Rutgers University

Capstone Senior Design Project

14:332:492

Scarletshield

A Lightweight, Linux-based Network Security Solution Suite

Team:
Jeff Adler
Eric Cuiffo
Parth Desai
Jeff Rabinowitz
Val A. Red

Advisor:
Dr. Manish Parashar

May 1, 2014

Contents

1 Abstract **2**
1.1 Introduction 2

2 Hardware and Software Architecture **5**
2.1 Evaluation of System Hardware 5
2.2 System Software Architecture 6

3 Approach and Methodology **9**
3.1 Approach 9
3.2 Employing Two-point Authentication over SSH 10
3.3 Installation 10

4 Administration Suite & Features **16**
4.1 Web Front-end Integration 16
4.2 Analytics 17
4.3 Response Options 19

5 Conclusion **21**
5.1 Cost/Sustainability Analysis and Scalability 21
5.2 Future Work 22
5.3 Summary and Concluding Remarks 23
5.4 Contribution Breakdown 24

A jQuery for the Scarlet Suite **27**

Chapter 1

Abstract

"Scarletshield" is a customized Ubuntu 12.04 LTS image featuring a uniquely layered cyber security deployment of several open source services; most notably the **"Snort" Intrusion Detection System (IDS)** utilizing packet inspection and the **"Fail2ban" Intrusion Prevention System (IPS)** utilizing log inspection. It is optimized for synergetic network defense and complemented by a dynamic, modern web frontend providing the utmost flexibility for network administrators to monitor and quickly react to any and all threats against a network. The design of was conceived as a proof-of-concept to expand upon the more orthodox but aging Defense in Depth (DiD) strategy that layers *overlapping* technologies rather than presenting varied layers of *complement* defense services working in *synergy*, which is the approach use to expand upon the defense-in- depth strategy. In addition, Scarletshield enables the potential for a user to interface with interconnected subnets and gateways over a private network to enable scalability and mitigate large-scale attacks. Overall, Scarletshield is intended to be a flexible, open-source system for preventing and thwarting evolving Distributed Denial-of-service (DDoS) Attacks and Advanced Persistent Threat (APT) by implementing a synergy of various open source and custom designed services designed to be as flexible and as intuitive as possible for the network administrator to interface with, minimizing administrative overhead and maximizing mobility for the network administrator to track and react to threats.

1.1 Introduction

Attackers have the edge in the dynamically evolving field of cyber security. With a wider breadth of strategies, near-infinite resources, and a virtually untraceable mask of anonymity; hackers have long boasted the advantage, successfully rendering useless the generally accepted DiD layered defense mechanisms most organizations deployed for network defense. Essentially, is analogous to an onion: although it is deeply layered, it can be peeled. Utilizing an even wider breadth of tools and resources, hackers have successfully and continuously

developed new and advanced approaches, exploits, Denial-of-service (DoS) methods, etc. to employ a persistent, peeling threat designed to break down or even bypass the DiD approach employed by network administrators. In essence, this summarizes the APT , which is not a single exploit/attack or even a collection of exploits/attacks: APT is a *campaign* involving possibly several of different collections of exploits and attack methodologies and even multiple actors across different machines that may exist in different countries.

Defense in Depth (DiD) [1], as described by the National Security Agency (NSA) , is "[a] practical strategy for achieving Information Assurance in today's highly networked environments. It is a "best practices" strategy in that it relies on the intelligent application of techniques and technologies that exist today."

While DiD appears sound in its introduction, a closer and modern interpretation of later details in the NSA strategy employing defense-in-depth shows some of its flaws that come with age, for example: "To effectively resist attacks [...] an organization needs to *characterize its adversaries*, their potential motivations, and their *classes of attack*."

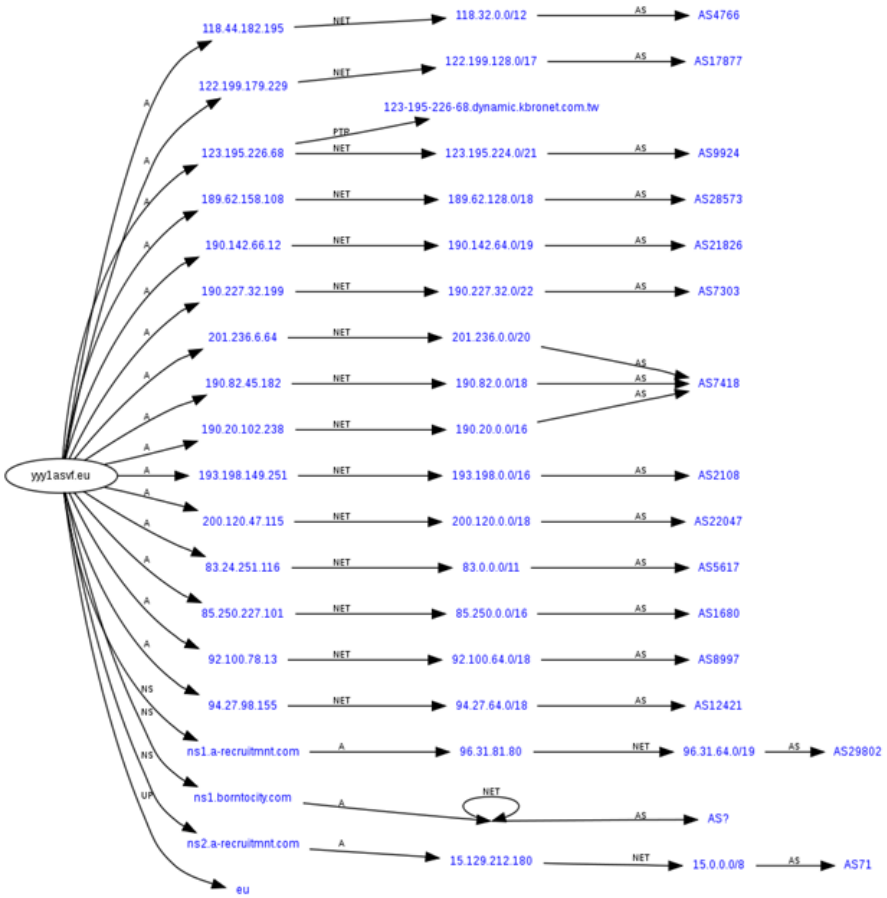

Figure 1.1: A visualization of the domain fast-flux. Note the large number of IP addresses and domains mapping to a single fully qualified domain name used for the C2 server.

The above notion presented by the NSA is idealistic at best in today's climate of virtually

untraceable, anonymous hackers employing robust Domain Name Service (DNS) techniques such as **fast flux** [2], which essentially accomplishes hiding a single Command-and-control (C2) server by utilizing numerous IP addresses and a single, rotatable (in an even more difficult-to-trace **double-flux** methodology) fully qualified domain name (something as simple as valred.com or hackr.ru, for example) to very quickly swap between the available IP addresses and DNS records to avoid detection and backtracing. Thanks to these particular methods, problems such as phishing will be omnipresent in the foreseeable future, exasperated by other countries boasting lax restrictions with regulating the domain name provisioning. As such, the NSA's notion of having organizations "characterize its adversaries," is among the highly impractical and dated points that weaken the argument for the defense-in-depth strategy. The only likely moment adversaries are to be characterized is when it is too late and a network has already been compromised, such as when the Syrian Electronic Army (SEA) successfully executed a man-in-the-middle attack rerouting traffic to the U.S. Marines [3] in 2013. As such, we keep Scarletshield lightweight and practical by not addressing this issue; instead, we specifically handle threat prevention and detection.

Strengths of the defense-in-depth methodology are applied in the implementation of Scarletshield . Particularly, one directed focus is on protecting the most exposed web-facing systems that are typically the target of automated attacks. This is specifically handled by the robustness of the Snort IDS, which essentially sniffs all types of Internet Protocol (IP) packets (TCP, UDP, ICMP, etc.), and checks against patterns, or *rules* and *signatures* indicative of an attack. This is reinforced even further by the log-inspection capabilities of fail2ban, which can essentially pattern-match very common exploits and DoS traces against typical protocols such as HTTP, SSH (Secure SHell, which we utilize to remotely access and work on the Scarletshield server) , and FTP (File Transfer Protocol, commonly used for storing and transferring files from a remote server) and react by blocking IP addresses logged via "iptables", a powerful firewall built into most modern Linux distributions (distros) .

Even though iptables comes with modern Linux distros by default, many end users beyond actual system and network administrators fail to utilize iptables to its full potential. Scarletshield can partly fill the role for a network administrator when an end user does not have the personnel or resources to fully utilize iptables themselves. It can essentially be connected to any switch or router and be configured to act as a network gateway such that all traffic to any end user on a network will have to make it past Scarletshield first.

One issue that comes with defense-in-depth is overhead. How will a user interface with Snort and fail2ban? Scarletshield bridges the gap by presenting all IDS and IPS information and actions in a modern, sleek front-end accessible via private network (so only computers in the Scarletshield network may access it) in a way that minimizes the overhead of a network or system administrator having to manage everything via command line.

In summary, Scarletshield employs a depth of security systems including Snort and fail2ban while also boasting a breadth of options such as iptables handling and dropping of malicious sources' packets while interfacing with other gateways and subnets to maximize overall security in a network.

Chapter 2

Hardware and Software Architecture

2.1 Evaluation of System Hardware

Scarletshield in its current manifestation runs in the Microway server described below. Note how the hardware is actually dated by today's standards; this exemplifies the scalability of Scarletshield – it does not actually require a high-end server to operate efficiently, but can actually operate on smaller devices as well. A more detailed study of scalability factors occurs in Chapter 4.

The largest performance factor when deploying the server for monitoring gateways is the Network Card. In the small, local, web-facing network we employed, a one-gigabit network card was sufficient enough to handle all requests and still have Scarletshield simultaneously replicate and check the requests against our IDS and IPS configuration. However, for maximum scalability, a network administrator should mind the network card being used, and if a Scarletshield-esque design is to be employed, also ensure that a second network card is present to facilitate DHCP and network connectivity for local machines within the network.

Chipset	ASUSTek Computer INC. K8N-DRE
CPU	2x Dual Core AMD Opteron(tm) 275 : 2.2GHz 2x1 MB L2 Cache
RAM	4x8GB
OS	Ubuntu 12.04 LTS
HDD	60GB HDD
Network Card	2x NetXtreme BCM5721 Gigabit Ethernet PCI Express

Table 2.1: The hardware specifications for the Microway Server provided by Rutgers Engineering Computing Services (ECS).

2.2 System Software Architecture

As shown below, Scarletshield is composed of two interfacing subsystems – a packet inspection and logging system, composed of Snort, Barnyard2, Pulled Pork, fail2ban, and iptables; and a web-based administrator suite powered by Drupal. These two systems are married by a MySQL server, which holds both Snort's configurations, as well as the attack logs.

