Internet Systems Consortium

Black Hat USA 2009

Title: Internet Special Ops – Stalking Badness Through Data Mining

Submitted by: Andrew Fried <afried@isc.org> +1.703.362.0067

Presenters:
Paul Vixie, President, Internet Systems Consortium
Andrew Fried, Researcher, Internet Systems Consortium
Dr. Chris Lee, Researcher, Internet Systems Consortium

Today's Internet threats are global in nature. Identifying, enumerating and mitigating these incidents require the collection and analysis of unprecedented amounts of data, which is only possible through data mining techniques. We will provide an overview of what data mining is, and provide several examples of how it is used to identify fast flux botnets and how the same techniques were used to enumerate Conficker.

Overview of Data Mining

Wikipedia defines data mining as, "the process of extracting hidden patterns from data". In it's simplest form, data mining may involve the analysis of one data set, such as web log files. But the real power of data mining occurs when multiple datasets are combined. Analyzing data from disparate sets can often reveal anomalies and relationships that could never be found from a single data set.

Unlike traditional data analysis, data mining enables us to see correlations and relationships that would otherwise not be found. That's the real magic of data mining. It also enables you to find "needles in haystacks" using quantifiable and replicable methods.

Successful data mining involves several steps. First, you must identify relevant data that is available. Secondly, that data must often be normalized, which simply means that you form the data sets in some manner that's consistent. Third, the data must often be reduced. Data reduction is necessary to make the sets manageable and to minimize processing time involved in relational database type queries. The fourth step involves identifying derivative data sets that can be produced through the primary sets. Finally, the fifth step involves determining how the data can/will be analyzed. This is the step that transcends science and approaches that of an art form. We'll discuss each of these steps in detail.

Basic Tools for Data Mining

Data mining almost always involves the analysis of large data sets. Large is a relative term, but for our purposes we'll assume the average data sets have tens of millions of records and are multi-gigabytes in size. Most mining requires the use of relational databases, whose efficiency is directly related to the CPU speed, number of cores, amount of RAM and speed of the disks.

So the first requirement for low stress analysis would be fairly "high end" computers. A good rule of thumb is that your RAM should be equal to, or greater, that the size of the data set you're working with. This obviously isn't always possible, but good high-speed disk drives help to compensate. A modern quad core computer with 32 to 64 gigs of RAM would be a good start. Add to that multiple 15K SAS drives set up in a RAID 0 configuration and you'll be in business. In today's prices, that equates to a system somewhere around $7,500 and up.

Secondly, you'll need a good relational database. Unless you work for a company that also prints money, you probably should look at either MySQL or PostgreSQL, both of which are free. Databases can be like religions – you tend to stick with the one you grew up with and tend to reject others because they're not the same as yours. But there are two considerations that you should keep in mind. MySQL is far easier to use, better documented, better supported, more intuitive and generally faster at indexing large data sets than PostgreSQL is. However, if you're working with IP addresses and CIDRs, PostgreSQL is the way to go – MySQL does not support IP data types.

There is a workaround for address analysis in MySQL, and that involves converting IP addresses into integers and netblocks into integer ranges. The problem with this is that finding an IP address that falls between a starting IP address and an ending IP address (that integer range I just mentioned) requires the use of a "BETWEEN" option in the SQL SELECT statement. MySQL is horrifically slow performing those queries. PostgreSQL, with an additional add-on package called IP4R, will enable you to work with addresses much faster.

My personal preference for standard database work is MySQL, however I also use PostgreSQL for the reasons stated above. The good news is that you can run both simultaneously on your database server.

Next, you'll need some programming capability – PHP, Perl, Python and Ruby seem to the most commonly used. Since the majority of the heavy lifting is done through the database, the majority of programming that is done outside of SQL is generally done to normalize datasets. The speed and efficiency of the language is really not that critical – they're all very fast at reading in data and regurgitating it in a different format, which is what normalization is all about. Pick any programming language that you're proficient with.

Finally, you'll need some storage. In fact, you'll probably need lots of storage. As a general rule, you should only keep the raw data you're processing on the database server. Archival data should be stored on an external system. The major requirement for this system is storage capacity rather than brute speed. SAS drives are fast, but they don't come in very large sizes, so a system running a raid array of 1 terabyte SATA drives is more than adequate. Your storage server can be set up in RAID 5 or even RAID 6 to insure against data loss. Setting a database server up using the same would result in a substantial reduction is read/write speeds. So your database server processes the data, while your external storage device stores and archives it.

