Poster: Evading Web Malware Classifiers using Genetic Programming

Anant Kharkar
University of Virginia
agk7uc@virginia.edu

Abstract—**Malware classifiers based on machine learning models have become increasingly popular. These classifiers use a combination of structural and dynamic features to detect malware in various domains, including PDF, binaries, and web pages. We propose to use genetic programming techniques to automatically generate variants of malicious web pages that evade state-of-the-art classifiers. Our method builds on the approach Xu et al. (NDSS 2016) developed for successfully evading PDF classifiers. Adapting this method to web page classifiers poses additional challenges because of the dynamic and hybrid strategies used by those classifiers and the complex structure of web pages.**

I. Web Malware

Malware classifiers attempt to determine if a given sample is malicious or benign; they often use machine learning (ML) techniques to train models that classify malware based on structural features. In 2011, Curtsinger et al. created a classifier called Zozzle that uses mostly-static structural features as classification parameters [1]. However, static analysis methods have been shown to be vulnerable to various adversarial methods. Obfuscation is a particularly common practice in embedded JavaScript that often prevents static analysis. Furthermore, dynamic code generation poses challenges for structural feature analysis, since much of the code is not visible to the classifer. More recent web malware classifiers combine static and dynamic features of web pages to yield more effective detection results. Dynamic features of a code sample can be analyzed by running the code in a sandbox and examining the resulting trace. JStill, created in 2013, is an example of such a hybrid (static and dynamic) JavaScript malware classifier, specifically focusing on detection of obfuscated code [2]. Collecting dynamic features may be impractical for high-traffic systems because of the overhead required to execute code, and is also prone to malware that hides its behavior from test environments. Another type of feature used in many web malware classifiers is external information such as domain registration and website history. The scope of this project is limited to evasion attempts via content-based features and therefore excludes such external information, which may not be easily manipulated by an attacker.

Machine learning models are based on features of a training data set, but are applied for predicting unfamiliar testing data sets. Thus, the effectiveness of an ML model is dependent on the resemblance between the training and testing data sets. This is often straightforward for randomized training sets of static data. However, adversarial malware authors represent a dynamic data source that can alter code, resulting in malware with features that deviate from the training dataset of the classifier. The goal of this work is to evaluate the robustness of web malware classifiers against adaptive adversaries with access to the classifier by automatically exploring the space of possible evasive variants.

II. Evolving Evasive Variants

Our goal is to automatically find evasive variants that preserve the essential malicious behavior of a seed sample, but are misclassified as benign by a target classifier. We use genetic programming techniques to conduct a heuristic search for evasive variants. Genetic programming depends on a fitness function that defines a sample's progress toward some goal and attempts to maximize that fitness function by repeatedly performing mutations on samples and selecting samples with better fitness. This process is repeated over multiple generations until a threshold fitness is reached. Xu et al. showed that genetic programming techniques could be adapted to generate classifier-evading variants of PDF malware [3]. That experiment was able to achieve a 100% evasion rate for its target classifiers PDFRate and Hidost in no more than 16 hours using random mutation operations. This was possible because both target classifiers relied on superficial static features that could be modified without disrupting the malicious behavior.

This project applies the approach of Xu et al. toward evading web malware classifiers by attempting to evade state-of-the-art JavaScript-based classifiers. Whereas there exist PDF malware classifiers that utilize static structural features, all modern web malware classifiers examine both structural and dynamic features. This adds complexity, since there is now a level of indirection between the transformations that can be done by the genetic operators and the measured features. If classification is done primarily based on dynamic features, evasive web malware variants must have different runtime behavior than their non-evasive counterparts. This suggests the need for transformation operators that alter dynamic behavior, but without disrupting malicious behavior, such as inserting insignificant events, changing the order of idempotent events, and removing unnecessary events.

III. Design

The prototype system is shown in Figure 1. We target the creation of malicious JavaScript that can evade state-of-the-art web malware classifiers. We are still deciding on appropriate targets and may need to construct our own prototype classifiers; unfortunately, many published classifiers that would be well suited for our experiment are not available to researchers for experimentation. One representative classifier is JSDC [4], which uses both static and dynamic features in three categories: textual (DOM structure), inner-script (embedded AST), and inter-script (external resources). These features are used to train 4 ensemble classification models — Random Forest, J48, Nave Bayes, and Random Tree — to determine maliciousness. Samples that are classified as malicious are also categorized into attack patterns. Another possible target classifier, Lux0r [5], is a JavaScript malware classifier that was original implemented for PDF-embedded JavaScript. Lux0r primarily uses API reference patterns as features for its classifier. It has been tested against adversarial mimicry attacks involving addition of code, making it an ideal target for evasion.