The Administration Suite utilizes Drupal, a Content Management System (CMS), not only for its maturity, but also for its modularity and extensibility; PHP scripts can be added to Drupal's configuration and then served to web clients with minimal effort. This is actually how the Scarletshield Administration Suite operates – through a series of modular PHP scripts furnished to Drupal. This minimalistic approach has the advantage that adding a new page to the suite is as simple as writing another script and providing it to Drupal. In this way, Scarletshield can be customized by end users to meet individualized needs.

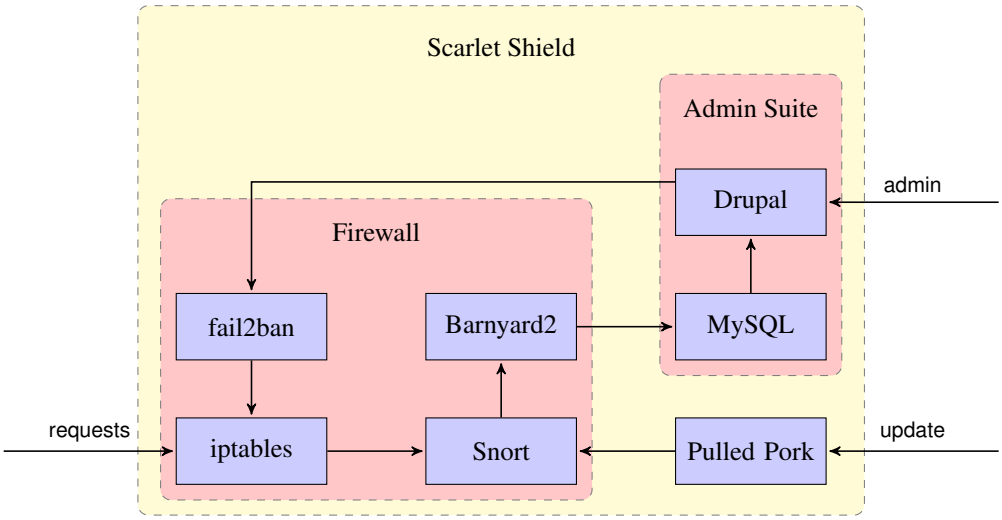

Figure 2.1: The architecture of Scarletshield. Incoming packets are routed through iptables, where they may be blocked per iptables' rule-set; Snort receives the packets and inspects them for signatures of attacks; offending packets are mirrored via Barnyard2 into a MySQL database; users can access the Administrator Suite provided by Scarletshield and hosted using Drupal in order to analyze attacks against the system; fail2ban automatically bans abusive foreign hosts. Pulled Pork is used to automatically update Snort's classification rulesets for offending packet patterns.

Scarletshield was implemented behind a Ubuntu 12.04 LTS Server edition for purposes of stability and universality. In addition, the "LTS" acronym stands for Long Term Support, a term coined by Canonical, the developers of Ubuntu, that essentially means that they are supporting and supplying security updates for the Ubuntu 12.04 LTS for five years from its release date. Ubuntu 12.04 LTS was released in April of 2012 (hence, it is version 12.04) and thus Scarletshield will have a completely secure operating system until April 2017. Note, however, that most of the subsystem software components to be listed in this section are

cross-platform, and thus compatible with other contemporary Linux distros and even modern Windows operating systems to an extent. This speaks to the scalability of the Scarletshield concept, which will be discussed in a later chapter.

Specifically, here is a breakdown of each component that builds the full software architecture behind Scarletshield, starting with Snort followed by each successive in the diagram component:

1. Snort 2.9.X (2.9.3 is used for Scarletshield) – A free and open source IDS developed by Sourcefire, Inc. suited towards network intrusion prevention, which is one of the core functions behind Scarletshield. Snort works by creating a separate, virtual and promiscuous (meaning that all packets received by Snort's separate interface gets directed to the central processing unit for inspection) interface that copies traffic off of the network card as it is received, enabling the simultaneous inspection of packets as they reach their intended destination in a timely manner. This means that Scarletshield is able to inspect packets without creating any latency or overhead experienced by clients actually intended to receive said packets, since Snort is only inspecting carbon copies of said packets, instead of the actual packets, as they pass through Scarletshield. To work effectively and stay up to date with the latest updates, Snort utilizes a ruleset of known and relevant exploits and employs pattern-matching to catch packets suspected of malicious activity. How is Snort kept up to date? It uses another critical component for updating known as Pulled Pork.

2. Pulled Pork (the Perl script is sometimes referred to in its concatenated form, Pulled-Pork) is not actual software, but an open source Perl script also by Sourcefire, Inc. that incorporates rule updating and management that facilitates the timely, monthly updating of the Snort IDS.

3. Barnyard2 – When Snort executes its packet inspection, and finds suspect packets which trigger a signature identifying a pattern of malicious activity, it presents a number of logging options. Out of its options, the most lightweight and most efficient method of logging and consolidating these suspect packets is via unified2 (u2) binary logging. The optimal means of reading and working with such logs is via an open source interpreter known as Barnyard2. Taking the u2 logs, barnyard2 will continuously process data that is written into them by the Snort IDS and output this data into readable data in a format that is then readable by an administrator via a database.

4. MySQL – A heavyweight, well-established, and open-source database management system, we utilize MySQL to integrate all records of packets captured by the Snort IDS and translated by the Barnyard2 interpreter.

5. Drupal/JavaScript – We used the PHP-based Drupal CMS to build scarletshield.rutgers.edu, our primary front-end for the presentation of our data. While we originally intended to integrate the MySQL database of our packets into Drupal directly, we actually found that utilizing javascript for the presentation of this data was a far more presentable

alternative. This is what facilitates administrators/users being able to view our data on a web browser.

6. fail2ban – Working in parallel with the Snort IDS, fail2ban is an IPS based on Python that utilizes log inspection to facilitate the proactive blacklisting of IP addresses that are the source of malicious activity, specifically brute force attacks out-of-the-box, and interfaces with the built-in firewall on modern Linux kernels managed by user space application, iptables, to drop packets from aforementioned malicious activity sources. By itself, it is unable to protect against more sophisticated exploits and attacks such as distributed denial-of-service. When reinforced with an IDS such as Snort, however, and integrated in our Scarletshield system, they can together log and stop a large volume of known attacks.

7. iptables – A user space application enabling management over the Linux kernel firewall via a system of chains and rules. The iptables name also refers to the entire kernel-level subsystems that make up the Linux kernel.

Chapter 3

Approach and Methodology

As we are designing Scarletshield from scratch with a Ubuntu 12.04 LTS server, there are several sections to our Approach and Methodology:

1. Design & Approach
2. Employing Two-point Authentication over SSH
3. Set up of Snort and fail2ban

3.1 Approach

To summarize the process in the previous chapter's system software architcture subsystem breakdown – Scarletshield takes all incoming packets and either accepts or drops them based on iptables, which is built in to modern Linux distros such as Ubuntu 12.04 LTS automatically. What makes Scarletshield unique is how we deploy an IDS and IPS through Snort and Fail2ban. Specifically, IDS mirrors all packets and analyzes them for patterns (via Snort rules, which are updated every month over the Internet) indicative of a potential threat, then logging threat signatures on a database accessible to the administrator and analyzed for new, more strict iptables rules banning offending IP addresses where necessary. We utilize fail2ban to detect obvious brute force, overflows, and exploits in the server logs (/var/log/auth.log, etc.) to also automatically filter and block IP addresses preventively, requiring no action from the administrator.

Thus, with the Snort IDS and fail2ban IPS working in concert with the administrator and the kernel-level firewall managed via iptables, the Scarletshield approach accomplishes the following two critical objectives:

1. Implements a level of automation to immediately act against obvious threats requiring

no action from the administrator.

2. Captures more ambiguous threats with the potential of attempted penetration and attacks against the Server and presents it to the administrator via a web-accessible front-end that gives the administrator an analysis enabling further policy control to help thwart potential threats.

3.2 Employing Two-point Authentication over SSH

Through our research as a group, we came to the conclusion that the safest way to protect our SSH server against brute forcing would be to change our SSH's port from 22 to create security through obscurity. However, in order to gather the largest possible set of logged attack data, we needed to present SSH on its standard port. This led us to a different approach of setting up two-factor authentication. By utilizing Google's "Google Authenticator API", in order to successfully gain access to ScarletShield's SSH server, one must first enter the correct password and the unique TOTP security token, described by RFC6238 and generated by the "Google Authenticator" app, for the specific user trying to login. This security measure makes it impossible for a malicious user to gain login access through brute fore.

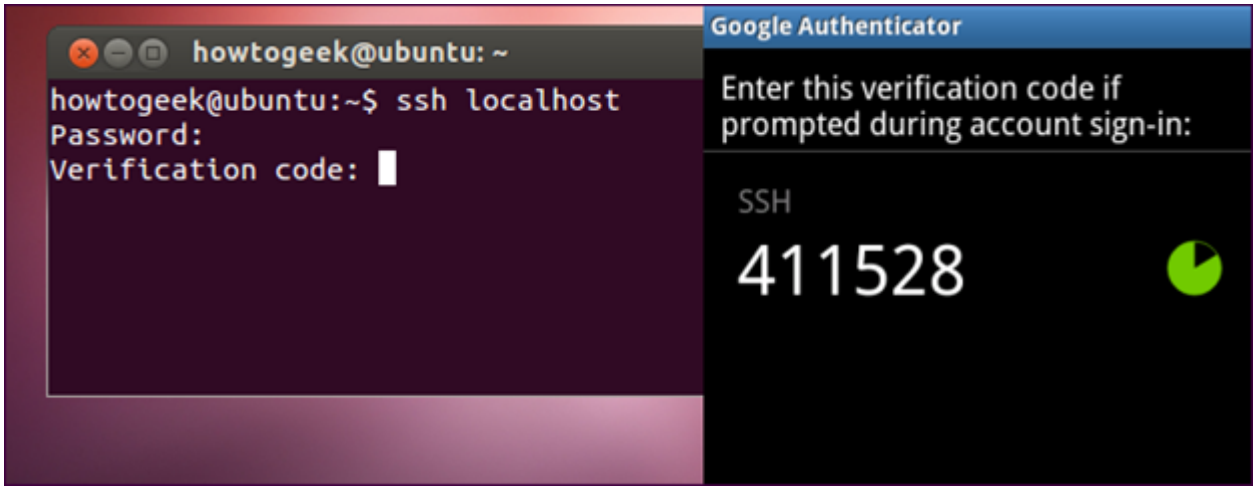

Figure 3.1: (Left) A prompt to enter a Google Authenticator verification code after typing in a password. (Right) Google Authenticator smart phone application generating a verification code.

3.3 Installation

Scarletshield covers the breadth of IPS and IDS by implementing both fail2ban and Snort and interfacing the former with iptables, a user space application for packet handling. This not

only enables the mobility for a network administrator to easily be able to monitor incoming traffic and potential threats on a switch, but also provides an automated mechanism via fail2ban to block obvious, trivial threats such as brute force attacks and denial-of-service.

