Using a Database to Store Data Captured with tcpdump

MIHALIS TSOUKALOS

Mihalis Tsoukalos is a UNIX system administrator who also knows programming, databases, and mathematics. Follow Mihalis on Twitter: @mactsouk
www.mtsoukalos.eu

In this article, I show you how to store network data in a MySQL database and how to take advantage of the SQL language to query stored data. I know that this has been done before, but I thought that it would be a good exercise to create my own solution.

Although I selected the MySQL [1] DBMS, you can use PostgreSQL, Oracle, or even a NoSQL database such as MongoDB [2]. For the network data capturing, I will use the trusty tcpdump [3] utility.

Advantages

Storing your network data into a database has many advantages, including the ability to query your data offline, being quicker than pure disk I/O because databases store their data optimized, embedding intelligence in your data by using a database, distributing your network data in many databases, using the reporting tools that support your database, easily querying your network data if you already know SQL, and giving a remote user access to your data using network access to the database.

Storing your network data in a database has some disadvantages, as well, including the fact that if you have a busy network, the database storage you will need will be big. Also, you will need a person to administer the database, if you do not already have one.

The Solution

Before being able to use any network data, you must first collect network data using the tcpdump utility. You can also use WireShark [4] or tshark [5] to capture network data, but tcpdump is more popular for network capture.

A Perl script will be used for selecting and inserting the data into the MySQL database. The DBI and DBD::mysql Perl modules must be pre-installed on your UNIX system for the Perl script to work. The script also uses the Data::Validate::IP Perl module for catching erroneous IP addresses.

The Perl script implies that the network data is already captured with tcpdump. If you want to tell tcpdump to capture 2000 network packets and exit, the following command will do the trick:

```
# tcpdump -i eth0 -c 2000 -w login.tcpdump
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture
   size 65535 bytes
2000 packets captured
2004 packets received by filter
0 packets dropped by kernel
```

You can alter the captured packets by adding more command-line arguments to the tcpdump command. Please note that you usually need root privileges to capture network traffic from a network interface.

The Perl script uses the following tshark [9] command to convert the network data into a more readable format:

```
$tshark -r login.tcpdump -T fields -e frame.number -e
 frame.time_relative -e ip.src -e ip.dst -e
 frame.protocols -e frame.len -E header=y -E quote=n
 -E occurrence=f
```

The format of the exported text file will look like the following:

```
frame.number frame.time_relative   ip.src          ip.dst          frame.protocols          frame.len
1            0.000000000           109.74.193.253  2.86.13.236     eth:ip:tcp:ssh           194
2            0.011654000           174.138.175.116 109.74.193.253  eth:ip:udp:dns           97
3            0.011733000           109.74.193.253  174.138.175.116 eth:ip:icmp:ip:udp:dns   125
4            0.048020000           208.67.217.17   109.74.193.253  eth:ip:udp:dns           106
5            0.048107000           109.74.193.253  208.67.217.17   eth:ip:icmp:ip:udp:dns   134
6            0.055475000           2.86.13.236     109.74.193.253  eth:ip:tcp               66
7            0.081615000           83.145.248.129  109.74.193.253  eth:ip:udp:dns           75
8            0.081692000           109.74.193.253  83.145.248.129  eth:ip:icmp:ip:udp:dns   103
```

As the `frame.protocols` column may contain many values, we will use the last one.

The Implementation

The table that holds the network data contains a field called "id" that auto increments and acts as the primary key for the table. The "packetNumber" field is the packet index for a given network traffic capture, whereas the "dt" field is the time that the packet appeared since the first packet of the given network capture. The "sourceIP" and "destIP" fields contain the source and destination IP values, respectively. The "protocol" fields holds the protocol name of the network packet. Last, the "length" field holds the packet size in bytes.

You can choose to include additional fields depending on your network and the problem(s) you want to examine.

Importing the data into the MySQL data is also performed by the Perl script. If you have a fast computer and a network without extremely high traffic, you can even store your network data in (near) real-time.

The Perl script is called netData.pl and takes a single argument that is the name of the file that holds the tcpdump network data.

Querying the Database

The Perl script executes all the following queries and displays their results.

