

Poster: Take it with a pinch of harmless—studying the robustness of static malware classifiers in the presence of benign code

David Pfaff

CISPA Helmholtz Center for Information Security
david.pfaff@cispa.saarland

Michael Backes

CISPA Helmholtz Center for Information Security
backes@cispa.saarland

I. EXTENDED ABSTRACT

Due to the growing popularity of adversarial machine learning, renewed interest has been shown towards malware classification. In return, the evasion of malware classifiers at test-time using machine learning specific attacks has also been invigorated, as adversarial examples have been shown to subvert a great number of different machine learning classifiers despite repeated attempts at mitigating their threat.

However, the adversarial example attacks imply drawbacks that make them impractical to apply and validate for actual malware samples: adversarial examples seek to limit the amount of perturbation applied to the sample, while provably shifting the sample closer (and eventually over) the classification boundary of a substitute classifier. In malware samples, limiting the perturbation does not positively affect their utility. Furthermore, realizing the small perturbation computed in feature-space as a perturbation on the actual sample is difficult and assumes knowledge of the feature-extraction process.

In this research project, we aim to understand how malware attackers can fool static feature malware classifiers with unbounded perturbation attacks [1]–[3] applied directly on JavaScript malware code. We simultaneously relax attacker assumptions, until we arrive at a fully model-independent and partially data-independent adversary. Our study shows that practical evasion attacks for malware classifiers can be performed with fewer assumptions than previously considered, while realizing strong adversarial goals such as *transferability* and *universality*.

We perform an extensive experimental evaluation by studying a range of different malware classifiers for JavaScript samples. Most notably, we study how different feature extraction and embedding methods perform: most previous research investigated how different machine learning algorithms perform over the same feature sets. Instead, we restrict our study to a single target dataset of malware and benign samples, and focus on different feature extraction methods that have been used in recent literature [4]–[7].

II. SNIPPET-BASED ADVERSARIES

The first adversary assumes label-query access to the classifier and access to the training data distribution, similar to

black-box attackers in adversarial example scenarios. However, different from attacks studied under these conditions, we also relax the noise constraint fool the target classifier.

We start by replacing the noise-limiting constraint by a generic utility constraint: the adversarial modifications should not inhibit the functionality of the malware sample. As adversarial examples are typically computed by iteratively optimizing the noise subject to such constraints, this notably increases the difficulty of producing adversarial examples. In general, we cannot guarantee the preservation of functionality. However, using static analysis, we can determine a restricted set of admissible perturbations. By using domain-specific knowledge, we can construct an attacker that realizes *transferable* and *universal* perturbations across different feature representations and machine learning classifiers.

We propose a *benign snippet transferring* attacker for classifier evasion, that is instantiated by different levels of access to the target classifier. For all instantiations, we assume that the attacker is realized by mining snippets from a (likely) benign substitute dataset. Snippets are connected code segments that can be provably inserted into the target sample using different insertion strategies without interference. We automatically mine snippets by examining the contents of popular web-pages for fully self-contained code that is free of side effects using a light-weight static analysis. Additionally, the malware samples are analyzed for insertion points, where these snippets can be included. The combined results of both analyses yields the parameter space over which the adversarial perturbation is computed.

Historically, optimizations over an appropriate loss function have been the tool of choice to compute adversarial perturbations. Due to our setting, we have no access to the target model, and cannot phrase the optimization problem in a way that makes it amendable to first order methods without introducing additional assumptions. Instead, we utilize random sub-sampling and localized grid-search to efficiently compute perturbations. At this point, we distinguish between three attackers with increasingly restricted access to the target model: ADV-LBL, ADV-QL and ADV-NOBOX. We hereby gradually relax the adversarial assumptions down to a model-independent adversary. ADV-LBL retains label-access to the target model. They can perform an unlimited amount of

queries for samples, returning the classifier result in form of a label. As access to the model is unbounded, this attacker can leverage a grid search over the parameter space of snippet and insertion methods to produce an adversarial perturbation. In contrast to a grid-search over pixel values (or general L-norm constrained adversaries), this method is feasible computationally since the search base is discrete. ADV-QL is given limited query-access to the target model. Since a grid search over the parameter space is no longer feasible, the attacker instead randomly samples parameter choices randomly and tests them against the model until the number of queries is reached. ADV-NOBOX is a fully non-adaptive attacker, and does not have access to the model. We show that no classical training procedure over the target classifier is required. Instead, the attacks rely on statistical measures and similarity metrics over the benign reference dataset.