Fail2ban is a very simple installation in Ubuntu 12.04 LTS, since it is included in the advanced packaging tool. Using its internal mechanism known as "jails", it essentially utilizes filters implemented based on obvious threats (brute force over SSH, flooding over apache, etc.) and, over a user-set period of time (default period is several minutes long), bans offending IP addresses for the duration after it gets caught by Fail2ban via iptables directives packaged with the program.

Snort is a larger beast with many different variations of configurations available (types of relational databases used, logging mechanism, etc.); at the very least, an administrator separately needs to acquire and install libdnet and the Data Acquisition API [4]. In addition, an administrator must choose a database and correctly install and configure its respective libraries and server application. Due to how fully featured it has become over the years, we chose MySQL.

In our Ubuntu 12.04 LTS Server Edition and extensible to any Debian based system, fail2ban is available via the built-in Advanced Packaging Tool (APT), and can be installed practically instantaneously via the following command-line directive:

```
    apt-get install fail2ban
```

Due to how comprehensive the Snort IDS is and how it integrates with Pulledpork and Barnyard2, there are several of steps to successfully deploying Snort, Pulledpork, and Barnyard2 on Linux distributions, and thus the instructions typically vary depending on the specific operating system involved. Our instructions is largely derived from the Snort Install Guide 2.9.3 for Ubuntu Linux, and the process we took is as follows:

The Snort IDS has several of requirements, thus, we thoroughly utilized the Advanced Packaging Tool that comes packaged in Ubuntu 12.04 LTS to install the following programs via the following apt-get install directive:

```
1 sudo apt-get install nmap, nbtscan, apache2, php5, php5-mysql, php5-gd,
2     libpcap0.8-dev, libpcre3-dev, g++, bison, flex, libpcap-ruby, make,
3     autoconf, libtool, mysql-server, libmysqlclient-dev
```

Next, ensuring maximum security, we applied all software and distribution updates as follows:

```
sudo apt-get install update
sudo apt-get install upgrade
```

There are two more requirements for Snort that are unfortunately not available through the Advanced Packaging Repository. They are the Data Acquisition API and libdnet. These are available directly via the snort.org website and libdnet google code repository, respectively, and we acquired and installed them as follows:

```
sudo wget http://www.snort.org/downloads/1806 -O daq.tar.gz
sudo tar xzvf daq.tar.gz
cd libdnet
sudo ./configure
sudo make
sudo make install
```

The above commands round out the Data Acquisition API installation. We moved on to installing libdnet as follows:

```
sudo wget http://libdnet.googlecode.com/files/libdnet-1.12.tgz -O libdnet.tar.gz
sudo tar xzvf libdnet.tar.gz
cd daq
sudo ./configure
sudo make
sudo make install
sudo ln -s /usr/local/lib/libdnet.1.0.1 /usr/lib/libdnet.1
```

Now all the required software components are installed and up-to-date, we moved on to installing Snort 2.9.3 directly!

```
sudo wget http://www.snort.org/downloads/1814 -O snort.tar.gz
sudo tar xzvf snort.tar.gz
cd snort
sudo ./configure --prefix=/usr/local/snort --enable- sourcefire
sudo make
sudo make install
sudo mkdir /var/log/snort
sudo mkdir /var/snort
```

```
sudo groupadd snort
sudo useradd -g snort snort
sudo chown snort:snort /var/log/snort
```

At this point, Snort needs a rule set. This is when Pulledpork comes to play. Pulledpork, being perl-based, has additional software requirements, which are also thankfully available via APT as follows:

```
apt-get install libcrypt-ssleay-perl liblwp-useragent-determined-perl -y
```

Pulledpork was retrieved and installed as follows into the Snort install folder:

```
cd /usr/local/snort
wget https://pulledpork.google.com/files/pulledpork-0.6.1.tar.gz
tar -xzvf pulledpork-0.6.1.tar.gz
cd pulledpork-0.6.1
mv pulledpork.pl /usr/local/bin/
```

From this point forward, there are several of fine-tuned configurations that we made. Specifically, we created an /etc/snort directory and changed into it, inserting a configuration file which we also have available on our web server as follows:

```
sudo mkdir /etc/snort
cd /etc/snort
wget http://scarletshield.rutgers.edu/snort.conf
chmod og-wr snort.conf
```

The last command above ensured maximum security by preventing unauthorized users from prying into or possibly editing your snort configuration. Next, we created a rules folder and populated it as follows:

```
sudo mkdir rules
wget http://scarletshield.rutgers.edu/snort.rules
```

A pulledpork configuration directory was downloaded and unpacked via the following configuration files:

```
sudo mkdir /etc/pulledpork
cd /etc/pulledpork/
sudo wget http://scarletshield.rutgers.edu/pulledpork.tar.gz
sudo tar xzvf pulledpork.tar.gz
```

Updates via our new Pulledpork configuration are asserted as follows:

```
pulledpork.pl -c /etc/pulledpork.conf
```

For our last software subsystem component, we installed barnyard2 as follows:

```
wget https://nodeload.github.com/firnsy/barnyard2/tarball/master
    -O barnyard2-2.10.tar.gz
sudo tar zxvf barnyard2-2.10.tar.gz
cd firnsy-barnyard2*
sudo autoreconf -fvi -I ./m4
sudo ./configure --with-mysql --with-mysql-libraries=/usr/lib/i386-linux-gnu
sudo make
sudo make install
sudo cp etc/barnyard2.conf /usr/local/snort/etc
sudo mkdir /var/log/barnyard2
sudo chmod 666 /var/log/barnyard2
sudo touch /var/log/snort/barnyard2.waldo
sudo chown snort.snort /var/log/snort/barnyard2.waldo
```

The last components of the installation are with regards to the database, which we generated as follows:

```
echo “create database snort;” | mysql -u root -p
mysql -u root -p -D snort < ./schemas/create\_mysql
echo “grant create,insert, select, delete, update on snort.* to
    snort@localhost identified by ‘OURSECRETPASSWORD’” | mysql -u root -p
```

Finally, we set up the network cards:

```
sudo vi /etc/network/interfaces
```

In the vi text editor, we added the following lines to add the mirrored, promiscuous interface that Snort would be utilizing:

```
auto eth1 iface eth1
inet manual
    up ifconfig \$IFACE 0.0.0.0 up
    up ip link set \$IFACE promisc on
    down ip link set \$IFACE promisc off
    down ifconfig \$IFACE down
```

This is the rigorous installation process we undertook for Scarletshield. Having installed the entire back-end, we can now continue on to the Administration Suite and Features.

Once Snort and the database being used are installed, the user must then acquire and/or update the Snort rule-set, which is essentially how Snort is able to parse and detect threats within the headers and contents of packets entering the machine. We chose "Pulled Pork", as it is the most reliable Snort rule management application, recommended by the creators of Snort themselves. Finally, the administrator must interface Snort with the MySQL database. As of Snort 2.9.3.0, the latest stable version available for Ubuntu 12.04 LTS, it is required to install the application "Barnyard2" for a Snort daemon to record information into the MySQL database created by a user.

Chapter 4

Administration Suite & Features

4.1 Web Front-end Integration

Scarletshield features a web frontend utilizing a simple but powerful Drupal Content Management System (CMS). In addition, several PHP scripts are utilized to help analyze and serve the Snort MySQL database.

Scarletshield has three powerful tools for a user that wishes to learn more about the offending attacks against their server. These three tools, the "Attack History", "Analytics" and "Heatmap" pages, all come comprise Scarletshield Suite. Each tool features a different way for the user to view the logs stored by Scarletshield. The link for the Scarletshield Suite is http://scarletshield.rutgers.edu/suite.[1]

When a user tries to access any page of the suite, a JavaScript function runs to see if the user has an active login cookie with the correct password (the password is "scarlet"). If the user does not have this cookie, they are redirected to the suite login page seen below.

The login page makes use of two JQuery packages. The first package, "Tubular", allows a streamed video to be the page background. The other package, "Toastr", makes use of JavaScript's non-blocking architecture. Toastr is responsible for all asynchronous pop-up alerts. These can be seen at page load, and when the access key form's "Submit" button is pressed. Upon entering the correct password, the login cookie is generated and the user is then brought to the "Attack History" page. When first accessing this page, a loading animation is shown, while the back-end queries data from the MySql database containing all Snort logs. As a large dataset is loaded from the server, a screen similar to the one shown below is displayed.

[1]Note that the Scarletshield Suite was written for the Chrome browser, and some small details will not work on other browsers.

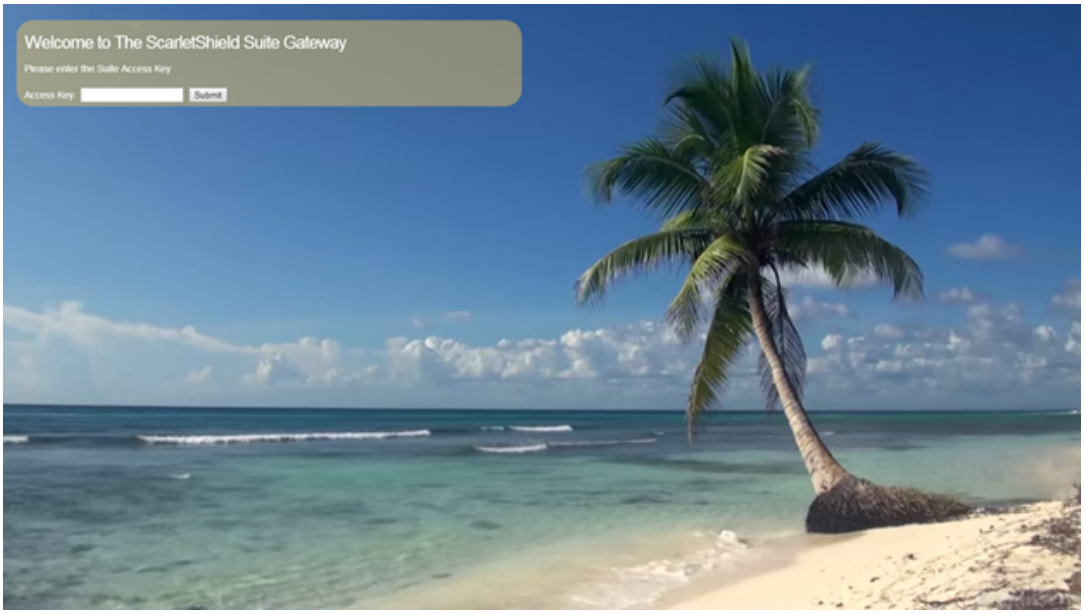

Figure 4.1: The Scarletshield Suite login page.