Getting the data from the storage device to the database and back can be done several ways, the most common being NFS, FTP or rsync. If you're running Linux, another excellent choice is a distributed file system called Gluster. Gluster works very much like NFS, but it tends to produce faster throughput with lower CPU overhead than NFS. Rsync is by far the fastest, but lacks the ability to "mount" filesystems from your storage server onto the database server. If you're running one centralized data storage server that feeds multiple database systems, consider using 10 gig Ethernet or Infiniband.

The final requisite tool is good intuition. This is a capability that you develop, rather than purchase. Probably no other aspect of computing relies more heavily on good intuition and "hunches" than data mining. The good news is that the more data you process, the better you'll get at this.

Data Collection

Once you've established an adequate infrastructure, it's time to start looking for data. There are a few important points to keep in mind. First, you need to identify what data is available. Don't necessarily limit yourself to the specific data sets you think you'll need, find out all of the data sets that you can readily get your hands on, even if they seem totally unrelated. If your organization stores firewall logs, dns logs, dhcp logs, ftp logs, whatever, get them all!

Next, you need to develop some methodology for archiving the data. If you're receiving firewall logs every hour, concatenating them into one big file is not the way to go. Files should be split into smaller chunks, and ideally the file names for each data set should contain some kind of timestamp. A common practice is to use epoch timestamps.

For example, if you retrieve a firewall log on August 1, 2009 at 12:00, a suitable name for the file might be "firewall-1249142400.log". "1249142400" is the Unix epoch time for August 1, 2009 at 12:00. Alternatively, you could also name the files "firewall-200908011200.log". Use whatever naming convention makes sense to you. Keep in mind that if you're going to write code to traverse directories that contain data, using epoch timestamps enables you to identify the exact date and time of the data using standard libraries without relying on the file timestamps, which can change as data is migrated from system to system.

And while we're on the topic of timestamps, keep in mind that while most log files contain dates and times of activity, not all clarify the timezone used. Back to our previous example of firewall logs – there are two important pieces of information needed to work with them. Number one – is the firewall using some kind of reliable time source such as NTP. In other words, are the times reflected in the logs precise simply approximate. Secondly, you need to know what time zone the logs used. Does the firewall use GMT, EST, etc. This is really important if you're comparing two or more log files together. In fact, its not merely important, it's critical.

Data Normalization

In the simplest terms, data normalization involves transforming disparate data sets so they have some commonality is both structure and content.

There are different definitions for normalization, depending on the context in which it's used. For example, Wikipedia defines data normalization as, "a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics". When preparing data to be imported into a database, you must insure that the data you send in matches the type data the database expects. If the database field you're populating expects an integer like '5', you can't send in a string like "five". Fortunately, most modern databases aren't shy about telling you that a data import didn't go well.

When defining data normalization in regards to data mining, the previous definition is insufficient. In this context, data normalization involves removing extraneous information ("pruning"), providing commonality between different data sets, compacting the data for speed of processing and performing sanity checks to prevent bad data from being used.

Going back to our example of firewall logs, if you were trying to match firewall logs against DHCP logs to determine who accessed what, you would need to insure that the times in both files are using the same timezone and that the ip addresses are in the same format. This may involve transforming the time from a timestamp into an epoch "integer" on both data sets, then adding or subtracting the correct number of seconds from one or the other to make them compatible.

As mentioned above, data normalization often involves modifying the data so it can be processed more efficiently. This is generally performed as an intermediary step between loading the raw data and preparing it for import into your relational database. For example, if you were working with domain data, your data sets could include a domain name and its nameserver(s). Domain names can be very long, as can be the names of nameservers. Producing indexes on long strings and doing subsequent indexed lookups on multiple fields can be very expensive in terms of cpu utilization and disk IO. However, if you take the domain name and append the nameserver to that, then create an md5 hash of the two fields, you end up with a 32 character string that is far faster to both index and search on. Little tricks like this

can make the difference between a query running in one minute versus the same one running in one hour; differences can be even greater as the amount of data increases.

Data Reduction

For our purposes, data reduction can refer to two different activities. The first definition would refer to pre-processing raw logs and striping out, or pruning, irrelevant information prior to importing the data set into your database. There is no reason to import data just because it's there. If each record you're importing has fields that are of no use in the queries you're running, don't bring them in. Processing speed is always related to data size, so you want to keep the non-essential data out of your data imports.