Find the Connections per Protocol:

```
mysql> select count(protocol), protocol FROM NetData GROUP BY protocol;
+-----------------+----------+
| count(protocol) | protocol |
+-----------------+----------+
|            1084 | DNS      |
|             794 | ICMP     |
|               1 | ICMPv6   |
|               1 | SSH      |
|              70 | SSHv2    |
|              50 | TCP      |
+-----------------+----------+
6 rows in set (0.04 sec)
```

Find the average packet length per protocol. Also print the number of packets:

```
mysql> select COUNT(*), protocol, avg(length) from NetData
GROUP BY protocol;
+----------+-----------+-------------+
| COUNT(*) | PROTOCOL | avg(length) |
+----------+-----------+-------------+
|     1084 | DNS      |     95.8127 |
|      794 | ICMP     |    123.0743 |
|        1 | ICMPv6   |    118.0000 |
|        1 | SSH      |    194.0000 |
|       70 | SSHv2    |    356.3571 |
|       50 | TCP      |     66.6400 |
+----------+-----------+-------------+
6 rows in set (0.04 sec)
```

Find the number of different Destination Hosts used in the destIP column:

```
mysql> select count(distinct(destIP)) from NetData;
+-------------------------+
|count(distinct(destIP))|
+-------------------------+
|                     279 |
+-------------------------+
1 row in set (0.01 sec)
```

Find the Top-10 Source IPs:

```
  mysql> select count(*) as TOTAL, SourceIP
  from NetData
  GROUP BY SourceIP
  ORDER BY TOTAL DESC
  LIMIT 10;
+-------+----------------+
| TOTAL | SourceIP       |
+-------+----------------+
|   841 | 109.74.193.253 |
|    76 | 2.86.13.236    |
|    38 | 204.74.106.104 |
|    25 | 175.41.186.83  |
|    24 | 89.149.6.76    |
|    22 | 90.155.53.34   |
|    20 | 218.248.241.3  |
|    20 | 114.134.15.205 |
|    19 | 200.29.243.21  |
|    17 | 14.139.5.22    |
+-------+----------------+
10 rows in set (0.03 sec)
```

Here is the output of the Perl script for the network data I captured:

```
$ ./netData.pl login.tcpdump
Erroneous IP!!
   count(protocol) protocol
   3756            DNS
   142             SSH
   100             TCP

   COUNT(*)        protocol      avg(length)
   3756            DNS           107.3387
   142             SSH           354.0704
   100             TCP           66.6400

   count(distinct(destIP))
   278

   TOTAL  SourceIP
   1682   109.74.193.253
   152    2.86.13.236
   76     204.74.106.104
   50     175.41.186.83
   48     89.149.6.76
   44     90.155.53.34
   40     218.248.241.3
   40     114.134.15.205
   38     200.29.243.21
   34     14.139.5.22
   Number of rows inserted: 999
```

If you connect to MySQL using the command line shell or a GUI application, you can execute whatever SQL query you wish. Only your imagination and SQL can limit the way you can utilize the data.

If you are using a NoSQL database, such as MongoDB, which is my favorite NoSQL database, you may need to learn how to use the MapReduce [6] technique to query the database. The results will be the same, but the implementation will be a little different. MongoDB is more focused on storing semi-structured data.

In conclusion, determining your own requirements and creating the queries that fit your needs is the first thing to do before inserting any data in a database. A statistical package such as R [7, 8] can also be used for visualizing and exploring your data. Download the netData.pl script from the USENIX site [10].

References

[1] MySQL site: http://www.mysql.com/.

[2] Kristina Chodorow, *MongoDB: The Definitive Guide, 2nd Edition* (O'Reilly Media, 2013).

[3] tcpdump: http://www.tcpdump.org.

[4] WireShark: http://www.wireshark.org/.

[5] tshark: http://www.wireshark.org/docs/man-pages/tshark.html.

[6] MapReduce: http://docs.mongodb.org/manual/reference/method/db.collection.mapReduce/.

[7] R Project: http://www.r-project.org.

[8] Mihalis Tsoukalos, "Using the R Advanced Statistical Package," *Linux Journal*, August 2013.

[9] Using WireShark Command Line Tools & Scripting: http://www.youtube.com/watch?v=CWOCqGmu1aI.

[10] netData.pl: https://www.usenix.org/publications/login/february-2014-volume-39-number-1