4.2 Analytics

Immediately upon visiting the page, the user is given access to all the data that Snort has logged; the data are logically and visually organized and to maximize presentability. (For comparison, an early implementation of this page can be seen here: `http://scarletshield.rutgers.edu/demo/last100.php`.) Each log is grouped by date and

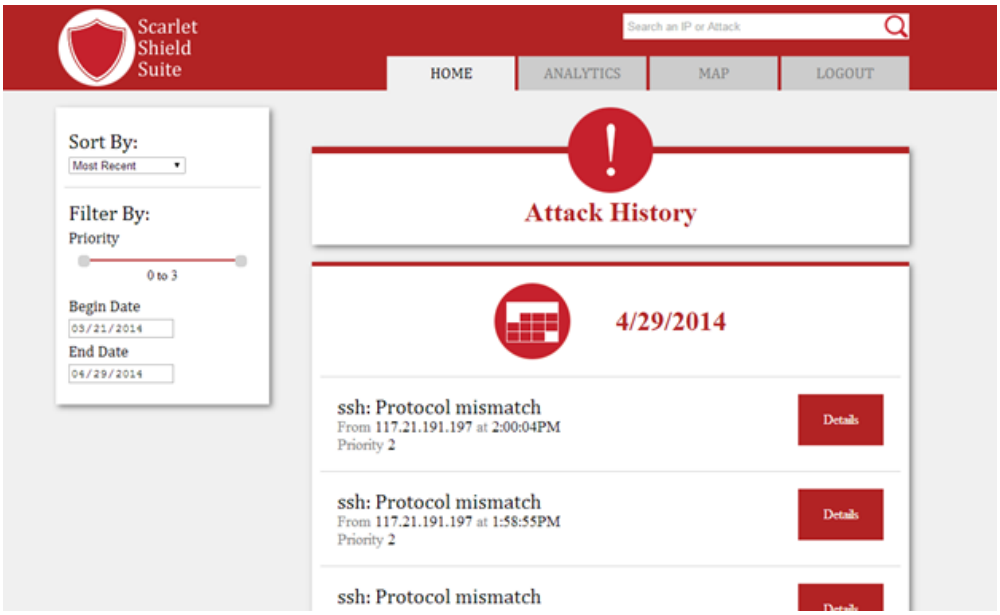

Figure 4.2: The Scarletshield Suite Attack History page.

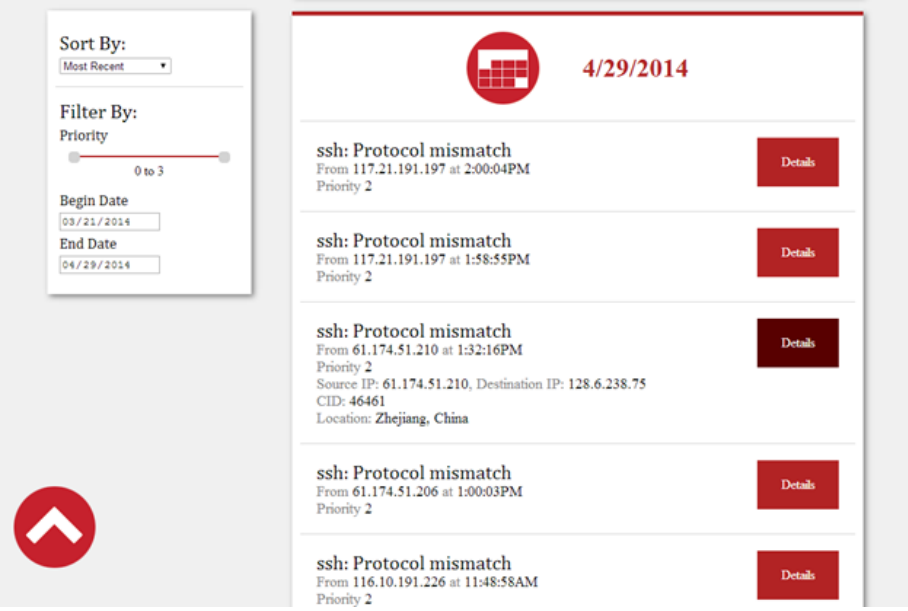

Figure 4.3: The Scarletshield Suite Attack History page, with an expanded description of the details of an attack originating from China.

contains information about the type of attack, the IP address it was originated from, the time it occurred and the priority of the attack (on a range from 0 to 3 with 0 being highest priority). If the user presses the "Details" button, extra information, such as the destination IP address (which may not be static if the server has more than one IP address), the ID number of the occurrence, and approximate location of the attack are expanded.

On the top left hand sides of the page, several tools allow the user advanced filtering capabilities. Logs can be sorted by date (oldest or newest) and priority (lowest first or highest first), and the search bar at the top can be used to only show information that either contains the IP address we search or show attacks that match the string that was searched. This is incredibly convenient because manually querying for signatures in MySQL (which involves several tedious SQL joins) would be a time-consuming task for an administrator, and by having it presented via an admin-accessible private frontend, not only could administrators react quicker, but even less-skilled users could discover and report an ongoing attack.

The Analytics page has three sections, as noted on the sidebar: the "Time Analysis", "Common IP's", and "Common Attacks". The Time Analysis section is simply a chart that shows a chart with the number of attacks that the server has undergone each day. This chart has a slider to focus on specific time intervals, and as one can see from the entire view, this is necessary because one single day may be so active that it dwarfs other days' activity.

The last page in the Scarletshield Suite is the Heatmap page. Essentially, Scarlet Shield processes IP addresses from all logged attack patterns and places it on a Google Map heatmap, so that the user may be able to geographically backtrace signatures. In certain configurations, the heatmap would be useful to backtrace potential ongoing threats and

attacks, such as DoS attacks within certain windows. This would also help administrators make informed decisions for creating temporary iptables rules for banning certain offending subnet servers for the duration of an ongoing threat. An example of the heatmap is shown below.

4.3 Response Options

The final portion enabling Scarletshield to cover the full spectrum of network defense is enabling the network administrator to utilize the frontend for analysis and decision-making options. Essentially, the an administrator should be able to take the Snort records, make an analysis, and decide the action (whether to throttle or drop all packets altogether from a suspicious IP address) to take to ensure network security.

Due to time constraints, this feature was not integrated into the administrator suite, although the capability is already inherently present because of the inclusion of fail2ban. Future work could easily incorporate the addition of such an administrator response suite.

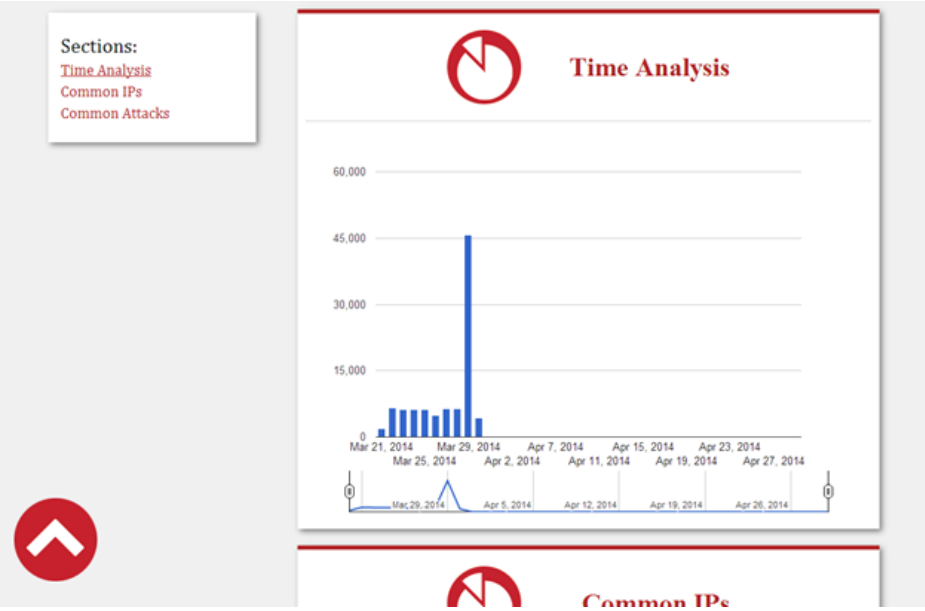

Figure 4.4: A chart of the rate of attacks over a certain interval. Users can use the slider at the bottom of the chart to analyze attacks over arbitrary intervals.

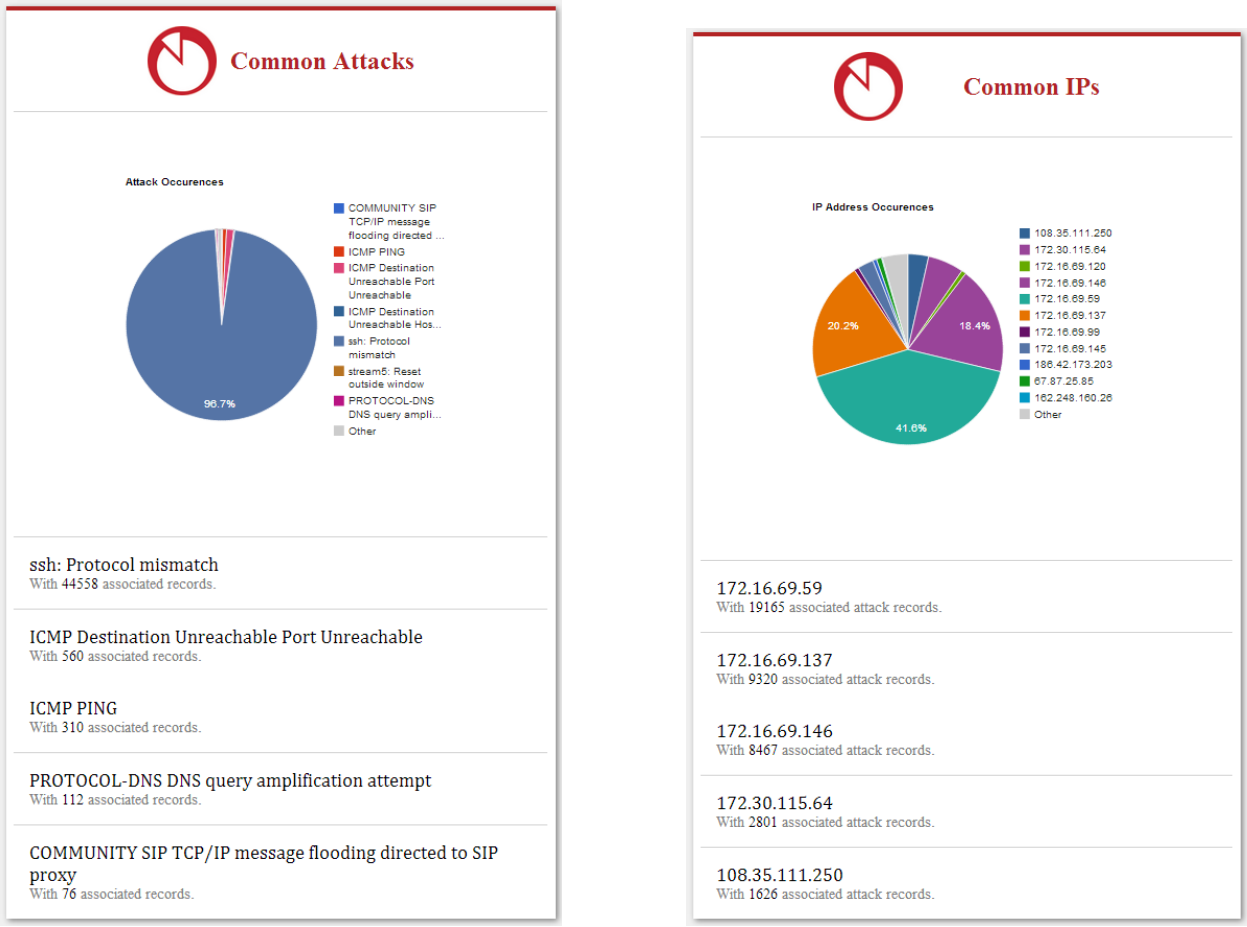

Figure 4.5: Pie charts featuring prominent attack varieties and IP addresses.

