Visual Geolocation Based Black and White Listing

Karl Cronburg*

Dept. of Computer Science, Tufts University, Medford MA 02155

Abstract

Various commercial and open-source tools are available for managing IP-based filtering of web traffic. A common open-source method is the use of static 'tricks' found on stackoverflow (e.g. defining Apache rewrite rules). A more involved approach is to create *iptables* rules. However, the availability of tools to interact with iptables from a high level is lacking. As such, we present a novel approach to managing iptables filtering through an interactive map of the world powered by Google Maps and IP2Location.com. Our approach focuses on effectively conveying web site traffic in a way that any content-provider can understand, regardless of prior experience with tools like iptables. We discuss the effectiveness of existing black and whitelisting tools, and where our approach fits into today's network security landscape. We conclude with a discussion of possible improvements to our approach, which should lead to increased productivity of website administrators in dealing with network attacks.

*Electronic address: `karl@cs.tufts.edu`

I. INTRODUCTION

The most common application of defense-in-depth is through the application of firewalls, in particular firewalls with rules either blacklisting or whitelisting traffic based on IP and DNS information associated with packets coming in and out of a network. The purpose of a blacklist is usually to predict the source of future network-based attacks, mitigating attacks before they have the chance to unfold. [1] The types of network attacks mitigated can range from Denial of Service (asymmetric attacks) to Cross-Site Scripting (application-specific attacks) and beyond.

Since blacklisting is predictive in nature, most research in this area is on pattern matching and the compiling of databases of suspected attack patterns. As a result blacklisting is susceptible to never before seen attacks coming from previously unsuspected sources (which, when possible, is mitigated using whitelists). The techniques for compiling and using databases of attack patterns are described by [2]. The primary techniques used today are known as GWOL and LWOL. *GWOL* stands for 'Global Worst Offenders List', which focuses on institutions and users world-wide collaborating in an effort to compile attack sources. Implementing GWOL is similar to implementing a *Web of Trust* for public key distribution; groups of institutions and users who inherently trust each other exchange their network logs in an effort to paint the fullest picture of the Internet. The second technique, *LWOL*, stands for 'Local Worst Offenders List', which focuses on profiling attack patterns on a specific machine / subnet of the Internet.

GWOL builds on the work of the already well-established LWOL technique, synthesizing the information compiled by LWOL machines. The work of [1] further builds on the work of [2] by looking for patterns *across* network logs rather than on an individual machine basis. The everlasting challenge with pattern matching, however, is being able to put network traffic into context. While an algorithm to match these patterns could take many years to perfect and subsequently implement, a human can be trained to pick up on such patterns in a matter of days. A human can synthesize many different sources of both subjective and objective data in a heartbeat whereas algorithms in IDSs today must be carefully fed every piece of information they require and employ some complicated algorithm to compute objective metrics. While long-term automation in this manner is certainly a noble pursuit, with CS grad students and professors meticulously developing algorithms and publishing

groundbreaking papers over the years, the challenges of detecting and preventing attacks *today* remains an issue we need to solve either *today* or in the not-to-distant future.

This paper presents a proof-of-concept implementation of human-driven pattern matching / blacklisting. The hope is that a complete implementation can be created in the course of a couple months.

II. TO THE COMMUNITY

A. Black & White Listing Applications

1. Most attempts to filter spam and block phishing incidents involve IP blacklists; statistics about long-term trends in network traffic can be used to correlate IP addresses into groups, specifically botnets. [3] Using a visualization tool similar to the one discussed in this paper is a possible way to look for patterns in the geographic layout of a botnet, and to be able to compare the size, extent, and characteristics of one botnet versus another.

2. Determining whether to use a whitelist or a blacklist is a key issue in various situations. For instance in [4] the efficienty of an academic document search engine is determined when using a blacklist versus a whitelist. In this research, efficiency was measured as the speed at which the crawler is able to index appropriate academic documents. In this situation whitelisting was found to be more efficient with higher quality documents indexed. In contrast, the blacklist had a low efficiency with many low quality documents, but has a lower false-negative rate.

3. In [5] the impact of Real-time Blackhole Lists (RBLs) is discussed. A RBL is simply a blacklist of IPs for which a subnet of the Internet will drop / ignore all requests going to or from that IP address. The discussion touches upon the affinity for certain RBLs to have very specific geographic qualities / distributions. That is to say, RBLs for SPAM and certain active attacks tend to contain a lot of Asian / Pacific and European IP addresses. In contrast phishing and malware RBLs tend to contain a lot of North American IP addresses. [5]

III. ACTION ITEMS

A. Data Collection and Analysis

Our implementation utilizes the ruby programming language to collect and analyze network packet data. Given a set of blacklisting or whitelisting rules attained from our data visualization technique (Section III B), we employ the *ip2location_ruby*, *nfqueue*, *haversine*, and *packetfu* gems available in ruby to enforce these rules.

