Honeypot HTTP Traffic Analysis

Dakota Nelson and Ian Hill
{dakota.nelson | ian.hill}@students.olin.edu
April 30, 2015

Introduction

Over the course of three months from July to September 2014, AWS honeypots created by L3 Data Tactics collected over 7500 HTTP hits. (The honeypot servers are each an AWS instance running a web server with no content on it set up to detect anomalous network traffic, and a hit is one HTTP request to a server.) Given that there were no legitimate uses for these servers, this traffic is, by definition, unusual. In this report, we investigate trends and relationships among this traffic with the goal of developing an understanding of the data that will enable further research and development in statistical modeling of background Internet traffic or identification and characterization of attacker groups.

The data was delivered in two files, one of web server logs, and one of additional data regarding each IP address that interacted with any of the servers.

Logfiles:

'timestamp', 'server', 'hit_type', 'server_ip', 'server_timestamp', 'http_method', 'requested_resource', 'http_version', 'resp_code', 'resp_bytes', 'hostname', 'user_agent', 'timestamp_obj'

Here is an example record:

Jul 29 03:31:15	dev-server	httpd_access	127.0.0.1	29/Jul/2014:03:31:15 +0000	GET	/.git/HEAD	HTTP/1.1	404	207	NaN	Mozilla/5.0 (compatible; Nmap Scripting Engine...	2014-07-29 03:31:15

IP data:

'ip', 'domain', 'org', 'isp', 'user_type', 'is_anonymous_proxy', 'accuracy_radius', 'continent', 'country', 'geoname_id', 'longitude', 'latitude', 'time_zone'

For example:

127.0.0.1	NaN	website.com	hosting	NaN	NaN	937	NaN	US	4744870	-77.4875	39.0437	America/New_York

note: The IP address 127.0.0.1 shown in the examples is there only to illustrate an example entry. IP addresses from the actual data were not used in this report.

Findings and Discussion

We computed basic counts and statistics about the honeypot servers and their traffic patterns to build a foundational understanding of the data. To get a sense of how much traffic hit the servers over the timeframe covered by the data, we binned the data into weeks of the year. Figure 1 is a time series which shows the number of hits the entire server collection received each week.

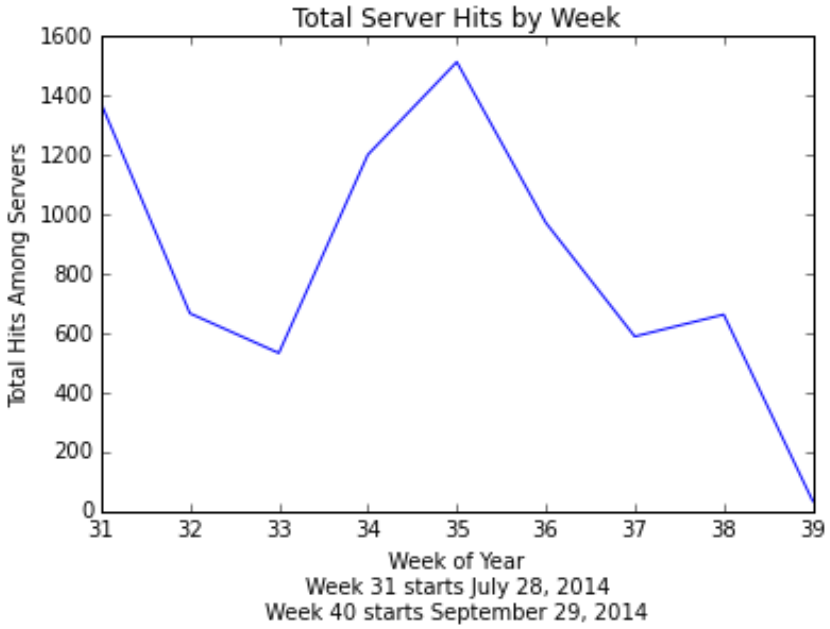


Figure 1: Volume of HTTP traffic to the honeypot servers.

Of course, the number of HTTP requests to each server is dependent on the server's uptime. Given the dates of the first and last HTTP requests, the servers seem to have been launched in two blocks: one server block that was launched in July and another that was launched in September. Figure 2 shows the first and last recorded HTTP requests on each honeypot server.

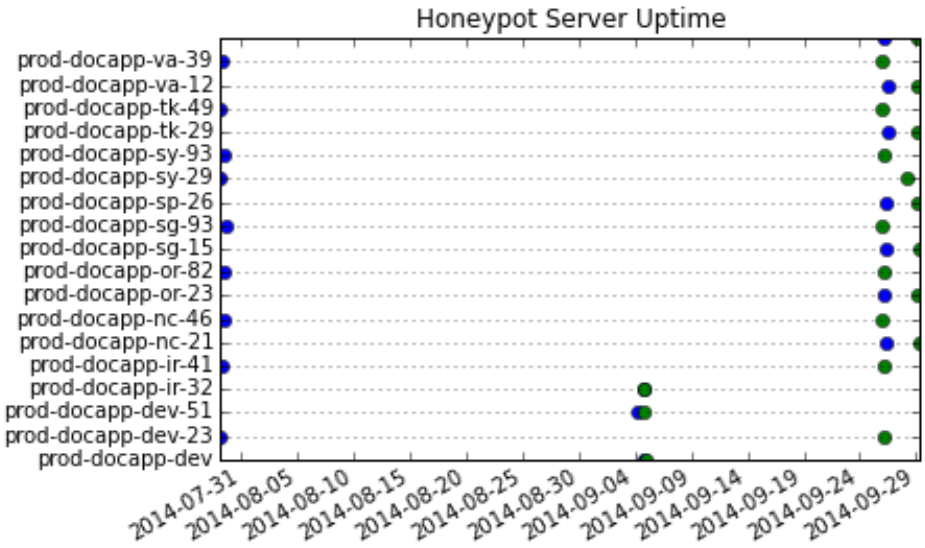


Figure 2: First and last log entry for each honeypot server. First observed entry is blue, last is green.

The data only contains records until the end of September, so naturally servers in the block launched in September have fewer HTTP requests on record. The existence of two launch blocks can be seen clearly in Figure 3, which shows the total hits per server.

