Brandon White
CSCI 390
July 26, 2016

Over the course of the summer and the duration of my time with the Summer Internships in Science and Technology program at Fermi National Accelerator Laboratory (Fermilab), I was part of the Computer Security Team (CST). I was assigned in March 2016 to work with the Computer Security Architect, Jason Ormes, on the Beholder security program. Beholder is a security software that has been developed at Fermilab by Jason over the past three years. A homegrown monitoring system for cybersecurity prevention and detection which spans an array of servers and operating systems, Beholder takes input from many different cybersecurity streams, detectors, and networking protocol analysis programs in order to facilitate the effective blocking of unauthorized connections and software. The software also blacklists malicious Uniform Resource Identifiers (URIs) from being accessed or responded to, while maintaining concurrent monitoring of systems inside the Fermilab domain. Over the course of Beholder's development, Jason had not produced a consolidated way to view or monitor the statistics produced by the application. My task over the summer was to create a web page for Beholder that displayed consolidated monitoring statistics that were formatted in such a way as to be pleasing to the eye as well as useful to the CST in future cybersecurity operations.

The structure of the CST is that of a small task-focused group, working in close proximity to other small task-focused teams assigned to other projects in order to facilitate communication. The team is led by Joe Klemencic, assisted by Arthur Lee who together head up operations conducted by Jason Ormes, Greg Cisko, and Wayne Baisley. Each member of the CST is responsible for a different area of expertise, while also having a rotating duty schedule that consists of a "Primary" and "Secondary". Together throughout the week the Primary and Secondary act as Fermilab's cybersecurity quick reaction force. Wayne is responsible for hardware management which includes the setup, maintenance, and update of the servers necessary to the operation of Beholder and other CST software applications. Greg is the Incident Response Coordinator, responsible for the proper handling of cybersecurity related occurrences in the daily operations of Fermilab as well as the subject matter expert on issues relating to the security of systems running Windows operating systems. Jason, as the Computer Security Architect is responsible for the development and debugging of software applications so as to process new emerging new cybersecurity threats and automate the security analysis of systems within the Fermilab domain. The managerial staff comprised of Joe and Arthur assist in finding new vulnerabilities, discovering emerging threat streams, and writing new detector software to be integrated into the Beholder monitoring suite, while simultaneously coordinating the operations of the CST with higher administrative offices at Fermilab.

The CST exists outside of the rest of the Computing Division at Fermilab, residing directly under the management of the Chief Information Officer in order to facilitate the implementation of effective computer security techniques from the top down in the Fermilab administration structure. I found the organized, detached, small focus group approach to computer security to be an effective method to establish an easily enforceable cybersecurity policy. In a small group environment such as that of the CST, communication and cooperation between members is easy and efficient. This promotes an increased response time to cybersecurity incidents and easier adaptation to emerging threats.

Development of Beholder is done in Ruby 2.2.4 utilizing the Rails 4.2.6 framework on a dedicated host server and associated powerful database server, with a large cadre of necessary software included via gems and packages. Some of these include Redis, an in-memory data structure used as a database cache, and Sidekiq Enterprise, a software that supports the running of multithreaded asynchronous worker scripts written in Ruby.

Also required beginning with my redesign of the user interface was a large amount of Javascript, extensively utilizing the Jquery library along with Ajax calls and various other libraries in order to facilitate the loading and rendering of data. Beholder monitors for various new cybersecurity events which have an Intrusion Detection Systems Signature (IDS Signature). A specific IDS Signature is indicative of a likely detection of malicious attack against Fermilab systems which will in turn trigger an appropriate response. Also performed under the watch of Beholder are scans such as NESSUS Vulnerability Scans. These check for a multitude of different software vulnerabilities such as poor passwords, DDOS attacks, system misconfiguration, and penetration attempts. Further performed are Netsparker Web Application Security Scans which are responsible for the scanning of Fermilab web servers, Splunk log queries for security analysis, and NMAP port scans which check for ports that are open to connections from outside the Fermilab domain. Finally, all of this information is summarized by the new web application user interface that I was responsible for building utilizing a combination of Ruby, Javascript, HTML, and Bootstrap CSS with supporting API integration with various systems for data collection.

This user interface page is structured into a modular system in order to effectively display information useful to the monitoring of Beholder's daily operations. It is also useful to the operations of the Primary and Secondary over the course of the week. Each module is constantly refreshed with new information on set time intervals via Javascript, making the dashboard a near real-time display of Beholder's machinations. Furthermore, the modules are coded with Javascript so that when new data comes in that requires immediate attention, the module pertaining to that data changes position on the screen and is displayed first in order to draw attention to itself. After a set time period, the module will return back to its original position in the module hierarchy displayed on the screen by default. There are currently nine separate modules with more currently in the design phase to be finished by the end of my internship on August 19th.

