

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

<b>1</b>	<b>Abstract</b>	<b>2</b>
1.1	Introduction . . . . .	2
<b>2</b>	<b>Hardware and Software Architecture</b>	<b>5</b>
2.1	Evaluation of System Hardware . . . . .	5
2.2	System Software Architecture . . . . .	6
<b>3</b>	<b>Approach and Methodology</b>	<b>9</b>
3.1	Approach . . . . .	9
3.2	Employing Two-point Authentication over SSH . . . . .	10
3.3	Installation . . . . .	10
<b>4</b>	<b>Administration Suite &amp; Features</b>	<b>16</b>
4.1	Web Front-end Integration . . . . .	16
4.2	Analytics . . . . .	17
4.3	Response Options . . . . .	19
<b>5</b>	<b>Conclusion</b>	<b>21</b>
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
<b>A</b>	<b>Source Code Highlights</b>	<b>26</b>
A.1	IPLocator.js . . . . .	26
A.2	HistoryGenerator.js . . . . .	30
A.3	Analytics.js . . . . .	39
A.4	main.js . . . . .	48

# 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*.”

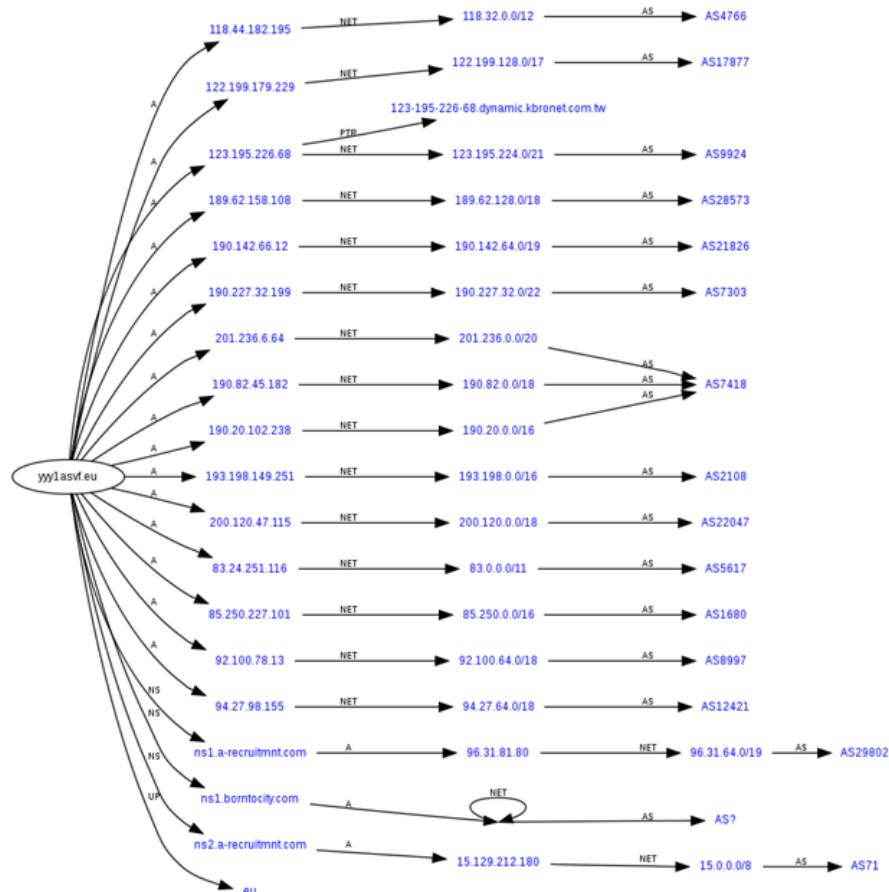


Figure 1.1: A visualization of the domain fast-flux. Note the large number of IP addresses and domains mapping to a single fully qualified domain name used for the C2 server.

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.

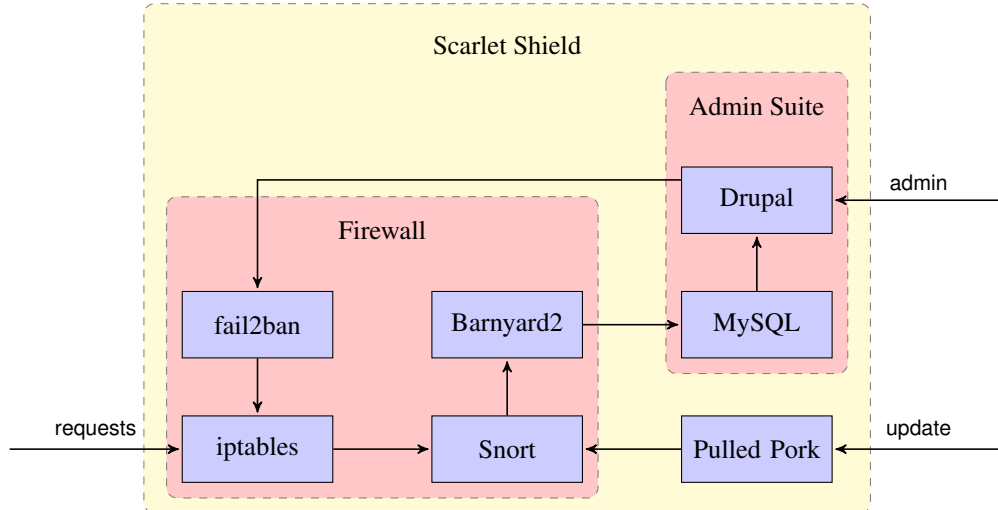


Figure 2.1: The architecture of Scarletshield. Incoming packets are routed through iptables, where they may be blocked per iptables’ rule-set; Snort receives the packets and inspects them for signatures of attacks; offending packets are mirrored via Barnyard2 into a MySQL database; users can access the 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](http://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 architecture 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 force.