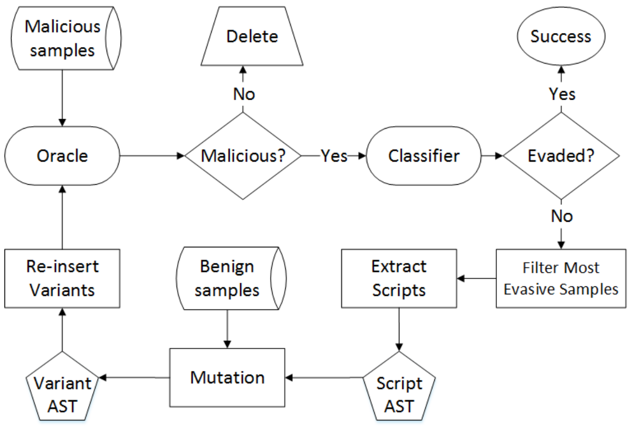

Fig. 1. The experimental system for generating evasive variants.

We begin the experiment with a set of web pages known to contain malicious JavaScript. These pages are verified to be malicious by our oracle, which is a virtual environment using Cuckoo sandbox [6]. Maliciousness is defined according to behavior indicative of malicious intention, such as an unrequested download or suspicious file system activity. Based on the web page's dynamic behavior in the sandbox, a signature is developed that captures the essential malicious behavior, but is robust enough to still match variants that alter the behavior in superficial ways. This behavioral signature is used to test all the generated variants in the experiment. Samples that are confirmed malicious are analyzed by the target classifiers, which return a binary result of malicious or benign for each sample. Samples that are classified as benign by the target classifiers but still match the malicious behavioral signature are considered evasive. Otherwise, we mutate the samples to create new variants.

In order to perform these mutations, we first extract all of the scripts using BeautifulSoup [7]. The scripts are then converted into AST form using the parser Esprima [8]. Mutations are conducted by randomly performing the insert, delete, replace, and crossover genetic programming operations on random nodes of each sample's AST. A set of benign webpages provides supplementary AST nodes. Thus, the mutations introduce nodes from the benign ASTs to the malicious trees. After mutation, the AST is then translated back to code form using the JavaScript code generator Escodegen [9]. Mutated variants are re-inserted in their respective web pages, replacing the original scripts. Since we expect that many of these variants will be corrupted or lose their malicious behavior, we verify their maliciousness in the oracle. Because of the challenges in finding transformations that preserve syntactic integrity of the JavaScript code while changing the dynamic behavior, we expect a random search similar to the approach for PDF malware will not be effective. Instead, language-specific transformation rules will be developed and combined with random transformations. Transformations that produce valid, but dynamically different code, will be collected and reused for other samples as inputs for the next cycle of the system. We will explore strategies that focus on identifying the script on the page primarily responsible for the malicious behavior, and modifying only that script, as well as strategies that transform all scripts on the page.

IV. Current Status

We are currently in the early stages of conducting this research, with many open questions to explore. We are nearly finished building a prototype evasive variant search system, and are exploring options for building a prototype target classifier for the experiment.

References

[1] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: fast and precise in-browser JavaScript malware detection. In *Proceedings of the 20th USENIX conference on Security*, 2011.

[2] W. Xu, F. Zhang, and S. Zhu. JStill: mostly static detection of obfuscated malicious JavaScript code. In *Proceedings of the third ACM conference on Data and application security and privacy*, 2013.

[3] W. Xu, Y. Qi, and D. Evans. Automatically Evading Classifiers: A Case Study on Structural Feature-based PDF Malware Classifiers. In *Network and Distributed Systems Symposium*, 2016.

[4] J. Wang, Y. Xue, Y. Liu, and T. Huat. JSDC: A Hybrid Approach for JavaScript Malware Detection and Classification. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015.

[5] I. Corona, D. Maiorca, D. Ariu, and G. Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript code through Discriminant Analysis of API References. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, 2014.

[6] https://www.cuckoosandbox.org/

[7] https://www.crummy.com/software/BeautifulSoup/

[8] http://esprima.org/

[9] https://github.com/estools/escodegen