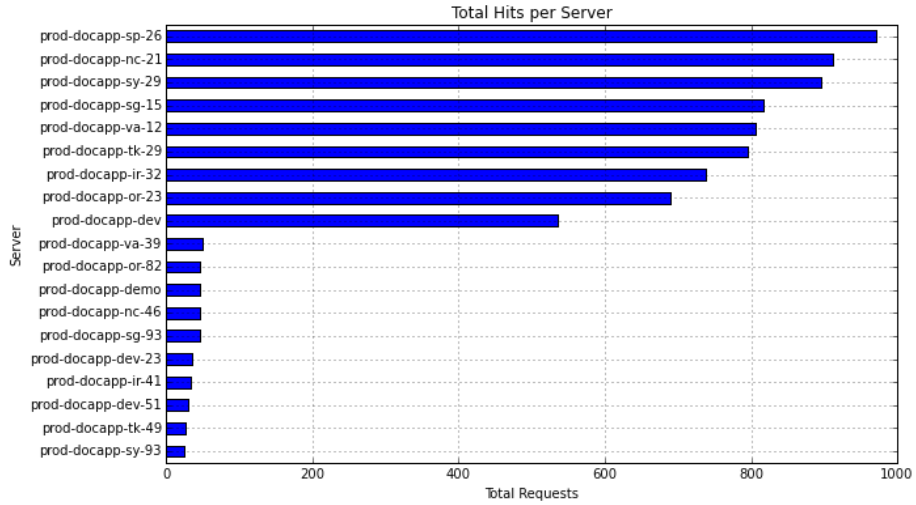


Figure 3: Total number of hits for each honeypot server.

Calculating rates provides us with a time-invariant view, allowing equal comparison of the servers. Each honeypot server has a roughly similar traffic rate of about ten hits per day. Three glaring exceptions exist; ‘prod-docapp-demo’, ‘prod-docapp-dev-23’, and ‘prod-docapp-dev-51.’ These three servers have much higher hit rates, but also have a

different naming convention from the rest, as well as unusual uptimes (see Figure 2), suggesting they were used for development and testing of the honeypot network.

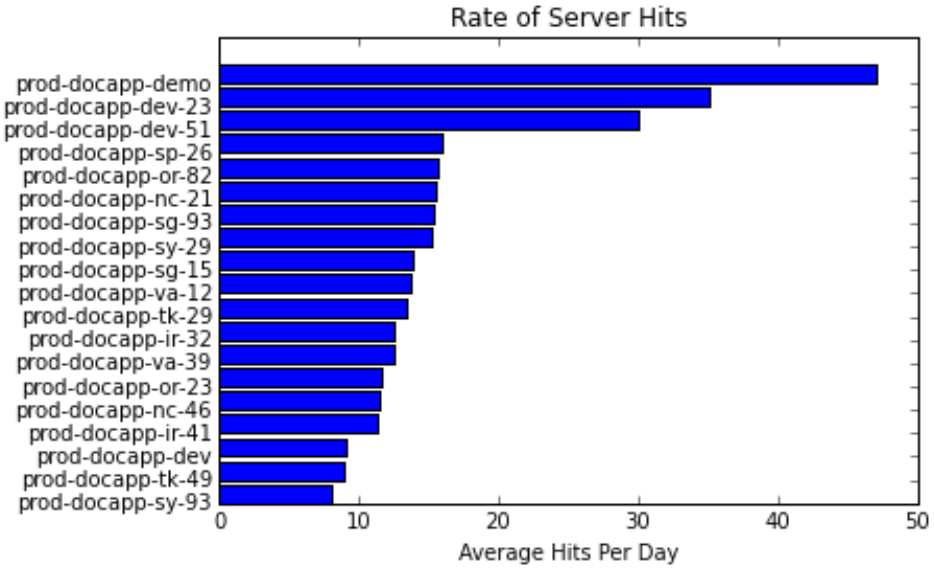


Figure 4: Average hits per hour over the course of each server's uptime, measured from the first to the last observed hit.

In addition to understanding the characteristics of the honeypot traffic, Figures 5 through 7 display information about client locations and requested resources. In this paper, 'clients' are the remote computers (represented to us by IP addresses) making requests of the honeypot servers. Requested resources can be used to determine the intent of the request, and we wanted to know if there was a relationship between the client locations and the quantity of anomalous network traffic.

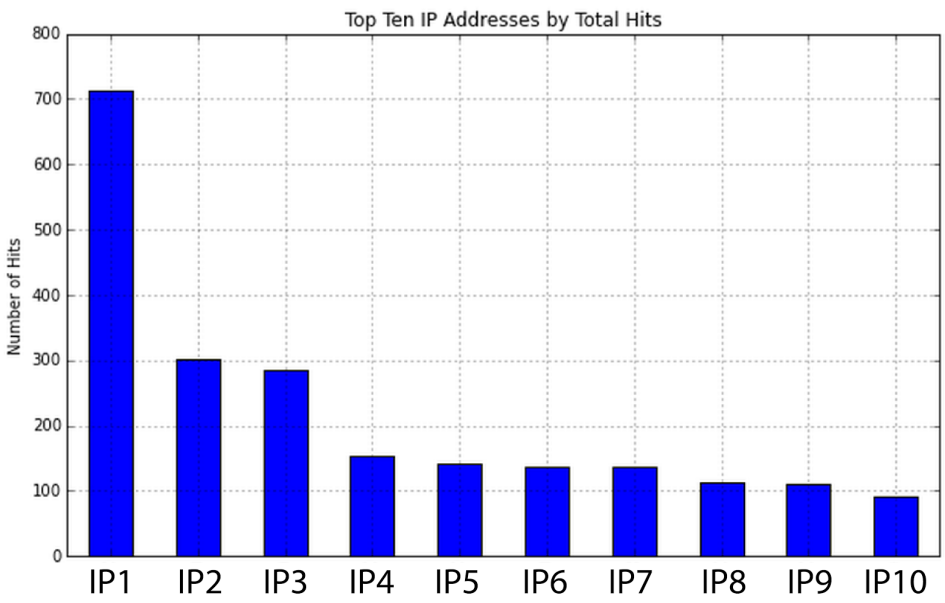


Figure 5: The top ten IP addresses by hit count.

Geolocation data for the clients is provided by L3 as part of the initial dataset. The actual method used for geolocation is unknown to us, but is likely to be a reverse IP lookup.

