# Using Perl, Postgres and SQL to build a Comprehensive, Searchable Database of Firewall Activity on a Checkpoint Firewall on The Cheap.

**Written by**

**Thomas J. Munn**
**CISSP**
**December 11, 2007**

Table of Contents:

## *Introduction*

One of the biggest problems in computer security is how to deal with logs. Firewall logs are especially difficult because they often run into the tens to hundreds of gigabytes, and searching them requires expensive, proprietary packages. The purpose of this paper is to help instruct the reader in how to use Postgres and Perl to have a queryable database using checkpoint log files. I do not cover other firewalls but the ideas herein could be applied to almost any firewall supporting output to text files. This paper will give the security practitioner to perform data mining on otherwise useless logfiles. Using the methods described in this document enabled the author to query for almost anything imaginable from a total of 450,000,000 records in under ten minutes. The cost: The cost of the hardware.

### A note on database servers

If you are running a fairly large data set, it is important to have some decent hardware to do this on. Database searches really pound on a system. The database profiled herein reached 430 million records, and queries would pound the server at 200+mbps for several to sometimes tens of minutes.

Setup described in this paper
* Centos 5.x
* Postgres 8.x
* Perl (stock install of centos)
* PAE kernel to use 8GB memory

## *Centos installation*:

Install a basic install of centos, using new hardware. Once installed, you probably will want to change your kernel to use the PAE, as machines with > 4gb will not be recognized. To install the PAE kernel package in CENTOS:

```
yum install kernel-pae
```

The "CENTOS EXTRAS" repository needs to be added. To do so edit the

```
/etc/yum.repos.d/CentOS-Base.repo
```

Go to the [centosplus] section and modify the enabled line to read:

```
enabled=1
```

After performing this be sure to reboot the system to use the new kernel and have access to all of the memory on the server. This step is only necessary if there is > 3.2gb installed in the machine. Also run a

```
yum update
```

to ensure the box is appropriately patched.

## *Installing Postgresql:*

The next step in the process is to install postgresql. The commands are listed below:

```
yum install postgresql postgresql-server
```

### Postgres preparation

You will be creating a new user (your login might be an ideal choice) that will be inserting the logfile data into the database. The commands below will give you FULL access to drop etc. so please use with great caution, as people with this account will be able to delete, modify and drop databases. Setup should include a more fine-grained user for, perhaps read only access etc. But this is beyond the scope of this paper.

```
su –postgres
createuser --superuser --createdb --createrole --login <USERNAME>
psql
create database logs;
\q
```

This prepares the database to receive the contents of the database itself. Basically this is a database containing the relevant information that we want to search via sql. The sql script is listed in the end of the document. Assuming that you created a file named create_table in the postgres directory, you could execute the following to create the database:

```
psql logs < create_table
```

Now that the database is created, and the postgres user setup, the database is ready to receive data from the checkpoint log server.

## *Transferring the Logfiles*

The easiest method of getting the logs converted is to convert on an intermediary server. Checkpoint will require that the ROOT user be used to view any of the files in the logfile directory. Install the checkpoint management server on the database server. This is beyond the scope of this document. Once the software is installed on the database software, put the keys of the username performing the transfer into the remote root's authorized_keys file. This, also is beyond the scope of this document. Look at SSH keys and how to implement passwordless ssh key exchange. Hint:

```
ssh-keygen –t rsa –b 2048
```

Assuming that the checkpoint software on the intermediary computer, transfers between the log server and the database server can be automated via an ssh job that will copy files. Alternatively, the log processing can be done on the log server, and then the post-processing done on the database server.

## Convert the Logfiles to SQL format

Issue the following command to convert the checkpoint logfiles to plaintext, separated by bars:

```
  fwm logexport -i /dir/to/export/file.log -d '|' -n -p -o
/dir/to/export/to/file.bsv
```

Once this is issued and done, the perl script can be invoked in the following manner:

```
cat file.bsv | convert.pl | gzip -9 > file.sql.gz
```

The reason the file is piped through gzip is because the resultant sql is HUGE and compression reduces the filesize by 97-99%.

