

Visualization of Infiltration Attempts

By: Ryan Dougherty

Advisor: Ming Chow

Abstract

One of the problems with using infiltration detection software is understanding what can be learned from the infiltration attempt. Location, frequency, timing, and infiltration technique details are all lost in the rapid-fire output from an infiltration detection program output. Visualizations for this data can help break down the overwhelming amount of data into digestible chunks. Attacks can be sorted by attacker location, categorized by attack type, and filtered by attack frequency. All of these visual filters can help a user understand how their system is infiltrated in a more contextual way. This proposed system maps a Ruby infiltration detection system to a web application using the Google Maps API to display infiltration attempts to a network on a map, and Processing to display infiltration details.

Introduction

What are infiltration attempts?

When designing an infiltration detection system, it is important to first consider what an infiltration attempt is. In the basic form, an infiltration attempt is anything that allows a user access to information on your machine. This can be achieved in multiple ways. One of the more common methods is by the method of port scanning. In this method, the attacker sends TCP packets to your server, and waits for a response. The packets are configured in such a way that your server will respond with information that allows the attacker to infer how you have configured your server.

What is Processing?

Processing is a library built for Java meant for creating visualizations. Processing has tools for creating text and shape based visualizations in both a 2D and 3D environment. In this project, I used Ruby-processing, a Ruby library, to create the primary visualization. Ruby-processing is a Ruby gem that calls native Processing functions.

What is Ruby?

Ruby is an interpreted language used for both online and offline applications. Ruby is used all over the web for content generation, database management, complex visualizations, and many more applications. I chose Ruby because of the universal knowledge of Ruby on the internet.

To The Community

This application was created to allow people to better understand their network security in an easy to use way. No security background is required to run the application, and there is no need to understand how it works. What's important is that the program is easy to use and understand. With only three commands, the user can run the intrusion detection system, update the Map database, and run the Map server.

I chose to use Ruby and Ruby on rails for this project because of the universal usage of Ruby all over the web. There are many tutorials online that users can go through if they run into any bugs with the program, or want to change anything about it. The main Ruby application is used to show attack information, while the rails server is used to show attack geolocation data.

Application

Detection

The main detection program is written in Ruby. It is built to detect basic known infiltration attempts to a server. The program searches the log for NULL, FIN, XMAS, Nikto, NMAP, and Masscan Scan attempts. The program also searches for Shell Shock exploit attempts, phpMyAdmin exploits, leaked credit cards, and anything that looks like shell code.

For the NULL, FIN, and XMAS scans, the program looks at the TCP header of the log entry and checks how the flags are set. NULL has no flags set, FIN has the FIN flag set, and XMAS has the FIN, URG, and PSH flag set. If the flags are set in this way, then the infiltration is detected and is written to file.

Nikto, NMAP, and Grahm's Masscan are easily detected scanning attempts. The program scans the log file searching for the words "nikto" "nmap" and "masscan" with varying capitalization. When an incident is detected, the infiltration attempt is written to file.

Shell Shock, phpMyAdmin, and shell code infiltration attempts are similar word searching techniques. Shell shock has a very distinct pattern of having `(){:;}` before bash scripts. The detector searches for that pattern and flags the line as dangerous. phpMyAdmin can be seen be searched for by looking for "phpmyadmin" in plaintext with varying capitalization. Compiled shell code is searched for by looking for hexadecimal values in the log file. Hexadecimal values can be seen by the telltale "\x" that comes before the numbers.

Credit card leaks can be detected by searching for the basic patterns for credit cards. The patterns for major credit card companies are as follows.

Regex	Carrier
/4\d{3}(\s -)?\d{4}(\s -)?\d{4}(\s -)?\d{4}/	Visa
/5\d{3}(\s -)?\d{4}(\s -)?\d{4}(\s -)?\d{4}/	MasterCard
/6011(\s -)?\d{4}(\s -)?\d{4}(\s -)?\d{4}/	Discover
/3\d{3}(\s -)?\d{6}(\s -)?\d{5}/	American Express

Visualization

The main visualization (Figure 1 & 2) is built using the Ruby-processing gem. Ruby-processing is a Ruby gem wrapper that calls native Processing functions. The main visualization shows a breakdown of the infiltration attempts to the system.

