Identifying API Access Behavior Anomalies

Remy LeWinter, Ashwin Rupakala, Rahul Suryadevara

Abstract

APIs can expose components of internal business logic which attackers may try to exploit. Attacks on APIs can have different API access patterns than normal API use. In this project we will evaluate various machine learning approaches to binary classification of observed behaviors as either normal or outlier. Our dataset[1] consists of a supervised training set with 1560 records and an unsupervised testing set of 34562 records, with 9 features representing API access patterns. We will use approaches possibly including but not limited to cross-validation, GD/SGD, MLE, logistic regression, naive Bayes, perceptrons, SVMs, neural networks, and decision trees.

Introduction

Microservices are applications structured as loosely coupled services as opposed to monoliths – structured as single self-contained artifacts, each service focusing on one aspect of business functionality. A company may use microservices for easier development, integration and maintenance of apps along with allowing for step-by-step building of applications so the entire application is not affected if mistakes are made. In the modern day market, consumer satisfaction and commitment are items that companies value dearly and companies use microservices to ensure customer satisfaction. This may involve the company having to be able to provide 24 hour customer support as well as questions to any small details any client may be having. As it may be hard to facilitate a human employee for extended durations and a lot of repeated questions, APIs (Application Programming Interface) have aided companies in being able to serve their customers during any part of the day and from any part of the world. Our group found it fascinating to be able to investigate the different behaviours users might have with an API.

First we must understand what an API is. An application programming interface is a connection between computers or between computer programs. It is a software interface that offers a service to other parts of the software. It also allows for internal and external developers to access an application's data or use an application's functionality. Some examples of APIs are weather snippets, dual factor authentication, Pay with PayPal, Twitter and FaceBook bots, and travel bookings.

Lately, however, there have been an increasing number of attacks on APIs. One of the main problems with APIs is the level of permissions often granted to them. Because APIs are not intended for human use, they are often configured with unrestricted access in web or mobile application environments. Typically, permissions are configured for the user making the original request, and those permissions are passed to the API. However, problems with this method of access authentication arise when an attack bypasses the user authentication process. Because the API itself has unrestricted access, successful API attacks can give hackers access to all of an organization's data. Securing APIs is very important because of the amount of information that is available to them, along with the financial/time impact to businesses and those relying on the services.

[1] https://www.kaggle.com/tangodelta/api-access-behaviour-anomaly-dataset

In order to avoid falling victim to hackers targeting APIs, it is recommended to take measures such as implementing API access controls for both users and applications, maintaining an API inventory, implementing encryption, logging API connections and using rate limits. This is where a process can be established for monitoring API connections. The use of API call graphs represent the abstraction of a program with reference to the sequence of API calls made by an executable has info such as total execution time, node name, time stamp, unique identifier.

Proposed Project

For our project, we plan to use API access pattern data to predict whether instances of API access are normal (non-malicious) or outlier (potentially malicious). API access patterns are captured as behavior metrics, which reflect users' API call graphs. While behavior metrics are typically represented as categorical entities, the dataset being used for this project has already quantified most of these metrics as continuous features.

Our dataset is from Kaggle and is called "API security: Access behavior anomaly dataset." This real-world data primarily comes from financial institutions and E-commerce groups. The data source is presented in the form of a "supervised dataset" with 10 columns and 1,560 rows, and a "remaining behavior" dataset with 11 columns and 34,562 rows. The former will be our training data and the latter our testing data, and we will be using the 9 features that are common between these datasets in our machine learning models, 7 continuous and 2 categorical. The features are the time interval between API accesses in a user session, ratio of distinct APIs in a user session to the total number of API calls made in that session, total number of API calls made in a session by a user (on average), duration of a user session within an observation window, type of IP the user came from, number of user sessions each with a different session ID, number of users generating the same type of API call sequences, number of unique APIs, and source of data. These 9 columns collectively represent an observed behavior. The final feature column, classification (training set) or behavior type (testing set), is the actual (training) or estimated (testing) outcome associated with each observed behavior (row). Behavior types in the test set are the results of an unidentified algorithm provided by the data uploader. Though we may wish to compare our test results to these, we will likely extract a validation set from our training set. Of the 1560 supervised records, 29% are classified as outlier, so a proportionally sampled 20% validation set would still contain ~90 records in the smallest class.

We have no missing data or data type mismatches and the data is already encoded into features, so we will not need to do any preprocessing. The outcome of our feature vectors is binarized into either a "normal" or "outlier" class, so we will use binary classification to address this task. Our plans will solidify as we cover more topics in class, but approaches will likely include some combination of cross-validation, (stochastic) gradient descent, maximum likelihood estimation, logistic regression, naive Bayes, perceptrons, support-vector machines, neural networks, and decision trees.