With limited access to the target classifier, pre-computing an adversarial perturbations that maximize the misclassification probability independent of the underlying malware sample becomes increasingly interesting. We therefore investigate *universal* perturbations and adapt the attackers to compute the perturbations over a set of malware samples rather than a single one.

III. EVALUATION AND PRELIMINARY RESULTS

We conduct a study of the adversarial examples produced the adversaries described in the previous section on a dataset containing 41027 malicious samples and 54021 benign samples mined from the alexa top5k websites. Furthermore, we evaluate across a wide range of different syntactic and lexicographic feature-representations of JavaScript source code used in previous work: N-grams, feature hashes and a AST-path attention model of code. This departs from how most prior work evaluates adversarial examples: As the malware setting is notably diverse in different feature representations, having a unified perturbation mechanism that works by computing perturbations over the input space rather than the feature space enables the study of transferability properties *across feature representations*. We further evaluate the transferability across different classifiers, notably deep neural networks, random forests, gradient-boosted decision trees and hierarchical attention networks (where applicable). To evaluate universal perturbations we give each attacker access to a set of malware samples that were not part of the classifiers training data.

Our preliminary results indicate that all classifiers we consider are susceptible to perturbations computed by each type of attacker. When analyzing the attack success rate as a function of the perturbation (which we compute as a percentage of code size), we see that attackers with less information typically need a larger amount of perturbation to achieve the same evasion rate. However, the fact that these types of attacks are possible at all, even for non-adaptive adversaries, is indicative of the brittleness of Machine Learning to large-scale changes. We believe that the effectiveness of these types of attack is due to multiple causes. First, the attacks generate samples that are very likely to be outside the classifiers training distribution.

Second, if given a sample containing features that are highly indicative of both benign and malicious samples, classifiers seemed to favour specific code segments from our snippet set. As we require a classifier to compute a single label during training, this could cause the classifier to overfit to few, highly salient features that are contained within only small segments of code in the overall document. We traced back this issue to source code re-use: classifiers increasingly tend to *memorizing* code snippets that are *common* in the dataset during training. Finally, code classifiers are forced to fit a document of arbitrary size into a fixed-size representation. This property makes such classifiers especially vulnerable to attacks that are not constrained in the size of their perturbation, as long as we can guarantee that any possible perturbation is viable (i.e. fits the utility constraint).

We further analyze how traditional adversarial training fares against these attacks. Our preliminary results are similar to the findings for image perturbations: adversarial training using PGD [8] does not scale to large perturbations computed by our adversaries. Adversarial training using the adversaries presented here is infeasible for ADV-LBL, for ADV-QL it remains feasible only for very small number of queries. Instead, we use our adversaries in adversarial training by directly augmenting the training dataset. However, we find that they tend to overfit strongly to the adversary used during training. We further investigate whether we can balance the training dataset by removing frequent clones, thereby mitigating the overfitting issue that we believe is one of the main causes why this type of attack works. Finally, we are currently in the process of investigating out-of-distribution statistics to detect perturbations.

REFERENCES

- [1] Y. Song, R. Shu, N. Kushman, and S. Ermon, "Constructing unrestricted adversarial examples with generative models," in *Advances in Neural Information Processing Systems*, 2018, pp. 8322–8333.
- [2] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "A rotation and a translation suffice: Fooling cnns with simple transformations," *arXiv preprint arXiv:1712.02779*, 2017.
- [3] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017.
- [4] A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, "JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2018.
- [5] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 40:1–40:29, Jan. 2019. [Online]. Available: <http://doi.acm.org/10.1145/3290353>
- [6] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 309–320.
- [7] J. Saxe, R. E. Harang, C. Wild, and H. Sanders, "A deep learning approach to fast, format-agnostic detection of malicious web content," in *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*, 2018, pp. 8–14.
- [8] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.


Take it with a pinch of harmless—studying the robustness of static malware classifiers in the presence of benign code

David Pfaff, Michael Backes
CISPA Helmholtz Center for IT-Security

Adversarial examples

Adversarial examples

- Obligatory adversarial Panda




Panda → Gibbon

- Limiting $|x - x'| < \delta$ preserves visual utility of images


Unconstrained adversarial examples

- “Natural” spatial perturbation [1]



Revolver → Mouse trap

- Universal localized perturbation [2]



Banana → Toaster

Adversarial perturbations for malware code

Attacker goals

- Evade detection, preserve functionality
- Adversarial perturbations need to be constrained to valid perturbations only

Code Representations in ML

- Difficulty: code features very diverse
- N-gram, hashed features, AST-path sampling, ...
- No unified model
- Point in feature space does not necessarily represent a valid code sample