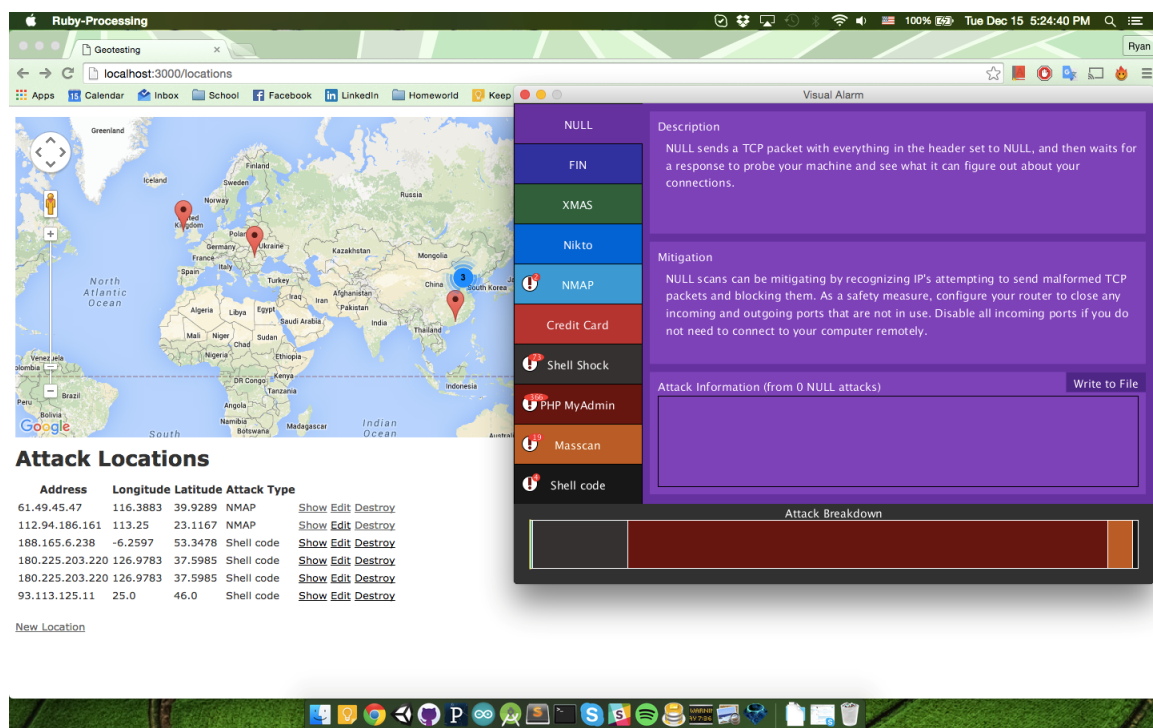


Figure 1 Server and Visual Alarm

The sidebar allows the user to select the different infiltration attempt types. If the user has not viewed an active infiltration attempt, a notification is displayed in the sidebar. If the attempt type is selected, the main view is populated with information related to the attempt.

For each infiltration type, there is a description of how the infiltration works, how to mitigate the infiltration attempt, and a list of IP addresses tagged with the time of infiltration (See Figure 2).