Data reduction can also refer to techniques used to reduce the raw data sets to more manageable chunks. For example, if you were running queries against firewall logs to produce utilization reports by user, you could try and bring in all the data for a full year, then run your queries against that humongous data set. In most enterprise environments, a years worth of logs would be in the hundreds of gigabytes and consist of trillions of records. It would be a very painful process regardless of how beefy your database server is.

So, with the above example, one data reduction technique would be to crunch the data on a monthly basis, then store the summary data into a different table. At the end of the year, you would have only 12 entries per user, one for each month. This would reduce your data an exponential amount without any resultant data loss.

One more method of data reduction is worth mentioning – partitioning. Both MySQL and PostgreSQL support data partitioning, which enables you to break a logical database table into multiple partitions. There are a number of methods for splitting up the data between the partitions (this is database dependent), but one of the most common is based on a date field. For example, if you set up a database table such that one partition stored data from January, one from February, etc, removing an entire month of data would involve simply "dropping" that partition. The database would not have to be taken offline, the indexes wouldn't have to be rebuilt and you'd incur no down time. This is an especially good technique to use if you're collecting data from live sensors, and the data goes from collector directly into your database.

Derivative Data

For illustrative purposes, lets say you were analyzing smtp mail logs. Contained within the logs were the date and time an email came in, the server name sending the email and the recipient's email address. So we have five fields in this table, and nothing else:

- Date
- Time
- Sender's server

- Recipient

Using this table, there are a number of simplistic queries we could develop. For example:

- How many emails per day
- How many emails per time of day
- How many emails per sender
- How many emails per recipient

These are nice for management reports, and you could make all sorts of colorful pie charts that would amaze and entertain your management. But suppose we wanted to know if we were getting an inordinate amount of emails from a particular ISP, or a particular country, or even a particular network. Could we do that with what we have? The answer is yes, but only if we add a little more detail to the logs.

We have the name of the sender's server. So the first order of business would be to use the power of our database to create a unique listing of servers, then feed that list into some kind of script or program that could perform automated DNS lookups (and format the output in such a manner that it can be easily imported into a database).

Next, we get a copy of our routers BGP tables and bring that into a PostgreSQL database table. With another program we can query the BGP data to determine the netblock the senders server is in, along with its ASN number. And finally, we can obtain a geoip database that maps IP addresses to countries and or city/country, run that against the server IP address, and obtain the location of the server that sent the emails. We then write some programs to import all this into our initial table and we now have:

- Date
- Time
- Sender's server
- Sender's IP address
- Sender's ASN
- Sender's Country
- Recipient

The data we added to the original data set is what I refer to as derivative data. It's data that was derived from our initial data set and used to augment our data. Now what can we do with this?

- How many emails from each country
- How many emails from each ASN
- Which user received the most email from "Hackzerastan"
- Many more….

Additional data sets could be leveraged against this, such as Spamhaus RBL data. Using RBL data could enable us to evaluate the efficiency of our inbound mail filtering, and can assist with modifying firewall rule sets.

Experienced data miners often have dozens of various "tool" tables residing on their database servers such as ASN data, geoip data, RBL lists, domain name data, zip code data, employee records, etc. Thanks to the power of relational databases, this data can be leveraged in queries without the need to insert it into each table. Maintaining a single database of ASN data enables us to keep the data current in one location and have the data immediately available throughout the rest of the databases on our server.

Derivative data enables us to process data with much greater granularity and specificity. It also enables us to see relationships not present within the original data set.

Developing an Analysis Strategy

One of my favorite quotes came from Albert Einstein, in which he said, "We can't solve problems by using the same kind of thinking we used when we created them." This is a very important concept as it relates to data mining. An experienced data miner will take data and use it to develop questions rather than simply answering questions posed to them. It's definitely a "think outside the box" kind of job.

For example, let's say you were tasked by your management to review employee accesses through the corporate VPN. The question, in their minds, is whether an expired or unknown user account was able to "hack" into their network. A data miner would immediately expand upon the question by wanting to see if any accesses looked suspicious and not limit their analysis strictly to user account resolution.

Once all the available logs were assembled and imported, the data miner would conduct a number of exploratory searches for the purposes of getting a "feel" of the data. The data miner would find derivative data that could be used in the queries, and would then strategize on what relationships can be gleened from the data.

What to look for and how to find it – that's the real challenge. And that's what data mining is all about!

There are two methods I employ that have proven helpful over the years. The first is to visualize various activity that I might want to find, then try and determine what that activity would look like in the logs. It's the , "If I were to do it, how would I do it" type strategizing, followed by the, "assuming I did that, what would get logged, where would it get logged and what would it look like".