The final step in the conversion process is to put the sql into the logs database and table that we created earlier.

Assuming the psql user:

```
su – postgres
cd /where/log/are/stored
zcat file.gz | psql logs > /dev/null
```

That's it. The convert process takes a long time, so automating this via cron would be a good idea. Also the script's logic is suboptimal, since it inserts the records one a time rather than using transactions.

Prior to getting into the sql queries, it is a good idea to create and index on the database. See appendix for the script to do this

## Useful SQL queries

The next section describes various ways of looking at the imported log data using structured query language.

## Top ports

The most basic things in security is knowing who did what. The queries below will display a log of the top ports and their counts. It also will make them distinct, so that they are summarized. It orders by the count, descending.

These commands are issued either by feeding a file into the psql interpreter, or by typing them by hand when in the psql command. Either way, here is the syntax for feeding a file into psql, and then append the output to the results

```
psql logs < input.query > results.txt
```

Below is the query that would be either in the file or at the psql prompt. psql is a command that lets the user interact with the database.

```
select distinct service,count (service) as count from logs
group by service
order by count desc;
```

Here is the output:

```
service |  count
---------+----------
      80 | 26509321
     443 | 13031794
    1352 | 11982547
      53 | 11366311
     389 |  6619088
     135 |  5005520
     161 |  4435975
     445 |  3663777
     139 |  3366915
     137 |  1984704
    1882 |  1141976
     515 |   991218
    6665 |   925032
    8080 |   820420
    3994 |   764949
    1433 |   753008
```

## Querying for a specific destination port

The next query allows for looking at a specific port once the first query is run. It is a 'drill down' function.

```
select  distinct src,dst,service, count (*) as total from logs
where service = 5798
group by src,dst,service
order by total desc;
```

This tells the top ports by source and destination. For 'servers' change the top line to only have destination:

Querying for most frequent destinations and a single port:

```
select  distinct dst,service, count (*) as total from logs
where service = 5798
group by src,dst,service
order by total desc;
```

Change the '5798' to whatever port is desired. Also programs can be written to query by the port from the top logs report.

## To query by Subnet:

To see all of the hosts within a 10.10.10.0/24 subnet:

```
Select src,dst,service from logs
Where src << inet '10.10.10/24'
```

## To see a list of possible portscanners:

To tell a list of 'portscanners' by looking for hosts with lots of different unique destination ports:

```
select src, count (distinct service) from logs
group by src
order by count desc;
```

Other things can be added to this query:

To tell the above limited by date > Nov 21, 2007 and rule #:

```
select src, count (distinct service) from logs
where cast (date_combined as DATE) > '2007-11-21'
and "rule" = 64 or "rule" = 62
group by src
order by count desc;
```

## Date range matching rules 64 or 62:

```
select src, count (distinct service) from logs
where cast (date_combined as DATE) between '2007-11-21' AND '2007-
12-04'
and "rule" = 64 or "rule" = 62
group by src
order by count desc;
```

## To delete specific IPs from the database that doesn't fall on subnet boundaries:

```
Delete from logs
where dst in ('1.1.1.1','2.2.2.2','3.3.3.3');
```

After deleting a large number of records, it is a good idea to drop the indexes, and to 'vacuum' the database. This helps with performance, and database size.

## To vacuum the database:

```
Vacuum;
```

Also when generating queries, you can run the 'explain' parameter in front of the query. This will tell you how 'expensive' a particular query is:

## Explaining the 'cost' of queries

For example:

```
Explain select * from logs limit 10;

------------------------------------------------------------------
 Limit  (cost=0.00..0.46 rows=10 width=316)
   ->  Seq Scan on logs  (cost=0.00..5619498.82 rows=121489782
width=316)
(2 rows)
```

The cost is 'zero' since things are pretty much in the order in which they come.