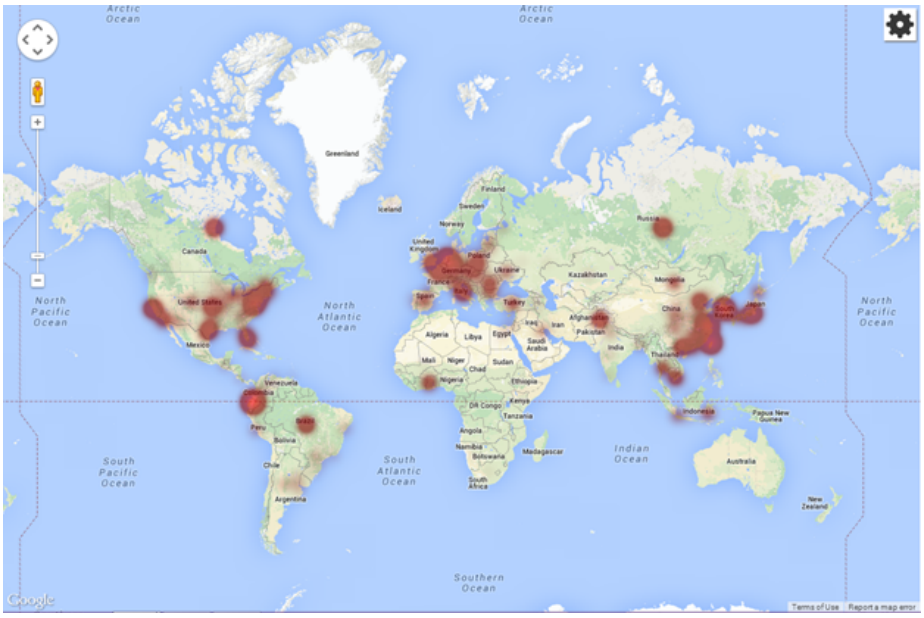

Figure 4.6: A geographic world map with red intensity markers indicating the presence of attack vectors from a geographic region, identified by IP address.

Chapter 5

Conclusion

5.1 Cost/Sustainability Analysis and Scalability

In spite of existing in an aged Microway rackmount server, the primary component necessary for utilizing Scarletshield on a network gateway is the Network Card. While a dedicated rackmount server is a costly investment, much of the hardware that Scarletshield actually runs off of is dated and can be purchased for very cheap prices off the shelf. One of its most important elements, a NetXtreme BCM5721 Gigabit Ethernet PCI Express card, actually goes for $15-$70 (depending on where you look, and on whether it is used or refurbished).

Scarletshield in its current manifestation operates off of a Microway rackmount server. An equivalent, but modern-day, rackmount server would range between $2,000-$5,000, which would be a large investment. A modern rackmount server running an operating system and network security suite comparable to Scarletshield would be able to scale to meet the demands of thousands and possibly even millions of clients and internal gateway server connections. To this end, however, the network cards would need to be far more powerful, possibly in the territory of 10Gbps+. Network cards that powerful could be hundreds to thousands of dollars or additional cost, but would only be necessary for the case of an ISP or server farm scale of operations.

It is important to note that Scarletshield does not require dedicated hardware to the aforementioned extent, however. Scarletshield could be installed into refurbished desktops with an extra network card and still be serviceable; this would actually be a very useful and cheap method of creating a homemade network security suite to use for a small business or home. An equivalent desktop computer with the hardware specifications that our dated Microway rackmount server brings forth would likely only be a few hundred dollars.

All the software utilized is free and, for the most part, open source. Thus, this is what makes the Scarletshield approach so desirable. As opposed to spending hundreds to thousands of dollars on a hardware-based intrusion detection and prevention system, a savvy

network or system administrator would be able to implement their own and be able to extend their server for functions other than dedicated intrusion detection or intrusion prevention. Scarletshield, for example, hosts its own web server from inside the server.

What makes Scarletshield so cost-effective is that the server is not necessarily restricted to the network security suite. You would be able to host additional web sites and databases on your gateway and maximize utility. The tradeoff would be that, being tied to the same server as the IDS/IPS, downtime may be somewhat higher due to the maintenance requirements of a network security system warranting updates and upgrades often to remain effective. Nevertheless, reboots and other functions of maintenance actually requiring downtime is not exclusive to the network security system alone; thus, a network or system administrator would not experience grief exclusive to integrating the IDS/IPS into a production server.

In summary, while Scarletshield in its current state can handle traffic from the scale of hundreds to thousands of connections, the core software architecture can actually be scaled down for a small business or home application by being installed in cheaper hardware and scaled up to handle thousands to millions of connections by being installed in enterprise-grade hardware. Utility with Scarletshield is maximized with pitted against a hardware IDS/IPS in that Scarletshield can double as a production server. Overall, that makes Scarletshield both cost-effective, sustainable, and scalable in the long-term!

5.2 Future Work

Although Scarletshield is already operational on a server within Rutgers Engineering, additional work is necessary to integrate it into private subnets and to give it world-wide-web scale. More robustness and failover capabilities are needed in order to communicate certain threats that may be large-scale, and which would otherwise disrupt significant cybersecurity infrastructure. This also requires fine tuning Scarletshield's frontend capabilities and adding more analysis tools and automation techniques to streamline the threat-analysis and threat-prevention workflow presented to the network administrator. Further integration of the subcomponents of Scarletshield will solidify its versatility, relevance, and overall strength. By successfully implementing a fully-featured frontend for monitoring and reacting to threats, we will maximize the potential of Scarletshield to be deployable to both private switches and enterprise-level networks.

We created a lightweight system of integrated network security via a front-end suite presenting the capabilities of both the Snort IDS and the fail2ban IPS in a way that maximizes automation and ease-of-use for the network and system administrator. Nevertheless, there are some features that could be added to make Scarletshield even more robust. Specifically, a distributed version of the threat analysis database and iptables could have been developed such that other networks beyond our own can share information on threat sources and types of attacks, being a more consolidated solution for types of threats such as DDoS. The scale of that type of feature, however, could warrant its own design project entirely! With regards

to the work we accomplished over the semester, there is great significance in the proof-of-concept Scarletshield represents. As opposed to utilizing a completely security-based Linux distribution such as how Security Onion presents itself, Scarletshield presents a subsystem of capabilities that can be appended to an existing gateway or DHCP server, such as how Scarletshield is going to be introduced to our wireless access point gateways. That alone speaks to success and confidence built in our final Capstone Design product.

5.3 Summary and Concluding Remarks

During this semester, we achieved the following goals:

1. Build from scratch a specialized Ubutnu 12.04 LTS Server Edition-based network security system integrating the Snort IDS and fail2ban IPS to interface with its kernel firewall via iptables to automatically log and potential threats and drop all packets from packet sources confirmed to be malicious.
2. Integrate the IDS and IPS into a specialized front-end intended to maximize the analytical options available to system and network administrators enabling them to make quick and well-informed decisions on their network security policies flexibly depending on the type of traffic that Snort has been capturing.

Our work in integrating the Snort IDS, fail2ban IPS, and iptables within the `http://scarletshield.rutgers.edu` web server has culminated in the final development and release of a robust frontend, the Scarletshield Suite, which is available via `http://scarletshield.rutgers.edu/suite` using the password "scarlet". Our overall framework presents a great proof-of-concept for an open-source solution for network security that can be applicable to small-time enterprises and individuals who would not prefer to invest in a proprietary or commercial IDS or IPS.

After a little over a month of being online, Scarletshield has caught over 100,000 suspicious packets suspected of attempting penetration or exploitation of our server. (Exemplifying how effective our work in Scarletshield has been this semester, Rutgers Engineering Computing Services (ECS) has tasked system administrator Val A. Red with replicating the Scarletshield Suite for internal use to monitor wireless access point gateways.) Scarletshield is a comprehensive proof-of-concept network security solution suite that has been proven to capture and eliminate potential threats from around the globe 24/7!

5.4 Contribution Breakdown

Scarletshield, an ambitious endeavor to present a large-scale network solution suite, was conceived and developed by a group of five undergraduate senior Electrical and Computer Engineering Students of Rutgers University's School of Engineering. They are Jeff Adler, Eric Cuiffo, Parth Desai, Jeff Rabinowitz, and Val Red. Due to the grand scale and major objectives of the projects, heavy contributions were made by each and every member. Thus, it can be estimated that each member had a specialized role that actually created a nice, even split of fifths such that without the fifth contributed by each member, the whole of Scarletshield would have failed. The contributions are broken down as follows, in the same alphabetical order:

Jeff Adler – Committed work on the extra-secure methodologies employed in Scarletshield, such as the two-point Google Authentication implemented for our Secure Shell (SSH) access, the Site Access Key authentication greeter for the administrative Scarletshield Suite.

Eric Cuiffo – Led the effort for an expansive, robust front-end for the Scarletshield based on javascript and jQuery. The suite scripts he predominantly led the effort on, also to be presented in the Appendix section, were instrumental for the front-end presentation of the Scarletshield Suite. He also had the greatest contribution to the Scarletshield Heat Map, which takes all potential threat connection sources and actually interpolates geolocation data into a map of the world!

Parth Desai – His experience with PHP and MySQL enabled our early, prototype work on the Scarletshield Suite backend to run smoothly. Specifically drafting how we would translate the database to our web frontend, Parth Desai was instrumental in the integration of the backend database and the frontend suite via queries.

Jeff Rabinowitz – Resident expert on relational databases and LaTeX, Jeff Rabinowitz contributed insight that ensured that not only our queries were as efficient as possible, but that this report was optimally presentable. His ability to work with databases specifically ensured that the Scarletshield Suite was a success in enabling real-time analysis of the data parsed by Snort.