In the standard display, with all modules in their default positions, the most important module is the Scanner Farm Status module. This module, styled with Bootstrap CSS as are the rest of the modules, maintains a display of the total number of systems currently being scanned by the Voyager server and all six of its worker nodes. Three of these nodes are within Fermilab's network, and scan for vulnerabilities that exist inside the domain. Conversely, the other three nodes exist in the "demilitarized zone" outside the Fermilab domain, allowing for scans of the Fermilab domain from a "rest of the world" point of view. On both sides, production and DEEMZ, these numbers are further broken down into the number of new scans and rescans. A new scan is conducted whenever a new device is detected on the network, usually within seconds of the device being connected to the Fermilab domain. Rescans are conducted on devices which have not left the network on a frequency of every three days. Normally all numbers are displayed

in green. However, should the number of scans drop below 25 at any given time, the text is changed to red in order to alert the operator that there is likely something broken. This module also has a dropdown menu to select between the numeric view and the historical view. When the historical view is selected, Javascript will on the fly change the data displayed in the module to a line chart rendered by the Google Charts API displaying the number of scans per hour over the last twenty-four hours. This has proven useful multiple times already as an easily identifiable indicator that there may be an issue with the proper operation of the scanner farm. Data acquisition for this module required me to familiarize myself with the Ruby API for the Splunk search and analytics software and ended up requiring a fair bit of debugging in order to work properly. Issues included connecting to Splunk in the first place and then ascertaining the appropriate logic in which to issue a very specific query to Splunk then in obtaining a usable dataset to be processed by an asynchronous Sidekiq worker for its content.

The next module in order is the Event Summary module. This module consists of Bootstrap datepicker fields that allow the selection of a date range with a default value of the last seven days. Utilizing this date range, the Beholder database is queried for events that have been created in this time frame in order to produce another Google rendered chart depicting all the various types of cybersecurity events that have occurred and the percentages associated with those types during the time specified. Once again, Javascript detects any change in the date range specified by the Bootstrap datepickers and then queries the database again through a series of jQuery and Ajax calls for data pertaining to the updated range. Some issues that occurred during the implementation of this module included implementing the datepickers correctly in the first place while posting the information contained in each field to the Ruby controller through Javascript correctly. Later issues occurred after implementation of manipulation of the module order, and pertained to Javascript binding to the datepicker fields at a time when that part of the DOM had not yet been rendered. This was due to the HTML being stored in a Javascript hash at the time for rendering and the Javascript attempting to pull values from datepicker fields that either did not exist, or outdated values in those fields.

Following this module in the default order is the Sidekiq Status Center. This module utilizing the Sidekiq API written in Ruby allows for monitoring of the Sidekiq Enterprise queues and job statuses. This module follows a similar styling to the Scanner Farm Status module in that during normal operation, all data is displayed in green. However, when Sidekiq jobs begin to hang, block, or fail all text within the module will turn red in order to alert the Beholder operator to the issues at hand. Currently Beholder runs approximately fifty-thousand Sidekiq jobs per day. This makes Sidekiq invaluable for writing short Ruby scripts for jobs that need to perform asynchronously in the background frequently while allowing Beholder to continue normal unblocking operation. This invaluability resulted in Fermilab upgrading from the free version of Sidekiq to Sidekiq Enterprise during my internship tenure, which took Sidekiq from one process running twenty-five threads to eight simultaneous processes cooperating in a swarm, each of which runs with up to twenty-five simultaneous threads. This allows Sidekiq to have a monstrous

throughput which is necessary for the speed at which Beholder operates. This module was actually relatively simple to implement. The only quandary which required a healthy amount of attention was during the upgrade from Sidekiq to Sidekiq Enterprise. This was due to things such as the enterprise version having built in support for UNIX style cron jobs, as well as slightly different configuration and initialization procedures.

Next in line is the Recent Black Hole Events module. This module contains a table that depicts the last six hours of events which have been "blackholed". When a DNS request resolves to, or an HTTP response is sent out from a system in the Fermilab domain to a blacklisted target location, that system is instead redirected to a safe Fermilab owned page which displays a warning that the site being accessed is blacklisted and that the event has been logged. This effectively blocks systems from not only accessing malicious web pages, but also serves to stop a malicious system from ever receiving a response from a probe of Fermilab's cyber security defenses. This module also contains a button labeled "Geolocation". When clicked, this button opens a modal Bootstrap window containing a Google Charts rendered map of the Earth. Displayed on this map is the location of each of the Black Hole events contained in the table, geocoded by IP address using a free IP geocoding service with a Ruby geocoding gem on GitHub. While quite interesting, this location data does have one weakness in that it cannot geolocate an address that originates from behind a proxy server. This module was interesting to implement in that creating the geocoding functionality appealed to me due to my background in Geospatial Information Systems. The most difficult part of this creating this module was in querying the Beholder database for the last six hours worth of Black Hole events. This was due to the fact that we had to query based not just on time in one field, but on time, event, and the IDS signature in a complicated join scenario. Also of some note in difficulty was that while writing this module, this was the first time utilizing Google Charts in the Beholder project since its inception. Figuring out how to convert a Ruby array of information into a Javascript array formatted in a way that Google's API could understand took a good chunk of time to ascertain.