The cost goes up the more items are added to the query. Here is the cost of the query when an index and a where clause is concerned:

```
explain select * from logs
logs-# where service = 80;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on logs  (cost=4145.07..1672452.69 rows=607449
width=316)
   Recheck Cond: (service = 80)
   ->  Bitmap Index Scan on index_service  (cost=0.00..4145.07
rows=607449 width=0)
         Index Cond: (service = 80)
(4 rows)
```

Finally, with a 'distinct' which requires going through the ENTIRE database:

```
Unique  (cost=1730805.47..1736879.96 rows=40001 width=68)
   ->  Sort  (cost=1730805.47..1732324.09 rows=607449 width=68)
         Sort Key: src, dst, service
         ->  Bitmap Heap Scan on logs  (cost=4145.07..1672452.69
rows=607449 width=68)
               Recheck Cond: (service = 80)
               ->  Bitmap Index Scan on index_service
(cost=0.00..4145.07 rows=607449 width=0)
                     Index Cond: (service = 80)
(7 rows)
```

The higher cost = a longer search time. It can be helpful in helping to see how well is made, and how it could be made more efficient.

## Query by date and protocol

Here is a query looking for udp port 389 on November 26, 2007. As an interesting aside Microsoft uses both TCP AND UDP for ldap....

```
select service,proto, count (*) as total from logs
where cast (date_combined as DATE) = '2007-11-26'
and service = 389
and proto = 'udp'
group by service,proto
order by total desc;
```

## This query sorts by distinct destination and destination port combination:

```
select  distinct dst,service, count (*) as total from logs
where service = 18
group by dst,service
order by total desc;
```

# *Table: logs*

**Prior to running this, need to run the 'psql' command. Then run: create database logs;**

**Descriptions**

Table Logs contains the output of the converted checkpoint logfiles.

**Fields**

| PK | Name | Data type | Not null | Unique | Description |
|---|---|---|---|---|---|
| | date_combined | timestamp | | | Combined Date stamp |
| | orig_fw | inet | | | Origin Firewall IP Address |
| | action | text | | | Firewall action (drop etc) |
| | if_name | text | | | Firewall Interface traffic recorded on |
| | if_dir | text | | | Inbound or Outbound |
| | src | inet | | | Source IP |
| | dst | inet | | | Destination IP address |
| | proto | text | | | Protocol Type ICMP, UDP etc. |
| | rule | integer | | | rule Number triggered |
| | service | integer | | | Destination Port |
| | s_port | integer | | | Source Port |
| | icmp_code | text | | | ICMP code type (#) |
| | icmp_type | text | | | ICMP type (text string) |
| ✓ | primary_key | bigint | ✓ | ✓ | |

**Indices**

| Name | Type | Function | Fields | Primary Key | Unique | Description |
|---|---|---|---|---|---|---|
| index_id | btree | | primary_key | ✓ | ✓ | |

**Definition**

```
CREATE TABLE "public"."logs" (
 "date_combined" TIMESTAMP WITHOUT TIME ZONE,
 "orig_fw" INET,
 "action" TEXT,
 "if_name" TEXT,
 "if_dir" TEXT,
 "src" INET,
 "dst" INET,
 "proto" TEXT,
 "rule" INTEGER,
 "service" INTEGER,
 "s_port" INTEGER,
```

```
 "icmp_code" TEXT,
 "icmp_type" TEXT,
 "primary_key" BIGSERIAL,
 CONSTRAINT "index_id" PRIMARY KEY("primary_key")
) WITHOUT OIDS;

COMMENT ON COLUMN "public"."logs"."date_combined"
IS 'Combined Date stamp';

COMMENT ON COLUMN "public"."logs"."orig_fw"
IS 'Origin Firewall IP Address';

COMMENT ON COLUMN "public"."logs"."action"
IS 'Firewall action (drop etc)';

COMMENT ON COLUMN "public"."logs"."if_name"
IS 'Firewall Interface traffic recorded on';

COMMENT ON COLUMN "public"."logs"."if_dir"
IS 'Inbound or Outbound';

COMMENT ON COLUMN "public"."logs"."src"
IS 'Source IP';

COMMENT ON COLUMN "public"."logs"."dst"
IS 'Destination IP address';

COMMENT ON COLUMN "public"."logs"."proto"
IS 'Protocol Type ICMP, UDP etc.';

COMMENT ON COLUMN "public"."logs"."rule"
IS 'rule Number triggered';

COMMENT ON COLUMN "public"."logs"."service"
IS 'Destination Port';

COMMENT ON COLUMN "public"."logs"."s_port"
IS 'Source Port';

COMMENT ON COLUMN "public"."logs"."icmp_code"
IS 'ICMP code type (#)';

COMMENT ON COLUMN "public"."logs"."icmp_type"
IS 'ICMP type (text string)';
```