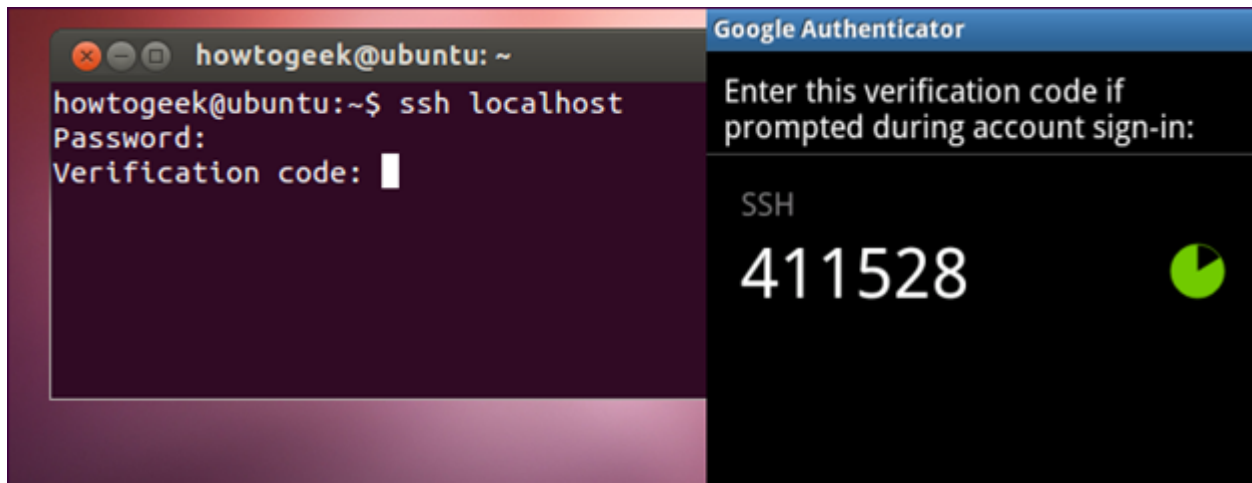


Figure 3.1: (Left) A prompt to enter a Google Authenticator verification code after typing in a password. (Right) Google Authenticator smart phone application generating a verification code.

## 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, libpcrc3-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:

---

```
1 sudo apt-get install update
2 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](http://snort.org) website and libdnet google code repository, respectively, and we acquired and installed them as follows:

---

```
1 sudo wget http://www.snort.org/downloads/1806 -O daq.tar.gz
2 sudo tar xzvf daq.tar.gz
3 cd libdnet
4 sudo ./configure
5 sudo make
6 sudo make install
```

---

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

---

```
1 sudo wget http://libdnet.googlecode.com/files/libdnet-1.12.tgz -O libdnet.tar.gz
2 sudo tar xzvf libdnet.tar.gz
3 cd daq
4 sudo ./configure
5 sudo make
6 sudo make install
7 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!

---

```
1 sudo wget http://www.snort.org/downloads/1814 -O snort.tar.gz
2 sudo tar xzvf snort.tar.gz
3 cd snort
4 sudo ./configure --prefix=/usr/local/snort --enable-sourcefire
5 sudo make
6 sudo make install
7 sudo mkdir /var/log/snort
8 sudo mkdir /var/snort
```

```
9 sudo groupadd snort
10 sudo useradd -g snort snort
11 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:

---

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

---

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

---

```
1 cd /usr/local/snort
2 wget https://pulledpork.google.com/files/pulledpork-0.6.1.tar.gz
3 tar -xzf pulledpork-0.6.1.tar.gz
4 cd pulledpork-0.6.1
5 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:

---

```
1 sudo mkdir /etc/snort
2 cd /etc/snort
3 wget http://scarletshield.rutgers.edu/snort.conf
4 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:

---

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

---

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

---

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

---

Updates via our new Pulledpork configuration are asserted as follows:

---

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

---

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

---

```
1 wget https://nodeload.github.com/firnsy/barnyard2/tarball/master
2     -O barnyard2-2.10.tar.gz
3 sudo tar xzvf barnyard2-2.10.tar.gz
4 cd firnsy-barnyard2*
5 sudo autoreconf -fvi -I ./m4
6 sudo ./configure --with-mysql --with-mysql-libraries=/usr/lib/i386-linux-gnu
7 sudo make
8 sudo make install
9 sudo cp etc/barnyard2.conf /usr/local/snort/etc
10 sudo mkdir /var/log/barnyard2
11 sudo chmod 666 /var/log/barnyard2
12 sudo touch /var/log/snort/barnyard2.waldo
13 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:

---

```
1 echo "create database snort;" | mysql -u root -p
2 mysql -u root -p -D snort < ./schemas/create\_mysql
3 echo "grant create,insert, select, delete, update on snort.* to
4     snort@localhost identified by 'OURSECRETPASSWORD'" | mysql -u root -p
```

---

Finally, we set up the network cards:

---

```
1 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:

---

```
1 auto eth1 iface eth1
2 inet manual
3     up ifconfig \${IFACE} 0.0.0.0 up
4     up ip link set \${IFACE} promisc on
5     down ip link set \${IFACE} promisc off
6     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>.<sup>1</sup>

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.

---

<sup>1</sup>Note that the Scarletshield Suite was written for the Chrome browser, and some small details will not work on other browsers.

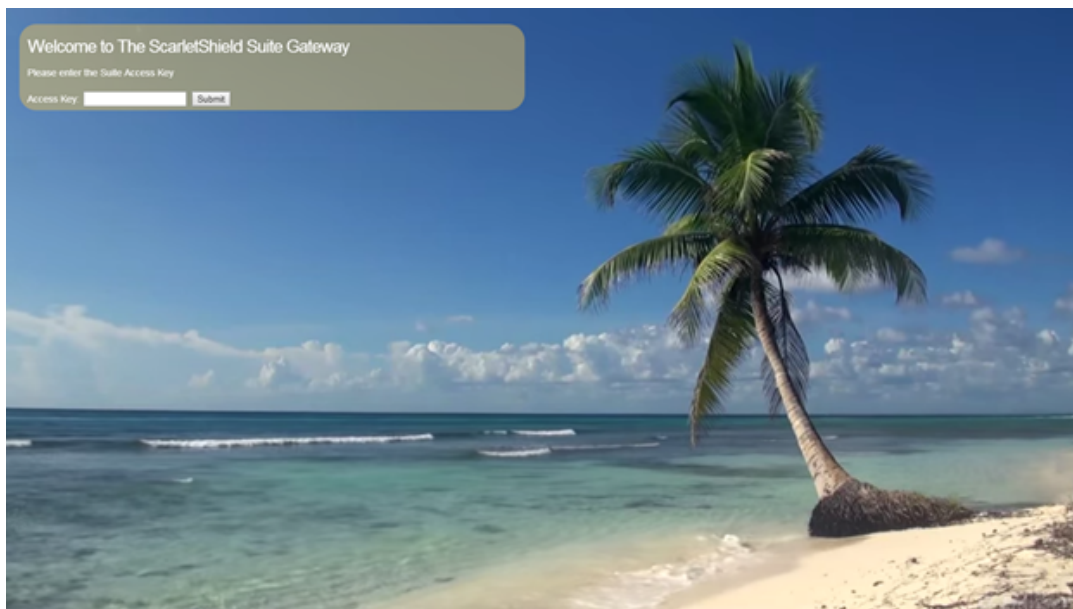


Figure 4.1: The Scarletshield Suite login page.