Perturbations on code

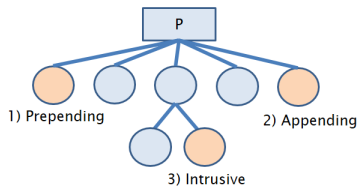
- Perturbation computed directly on source
- Use static analysis to guarantee validity
- However: not amendable to typical attack strategies of adversarial examples
- Guarantee valid samples:
 - Cannot remove code
 - Introduce only non-interfering code
 - Should be “natural” and likely indicative of benign samples

A code snippet transferring attacker

Snippet transferring

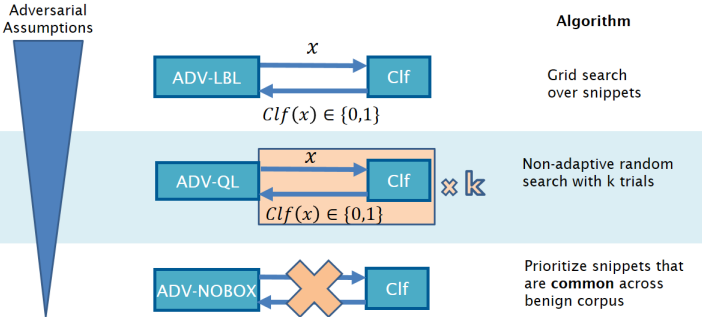
- Core assumption: code classifiers match presence of snippets to target class
- Mine non-interfering snippets from benign reference distribution
- JavaScript malware → code from likely benign web pages

Implantation strategies



Three adversaries

Adversarial Assumptions



Algorithm

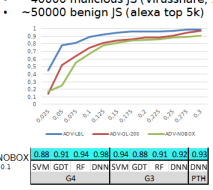
- Grid search over snippets
- Non-adaptive random search with k trials
- Prioritize snippets that are common across benign corpus

Results of attacks

Experiments

- 4 classifiers: Deep neural networks (DNN), Support Vector Machines (SVM), Gradient-boosted Decision Trees (GDT), Random Forest (RF)
- 3 featuresets: 4-gram, 3-gram, AST Paths (PTH)
- ~40000 malicious JS (Virusshare, ...)
- ~50000 benign JS (alexa top 5k)

Adversary	DNN	GDT	RF	SVM	PTH
ADV-LBL	0.73	0.92	0.67	0.73	0.87
ADV-QL	0.74	0.92	0.59	0.83	0.87
ADV-NOBOX	0.88	0.91	0.94	0.98	0.91



Source code re-use: an unseen threat to ML classifiers?

Identifying source code re-use across the web

- Token-based clone detection successful at identifying large-scale code re-use across GitHub [3]
- Feed snippet-level decomposition of benign files to SourcererCC [4] to re-identify snippets
- We find large-scale re-use across the benign dataset even with very conservative matching

Contain clone	Avg # of re-use	Unique code clones
71%	7.3	5704

- Common clones cause the **toxic snippet** problem: “as data is repeated more [...] optimized model will converge on repeated text” [5]
- Adversarial view: inserting toxic snippets maximizes probability of misclassification in model-independent manner

Evaluating potential mitigations

Adversarial training

- Adversarial training with PGD does not protect against natural perturbation
- Using adversaries during training infeasible for ADV-lbl. Adv-ql possible for very small k , but slows down training considerably
- Resulting classifiers overfit to specific samples and insertion strategies

Out-of-distribution statistics

- Preliminary results indicate that perturbations lead to OOD samples
- However, defenses are akin to security-by-obscurity: rely on the fact that attacker cannot optimize against OOS metric

Un-cloning

- Remove clones and train on resulting dataset
- Robustly protects against snippet re-use
- Are code clones not necessary to model the “true” distribution?

[1] Engstrom, Logan, et al. “A rotation and a translation suffice: Fooling cnns with simple transformations.” *arXiv preprint arXiv:1712.02779* (2017).
 [2] Brown, Tom B., et al. “Adversarial patch.” *arXiv preprint arXiv:1712.09665* (2017).
 [3] Lopes, Cristina V., et al. “DéjàVu: a map of code duplicates on GitHub.” *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017): 84.
 [4] Sajani, Hitesh, et al. “SourcererCC: scaling code clone detection to big-code.” *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016.
 [5] Schofield, Alexandra, Laure Thompson, and David Mimno. “Quantifying the Effects of Text Duplication on Semantic Models.” *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017