## *Perl script to extract data from checkpoint*

```
#!/usr/bin/perl -w
#checkpoint fw log export command: fw logexport -i
/path/to/logfileinput -d
#'|' -n -p -o /path/to/outputfile.bsv the character to use is the 'bar'
#character.  I normally use the cat file.log|./logslurp.pl | gzip >
sql.gz.
#  This keeps the sql file size down.
# use strict;
#use diagnostics;
$datematch = qr/(\d{1,2})([a-z]{3})(\d{4})/io;
%months = (
   Jan => "January",
   Feb => "February",
   Mar => "March",
   Apr => "April",
   May => "May",
   Jun => "June",
   Jul => "July",
   Aug => "August",
   Sep => "September",
   Oct => "October",
   Nov => "November",
   Dec => "December"
          );
while (<>) {
   my @fields = split (/\|/);
   if ($fields[1]eq "date") {
       $i=0;
       %generateindex=();
       foreach (@fields) {
           if (/product/) {
               $generateindex{'product'} = $i;
           }
           if (/^src$/) {
               $generateindex{'src'} = $i;
           }
           if (/^dst$/) {
               $generateindex{'dst'} = $i;
           }
           if (/^rule$/) {
```

```
            $generateindex{'rule'} = $i;
        }
        if (/^ICMP$/) {
            $generateindex{'icmp_text'} = $i;
        }
        if (/^ICMP Type$/) {
            $generateindex{'icmp_type'} = $i;
        }
        if (/^ICMP Code$/) {
            $generateindex{'icmp_code'} = $i;
        }
        if (/^proto$/) {
            $generateindex{'proto'} = $i;
        }
        if (/^s_port$/) {
            $generateindex{'s_port'} = $i;
        }
        if (/^service$/) {
            $generateindex{'service'} = $i;
        }
        if (/^date$/) {
            $generateindex{'date'} = $i;
        }
        if (/^time$/) {
            $generateindex{'time'} = $i;
        }
        if (/^orig$/) {
            $generateindex{'firewallip'} = $i;
        }
        if (/^action$/) {
            $generateindex{'action'} = $i;
        }
        if (/^i\/f_name$/) {
            $generateindex{'ifname'} = $i;
        }
        if (/^i\/f_dir$/) {
            $generateindex{'ifdir'} = $i;
        }
    $i++;
    }
    next;
}
```

```
   if ($fields[$generateindex{'action'}] eq "daemon") {
      next;
   }


   $fields[$generateindex{'date'}] =~ s/^\ //;
   $fields[$generateindex{'date'}] =~ s/$datematch/$months{$2}\
$1\,$3/;
   my $date = "$fields[$generateindex{'date'}]
$fields[$generateindex{'time'}]";
 #match origin firewall, source, or dest
   foreach $field ($generateindex{'firewallip'}, $generateindex{'src'},
$generateindex{'dst'}) {
      if ($fields[$field] eq "") {
         $fields[$field] = "0.0.0.0";
      }
   }
#match rule, service, or source port
   foreach $field ($generateindex{'rule'}, $generateindex{'service'},
$generateindex{'s_port'}) {
       if ($fields[$field] eq "") {
           $fields[$field] = "0";
       }
   }
   for ($i = 1; $i < $#fields; $i++) {
       $fields[$i] =~ s/\'//g;
   }
print "INSERT INTO logs (date_combined, orig_fw, action, if_name,
if_dir, src, dst,proto, rule, service, s_port, icmp_type,icmp_code)\n";
   print "VALUES ('$date','";
   print $fields[$generateindex{'firewallip'}],"','";
   print $fields[$generateindex{'action'}],"','";
   print $fields[$generateindex{'ifname'}],"','";
   print $fields[$generateindex{'ifdir'}],"','";
   print $fields[$generateindex{'src'}],"','";
   print $fields[$generateindex{'dst'}],"','";
   print $fields[$generateindex{'proto'}],"',";
   print $fields[$generateindex{'rule'}],",";
   print $fields[$generateindex{'service'}],",";
   print $fields[$generateindex{'s_port'}],",'";
   print $fields[$generateindex{'icmp_text'}],"','";
   print $fields[$generateindex{'icmp_code'}],"');\n";
@fields="";
}
```