## 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

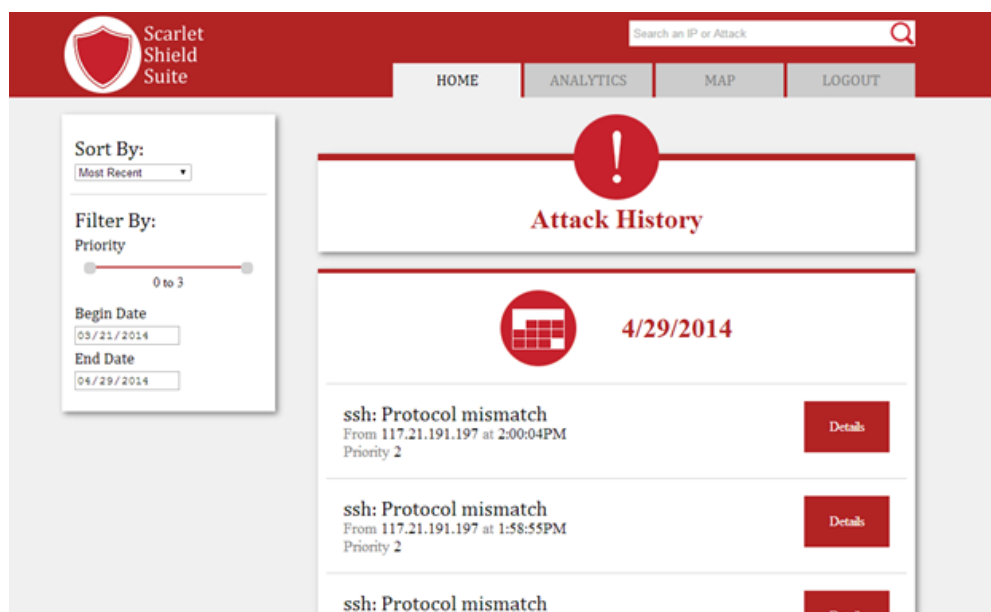


Figure 4.2: The Scarletshield Suite Attack History page.

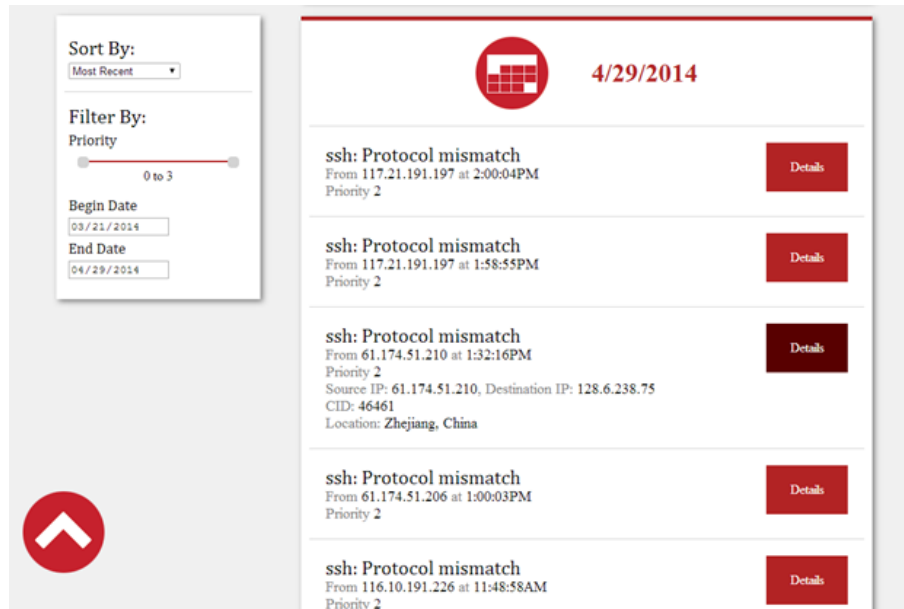


Figure 4.3: The Scarletshield Suite Attack History page, with an expanded description of the details of an attack originating from China.

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.

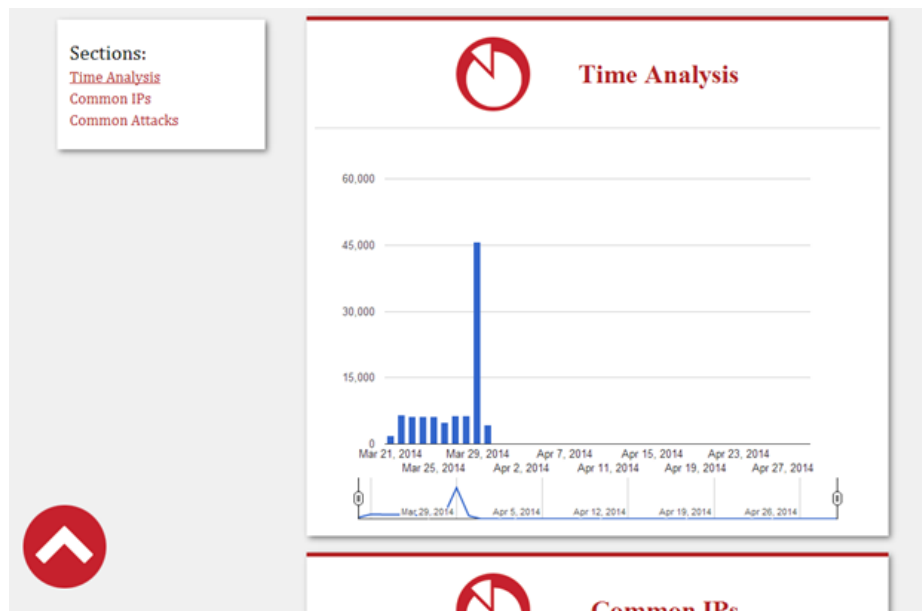


Figure 4.4: A chart of the rate of attacks over a certain interval. Users can use the slider at the bottom of the chart to analyze attacks over arbitrary intervals.