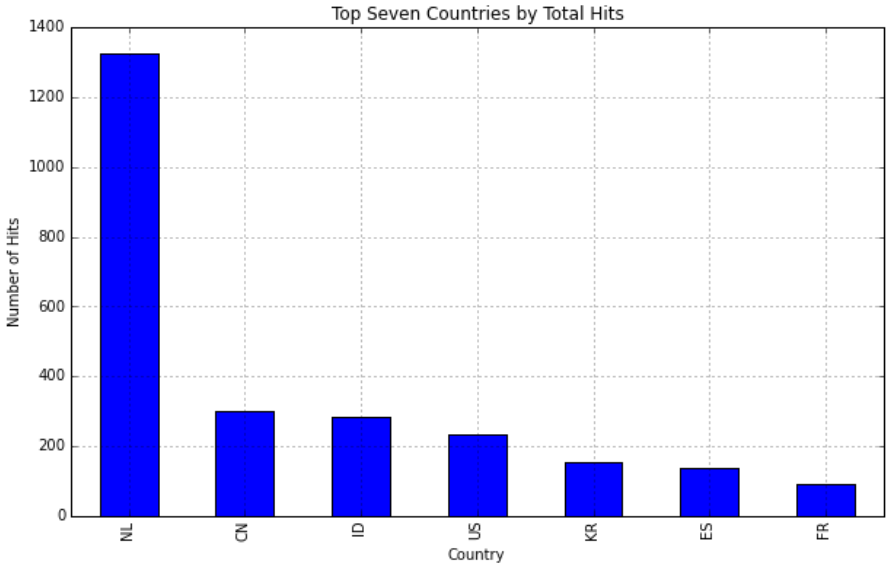


Figure 6: Top seven countries by hit count.

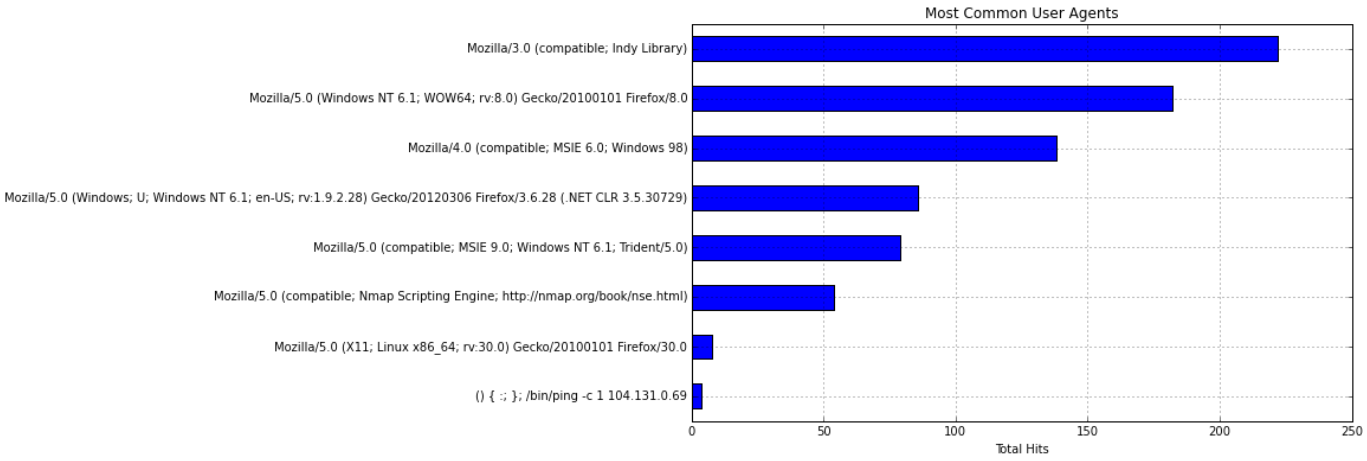


Figure 7: Eight most common user agent strings.

In most cases, the requested resource allowed us to estimate the legitimacy of the recorded network traffic because many clients requested to access to known restricted systems. The intent of the requests can largely be determined by a simple Google search of the requested resource. Request including “php” are largely probes or attacks on PHPmyadmin or PHP code injections. Other requests attempt to detect the existence of a git repository that could be exploited. Interactions requesting “/web-console/ServerInfo.jsp” or “/jmx-console/HtmlAdaptor” are attempts to access a JBOSS web application server. Similarly, interactions requesting “/manager/html” are attempts to access the management console for a Apache Tomcat web server. On the other hand, requests for “/robots.txt” and root are more likely to be non-malicious traffic attempting to map the web. Figure 8 shows the counts of the 15 most requested resources.