The next module in the lineup is a special case of module. Depending on conditions, this spot is either taken up by the Webscan Targets module, or the Current Web Scans module. When no current scans of HTTP servers at Fermilab are under way, the Webscan Targets module is displayed. This renders a table containing the upcoming schedule web scans, including the hostname, IP address, port to be scanned, and a column to monitor if the scan has occurred yet. When a scan of an HTTP server occurs, the module is replaced instead with the Current Web Scans module. This displays the time that the scan began at, along with the IP address and the target of the scan. This module is useful as many web scans occur in batches based on the time before a rescan of a particular node is due. The dynamic property of this module implementation allows for real-time monitoring of the status of the NMAP and Netsparker scans on the plethora of HTTP servers that Fermilab is running inside the domain at any given time. These two modules were quite simple to implement in that there was only a single query for each of them

due to the way that Beholder handles the processing of HTTP server scans through its various systems.

Following the Webscan Targets and Current Web Scans module is the Tissue Status Center. This allows for monitoring of tickets related to another Fermilab system that handles the blocking of computers that violate the cybersecurity policies in place here. This is useful to the Beholder operator in that sometimes Beholder is responsible for the creation of a Tissue event. These Tissue events may require immediate action by a member of the CST in order to respond to the variety of cybersecurity needs that arise on a daily basis. This module is implemented as another table that uses the API created by Fermilab's developers assigned to Tissue in order to gather data for display. Creation of this module, which required coordination with the Tissue developers, was a great insight into the interoperation of the development teams here at Fermilab. My implementation of this module was easy in that all that was required was collaborating with those developers, and having them slightly modify their JSON output in a way so that it could be accessed and utilized inside of Beholder.

The next module is titled Recent Webscan Information. This module takes the results of the web scans conducted by Netsparker and displays them color coded based on severity of the vulnerability discovered. Of the five categories of severity numbered one through five, one being delineated as a critical vulnerability and symbolized by turning that row of the table red. A level two vulnerability is categorized as important symbolized orange, and a level three is classified as medium symbolized as yellow. A medium vulnerability may be something such as an outdated version of SSL or an Open Client Access Policy. A critical vulnerability on the other hand would be a Confirmed SQL Injection attack or an out of date Content Management System. As was the case with the other modules relating to the status of the Web Scans being run by Beholder, implementation of this module was simple in that data acquisition only required a single simple query to obtain an informative dataset.

The final module that I designed for Beholder was the Netflow Information module. This module is responsible for keeping track of the total business internet traffic that crosses the border router inbound and outbound from Fermilab. This traffic does not include scientific data feeds from CERN or other scientific institutions as the volume of that data is far too large and causes too many false positives for the CST to manage. These numbers are broken down into inbound and outbound values, which are then broken down into minutely, hourly and daily values. This module also has the ability like the Scanner Farm Status module to switch to a historical view, in which data is sent to Google to be rendered as a line chart depicting the netflow information for the previous seven days. This module required some work to implement. Data acquisition starts on the Ebb and Flow servers, which are constantly fed netflow information through port 50001. By utilizing a script written by Jason, this information is continuously parsed on a by-the-minute basis and imported into Beholder for use. This data enters Beholder as a number of bytes, which are then converted to megabytes for easier reading.

Total traffic inbound and outbound related to normal day to day business activities at Fermilab averages between two and four terabytes daily.

On July 15th, I gave a presentation at the weekly Computer Security Board meeting in which I demoed the capabilities of the page I had designed. While giving briefs and presentations in front of higher echelons of leadership is not a new experience to me, it was a good experience to do it in a different environment than I had grown used to in the past. The entirety of the presentation and all the associated questions lasted around twenty-five minutes and was very well received by both leadership of the CST as well as that of the Computer Security Board.

This internship has been an extremely informative learning experience, allowing me to gain knowledge and extensive experience with a variety of operating systems, languages, networking protocols, and software packages, while simultaneously providing for undergraduate attendance at twice weekly lectures relating to particle physics topics such as Neutrino Mass Ordering, Gravitational Waves, and Particle Physics at the Compact Muon Solenoid located at CERN. Fermilab has a very capable cybersecurity presence in the form of the CST and is unsurprisingly without equal in the ability to provide students hands on experience with fantastic hardware and software suites managed by teams of extremely skilled professionals, while still providing an opportunity for students to produce meaningful work. I cannot understate the knowledge and professionalism of the employees here as from day one I knew that I would have to step up my own self learning in order to even begin to understand the daily machinations here and Fermilab and below the surface in the Beholder software.