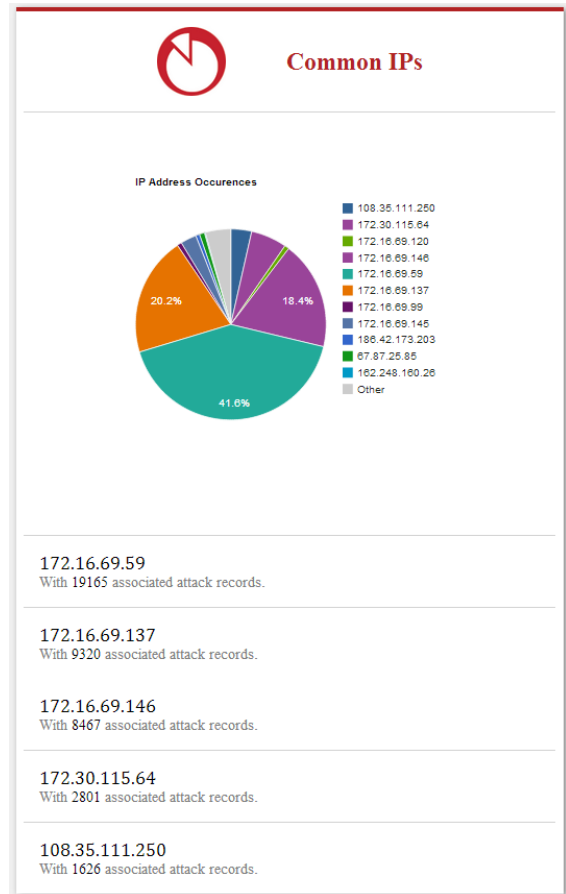
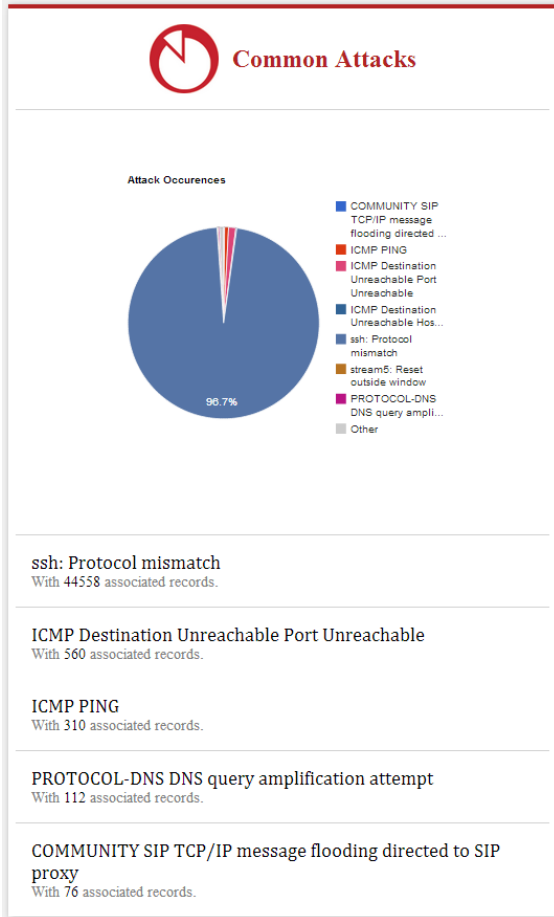


Figure 4.5: Pie charts featuring prominent attack varieties and IP addresses.

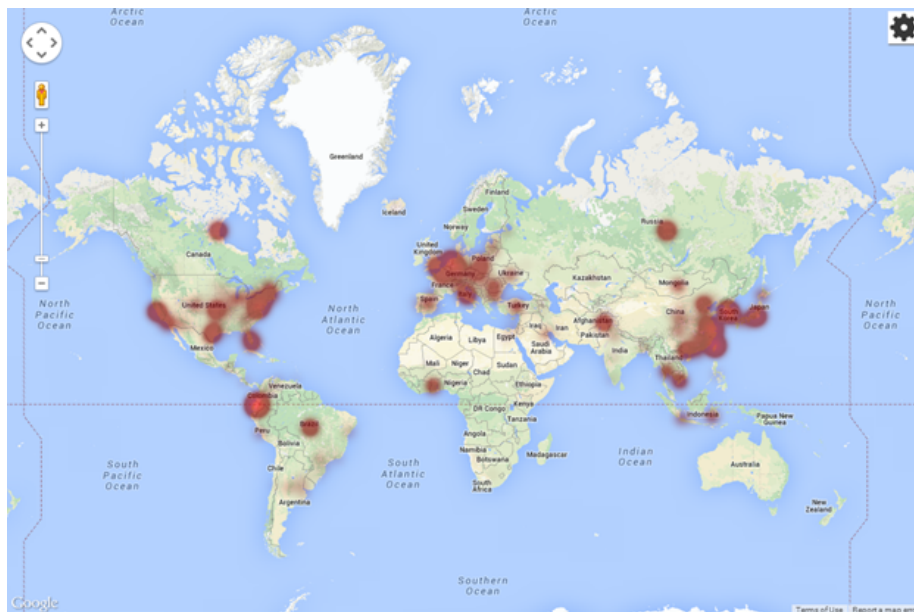


Figure 4.6: A geographic world map with red intensity markers indicating the presence of attack vectors from a geographic region, identified by IP address.