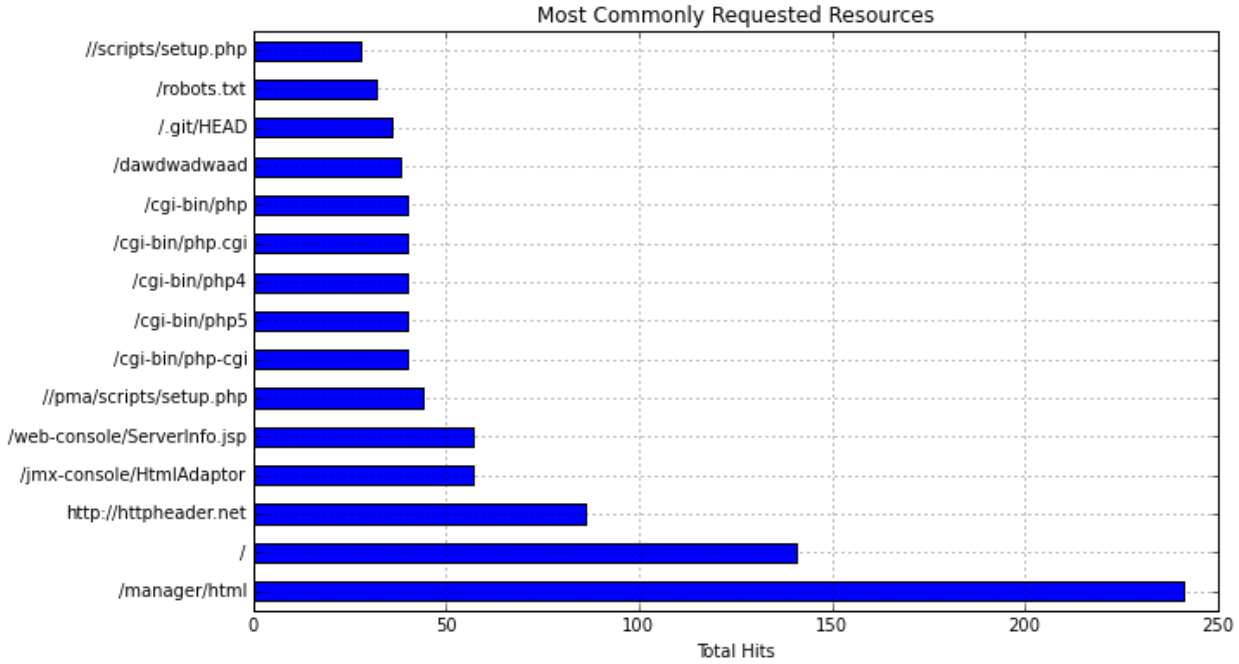


Figure 8: Most commonly requested resources.

More than 40% of hits on the honeypots probed PHP vulnerabilities, over 12% attempted to discover Apache Tomcat consoles, and 20% of the hits requested the server's root path. Figure 9 shows the prevalence of various types of HTTP hits across all of the honeypot servers. The criteria for request categorization are shown in Table 1.

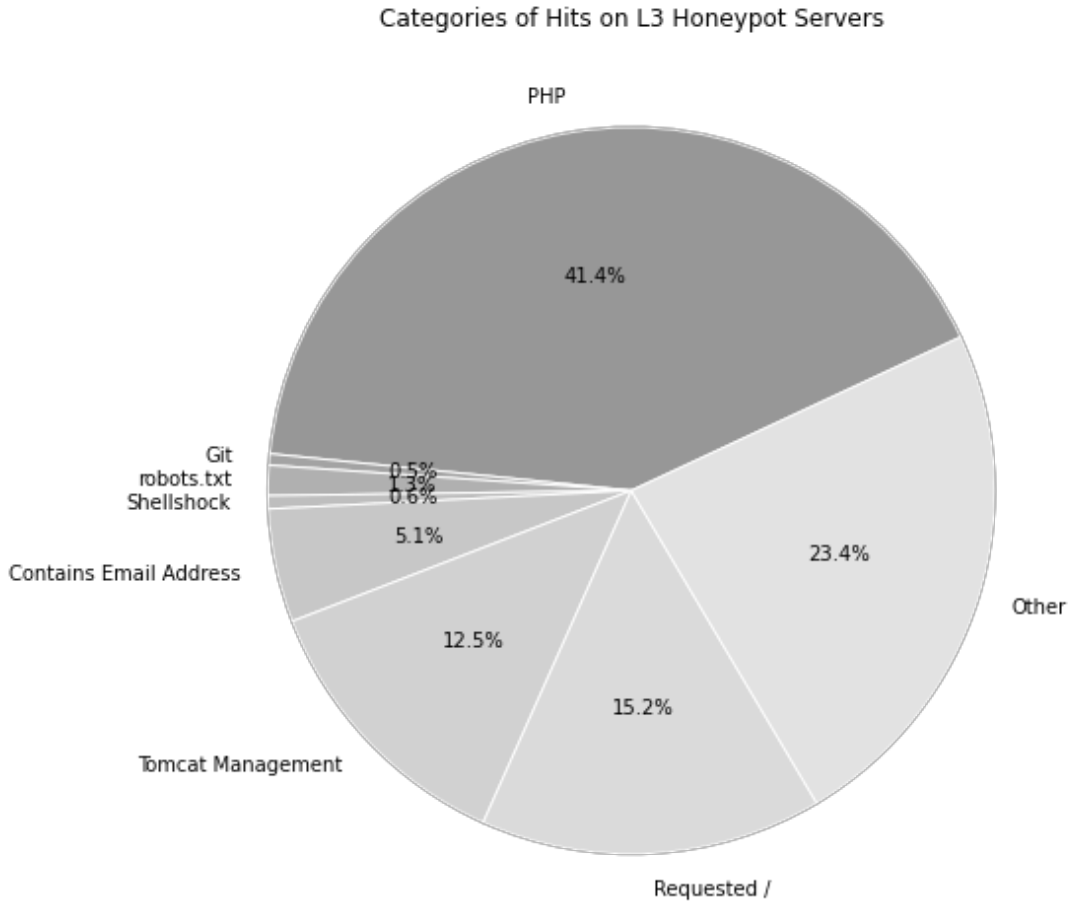


Figure 9: Server hits categorized.

Category	Criterion
PHP	Requested resource contains 'php'
Git	Requested resource contains 'git'
Robots.txt	Requested resource contains 'robots.txt'
Shellshock	Requested resource contains '{ :;};'
Contains Email	Requested resource matches '.+@.+\..+'

Tomcat Management	Requested resource is ‘/manager/html’
Requested Root	Requested resource is ‘/’
Other	All requests that don’t fit into any of the above categories

Table 1: Honeypot server hit categories.

Having categorized these hit types, we then discovered something odd about the data - when clients request the root path (‘/’), a variety of response codes are returned, with a variety of response lengths, as shown in Figures 10 and 11. Further exploration confirmed this was neither caused by differences in servers (all servers had a mix of response codes) nor by differences in request method (most HTTP methods had a mix of response codes). Without access to the data gathering machinery, we are unable to determine what caused this unexpected scenario.

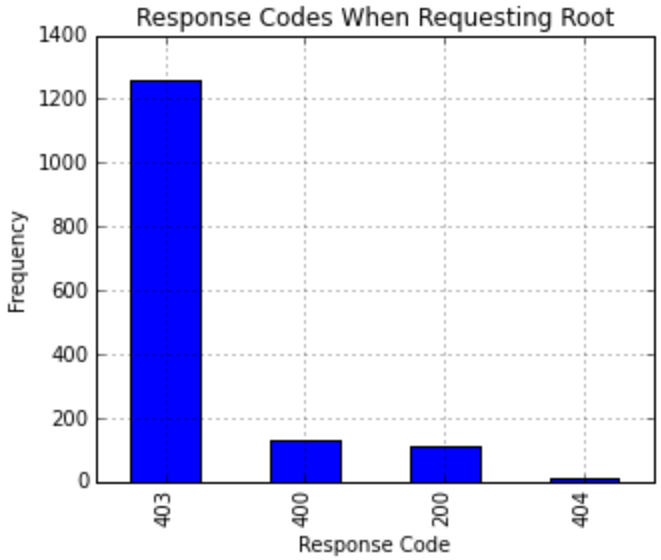


Figure 10: Response codes returned to clients after requesting root.