The *ip2location_ruby* gem is used to map IP addresses to (longitude, latitude) coordinates on the globe. Thus, given the rule 'blacklist IPs from this region' and an IP packet passing through the iptables chain of rules, we can decisively say *yes* this packet is OK to let through or *no* this packet should be dropped. See Section IV for other kinds of rules we would be able to enforce using *ip2location_ruby*.

The *haversine* gem is used to determine whether or not the coordinates of a given IP address is contained or not contained by a given rectangular region on the globe. The haversine formula allows us to calculate the 'great-circle distance' between two points. From this formula, any variety of filtering rules can be devised.

The *nfqueue* gem is a ruby interface to libnetfilter in the linux kernel. From a security standpoint this library is the place where any firewall or IDS is able to monitor and filter packets. The library lets you define a callback function for when packets matching a pattern enter the network stack. In ruby, this means being given a binary string containing each packet, and being able to issue accept and reject verdicts on each of the packets.

The *packetfu* ruby gem is a convenient tool for extracting information from packets. In the case of our prototype, we are able to extract the source IP address from the IP header and act accordingly. See Section IV for other possible uses of *packetfu* in the context of security and blacklisting.

The current functionality of our prototype is as follows:

1. Viewing of static historical apache log data. A content provider can, using our prototype, analyze / view an apache log file containing suspicious entries. This is useful for content providers who want to effectively figure out where all attacks on their machine are coming from.

2. Viewing live data taken from apache using a log parser. A content provider can thereby

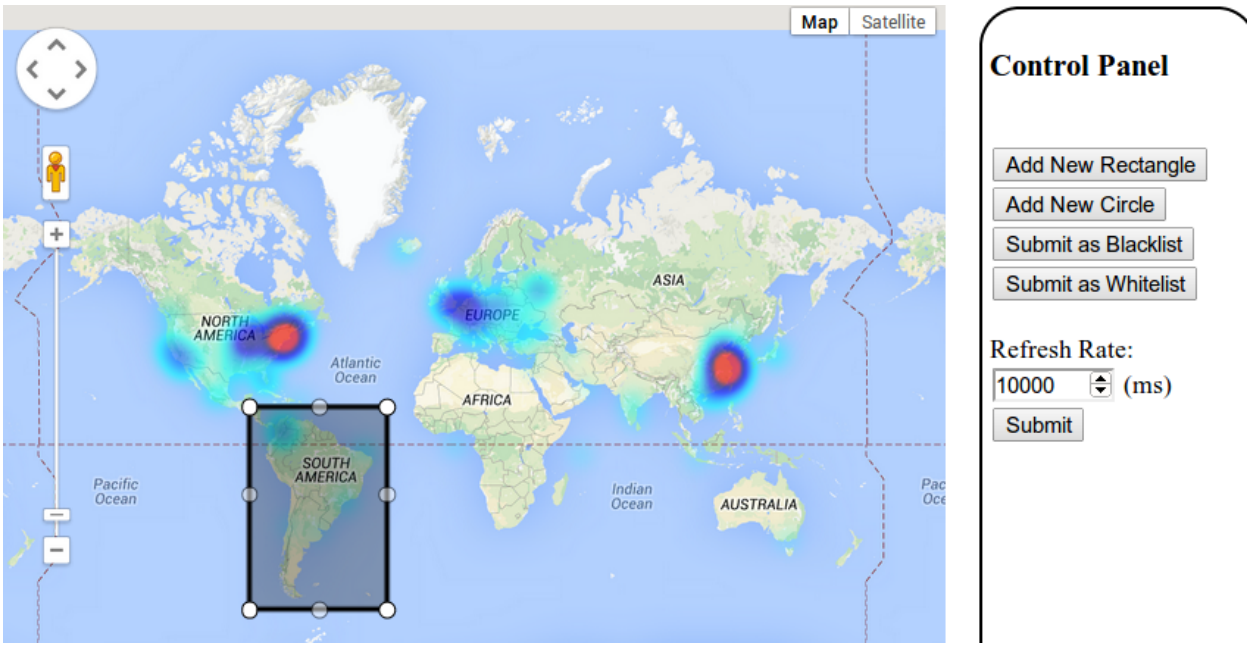

FIG. 1: A heatmap of static / historical data from a personal web-server open to the Internet for the past few years. The data was manually filtered to show only suspicious activity, based on known legitimate usage of the server. The data may however contain a number of false positives located in Pennsylvania and Boston, the two primary locations the server was used.

see in real time where HTTP requests are coming from, making them more informed when designing security policies for their network or website.

3. View and filter live packets taken from the linux network stack. This is the core feature of our prototype; this is also the feature with the most room for expansion. A content provider can currently specify filtering rules using the visualization technique (Section III B) which are then carried out either on a gateway machine or the content server itself.