# 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 sub-components 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 Ubuntu 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](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 IPLocator.js

---

```
1 /*
2  * The IPLocator class. Does exactly what it sounds like: translates
3  * the IP addresses which are passed into the locate function into
4  * an array of Objects containing google LatLng objects and the
5  * corresponding weight they have on the map.
6  *
7  * Arguments:
8  *   Nothing.
9  *
10 * Returns:
11 *   <Object> - this. The context of the IPLocator so that a function
12 *   that creates an IPLocator object may call the locate function.
13 */
14 var IPLocator = function() {
15   this.goalWeight = 0;
16   this.totalWeight = 0;
17   this.goalRequests = 0;
18   this.completeRequests = 0;
19   this.heatSpots = [];
20   this.container = document.getElementById('map-canvas');
21   this.callback = null;
22
23   return this;
24 }
25
```

```

26 /*
27  * Takes care of creating JSON requests for the IP address locations
28  * which in turn will trigger the map being drawn later on.
29  *
30  * Arguments:
31  *   IPs <Array> - Array of IP addresses which we will search for
32  *   the location of.
33  *   callback <Function> - The function we will call upon successful
34  *   translation.
35  *
36  * Returns:
37  *   Nothing.
38  */
39 IPLocator.prototype.locate = function(IPs, callback) {
40
41   // The site we use to get information about IP addresses.
42   var geoURL = 'http://162.248.161.124:8080/json/';
43
44   // An array we will keep the locations we wish to highlight
45   // in the form of google LatLng objects.
46
47   var hash = {};
48
49   for (var i = 0; i < IPs.length; ++i) {
50     var cur = IPs[i];
51     if (!hash[cur]) {
52       hash[cur] = 1;
53     } else {
54       ++hash[cur];
55     }
56   }
57
58   // Save some stuff for later.
59   this.callback = callback;
60   this.goalWeight = IPs.length;
61   this.goalRequests = Object.keys(hash).length;
62
63   var cur;
64   for (cur in hash) {
65     var weight = hash[cur];
66     $.getJSON(geoURL + cur,
67       IPLocator.successHandler(weight, this)
68     ).fail(IPLocator.failHandler(weight, this));
69   }
70

```

```

71 }
72
73
74 /*
75  * Static function which is called when a request is responded to.
76  */
77 IPLocator.successHandler = function(weight, iptrans) {
78     return function(data) {
79         iptrans.handler(weight, data);
80     }
81 }
82
83
84 /*
85  * Static function which is called when a request fails.
86  */
87 IPLocator.failHandler = function(weight, iptrans) {
88     return function() {
89         iptrans.error(weight);
90     }
91 }
92
93
94 /*
95  * Function that checks if the coordinates we are currently seeing
96  * are part of a location we wish to ignore: where we live.
97  */
98 IPLocator.prototype.isUs = function(data) {
99     return (Math.floor(Number(data.latitude)) === 40 &&
100         Math.floor(Number(data.longitude)) === -75)
101 }
102
103
104 /*
105  * A function that handles the returned request asking for
106  * the coordinates of an IP address. Simply adds the coordinate
107  * value as many times as the IP address is needed.
108  *
109  * Arguments:
110  *   weight <Number> - The number of times this IP address appeared
111  *   in the list passed in.
112  *   data <Object> - The data that was returned by the request.
113  *
114  * Returns:
115  *   Nothing.

```

```

116  */
117 IPLocator.prototype.handler = function(weight, data) {
118     if (!(data.latitude === 0 && data.longitude === 0 ) && !this.isUs(data)) {
119
120         // When we get it, we push the google LatLng object of the
121         // location to the heatSpots array.
122         this.heatSpots.push({
123             location: new google.maps.LatLng(data.latitude, data.longitude),
124             weight: weight
125         });
126     }
127
128     // Increment counter for how far we have progressed in our reqs.
129     this.totalWeight += weight;
130     this.completeRequests += 1;
131
132
133     var overlay = document.getElementById('progress');
134     //overlay.setAttribute('aria-valuenow', this.completeRequests/this.goalRequests*100);
135     overlay.style.width = this.completeRequests/this.goalRequests*100+'%'
136     //$('#overlay').html(
137     //    Math.floor(
138     //        this.completeRequests/this.goalRequests * 100
139     //    ).toString());
140
141     if (this.totalWeight === this.goalWeight) {
142         this.callback(this.heatSpots);
143     }
144 }
145
146
147 /*
148  * A function that is called when there is an error on the request.
149  * Simply increments the counter of the requests returned and also
150  * checks if this was the last item needed to be returned.
151  */
152 IPLocator.prototype.error = function(weight) {
153     this.totalWeight += weight;
154     this.completeRequests += 1;
155     if (this.totalWeight === this.goalWeight) {
156         this.callback(this.heatSpots);
157     }
158 }