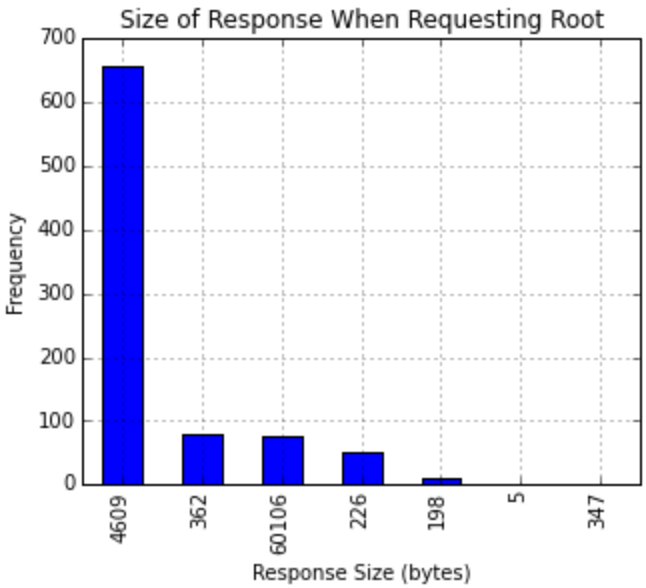


Figure 11: Size of responses to clients after requesting root.

Location Based Characteristics

We also investigated whether or not the location of a client impacts when its traffic to the honeypots occurs. Here, we compare traffic from the US to traffic from China. These countries have a large volume of traffic (1650 and 1572 hits, respectively) and a large time zone difference.

The difference in median server hit time, in seconds since midnight, between the US and China is 3019 seconds - just over 50 minutes. Given the small difference, we conclude that the time traffic arrives at the honeypot servers is independent of the time zone it originates from.

Length of Resource Requested

The length of the resource any given hit is requesting (e.g. ‘/’ vs ‘/manager/html’ vs ‘/robots.txt’ and so on) is valuable information. Long requested resources tend to be more suspicious, as no client should really know what would be on the server other than root. As the lengths start to get very long, it becomes more obvious that a POST request is attempting some sort of injection attack. The distribution of request lengths is shown in Figure 12.

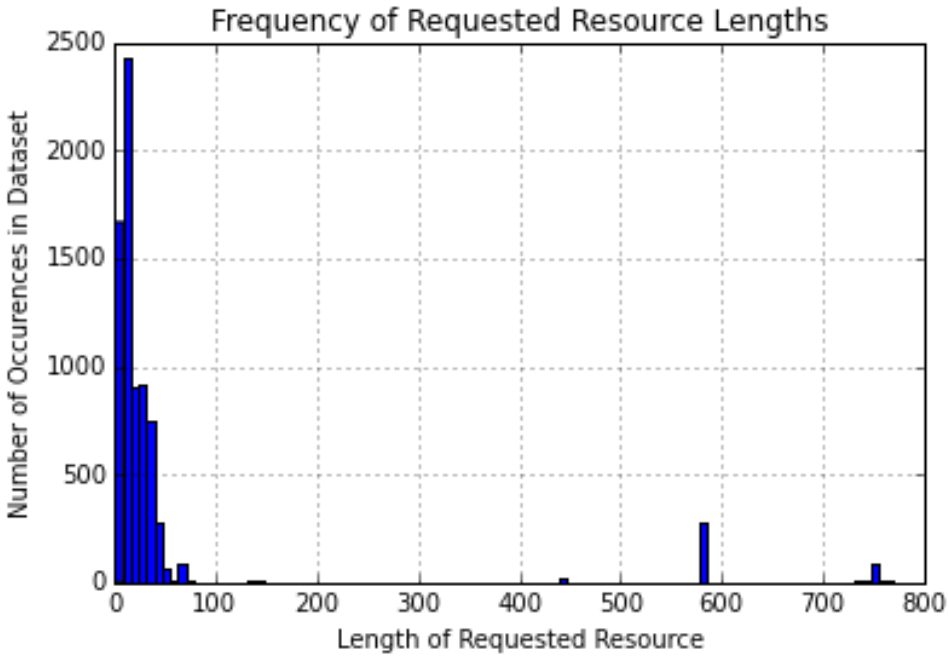
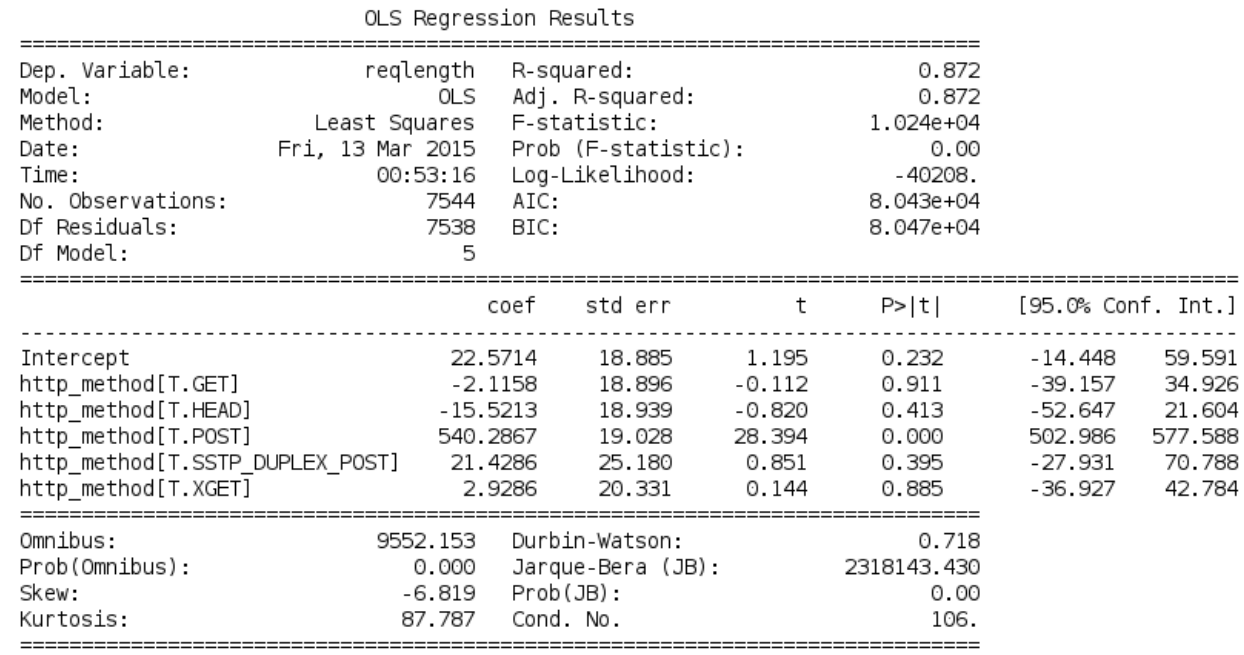