Val A. Red – A system administrator by trade, Val Red implemented the installation phase of the backend, ensuring Snort and fail2ban worked seamlessly and did not break the Ubuntu 12.04 LTS installation or Scarletshield's Microway Server altogether. Specifically, his rigorous configurations (.conf files will also be in the appendix) and fine-tuning maximized performance and allowed the Scarletshield Suite to work seamlessly by ensuring no errors or failures would occur in the extensive back-end, maximizing speed and efficiency between Snort, fail2ban, and the entire Scarletshield system.

All group members made great, synergistic contributions such that the whole of Scarletshield summed together amounted to a design far greater than its parts. In addition, there was a lot of overlap and cross-specializations such that in addition to the aforementioned

specialized contributions, practically every part of the final product was a result of all team members working together. We are proud of our work on Scarletshield, and have learned a great deal about network security and systems programming overall in building it!

Bibliography

[1] United States National Security Agency. (2012). Defense in Depth: A practical strategy for achieving Information Assurance in today's highly networked environments, [Online]. Available: http://www.nsa.gov/ia/_files/%20support/defenseindepth.pdf.

[2] J. Riden. (2008). How fast-flux service networks work, [Online]. Available: http://www.honeynet.org/node/132.

[3] A. Sternstein. (2013). Pentagon planned to secure web domains before syrians hijacked marine corps site, [Online]. Available: http://www.nextgov.com/cybersecurity/2013/09/pentagon-planned-secure-web-domains-syrians-hijacked-marine-corps-site/69830/.

[4] D. Gullett. (Jul. 22, 2012). Snort 2.9.3 and Snort Report 1.3.3 on Ubuntu 12.04 LTS Installation Guide, [Online]. Available: http://www.snort.org/assets/158/snortinstallguide293.pdf.

Appendix A

Source Code Highlights

A.1 IPLocater.js

```
/*
 * The IPLocator class. Does exactly what it sounds like: translates
 * the IP addresses which are passed into the locate function into
 * an array of Objects containting google LatLng objects and the
 * corresponding weight they have on the map.
 *
 * Arguments:
 *   Nothing.
 *
 * Returns:
 *   <Object> - this. The context of the IPLocator so that a function
 *   that creates an IPLocator object may call the locate function.
 */
var IPLocator = function() {
  this.goalWeight = 0;
  this.totalWeight = 0;
  this.goalRequests = 0;
  this.completeRequests = 0;
  this.heatSpots = [];
  this.container = document.getElementById('map-canvas');
  this.callback = null;

  return this;
}

```

```
/*
 * Takes care of creating JSON requests for the IP address locations
 * which in turn will trigger the map being drawn later on.
 *
 * Arguments:
 *   IPs <Array> - Array of IP addresses which we will search for
 *   the location of.
 *   callback <Function> - The function we will call upon successful
 *   translation.
 *
 * Returns:
 *   Nothing.
 */
IPLocator.prototype.locate = function(IPs, callback) {

  // The site we use to get information about IP addresses.
  var geoURL = 'http://162.248.161.124:8080/json/';

  // An array we will keep the locations we wish to highlight
  // in the form of google LatLng objects.

  var hash = {};

  for (var i = 0; i < IPs.length; ++i) {
    var cur = IPs[i];
    if (!hash[cur]) {
      hash[cur] = 1;
    } else {
      ++hash[cur];
    }
  }

  // Save some stuff for later.
  this.callback = callback;
  this.goalWeight = IPs.length;
  this.goalRequests = Object.keys(hash).length;

  var cur;
  for (cur in hash) {
    var weight = hash[cur];
    $.getJSON(geoURL + cur,
      IPLocator.successHandler(weight, this)
    ).fail(IPLocator.failHandler(weight, this));
  }

```

```
}


/*
 * Static function which is called when a request is responded to.
 */
IPLocator.successHandler = function(weight, iptrans) {
  return function(data) {
    iptrans.handler(weight, data);
  }
}


/*
 * Static function which is called when a request fails.
 */
IPLocator.failHandler = function(weight, iptrans) {
  return function() {
    iptrans.error(weight);
  }
}


/*
 * Function that checks if the coordinates we are currently seeing
 * are part of a location we wish to ignore: where we live.
 */
IPLocator.prototype.isUs = function(data) {
  return (Math.floor(Number(data.latitude)) === 40 &&
          Math.floor(Number(data.longitude)) === -75)
}


/*
 * A function that handles the returned request asking for
 * the coordinates of an IP address. Simply adds the coordinate
 * value as many times as the IP address is needed.
 *
 * Arguments:
 *   weight <Number> - The number of times this IP address appeared
 *   in the list passed in.
 *   data <Object> - The data that was returned by the request.
 *
 * Returns:
 *   Nothing.
```

```
 */
IPLocator.prototype.handler = function(weight, data) {
  if (!(data.latitude === 0 && data.longitude === 0 ) && !this.isUs(data))  {

    // When we get it, we push the google LatLng object of the
    // location to the heatSpots array.
    this.heatSpots.push({
      location: new google.maps.LatLng(data.latitude, data.longitude),
      weight: weight
    });
  }

  // Increment counter for how far we have progressed in our reqs.
  this.totalWeight += weight;
  this.completeRequests += 1;


  var overlay = document.getElementById('progress');
  //overlay.setAttribute('aria-valuenow', this.completeRequests/this.goalRequests*
  overlay.style.width = this.completeRequests/this.goalRequests*100+'%'
  //$('#overlay').html(
   //   Math.floor(
   //     this.completeRequests/this.goalRequests * 100
    //  ).toString());

  if (this.totalWeight === this.goalWeight) {
    this.callback(this.heatSpots);
  }
}


/*
 * A function that is called when there is an error on the request.
 * Simply increments the counter of the requests returned and also
 * checks if this was the last item needed to be returned.
 */
IPLocator.prototype.error = function(weight) {
  this.totalWeight += weight;
  this.completeRequests += 1;
  if (this.totalWeight === this.goalWeight) {
    this.callback(this.heatSpots);
  }
}
```