```

---

## A.2 HistoryGenerator.js

---

```
1 var HistoryGenerator = function(data) {
2   this.data = data;
3   this.cur = data.length - 1;
4   this.container = document.getElementById('dataArea');
5   this.lastDate = new Date(0);
6   this.lastCard = null;
7   this.sorting = 'recent';
8   this.lowPriority = 3;
9   this.highPriority = 0;
10  var startdate = document.getElementById('startdate');
11  var enddate = document.getElementById('enddate');
12  var sdateVal = HistoryGenerator.getStartDate(startdate.value);
13  var edateVal = HistoryGenerator.getEndDate(enddate.value);
14  this.startDate = sdateVal.valueOf();
15  this.endDate = edateVal.valueOf();
16  this.search = '';
17
18  this.container.innerHTML = '';
19
20
21  return this;
22 }
23
24 HistoryGenerator.prototype.next20 = function() {
25
26   // Get some common variables to use a lot.
27   var data = this.data;
28   var lastDate = this.lastDate;
29   var lastCard = this.lastCard;
30
31   // Find where we left off.
32   var i = this.cur;
33
34   // We only want to do 20 at a time so keep a counter.
35   var numDone = 0;
36   while (numDone < 20 && i >= 0) {
37
38     // Get the current object.
39     var cur = data[i];
40
41     if (this.search !== '') {
```



```

42     var message = cur.msg.toLowerCase();
43     var src = cur.src;
44     var dst = cur.dst;
45     var search = this.search.toLowerCase();
46     console.log();
47     if (message.indexOf(search) === -1 &&
48         src.indexOf(search) === -1 &&
49         dst.indexOf(search) === -1) {
50         --i;
51         continue;
52     }
53 }
54 // If we happen to be filtering out this priority number, skip
55 // this whole block.
56 var priority = cur.priority;
57 if (priority < this.highPriority || priority > this.lowPriority) {
58     --i;
59     continue;
60 }
61
62 // Get the date from it and create a Date object from it.
63 var splitDate = cur.datetime.split(/\s:-/);
64 var year = Number(splitDate[0]);
65 var month = Number(splitDate[1]);
66 var day = Number(splitDate[2]);
67 var hour = Number(splitDate[3]);
68 var minutes = Number(splitDate[4]);
69 var seconds = Number(splitDate[5]);
70 var curDate = new Date(year, month-1, day, hour, minutes, seconds, 0);
71
72 if (curDate.valueOf() > this.endDate ||
73     curDate.valueOf() < this.startDate) {
74     --i;
75     continue;
76 }
77
78 // If this is the same date as the last one, we want to continue on the same
79 // card. If not, we want to make a new one.
80 if (lastDate.toString() === curDate.toString()) {
81     this.appendEntry(cur, lastCard, hour, minutes, seconds);
82 } else {
83     var card = this.appendCard(cur, year, month, day, hour, minutes, seconds);
84
85     // Since this is a new card, we have to replace the old variables which no
86     // 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 =
107     "<div class='card shadow'>
108         <div class='redline'></div>
109         <div class='inner-card'>
110             <div class='iconandtext'>
111                 <img class='iticon' src='images/sad.png' alt='Calendar'/>
112                 <h1 class='ittext'>No Results!</h1>
113             </div>
114             <div class='error-space'></div>
115         </div>
116     </div>";
117 }
118
119
120 HistoryGenerator.prototype.appendCard = function(
121     cur, year, month, day, hour, minutes, seconds) {
122     var card = document.createElement('div');
123     card.classList.add('card', 'shadow');
124     this.container.appendChild(card);
125
126     var redline = document.createElement('div');
127     redline.className = 'redline';
128     card.appendChild(redline);
129
130     var innerCard = document.createElement('div');
131     innerCard.className = 'inner-card';

```

```

132     card.appendChild(innerCard);
133
134     var iconandtext = document.createElement('div');
135     iconandtext.className = 'iconandtext';
136     innerCard.appendChild(iconandtext);
137
138     var iticon = document.createElement('img');
139     iticon.className = 'iticon';
140     iticon.src = 'images/cal.png';
141     iticon.alt = 'Calendar';
142     iconandtext.appendChild(iticon);
143
144     var ittext = document.createElement('h1');
145     ittext.className = 'itttext';
146     ittext.innerHTML = month + '/' + day + '/' + year;
147     iconandtext.appendChild(ittext);
148
149     this.appendEntry(cur, innerCard, hour, minutes, seconds);
150
151
152     return innerCard;
153 }
154
155 HistoryGenerator.prototype.appendEntry = function(
156     cur, lastCard, hour, minutes, seconds) {
157     var innerCard = lastCard;
158
159     var line = document.createElement('div');
160     line.classList.add('onepixline', 'clear');
161     innerCard.appendChild(line);
162
163     var entry = document.createElement('div');
164     entry.className = 'entry';
165     innerCard.appendChild(entry);
166
167     var det = document.createElement('div');
168     det.className = 'detButton';
169     det.innerHTML = 'Details';
170     entry.appendChild(det);
171
172
173     var title = document.createElement('h2');
174     title.className = 'entryTitle';
175     title.innerHTML = cur.msg;
176     entry.appendChild(title);

```

```

177
178 var info = document.createElement('p');
179 info.className = 'entryDes';
180 var m = 'AM';
181 if (hour === 0) {
182     hour = '12';
183 } else if (hour > 12) {
184     hour = hour - 12;
185     m = 'PM'
186 }
187 if (minutes < 10) {
188     minutes = '0' + minutes;
189 }
190 if (seconds < 10) {
191     seconds = '0' + seconds;
192 }
193 info.innerHTML = '<span>From</span> ' + cur.src +
194     ' <span>at</span> ' + hour + ':' + minutes + ':' + seconds + m;
195 entry.appendChild(info);
196
197 var prio = document.createElement('p');
198 prio.className = 'entryDes';
199 prio.innerHTML = '<span>Priority</span> ' + cur.priority;
200 entry.appendChild(prio);
201
202 var popout = document.createElement('div');
203 popout.style.display = 'none';
204 entry.appendChild(popout);
205
206 var srcdst = document.createElement('p');
207 srcdst.className = 'entryDes';
208 srcdst.innerHTML = '<span>Source IP: </span>' + cur.src +
209     '<span>, Destination IP: </span>' + cur.dst;
210 popout.appendChild(srcdst);
211
212 var cid = document.createElement('p');
213 cid.className = 'entryDes';
214 cid.innerHTML = '<span>CID: </span>' + cur.cid;
215 popout.appendChild(cid);
216
217 var loc = document.createElement('p');
218 loc.className = 'entryDes';
219 popout.appendChild(loc);
220
221

```

```

222 var expanded = false;
223 var foundLocation = false;
224 det.onclick = function() {
225     if (expanded === false) {
226         if (foundLocation === false) {
227             var geoURL = 'http://162.248.161.124:8080/json/';
228
229             get(geoURL + cur.src, function() {
230                 var response = JSON.parse(this.responseText);
231                 var string = '';
232                 if (response.region_name !== '') {
233                     string += response.region_name + ', ';
234                 }
235                 if (response.country_name !== '') {
236                     string += response.country_name;
237                 }
238                 if (string === '') {
239                     string = 'Location unknown.';
240                 } else {
241                     string = '<span>Location: </span>' + string;
242                 }
243                 loc.innerHTML = string;
244             });
245
246             foundLocation = true;
247         }
248         popout.style.display = 'block';
249         expanded = true;
250     } else {
251         popout.style.display = 'none';
252         expanded = false;
253     }
254 }
255
256 }
257
258 HistoryGenerator.prototype.registerScroll = function() {
259     var content = document.getElementById('content');
260     var bottom = document.createElement('div');
261     bottom.className = 'bottom-space';
262     content.appendChild(bottom);
263
264     var pageupShown = false;
265     var hg = this;
266     window.onscroll = function(e) {

