73:326–344, 2018.

[9] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[10] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[11] Maryam Shahpasand, Len Hamey, Dinusha Vatsalan, and Minhui Xue. Adversarial attacks on mobile malware detection. In *International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, 2019.

[1]https://github.com/hspen4/adversarial-attacks

[12] Sen Chen, Minhui Xue, Lingling Fan, Lei Ma, Yang Liu, and Lihua Xu. How can we craft large-scale Android malware? an automated poisoning attack. In *International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, 2019.

[13] Jialin Wen, Benjamin Zi Hao Zhao, Minhui Xue, Alina Oprea, and Haifeng Qian. With great dispersion comes greater resilience: Efficient poisoning attacks and defenses for linear regression models. *IEEE Transactions on Information Forensics and Security*, 2021.

[14] Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu. Hidden backdoors in human-centric language models. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[15] Jing Xu, Minhui Xue, and Stjepan Picek. Explainability-based backdoor attacks against graph neural networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, 2021.

[16] Shaofeng Li, Minhui Xue, Benjamin Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[17] Alvin Chan, Lei Ma, Felix Juefei-Xu, Yew-Soon Ong, Xiaofei Xie, Minhui Xue, and Yang Liu. Breaking neural reasoning architectures with metamorphic relation-based adversarial examples. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[18] Wei Wang, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, and Haojin Zhu. Exposing weaknesses of malware detectors with explainability-guided evasion attacks. *arXiv preprint arXiv:2111.10085*, 2021.

[19] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European symposium on research in computer security*, pages 62–79. Springer, 2017.

[20] Luca Demetrio and Battista Biggio. Secml-malware: A python library for adversarial robustness evaluation of windows malware classifiers. *arXiv preprint arXiv:2104.12848*, 2021.

[21] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.