B. Data Visualization

Our prototype implementation has a javascript-driven web front-end which queries the Google Maps API as seen in Fig 1 with Fig 2 showing just the New England region. The figure contains a heatmap of, in this case, static / historical data from a personal web-server. The more red a region is, the more incidents there were that came from that region. This

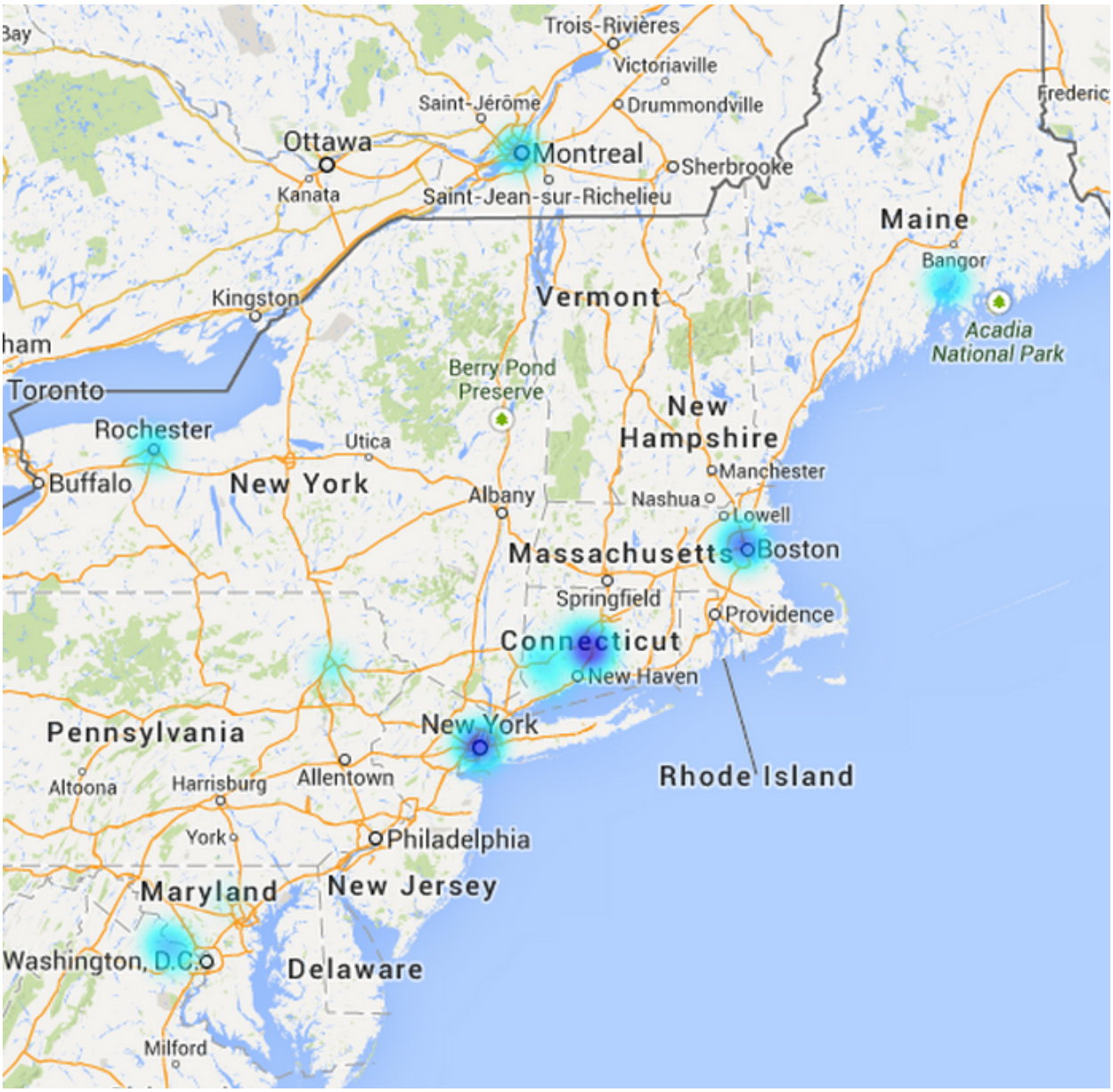

FIG. 2: A zoomed-in version of Fig 1 focusing on the New England region. It is more apparent in this image the primary limitation of our method. That is, the granularity of the geolocation data is limited to the zip code in which an IP address is located.

view of the data gives the user a good feel for geographically similar incidents as well as number of incidents occuring there.

C. Interactivity

Users can interact with the front-end by doing the following:

1. Create a rectangular or circular region on the map which can then be submitted as a blacklisted or whitelisted region.