A.2 HistoryGenerator.js

```
var HistoryGenerator = function(data) {
  this.data = data;
  this.cur =  data.length - 1;
  this.container = document.getElementById('dataArea');
  this.lastDate = new Date(0);
  this.lastCard = null;
  this.sorting = 'recent';
  this.lowPriority = 3;
  this.highPriority = 0;
  var startdate = document.getElementById('startdate');
  var enddate = document.getElementById('enddate');
  var sdateVal = HistoryGenerator.getStartDate(startdate.value);
  var edateVal = HistoryGenerator.getEndDate(enddate.value);
  this.startDate = sdateVal.valueOf();
  this.endDate = edateVal.valueOf();
  this.search = '';

  this.container.innerHTML = '';


  return this;
}

HistoryGenerator.prototype.next20 = function() {

  // Get some common variables to use a lot.
  var data = this.data;
  var lastDate = this.lastDate;
  var lastCard = this.lastCard;

  // Find where we left off.
  var i = this.cur;

  // We only want to do 20 at a time so keep a counter.
  var numDone = 0;
  while (numDone < 20 && i >= 0) {

    // Get the current object.
    var cur = data[i];

    if (this.search !== '') {
```

```
      var message = cur.msg.toLowerCase();
      var src = cur.src;
      var dst = cur.dst;
      var search = this.search.toLowerCase();
      console.log();
      if (message.indexOf(search) === -1 &&
          src.indexOf(search) === -1 &&
          dst.indexOf(search) === -1) {
        --i;
        continue;
      }
    }
    // If we happen to be filtering out this priority number, skip
    // this whole block.
    var priority = cur.priority;
    if (priority < this.highPriority || priority > this.lowPriority) {
      --i;
      continue;
    }

    // Get the date from it and create a Date object from it.
    var splitDate = cur.datetime.split(/[\s:-]/);
    var year = Number(splitDate[0]);
    var month = Number(splitDate[1]);
    var day = Number(splitDate[2]);
    var hour = Number(splitDate[3]);
    var minutes = Number(splitDate[4]);
    var seconds = Number(splitDate[5]);
    var curDate = new Date(year, month-1, day, hour, minutes, seconds, 0);

    if (curDate.valueOf() > this.endDate ||
        curDate.valueOf() < this.startDate) {
      --i;
      continue;
    }

    // If this is the same date as the last one, we want to continue on the same
    // card. If not, we want to make a new one.
    if (lastDate.toDateString() === curDate.toDateString()) {
      this.appendEntry(cur, lastCard, hour, minutes, seconds);
    } else {
      var card = this.appendCard(cur, year, month, day, hour, minutes, seconds);

      // Since this is a new card, we have to replace the old variables which no
      // longer matter.
```

```
87        lastCard = card;
88        lastDate = curDate;
89      }
90      --i;
91      ++numDone;
92    }
93
94    if (i < 0 && lastCard === null) {
95      this.showError();
96    }
97
98    // Save these variables for after we scroll down.
99    this.cur = i;
100   this.lastDate = lastDate;
101   this.lastCard = lastCard;
102 }
103
104
105 HistoryGenerator.prototype.showError = function() {
106   this.container.innerHTML = "<div class='card shadow'><div class='redline'></div>
107 }
108
109
110 HistoryGenerator.prototype.appendCard = function(
111     cur, year, month, day, hour, minutes, seconds) {
112   var card = document.createElement('div');
113   card.classList.add('card', 'shadow');
114   this.container.appendChild(card);
115
116   var redline = document.createElement('div');
117   redline.className = 'redline';
118   card.appendChild(redline);
119
120   var innerCard = document.createElement('div');
121   innerCard.className = 'inner-card';
122   card.appendChild(innerCard);
123
124   var iconandtext = document.createElement('div');
125   iconandtext.className = 'iconandtext';
126   innerCard.appendChild(iconandtext);
127
128   var iticon = document.createElement('img');
129   iticon.className = 'iticon';
130   iticon.src = 'images/cal.png';
131   iticon.alt = 'Calendar';
```

```
  iconandtext.appendChild(iticon);

  var ittext = document.createElement('h1');
  ittext.className = 'ittext';
  ittext.innerHTML = month + '/' + day + '/' + year;
  iconandtext.appendChild(ittext);

  this.appendEntry(cur, innerCard, hour, minutes, seconds);


  return innerCard;
}

HistoryGenerator.prototype.appendEntry = function(
    cur, lastCard, hour, minutes, seconds) {
  var innerCard = lastCard;

  var line = document.createElement('div');
  line.classList.add('onepixline', 'clear');
  innerCard.appendChild(line);

  var entry = document.createElement('div');
  entry.className = 'entry';
  innerCard.appendChild(entry);

  var det = document.createElement('div');
  det.className = 'detButton';
  det.innerHTML = 'Details';
  entry.appendChild(det);


  var title = document.createElement('h2');
  title.className = 'entryTitle';
  title.innerHTML = cur.msg;
  entry.appendChild(title);

  var info = document.createElement('p');
  info.className = 'entryDes';
  var m = 'AM';
  if (hour === 0) {
    hour = '12';
  } else if (hour > 12) {
    hour = hour - 12;
    m = 'PM'
  }
```

```
177   if (minutes < 10) {
178     minutes = '0' + minutes;
179   }
180   if (seconds < 10) {
181     seconds = '0' + seconds;
182   }
183   info.innerHTML = '<span>From</span> ' + cur.src +
184     ' <span>at</span> ' + hour + ':' + minutes + ':' + seconds + m;
185   entry.appendChild(info);
186
187   var prio = document.createElement('p');
188   prio.className = 'entryDes';
189   prio.innerHTML = '<span>Priority</span> ' + cur.priority;
190   entry.appendChild(prio);
191
192   var popout = document.createElement('div');
193   popout.style.display = 'none';
194   entry.appendChild(popout);
195
196   var srcdst = document.createElement('p');
197   srcdst.className = 'entryDes';
198   srcdst.innerHTML = '<span>Source IP: </span>' + cur.src +
199     '<span>, Destination IP: </span>' + cur.dst;
200   popout.appendChild(srcdst);
201
202   var cid = document.createElement('p');
203   cid.className = 'entryDes';
204   cid.innerHTML = '<span>CID: </span>' + cur.cid;
205   popout.appendChild(cid);
206
207   var loc = document.createElement('p');
208   loc.className = 'entryDes';
209   popout.appendChild(loc);
210
211
212   var expanded = false;
213   var foundLocation = false;
214   det.onclick = function() {
215     if (expanded === false ) {
216       if (foundLocation === false) {
217         var geoURL = 'http://162.248.161.124:8080/json/';
218
219         get(geoURL + cur.src, function() {
220           var response = JSON.parse(this.responseText);
221           var string = '';
```

```
222           if (response.region_name !== '') {
223             string += response.region_name + ', ';
224           }
225           if (response.country_name !== '') {
226             string += response.country_name;
227           }
228           if (string === '') {
229             string = 'Location unknown.';
230           } else {
231             string = '<span>Location: </span>' + string;
232           }
233            loc.innerHTML = string;
234         });
235
236         foundLocation = true;
237       }
238       popout.style.display = 'block';
239       expanded = true;
240     } else {
241       popout.style.display = 'none';
242       expanded = false;
243     }
244   }
245
246 }
247
248 HistoryGenerator.prototype.registerScroll = function() {
249   var content = document.getElementById('content');
250   var bottom = document.createElement('div');
251   bottom.className = 'bottom-space';
252   content.appendChild(bottom);
253
254   var pageupShown = false;
255   var hg = this;
256   window.onscroll = function(e) {
257     e.preventDefault();
258     var winHeight = window.innerHeight;
259     var divLocation =
260       document.getElementsByClassName('bottom-space')[0].getBoundingClientRect().t
261
262     if (divLocation < winHeight) {
263       hg.next20();
264     }
265
266     var side = document.getElementById('sidebar');
```

```
    if (window.scrollY > 100) {
      if (pageupShown === false) {
        var pageup = document.getElementById('pageup');
        moveToX(pageup, 20, 175);
        pageupShown = true;
      }
      var side = document.getElementById('sidebar');
      side.style.position = 'fixed';
      side.style.marginTop ='-100px';
    } else {
      if (pageupShown === true) {
        var pageup = document.getElementById('pageup');
        moveToX(pageup, -150, 175);
        pageupShown = false;
      }
      side.style.marginTop ='0px';
      side.style.position = 'static';
    }
    return false;
  };
}

HistoryGenerator.prototype.sortBy = function(type) {
  if (type === 'recent') {
    if (this.sorting === 'oldest') {
      this.data.reverse();
    } else {
      this.data.sort(function(a, b) {
        return a.id - b.id;
      });
    }
  } else if (type === 'oldest') {
    if (this.sorting === 'recent') {
      this.data.reverse();
    } else {
      this.data.sort(function(a, b) {
        return b.id - a.id;
      });
    }
  } else if (type === 'pinc') {
    this.data.sort(function(a, b) {
      var retval = a.priority - b.priority;
      if (retval === 0) {
        return a.id - b.id;
      }
```

```
      return retval
    });
  } else if (type === 'pdec') {
    this.data.sort(function(a, b) {
      var retval = b.priority - a.priority;
      if (retval === 0) {
        return a.id - b.id;
      }
      return retval
    });
  }
  this.refreshUs();
  this.sorting = type;
  console.log(this);
}

HistoryGenerator.prototype.refreshUs = function() {
  window.scrollTo(0,0);
  this.cur = this.data.length - 1;
  this.container.innerHTML = '';
  this.lastDate = new Date(0);
  this.lastCard = null;
  this.next20();

}

HistoryGenerator.prototype.registerSelector = function() {
  var selector = document.getElementById('selector');
  var hg = this;
  selector.onchange = function() {
    hg.sortBy(selector.value);
  }
}

HistoryGenerator.prototype.registerDates = function() {
  var startdate = document.getElementById('startdate');
  var enddate = document.getElementById('enddate');
  var hg = this;
  startdate.onblur = function() {
    var date = HistoryGenerator.getStartDate(startdate.value);
    hg.startDate = date.valueOf();
    hg.refreshUs();
  }
  enddate.onblur = function() {
    var date = HistoryGenerator.getEndDate(enddate.value);
```

```
357     hg.endDate = date.valueOf();
358     hg.refreshUs();
359   }
360
361 }
362
363 HistoryGenerator.prototype.registerSearch = function() {
364   var bar = document.getElementById('searchbox');
365   var button = document.getElementById('searchsubmit');
366   var hg = this;
367   button.onclick = function() {
368     var value = bar.value;
369     hg.search = value;
370     hg.refreshUs();
371   }
372   bar.onkeypress = function(e) {
373     if (e.keyCode === 13) {
374       var value = bar.value;
375       hg.search = value;
376       hg.refreshUs();
377     }
378   }
379
380 }
381
382 HistoryGenerator.getStartDate = function(value) {
383     var tokenized = value.split('-');
384     var year = Number(tokenized[0]);
385     var month = Number(tokenized[1]);
386     var day = Number(tokenized[2]);
387     var date = new Date(year, month-1, day, 0, 0, 0, 0);
388     return date;
389
390 }
391 HistoryGenerator.getEndDate = function(value) {
392     var tokenized = value.split('-');
393     var year = Number(tokenized[0]);
394     var month = Number(tokenized[1]);
395     var day = Number(tokenized[2]);
396     var date = new Date(year, month-1, day, 23, 59, 59, 999);
397     return date;
398 }
399
400 HistoryGenerator.prototype.setHighPriority = function(value) {
401   this.highPriority = value;
```

```
  this.refreshUs();
}
HistoryGenerator.prototype.setLowPriority = function(value) {
  this.lowPriority = value;
  this.refreshUs();
}
```

A.3 Analytics.js

```
/*
 * The Analytics class. Starts up our analytics in the div with id equal
 * to dataArea. Use Analytics.init() to kick off the process.
 */
var Analytics = function(data) {
  this.data = data;
  this.container = document.getElementById('dataArea');

  return this;
}


/*
 * Initializes all of the boxes in the Analytics class.
 */
Analytics.prototype.init = function() {

  // Clear the area for our data.
  this.container.innerHTML = '';

  // Create the time thing
  var tc = this.createTimeChart();

  // Create common IP thing
  var ip = this.createCommonIPs();

  // Create common attacks thing
  var att = this.createCommonAttacks();

  // Hotlinks to divs.
  this.hotlinks(tc, ip, att);
}
```

```
33
34
35 /*
36  * Creates the first box, the time chart, for the Analytics class.
37  * Uses Google Charts Chart Control on a Column Graph to map out
38  * all the attacks according to the day they were created.
39  */
40 Analytics.prototype.createTimeChart = function() {
41   var card = document.createElement('div');
42   card.classList.add('card', 'shadow');
43   this.container.appendChild(card);
44
45   var innerCard = this.initCard(card, 'Time Analysis');
46
47   var line = document.createElement('div');
48   line.classList.add('onepixline', 'clear');
49   innerCard.appendChild(line);
50
51   var entry = document.createElement('div');
52   entry.className = 'entry';
53   innerCard.appendChild(entry);
54
55   var dash = document.createElement('div');
56   dash.id = 'dashboard';
57   entry.appendChild(dash);
58
59   var cha = document.createElement('div');
60   cha.id = 'chart';
61   cha.style.width = '100%';
62   cha.style.height = '400px';
63   dash.appendChild(cha);
64
65   var con = document.createElement('div');
66   con.id = 'control';
67   con.style.width = '100%';
68   con.style.height = '50px';
69   dash.appendChild(con);
70
71   var dashboard = new google.visualization.Dashboard(dash);
72
73   var control = new google.visualization.ControlWrapper({
74     'controlType': 'ChartRangeFilter',
75     'containerId': 'control',
76     'options': {
77       'filterColumnIndex': 0,
```

```
78        'ui': {
79          'chartType': 'LineChart',
80          'chartOptions': {
81            'chartArea': {'width': '90%'},
82            'hAxis': {'baselineColor': 'none'}
83          },
84          'minRangeSize': 86400000
85        }
86      },
87    });
88
89    var chart = new google.visualization.ChartWrapper({
90      'chartType': 'ColumnChart',
91      'containerId': 'chart',
92      'options': {
93        'chartArea': {'height': '80%', 'width': '80%'},
94        'hAxis': {'slantedText': false},
95        'legend': {'position': 'none'}
96      },
97      'view': {
98        'columns': [
99          {
100           'calc': function(dataTable, rowIndex) {
101             return dataTable.getFormattedValue(rowIndex, 0);
102           },
103           'type': 'string'
104         }, 1]
105     }
106   });
107
108   var data = new google.visualization.DataTable();
109   data.addColumn('date', 'Date');
110   data.addColumn('number', 'Number of Attacks');
111
112   var hash = {};
113   var i;
114   for (i = 0; i < this.data.length; ++i) {
115     var cur = this.data[i];
116     var splitDate = cur.datetime.split(/[\s:-]/);
117     var year = Number(splitDate[0]);
118     var month = Number(splitDate[1]);
119     var day = Number(splitDate[2]);
120     var curDate = year + ' ' + month + ' ' + day;
121     if (hash[curDate]) {
122       hash[curDate]++;
```

```
123     } else {
124       hash[curDate] = 1;
125     }
126   }
127
128   var objs;
129   for (objs in hash) {
130     var date = new Date(objs);
131     data.addRow([date, hash[objs]]);
132   }
133
134   dashboard.bind(control, chart);
135   dashboard.draw(data);
136
137   return card;
138 }
139
140
141 /*
142  * Shows a pie chart all the IP addresses weighing how
143  * often they show up. After this, we add on the five most
144  * common ones along with some details about them.
145  */
146 Analytics.prototype.createCommonIPs = function() {
147
148   var card = document.createElement('div');
149   card.classList.add('card', 'shadow');
150   this.container.appendChild(card);
151
152   var innerCard = this.initCard(card, 'Common IPs');
153
154   var hash = {};
155   var i;
156   for (i = 0; i < this.data.length; ++i)  {
157     var cur = this.data[i];
158     if (cur.src.indexOf('128.6.238') !==  -1) {
159       continue;
160     }
161     if (hash[cur.src]) {
162       hash[cur.src]++;
163     } else {
164       hash[cur.src] = 1;
165     }
166   }
167
```