```

```

267     e.preventDefault();
268     var winHeight = window.innerHeight;
269     var divLocation =
270         document.getElementsByClassName('bottom-space')[0].getBoundingClientRect().t
271
272     if (divLocation < winHeight) {
273         hg.next20();
274     }
275
276     var side = document.getElementById('sidebar');
277     if (window.scrollToY > 100) {
278         if (pageupShown === false) {
279             var pageup = document.getElementById('pageup');
280             moveToX(pageup, 20, 175);
281             pageupShown = true;
282         }
283         var side = document.getElementById('sidebar');
284         side.style.position = 'fixed';
285         side.style.marginTop = '-100px';
286     } else {
287         if (pageupShown === true) {
288             var pageup = document.getElementById('pageup');
289             moveToX(pageup, -150, 175);
290             pageupShown = false;
291         }
292         side.style.marginTop = '0px';
293         side.style.position = 'static';
294     }
295     return false;
296 };
297 }
298
299 HistoryGenerator.prototype.sortBy = function(type) {
300     if (type === 'recent') {
301         if (this.sorting === 'oldest') {
302             this.data.reverse();
303         } else {
304             this.data.sort(function(a, b) {
305                 return a.id - b.id;
306             });
307         }
308     } else if (type === 'oldest') {
309         if (this.sorting === 'recent') {
310             this.data.reverse();
311         } else {

```

```

312     this.data.sort(function(a, b) {
313         return b.id - a.id;
314     });
315 }
316 } else if (type === 'pinc') {
317     this.data.sort(function(a, b) {
318         var retval = a.priority - b.priority;
319         if (retval === 0) {
320             return a.id - b.id;
321         }
322         return retval
323     });
324 } else if (type === 'pdec') {
325     this.data.sort(function(a, b) {
326         var retval = b.priority - a.priority;
327         if (retval === 0) {
328             return a.id - b.id;
329         }
330         return retval
331     });
332 }
333 this.refreshUs();
334 this.sorting = type;
335 console.log(this);
336 }
337
338 HistoryGenerator.prototype.refreshUs = function() {
339     window.scrollTo(0,0);
340     this.cur = this.data.length - 1;
341     this.container.innerHTML = '';
342     this.lastDate = new Date(0);
343     this.lastCard = null;
344     this.next20();
345 }
346 }
347
348 HistoryGenerator.prototype.registerSelector = function() {
349     var selector = document.getElementById('selector');
350     var hg = this;
351     selector.onchange = function() {
352         hg.sortBy(selector.value);
353     }
354 }
355
356 HistoryGenerator.prototype.registerDates = function() {

```

```

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

```



```

402     var tokenized = value.split('-');
403     var year = Number(tokenized[0]);
404     var month = Number(tokenized[1]);
405     var day = Number(tokenized[2]);
406     var date = new Date(year, month-1, day, 23, 59, 59, 999);
407     return date;
408 }
409
410 HistoryGenerator.prototype.setHighPriority = function(value) {
411     this.highPriority = value;
412     this.refreshUs();
413 }
414 HistoryGenerator.prototype.setLowPriority = function(value) {
415     this.lowPriority = value;
416     this.refreshUs();
417 }

```

---

## A.3 Analytics.js

---

```

1  /*
2   * The Analytics class. Starts up our analytics in the div with id equal
3   * to dataArea. Use Analytics.init() to kick off the process.
4   */
5  var Analytics = function(data) {
6      this.data = data;
7      this.container = document.getElementById('dataArea');
8
9      return this;
10 }
11
12
13 /*
14 * Initializes all of the boxes in the Analytics class.
15 */
16 Analytics.prototype.init = function() {
17
18     // Clear the area for our data.
19     this.container.innerHTML = '';
20
21     // Create the time thing
22     var tc = this.createTimeChart();

```

```

23
24 // Create common IP thing
25 var ip = this.createCommonIPs();
26
27 // Create common attacks thing
28 var att = this.createCommonAttacks();
29
30 // Hotlinks to divs.
31 this.hotlinks(tc, ip, att);
32 }
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
168 var data = new google.visualization.DataTable();
169 data.addColumn('string', 'IP Address');
170 data.addColumn('number', 'Number of Occurrences');
171
172 for (i in hash) {
173     if (hash[i]) {
174         data.addRow([i, hash[i]]);
175     }
176 }
177 var options = {
178     title: 'IP Address Occurences'
179 };
180 var divi = document.createElement('div');
181 divi.classList.add('onepixline', 'clear');
182 innerCard.appendChild(divi);
183
184 var pie = document.createElement('div');
185 pie.className = 'entry';
186 pie.style.height = '500px';
187 innerCard.appendChild(pie);
188
189 var chart = new google.visualization.PieChart(pie);
190 chart.draw(data, options);
191
192 for (i = 0; i < 5; ++i) {
193     var j, max = 0, maxIndex = -1;
194     for (j in hash) {
195         if (hash[j] > max) {
196             max = hash[j];
197             maxIndex = j;
198         }
199     }
200
201     var line = document.createElement('div');
202     line.classList.add('onepixline', 'clear');

```

```

203     innerCard.appendChild(line);
204
205     var entry = document.createElement('div');
206     entry.className = 'entry';
207     innerCard.appendChild(entry);
208
209     var title = document.createElement('h2');
210     title.className = 'entryTitle';
211     title.innerHTML = maxIndex;
212     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) {
258     if (hash[i]) {
259         data.addRow([i, hash[i]]);
260     }
261 }
262 var options = {
263     title: 'Attack Occurences'
264 };
265 var divi = document.createElement('div');
266 divi.classList.add('onepixline', 'clear');
267 innerCard.appendChild(divi);
268
269 var pie = document.createElement('div');
270 pie.className = 'entry';
271 pie.style.height = '500px';
272 innerCard.appendChild(pie);
273
274 var chart = new google.visualization.PieChart(pie);
275 chart.draw(data, options);
276
277 for (i = 0; i < 5; ++i) {
278     var j, max = 0, maxIndex = -1;
279     for (j in hash) {
280         if (hash[j] > max) {
281             max = hash[j];
282             maxIndex = j;
283         }
284     }
285
286     var line = document.createElement('div');
287     line.classList.add('onepixline', 'clear');
288     innerCard.appendChild(line);
289
290     var entry = document.createElement('div');
291     entry.className = 'entry';
292     innerCard.appendChild(entry);