Figure 12: Distribution of requested resource lengths in the dataset.

To take this analysis further, ordinary least squares linear regression is used to determine what factors most strongly affect request length. The length of the requested resource is computed, and then regression run on other chosen factors to determine their effect on request length.

The results shown in Figure 13 below display the effect of each request type upon request length. The ‘coef’ column provides the effect size - in essence, to get a best estimate of a request’s length if you only know its method, start with the intercept and add the corresponding coefficient. The other columns provide error and certainty information.

For example, in the regression on http_method shown in Figure 13, it can be seen that if you know a request’s method is POST, you should add 540 characters to your estimate of what the requested resource length will be - strong information that POST requests are longer than the other methods. This is not particularly surprising information, but nonetheless supports the observation that there are many attempted POST request injection attacks, which tend to be far longer than other request types. None of the other results are statistically significant.



OLS Regression Results

Dep. Variable:	reqlength	R-squared:	0.872
Model:	OLS	Adj. R-squared:	0.872
Method:	Least Squares	F-statistic:	1.024e+04
Date:	Fri, 13 Mar 2015	Prob (F-statistic):	0.00
Time:	00:53:16	Log-Likelihood:	-40208.
No. Observations:	7544	AIC:	8.043e+04
Df Residuals:	7538	BIC:	8.047e+04
Df Model:	5		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
Intercept	22.5714	18.885	1.195	0.232	-14.448	59.591
http_method[T.GET]	-2.1158	18.896	-0.112	0.911	-39.157	34.926
http_method[T.HEAD]	-15.5213	18.939	-0.820	0.413	-52.647	21.604
http_method[T.POST]	540.2867	19.028	28.394	0.000	502.986	577.588
http_method[T.SSTP_DUPLEX_POST]	21.4286	25.180	0.851	0.395	-27.931	70.788
http_method[T.XGET]	2.9286	20.331	0.144	0.885	-36.927	42.784

Omnibus:	9552.153	Durbin-Watson:	0.718
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2318143.430
Skew:	-6.819	Prob(JB):	0.00
Kurtosis:	87.787	Cond. No.	106.

Figure 13: Ordinary least squares linear regression run on request length vs. request method.

Time Based Characteristics

We hypothesize that there might be a relationship between the number of attacks on a given day on one server and the number of attacks on the other servers. More specifically, we wondered if server A was attacked n times by a certain category of attack on a given day, would server B be attacked approximately n times by that same category on the same day.

The data contains records of every outside interaction with each server. We separated these interactions, or ‘hits’, into separate pandas DataFrames each representing a different kind of attack. We then computed the number of hits each server had per day within a given DataFrame. We selected a benchmark server against which the hits of other servers on a given day could be compared. We created a scatter plot of points representing every day where the benchmark server and another server had been attacked by a certain method. For example, data point (10,8) represents a day where the benchmark server was attacked by php injection 10 times on the same day that another server was attacked by php injection 8 times. Figure 14 shows the days where any of the other servers also had attacks on the same day as prod-docapp-sp-26, the benchmark server for this plot.

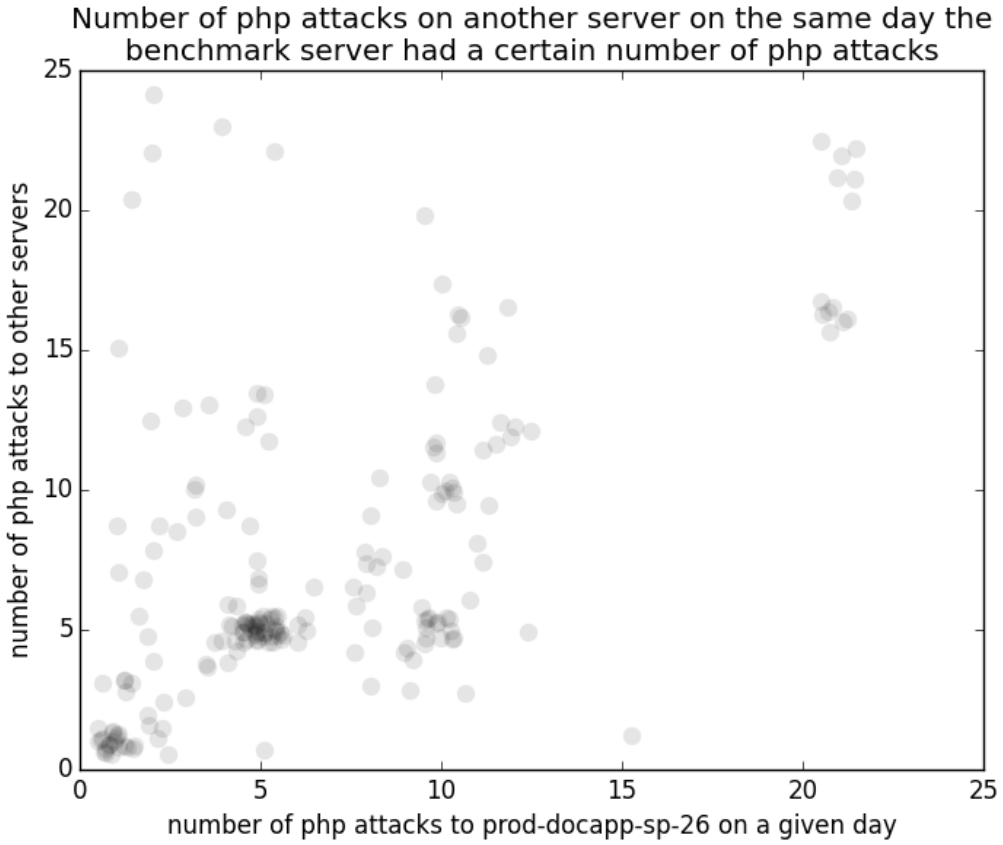


Figure 14: A scatter plot of points representing every day where prod-docapp-sp-26 and another server had been attacked by php injection.