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'IP Address');
data.addColumn('number', 'Number of Occurrences');

for (i in hash) {
  if (hash[i]) {
    data.addRow([i, hash[i]]);
  }
}
var options = {
  title: 'IP Address Occurences'
};
var divi = document.createElement('div');
divi.classList.add('onepixline', 'clear');
innerCard.appendChild(divi);

var pie = document.createElement('div');
pie.className = 'entry';
pie.style.height = '500px';
innerCard.appendChild(pie);

var chart = new google.visualization.PieChart(pie);
chart.draw(data, options);

for (i = 0; i < 5; ++i) {
  var j, max = 0, maxIndex = -1;
  for (j in hash) {
    if (hash[j] > max) {
      max = hash[j];
      maxIndex = j;
    }
  }

  var line = document.createElement('div');
  line.classList.add('onepixline', 'clear');
  innerCard.appendChild(line);

  var entry = document.createElement('div');
  entry.className = 'entry';
  innerCard.appendChild(entry);

  var title = document.createElement('h2');
  title.className = 'entryTitle';
  title.innerHTML = maxIndex;
  entry.appendChild(title);
```

```
213
214     var info = document.createElement('p');
215     info.className = 'entryDes';
216     info.innerHTML = '<span>With</span> ' + max + ' <span>associated attack record
217     entry.appendChild(info);
218
219     hash[maxIndex] = undefined;
220   }
221
222   return card;
223 }
224
225
226 /*
227  * Creates a way to see the most common attacks. Shows a
228  * pie chart of all values as well as listing the top
229  * five underneath with some details.
230  */
231 Analytics.prototype.createCommonAttacks = function() {
232   var card = document.createElement('div');
233   card.classList.add('card', 'shadow');
234   this.container.appendChild(card);
235
236   var innerCard = this.initCard(card, 'Common Attacks');
237
238   var hash = {};
239   var i;
240   for (i = 0; i < this.data.length; ++i)  {
241     var cur = this.data[i];
242     if (cur.src.indexOf('128.6.238') !==  -1) {
243       continue;
244     }
245     if (hash[cur.msg]) {
246       hash[cur.msg]++;
247     } else {
248       hash[cur.msg] = 1;
249     }
250   }
251
252
253   var data = new google.visualization.DataTable();
254   data.addColumn('string', 'Name of Attack');
255   data.addColumn('number', 'Number of Occurrences');
256
257   for (i in hash) {
```

```
    if (hash[i]) {
      data.addRow([i, hash[i]]);
    }
  }
  var options = {
    title: 'Attack Occurences'
  };
  var divi = document.createElement('div');
  divi.classList.add('onepixline', 'clear');
  innerCard.appendChild(divi);

  var pie = document.createElement('div');
  pie.className = 'entry';
  pie.style.height = '500px';
  innerCard.appendChild(pie);

  var chart = new google.visualization.PieChart(pie);
  chart.draw(data, options);

  for (i = 0; i < 5; ++i) {
    var j, max = 0, maxIndex = -1;
    for (j in hash) {
      if (hash[j] > max) {
        max = hash[j];
        maxIndex = j;
      }
    }

    var line = document.createElement('div');
    line.classList.add('onepixline', 'clear');
    innerCard.appendChild(line);

    var entry = document.createElement('div');
    entry.className = 'entry';
    innerCard.appendChild(entry);

    var title = document.createElement('h2');
    title.className = 'entryTitle';
    title.innerHTML = maxIndex;
    entry.appendChild(title);

    var info = document.createElement('p');
    info.className = 'entryDes';
    info.innerHTML = '<span>With</span> ' + max + ' <span>associated records.</spa
    entry.appendChild(info);
```

```
303
304     hash[maxIndex] = undefined;
305   }
306
307   return card;
308 }
309
310
311 /*
312  * A helper class to start up the creation of a card since it
313  * is so common.
314  */
315 Analytics.prototype.initCard = function(card, title) {
316   var redline = document.createElement('div');
317   redline.className = 'redline';
318   card.appendChild(redline);
319
320   var innerCard = document.createElement('div');
321   innerCard.className = 'inner-card';
322   card.appendChild(innerCard);
323
324   var iconandtext = document.createElement('div');
325   iconandtext.className = 'iconandtext';
326   iconandtext.style.width = '50%';
327   innerCard.appendChild(iconandtext);
328
329   var iticon = document.createElement('img');
330   iticon.className = 'iticon';
331   iticon.src = 'images/pie.png';
332   iticon.alt = 'Mmm... pie.';
333   iconandtext.appendChild(iticon);
334
335   var ittext = document.createElement('h1');
336   ittext.className = 'ittext';
337   ittext.innerHTML = title;
338   iconandtext.appendChild(ittext);
339
340   return innerCard;
341 }
342
343
344 /*
345  * A class that starts up the hotlinks functionality: scrolling
346  * to the section when it is clicked and as the user is scrolling,
347  * update which section we are in with an underline.
```

```
 */
Analytics.prototype.hotlinks = function(tc, ip, att) {
  var hottc = document.getElementById('tc');
  var hotip = document.getElementById('ip');
  var hotatt = document.getElementById('att');

  hottc.onclick = function() {
    scrollTo(document.body, tc.getBoundingClientRect().top + window.scrollY, 200);
  };
  hotip.onclick = function() {
    scrollTo(document.body, ip.getBoundingClientRect().top + window.scrollY, 200);
  };
  hotatt.onclick = function() {
    scrollTo(document.body, att.getBoundingClientRect().top + window.scrollY, 200)
  };

  window.addEventListener('scroll', function(e) {
    e.preventDefault();
    if (att.getBoundingClientRect().top <= 0) {
      hotip.classList.remove('underlined');
      hottc.classList.remove('underlined');
      if (!hotatt.classList.contains('underlined')) {
        hotatt.classList.add('underlined');
      }
    } else if (ip.getBoundingClientRect().top <= 0) {
      hotatt.classList.remove('underlined');
      hottc.classList.remove('underlined');
      if (!hotip.classList.contains('underlined')) {
        hotip.classList.add('underlined');
      }
    } else {
      hotip.classList.remove('underlined');
      hotatt.classList.remove('underlined');
      if (!hottc.classList.contains('underlined')) {
        hottc.classList.add('underlined');
      }
    }
    return false;
  });
}
```

A.4 main.js

```
/*
 * A function that loads a new heatmap data layer and draws it onto
 * the current map.
 *
 * Arguments:
 *   heatmap <Object> - The google heatmap layer object.
 *
 * Returns:
 *   <Function> - A closure for the heatmap variable.
 *   Has the coordinates <Array> variable which must contain
 *   an array of {location: google LatLng Object,
 *   weight: Number} objects.
 */
var updateData = function(heatmap) {
  return function(coordinates) {

    $('#overlay').hide();

    // Convert heat list to google maps array.
    var pointArray = new google.maps.MVCArray(coordinates);

    // Set the heatmap to be updated with the new points.
    heatmap.setData(pointArray);
  }
}


/*
 * A function that takes a list of IPs and translates them into coordinates
 * and weights on each coordinate. At completion, the callback function that
 * was passed in is called.
 *
 * Arguments:
 *   callback <Function> - The callback function to be called on completion.
 *
 * Returns:
 *   <Function> - Creates a closure that is passed back to the getLatestIPs
 *   function so that it can call this asynchronously. Argument to the closure
 *   is an <Array> that contains all IP addresses we want to translate.
 */
var translate = function(callback) {
```

```
  return function(IPs) {
    $('#overlay').show();
    var iplocator = new IPLocator();
    iplocator.locate(IPs, callback);
  }
}


/*
 * A function that gets the list of the latest IPs which have connected to
 * our server. Passes on the callback function to our translate function
 * so that it can return context later if the map has already been drawn.
 *
 * Arguments:
 *   callback <Function> - The callback function. It tells the program where
 *   to return to once the data has been process but is not dealt with
 *   directly in this function.
 *
 * Returns:
 *   Nothing.
 */
var getLatestIPs = function(callback) {
  var ipgetter = new IPGetter();
  ipgetter.get(translate(callback));
}


/*
 * Kicks off our program by first creating our map and then fetching the
 * latest&greatest IP information from the server. After the map is loading,
 * we are taken to a function that will download the server info.
 *
 * Arguments:
 *   None.
 *
 * Returns:
 *   Nothing.
 */
var initialize = function() {

  // Set some rough map options.
  //var mapCenter = new google.maps.LatLng(37.8282, -95.5795);
  var mapCenter = new google.maps.LatLng(40,0);
  var mapOptions = {
    mapTypeControl: false,
```

```
    center: mapCenter,
    zoom: 2
  };

  // Get the map container.
  var container = document.getElementById('map-canvas');

  // Draw the map.
  var map = new google.maps.Map(container, mapOptions);

  // Create an empty MVC array to pass into Google Maps.
  var pointArray = new google.maps.MVCArray([]);

  // Generate an empty heatmap layer.
  var heatmap = new google.maps.visualization.HeatmapLayer({
    data: pointArray,
    radius: 20
  });

  // Set the heatmap layer down.
  heatmap.setMap(map);

  // Set some colors so that our map is more visible.
  var gradient = [
    'rgba(125, 0, 0, 0)',
    'rgba(135, 0, 0, 1)',
    'rgba(145, 0, 0, 1)',
    'rgba(155, 0, 0, 1)',
    'rgba(165, 0, 0, 1)',
    'rgba(175, 0, 0, 1)',
    'rgba(185, 0, 0, 1)',
    'rgba(195, 0, 0, 1)',
    'rgba(205, 0, 0, 1)',
    'rgba(215, 0, 0, 1)',
    'rgba(225, 0, 0, 1)',
    'rgba(235, 0, 0, 1)',
    'rgba(245, 0, 0, 1)',
    'rgba(255, 0, 0, 1)'
  ]
  heatmap.set('gradient', gradient);

  // Create the button at the top right for settings.
  var opsDiv = document.createElement('div');
  var ops = new OptionsButton(opsDiv, map, heatmap);
  opsDiv.index = 1;
```

```
132   map.controls[google.maps.ControlPosition.TOP_RIGHT].push(opsDiv);
133
134   // Get the latest information.
135   getLatestIPs(updateData(heatmap));
136 }
137
138 // Create a listener for when the DOM has loaded.
139 $(document).ready(initialize);
```