```

```

293
294     var title = document.createElement('h2');
295     title.className = 'entryTitle';
296     title.innerHTML = maxIndex;
297     entry.appendChild(title);
298
299     var info = document.createElement('p');
300     info.className = 'entryDes';
301     info.innerHTML = '<span>With</span> ' + max + ' <span>associated records.</spa
302     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 = 'itttext';
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.
348  */
349 Analytics.prototype.hotlinks = function(tc, ip, att) {
350     var hottc = document.getElementById('tc');
351     var hotip = document.getElementById('ip');
352     var hotatt = document.getElementById('att');
353
354     hottc.onclick = function() {
355         scrollTo(document.body, tc.getBoundingClientRect().top + window.scrollY, 200);
356     };
357     hotip.onclick = function() {
358         scrollTo(document.body, ip.getBoundingClientRect().top + window.scrollY, 200);
359     };
360     hotatt.onclick = function() {
361         scrollTo(document.body, att.getBoundingClientRect().top + window.scrollY, 200);
362     };
363
364     window.addEventListener('scroll', function(e) {
365         e.preventDefault();
366         if (att.getBoundingClientRect().top <= 0) {
367             hotip.classList.remove('underlined');
368             hottc.classList.remove('underlined');
369             if (!hotatt.classList.contains('underlined')) {
370                 hotatt.classList.add('underlined');
371             }
372         } else if (ip.getBoundingClientRect().top <= 0) {
373             hotatt.classList.remove('underlined');
374             hottc.classList.remove('underlined');
375             if (!hotip.classList.contains('underlined')) {
376                 hotip.classList.add('underlined');
377             }
378         } else {
379             hotip.classList.remove('underlined');
380             hotatt.classList.remove('underlined');
381             if (!hottc.classList.contains('underlined')) {
382                 hottc.classList.add('underlined');

```

```

383     }
384   }
385   return false;
386 });
387 }

```

---

## A.4 main.js

---

```

1  /*
2   * A function that loads a new heatmap data layer and draws it onto
3   * the current map.
4   *
5   * Arguments:
6   *   heatmap <Object> - The google heatmap layer object.
7   *
8   * Returns:
9   *   <Function> - A closure for the heatmap variable.
10  *   Has the coordinates <Array> variable which must contain
11  *   an array of {location: google LatLng Object,
12  *   weight: Number} objects.
13  */
14  var updateData = function(heatmap) {
15    return function(coordinates) {
16
17      $('#overlay').hide();
18
19      // Convert heat list to google maps array.
20      var pointArray = new google.maps.MVCArray(coordinates);
21
22      // Set the heatmap to be updated with the new points.
23      heatmap.setData(pointArray);
24    }
25  }
26
27
28  /*
29   * A function that takes a list of IPs and translates them into coordinates
30   * and weights on each coordinate. At completion, the callback function that
31   * was passed in is called.
32   *
33   * Arguments:

```

```

34  *   callback <Function> - The callback function to be called on completion.
35  *
36  * Returns:
37  *   <Function> - Creates a closure that is passed back to the getLatestIPs
38  *   function so that it can call this asynchronously. Argument to the closure
39  *   is an <Array> that contains all IP addresses we want to translate.
40  */
41  var translate = function(callback) {
42      return function(IPs) {
43          $('#overlay').show();
44          var iplocator = new IPLocator();
45          iplocator.locate(IPs, callback);
46      }
47  }
48
49
50  /*
51  * A function that gets the list of the latest IPs which have connected to
52  * our server. Passes on the callback function to our translate function
53  * so that it can return context later if the map has already been drawn.
54  *
55  * Arguments:
56  *   callback <Function> - The callback function. It tells the program where
57  *   to return to once the data has been process but is not dealt with
58  *   directly in this function.
59  *
60  * Returns:
61  *   Nothing.
62  */
63  var getLatestIPs = function(callback) {
64      var ipgetter = new IPGetter();
65      ipgetter.get(translate(callback));
66  }
67
68
69  /*
70  * Kicks off our program by first creating our map and then fetching the
71  * latest&greatest IP information from the server. After the map is loading,
72  * we are taken to a function that will download the server info.
73  *
74  * Arguments:
75  *   None.
76  *
77  * Returns:
78  *   Nothing.

```

```

79  */
80  var initialize = function() {
81
82      // Set some rough map options.
83      //var mapCenter = new google.maps.LatLng(37.8282, -95.5795);
84      var mapCenter = new google.maps.LatLng(40,0);
85      var mapOptions = {
86          mapTypeControl: false,
87          center: mapCenter,
88          zoom: 2
89      };
90
91      // Get the map container.
92      var container = document.getElementById('map-canvas');
93
94      // Draw the map.
95      var map = new google.maps.Map(container, mapOptions);
96
97      // Create an empty MVC array to pass into Google Maps.
98      var pointArray = new google.maps.MVCArray([]);
99
100     // Generate an empty heatmap Layer.
101     var heatmap = new google.maps.visualization.HeatmapLayer({
102         data: pointArray,
103         radius: 20
104     });
105
106     // Set the heatmap Layer down.
107     heatmap.setMap(map);
108
109     // Set some colors so that our map is more visible.
110     var gradient = [
111         'rgba(125, 0, 0, 0)',
112         'rgba(135, 0, 0, 1)',
113         'rgba(145, 0, 0, 1)',
114         'rgba(155, 0, 0, 1)',
115         'rgba(165, 0, 0, 1)',
116         'rgba(175, 0, 0, 1)',
117         'rgba(185, 0, 0, 1)',
118         'rgba(195, 0, 0, 1)',
119         'rgba(205, 0, 0, 1)',
120         'rgba(215, 0, 0, 1)',
121         'rgba(225, 0, 0, 1)',
122         'rgba(235, 0, 0, 1)',
123         'rgba(245, 0, 0, 1)',

```

```
124     'rgba(255, 0, 0, 1)'
125 ]
126 heatmap.set('gradient', gradient);
127
128 // Create the button at the top right for settings.
129 var opsDiv = document.createElement('div');
130 var ops = new OptionsButton(opsDiv, map, heatmap);
131 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);
```

---