The second method involves looking for statistical anomalies. It's sorting, averaging and comparing data sets to see if something appears outside the norm. As related to our review of VPN accesses, those statistics could include average bytes downloaded, average bytes uploaded, where connections originated from, when the connections were made from, number of connections, and could even include some analysis of accesses by time and distance (e.g. user logs in at 7:00 am from Los Angeles then logs back in again two hours later from New York City). It could include the number

of failed logins by user account, the number of logins per IP address or even the number of different users that logged in from the same IP address. And of course accesses that originated from outside our country would be something I would scrutinize, especially if employees didn't travel abroad.

Real World Examples – Fast Flux Botnets

One activity that I've been heavily involved in over the past five years involves phishing and fraudulent websites. In the early days, some non-descript website would get hacked and the bad guys would upload a phish kit, then send out spammed emails directing prospective victims to the site under a number of different guises. Anyone with an email account has received the enticement emails, often several times a day, in fact!

As time progressed, the complexity of the schemes increased. The phishing sites were hosted on dedicated servers that were obfuscated through botnets of proxies. The domain names used in the schemes were registered specifically for the sites, and the takedowns became more time consuming.

The challenge was to find the botnets before the victims could. The Internet is a big place, so finding them required more than brute force tactics. It required data mining, and data mining on a very large scale.

So lets dissect the problem and look at it step by step. First, the bad guys register a domain name. Next, they set up the site and get the DNS records to point to the site. Then they send out the enticement emails advertising the site. From a data mining perspective, they actually provided a number of options for finding their sites.

Let's now start to assemble the data and derivative data sets we can use to start stalking the bad guys on the internet.

First, they register domain names. Well, I have several good tools that I can leverage that deal with domain names. First, I have a very large database of all gTLD domain names that's updated daily. So when they register that domain, it should be in my database – the trick is to know which one of the 400,000 or so changes I see each day belongs to them.

Secondly, the domain name has to resolve. Simply put, when someone queries the domain, it goes to a nameserver for resolution. With sensors on certain large resolvers, I have a feed of that data, which is referred to as passive dns. The passive dns provides host names, domain names and IP addresses in virtually real time.

Since they're using botnets, each domain name resolution will contain multiple IP addresses. This is not the norm for the vast majority of dns address resolutions. So by focusing in on dns "A" record resolutions from both domain data and passive dns that return multiple IP addresses the botnets are readily visible.

Additionally, the botnets appear to be located on randomly selected compromised servers that are geographically dispersed. By analyzing DNS data and combining those queries against BGP routing data and geoip location databases, I can mathematically determine ratios of addresses to autonomous system, ratios of IP addresses to countries, and ratio of IP addresses to subnet allocation sizes. Once again, when that data is normalized in relation to the overall norms found on the Internet at large, the botnets stand out. Data mining is truly a wonderful thing.

Some of the phishing gangs take extraordinary efforts by implementing countermeasures against both DNS resolution and scans. This actually results in a very counterproductive consequence – sites that have passive DNS resolution that cannot be subsequently accessed are scanned through our proxies, and if one access is unsuccessful but the other is, they're found – proof positive.

Finally, a phishing site is no good unless victims can find it. Unfortunately for the phishers, if a hapless victim can find their site, so can I. Spamtraps are used extensively to analyze links appearing in spam, and this data is augmented with spam reports from a number of the realtime blackhole lists (RBLs). Spamtrap data is collected, parsed, augmented, databased and automated reports are generated from them which, in turn, is fed into the DNS and web site scanners.

The phishing sites I'm looking for have distinctive keywords and phrases. By spidering websites identified through passive DNS, active DNS scans, spam trap data and external RBL reports, I can positively identify the sites and generate automated alerts.

Certain phishing groups use automated programs to create and register the domain names. By analyzing the domain names in relation to nameservers, certain telltale fingerprints can be identified. Examples of domain name analysis would include ratios of vowels to consonants, number of consecutive consonants, perentage of domains that have special characters such as hyphens, etc.

Despite the fact that there are hundreds of millions of active domains on the Internet, and up to half a million new domains added every day, data mining techniques enable us to effectively isolate and identify a myriad of threats that appear on a daily basis.

Summary

Data mining is as much an art form as it is a science. And while there is a standard cadre of available tools and data, it's the skills, availability of data and resourcefulness of the data miner that determine its effectiveness. Data mining is the only method that can successfully and reliably analyze large data sets in a timely manner, which is imperative when using it in a defensive posture.