2. Changing the refresh rate of the page data.

3. Scroll and zoom the map just like any Google Map.

IV. FUTURE WORK

In this section we give lists of possible ways to improve on our proof-of-concept prototype, in particular what analysis can be easily automated without the need for complicated algorithms, and in what ways we can better give the user feedback to how the rules he or she came up with are playing out on the network.

In Section III A, we described the most basic kind of blacklisting rule possible using *ip2location_ruby*; regional IP address based blacklisting. The database available using this ruby gem however is not limited to (IP,geolocation) tuples. The following rules could also be implemented:

1. Regional country / nation based filtering. Although the content provider can easily select a rectangular region around a particular country, it may be more useful in certain situations to be able to select a region by country. The ip2location ruby gem gives a mapping from IP address to country / region which would allow for such filtering.

2. A content provider with a user-base concentrated in a specific locality / city may wish to handle requests from the city in question on one powerful server, and other requests on a separate slower server. The 'city' field available in the ip2location database gives us this capability. The content provider can select a city from the map or dropdown menu, and have the requests not from that city redirected to the secondary server. From a security / defensive point-of-view this eliminates the useability impact of the website on the majority of the user-base when an attack comes from outside the target city.

3. Along with the city-based filtering, *ip2location_ruby* has a zip code field, giving the content provider more flexibility in how he or she provides the system with filtering rules.

4. A 'timezone' field is also available in *ip2location_ruby*, giving us easy access to the time of day in the region from which a packet originated. A content provider might come to expect certain traffic patterns based on the time of day from a particular region. Since network incidents tend to occur at random, without regard for normal traffic patterns, the timezone field allows us to alert the content provider of anomalous-looking traffic and let the content provider act accordingly.

In Section III A we also described how *packetfu* is used to parse packets taken from the network stack. Given the other information *packetfu* makes readily available (all TCP and IP header fields of a given packet) a more versatile implementation would compile useful statistics and trends seen over time. In the front-end (the Google Maps visualization) we would then be able to present the statistics in a meaningful way in relation to everything else being presented. The content provider could for instance flag packets with high RTTs (round-trip times) and further investigate where they are coming from and what the requests are for.

The ip2location database lookup API we used in our prototype uses a binary search tree to store the IP address to geolocation information mapping. Since people rely on and have grown to expect low latency over networks, the use of a binary search tree in ruby is likely not a scalable solution when it comes to analyzing every single packet on a network. One possibility is to pre-compute a hash table each time the content-provider submits a new set of blacklist / whitelist rules from the web interface. This hash table would map IP addresses to geolocation information in $O(1)$ time, making it a scalable solution on high bandwidth networks. The hash table then need only be distributed to all the gateway machines available, dividing up the workload while maintaining consistency of the desired security policy.

Some of the future improvements to make on the visualization side are as follows.

1. Show live packet statistics and / or live graphs to help the user correlate large traffic bursts with packet and data types.

2. Allow advanced users to edit `iptable` rules themselves. Would need to do syntax verification and input cleansing to make sure only `iptable` rules are performed.

3. Add a color-bar to indicate the exact rate at which requests are originating from a given location.

4. A hover textbox feature for telling the user the exact rate at which requests are originating from the location over which the mouse is hovering.

V. CONCLUSIONS

Although geolocation based blacklisting is, in its' present form, a form of unverifiable authentication (IP addresses can be spoofed), its' economic benefits cannot be denied. As such, firewalls, blacklists, and IDSs are here to stay. It is therefore in our best interest to come up with quick and easy techniques for improving the security of our networks today rather than waiting for vulnerability patches and smarter algorithms.

---

[1] F. Soldo, A. Le, and A. Markopoulou, CoRR **abs/0908.2007** (2009).

[2] J. Zhang, P. Porras, and J. Ullrich, in *Proc. of USENIX Security '08* (2008), pp. 107–122.

[3] A. G. West and I. Lee, in *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference* (ACM, New York, NY, USA, 2011), CEAS '11, pp. 73–82, ISBN 978-1-4503-0788-8, URL `http://doi.acm.org/10.1145/2030376.2030385`.

[4] J. Wu, P. Teregowda, J. P. F. Ramírez, P. Mitra, S. Zheng, and C. L. Giles, in *Proceedings of the 3rd Annual ACM Web Science Conference* (ACM, New York, NY, USA, 2012), WebSci '12, pp. 340–343, ISBN 978-1-4503-1228-8, URL `http://doi.acm.org/10.1145/2380718.2380762`.

[5] J. Zhang, A. Chivukula, M. Bailey, M. Karir, and M. Liu, in *Proceedings of the 14th international conference on Passive and Active Measurement* (Springer-Verlag, Berlin, Heidelberg, 2013), PAM'13, pp. 218–228, ISBN 978-3-642-36515-7, URL `http://dx.doi.org/10.1007/978-3-642-36516-4\_22`.