We then computed the Pearson Correlation Coefficient for each attack type using various servers as the benchmark.

Attack Type	Pearson Correlation
PHP with prod-docapp-va-12 as benchmark	0.71
Tomcat Manager with prod-docapp-sp-26 as benchmark	0.47
Requested / with prod-docapp-sp-26 as benchmark	0.38

Table 2. Pearson's correlation coefficient of the data from similar scatter plots showing various types of traffic.

These results support our hypothesis. The correlation coefficients are sufficiently high with some types of attacks to be meaningful but are not high enough to declare that here is a definite relationship. It is worth noting that many attacks came from a small number of sources whose simultaneous broadcast attacks might have influenced the results. More data would be necessary to further explore this hypothesis.

Having made curious discoveries while exploring the timing of attacks on the servers, we analyzed the data to determine if there was a pattern among repeat clients to the servers. For the server, prod-docapp-sp-26, we found there to be some seasonality with which clients interacted with the server again after having not interacted for at least one day. Figure 15 shows a PMF of the number of days between the first HTTP hit from a unique IP address and other repeat hits from the same IP address that were at least a day apart. For example, if a client attempted a php injection on Monday and then attempted to do the same thing on Friday, that repeated attempt would count as a follow up attack 4 days after the first interaction with the server.

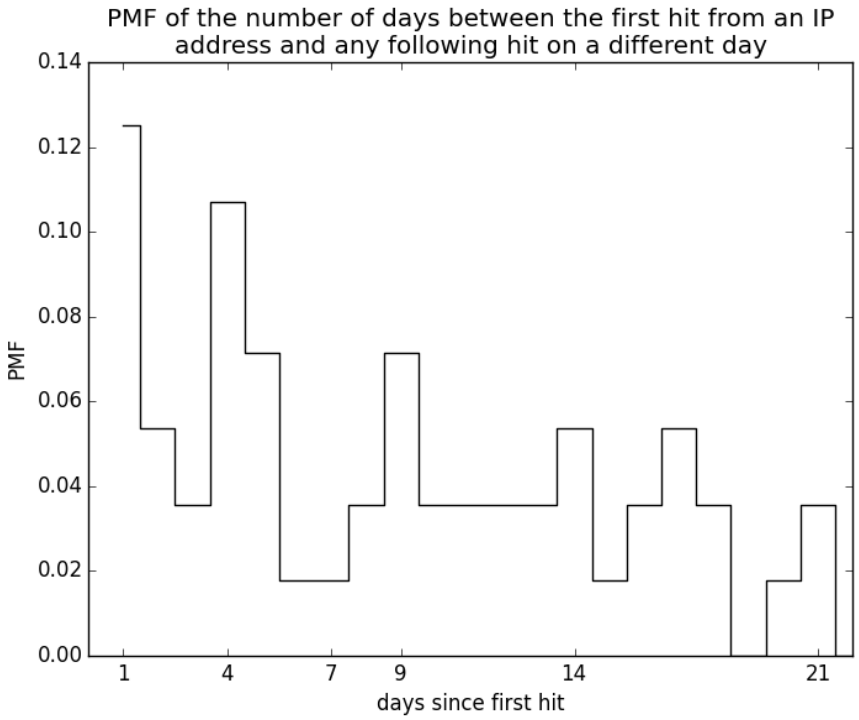


Figure 15: PMF of the number of days between the first HTTP interaction from any unique IP address and any interaction after the initial interaction from the same IP address on a different day with server, prod-docapp-sp-26.

Notice the peaks at 14 and 21 days. Based on the limited data displayed in the PMF above, clients appear to following up with the server after 2 weeks and 3 weeks. While the valley at 7 days does not support this conjecture, the seasonality of client interactions appears to be weekly after the first week that the server was discovered by the client. Seasonality could be useful for predicting future attacks.

Text Field Comparison

Much of the information in each hit, and especially information-dense fields such as user agent strings and requested resource, is raw text, necessitating different comparison and analysis methods. Figures 16 and 17 are matrices displaying the Levenshtein ratio (normalized Levenshtein distance) between the requested resource and user agent strings for 500 hits. Every string is compared against each of the others - along the diagonal, the strings are compared against themselves, always resulting in a ratio of 1, or a perfect match.

Figure 16: Levenshtein ratio between requested resources. Note that this plot is symmetrical about its diagonal, and that the diagonal cells are all 1. A higher ratio indicates a more similar pair of strings.

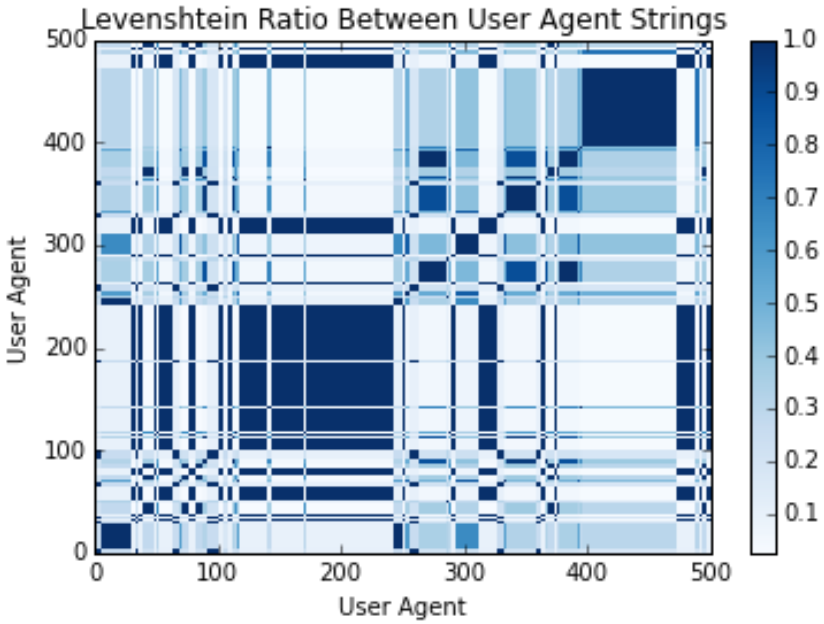


Figure 17: Levenshtein ratio between user agents. As in Figure 16, this plot is symmetrical about its diagonal. A higher ratio indicates a more similar pair of strings.

