Calvin Chen, Lauren Cheong, Brian Walker

CSE331

Final Project Report

Introduction

Our group has been assigned project #1, specifically the Web Application Firewall (WAF). As suggested by the project description, we have decided to use mitmproxy as a reverse proxy due to lack of knowledge about Apache modules. In addition to mitmproxy, we will also use Joomla CMS as suggested to demonstrate our WAF’s capabilities regarding signature and anomaly detections. Finally, we will be using WampServer in order to host our local website that uses Joomla CMS. The entire project will be set up locally.

Procedure

First order of business was to download Joomla CMS, along with WampServer. We then placed all of the Joomla files within the www folder of WampServer in a folder we named cse331. This gave us the starting point. Upon opening localhost/331, this will bring us to the installation of Joomla. Following the instructions on screen, we created a super user, as well as linked WampServer’s included PHPMyAdmin, using the username root without a password. Joomla allowed for the initialization of our database, filling it with tables prefixed by what it randomly chose. In this case, we named the schema cse331 with the table prefix b4hi8. This completes the initial configuration of our Joomla CMS.

As mentioned, we will also be using mitmproxy. With Python 3.x installed, we can simply utilize pip to install mitmproxy, running the command pip install mitmproxy. Unfortunately, running mitmproxy on a Windows operating system leaves us unable to use the terminal console for mitmproxy, leaving us only mitmweb, the web console, and mitmdump. Running mitmproxy as a reverse proxy is quite simple, requiring only a single --mode reverse:[url]. By default, mitmproxy will listen in on port 8080 so we will have to set our wampserver to be listening on port 8081. By specifying the url in the reverse flag, we will have properly set up the reverse proxy, grabbing requests from port 8080 and then routing it to port 8081. There are however some consequences of this. Since we have set the proxy to only be listening on port 8080 and the web server to be listening on 8081, any static link in our web server will bypass our reverse proxy. This causes quite a few problems, especially when testing post methods.

Mitmproxy also allows the inclusion of a Python script that works as an addon. Theoretically, this Python file will filter through the HTTP Requests according the rules in a “database of malicious signatures,” or simply a text file. Our script defines a new class that will be an addon, set by using the -s [path] flag. Notably, the last line of the file adds an addons list, with the instantiation of our new addon class.

Our class will write a new log.txt for the sake of debugging. This done by a simple file open call within Python. The class contains a single function, with the signature request(self, flow). The request function will read all HTTP Requests sent through the proxy, with the flow being the HTTP Request Flow, thus containing all the information. We will loop through each line of the rules file and match the rules with the text specified. This part was quite difficult since mitmproxy’s API is not very well documented. Many of the objects returned can only be understood under the inspection of the source code. As a matter of fact, mitmproxy is on version 4 while the documentation is on version 2. Overall, most of the information required will be in flow.request. For attacks involving headers, we will be checking the header attribute of the flow.request object. On the other hand, attacks regarding the HTTP Method will require some more checking of the parameters and the HTTP method. This will conclude the signature detection portion of our project.

Complications