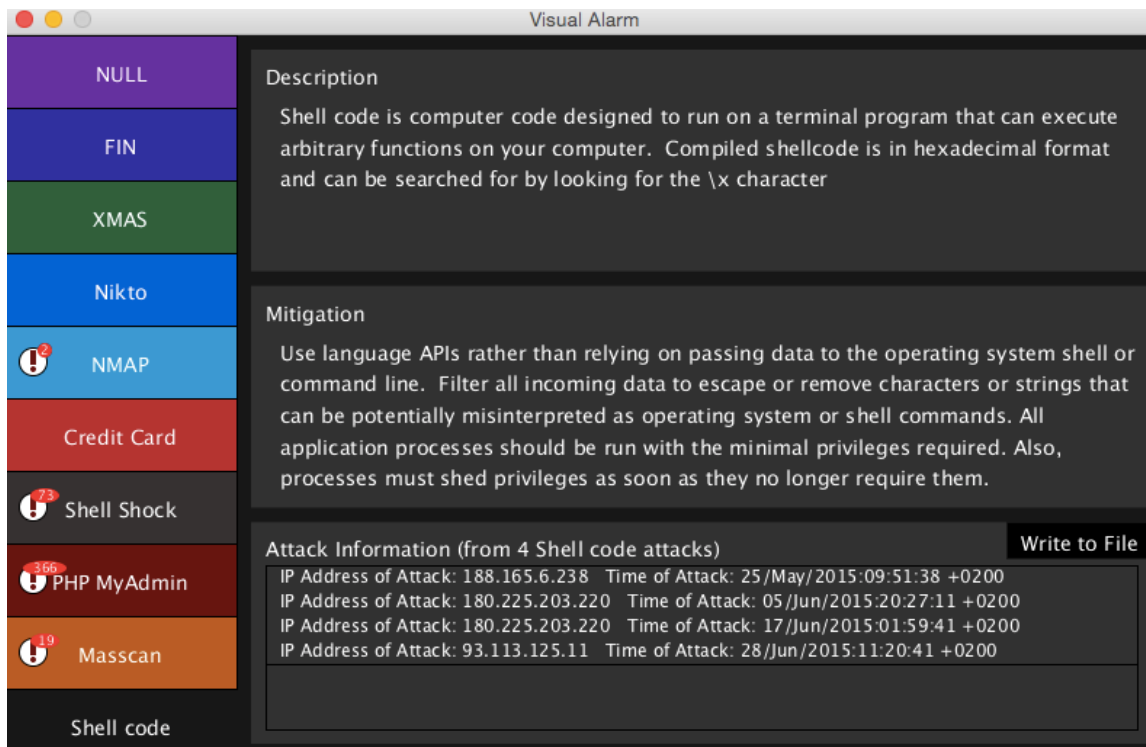


Figure 2 Visual Alarm with IP Information

At the bottom of the visualization is a bar that shows a ratio breakdown of infiltration attempts. The bar is highlighted when the user has selected an infiltration attempt that was detected in a Multiple Coordinated Views fashion. If the user mouses over the attack breakdown section at the bottom, they can see how many times each infiltration technique was attempted.

The Ruby-processing visualization generates files for the rails server in the pattern {EXPLOIT_NAME}.log. That are filled with comma separated values with the IP address and infiltration attack type of each attack.

Ruby Server

The Ruby rails server uses the geocoder and gmaps4rails gems to map IP information to specific longitudes and latitudes. The application looks in the app/IPlogs folder, and adds all log information to the database using the Ruby make command, rake ("rake db:populate"). After that, the data is set up, and the server can be run with "rails s" and viewed on localhost:3000.

User Experience

This program was designed for people without a security background, but still requires a bit of a computer science background. The user needs basic understanding of ruby gems and rails in order to build and run the visual alarm and the geolocator server.

The program was created with a minimalistic design in mind. This was to reduce distraction, and make sure the only things being presented to the user is information they need to know. Google maps was used for the geolocator server's base because it is a technology that is familiar to many users. Markers were clustered to reduce clutter when attacks were attempted from the same area multiple times.

Going Forward

This system can be reworked in the future in many ways. In the future, it is planned to integrate a live packet sniffing Ruby gem packetfu with the application. Unfortunately, the jruby that runs Ruby-processing made the current application unable to process real time packet information, and all information had to be fed in from a server log file. Additionally, the attack type detection system is expandable for future use. The project would also benefit from a single shell script to run all of the build and make commands.

Conclusion

Security information overload can be mitigated with the usage of visualizations. Network infiltration attempt detection can leave the user flooded with information they do not understand or are unable to deal with in size. Visualizations can be used to present large amounts of data in a more palatable way to a larger audience. The melding of these two softwares allows the user a streamlined experience with a much lower barrier to entry, and an easy to use and understand system.

Reference

1. <https://www.sans.org/reading-room/whitepapers/services/requirements-managing-security-information-overload-1147>
2. <https://www.giac.org/paper/gsec/3499/information-overload-techniques-tools/103258>
3. [http://www.academia.edu/3949995/Minimalism in information visualization on attitudes towards maximizing the data-ink ratio](http://www.academia.edu/3949995/Minimalism_in_information_visualization_on_attitudes_towards_maximizing_the_data-ink_ratio)
4. <https://capec.mitre.org/>
5. <http://www.businessinsider.com/heres-why-ruby-on-rails-is-hot-2011-5>