The matrices above are reduced use principal component analysis, from *n* features to 5, then used as inputs for a k-means algorithm, which we use to cluster the data into five distinct groups. (The number five was chosen arbitrarily after some experimentation.) The resulting clusters appear to effectively group the honeypot hits into clusters with similar characteristics (especially interesting given that similarity between requested resource and user agent are the only features used).

This clustering represents the first possible step toward clustering and characterizing incoming traffic, potentially identifying groups of attackers working in concert. However, there is no available ground truth for the clusters, making any sort of evaluation an almost entirely subjective exercise.

Temporal Clustering

With the goal of finding botnets or other coordinated malicious sources of internet traffic, we attempted to group IP addresses by running K-means clustering on a scoring matrix derived from re-codes and comparisons of the timestamps of the http requests of each IP address. We created an *n* by *n* scoring matrix where *n* was the number of unique IP addresses. Each IP address was mapped to each other IP address, and each mapping was scored with our own homemade algorithm which quantified how likely we thought the 2 IP addresses might be colluding. The scoring algorithm measured three qualities re-coded

exclusively from the timestamps of the http requests when comparing each pair of IP addresses. We fed the scoring matrix into the K-means algorithm implemented in the scikit-learn library in order to produce clusters which we hoped would represent botnets.

We developed the scoring algorithm and its metrics by drawing on our domain knowledge and intuition of malicious networks and botnets. Ultimately, we were trying to extract additional information from a single one-dimensional attribute of our http request data set. We assumed that temporal distance, request order, and seasonality would have some correlation to whether or not two IP addresses were colluding. Admittedly, we have relatively limited domain knowledge, but we are hopeful that this method might allow a data scientist at L3 Data Tactics with more extensive domain knowledge to uncover insights from the data.

The first scoring metric was simply a binary score of 1 or 0 determined by whether or not the first request from the second IP address occurred after the first IP address. For example, the scoring element (4,6) would be increased by 1 scoring unit if the first recorded request from the 6th IP address occurred after the first recorded request from the 4th IP address. Here, we assumed that an IP address was more likely to be collaborating with a previously seen IP address because the first IP address could have communicated behind the scenes the existence of the honeypot server to the new IP address. Given our relative uncertainty with this assumption, this metric was given a low maximum weight of 1 scoring unit.

The second scoring metric quantified the temporal distance between the nearest http requests of 2 IP addresses. If the 2 http requests were within 7 days of each other, the mapping would be scored an additional number of points between 0 and 5 depending linearly on how close they were. For example, the scoring element (4,6) would be increased 2.5 scoring units if their nearest http requests were 3.5 day apart. We assumed that 2 IP addresses might be related if they each made requests to a server within 7 days of each other. Admittedly, our choice of a 7 day cut-off was completely arbitrary. We decided that the temporal proximity of the http requests indicated a high likelihood of a connection between the 2 IP addresses and therefore weighted it relatively highly with a maximum weight of 5 scoring units.

The third scoring metric was difficult to apply to the matrix because it only applies to a fraction of the IP addresses in the dataset. If an IP address made multiple http requests, we computed the average time, in seconds, between each request and assigned that average “delta” to the IP address. We then compared that average delta to the average deltas of all other IP addresses for which we could compute the delta and scored the pair highly if their average deltas were similar. If the 2 average deltas were within 2 days of each other, the mapping would be scored an additional number of points between 0 and 5 depending linearly on how close they were. For example, the scoring element (4,6) would be increased 2.5 scoring units if the average delta for the 4th IP address was 7 days and the average delta for the 6th IP address was 6 days. Since this average delta was effectively the seasonality of the

http requests from a particular IP address, we scored this metric highly as well with a maximum of 5 scoring units.

Once the scoring was computed, we used the matrix as input for the K-means object implemented by the scikit-learn machine learning library. Arbitrarily, we asked it to produce 10 clusters. Unfortunately, it is impossible to tell if the clustering actually detected a botnet because if we knew IP addresses comprised a botnet we would not be looking for them. That said, the attributes of the IP addresses within the clusterings did seem to show some similarities even though the scoring was exclusively based on the timestamp of the http requests of each IP address. For example, the method described here clustered a large number of Amazon.com hosted IP addresses into cluster 0 without ever using the country, domain, or organization in its scoring algorithm.

In the final results of this temporal clustering exploration, there were 10 clusters. The results of the clusterings here are inconclusive, but we believe that further analysis of the timestamps of http requests deserves further study.

Future Work

Several areas of promise exist to carry on the work conducted so far.

More Diverse Data

The addition of further data sources - for example, non-HTTP traffic - would enable further probing into a broader definition of “interesting activity.” It’s very possible that these HTTP requests are coordinated with different types of activity that fuller monitoring and datasets would uncover.

Better Software

Finally, operationalization of the software developed in this exploration would likely incur prohibitive computational costs. The current Levenshtein distance computation in an $O(n^2)$ operation by itself - just for a single feature, before clustering is even carried out. Transitioning into a viable operational software package would require exploration of methods to reduce the complexity of the process used.

Summary

In exploring the dataset given to us by L3 Data Tactics, we have found a variety of ways to re-code the attributes of HTTP requests and extract non-obvious value from them. We have gained an understanding of various forms of potentially malicious internet traffic and the patterns in which they are used.

We learned that, particularly in the case of honeypot servers, the length of a requested resource string can indicate malicious intent because there should not be anything on the server to legitimately request. The content of the requested resource can also be used to determine the intent of the request, malicious or otherwise.

We also dug deeper into the data and found some avenues of exploration that have not yet yielded conclusive results, but we believe warrant further study. We clustered client IP addresses by comparing text fields as well as HTTP request timestamps. As of this writing, it is impossible to tell whether or not the clusterings actually discovered useful results, but we believe those avenues of exploration could yield more conclusive results in the future.