## *Perl wrapper script to automate conversion of logfiles*

**This uses the above script and put into postgres. Not related to main program, run prior to running above script. Change the chdir to wherever you are downloading the logfiles to.**

```
#!/usr/bin/perl -w
chdir "/where/logs/are/";
#this line should be changed to match the pattern of the logfiles you
#want to process
my @dir = `ls -D -1 emp\*.gz`;
$i=0;
foreach (@dir) {
        chomp ($dir[$i]);
        `gunzip $dir[$i]`;
        $i++;
}
$i=0;
my @dir1 = `ls -D -1 emp\*.log`;
foreach (@dir1) {
        chomp ($dir1[$i]);
        `fwm logexport -i ./$dir1[$i] -d '|' -n -p -o ./$dir1[$i].bsv`;
        $i++;
}
#again, modify to match the pattern of YOUR logfiles!
my @dir2 = `ls -D -1 emp\*.log.bsv`;
$i=0;
foreach (@dir2) {
        chomp ($dir2[$i]);
        `cat $dir2[$i] | ./logslurp.pl | gzip > $dir2[$i].sql.gz`;
        $i++;
}
```

## *Final program to input data into postgres database*

```
#!/usr/bin/perl -w
#Match on the logfiles that you want to process!
my @dir3 = `ls -D -1 emp\*.log.bsv.sql.gz`;
$i=0;
foreach (@dir3) {
        chomp ($dir3[$i]) ;
        `zcat $dir3[$i] \| psql > /tmp/$dir3[$i].out`;
```

```
        $i++;
}
```

## *Sql to create indexes on newly created database:*

```
CREATE INDEX "index_date" ON logs
  USING btree ("date_combined");
CREATE INDEX "index_action" ON logs
  USING btree ("action");
CREATE INDEX "index_dst" ON logs
  USING btree ("dst");
CREATE INDEX "index_src" ON logs
  USING btree ("src");
CREATE INDEX "index_service" ON logs
  USING btree ("service");
CREATE INDEX "index_proto" ON logs
  USING btree ("proto");
CREATE INDEX "index_service" ON logs
  USING btree ("service");
```

## *Sql to drop indexes prior to mass deleting*

```
DROP index index_date;
DROP index index_action;
DROP index index_dst;
DROP index index_src;
DROP index index_service;
DROP index index_proto;
DROP index index_service;
```

## *Perl script that will query based upon a texfile, ports, in its current directory containing a list of ports to be queried:*

```
#!/usr/bin/perl -w
open (PORTS, "ports");

while ($line=<PORTS>) {
   open (GQ, ">gquery.sql");
   open (SRC, ">src.sql");
```

```
   open (DST, ">dst.sql");
   chomp ($line);
   #print $line;
   print GQ "select distinct src,dst,service, count (*) as total from
logs\n";
   print GQ "where service = $line\n";
   print GQ "group by src,dst,service\n";
   print GQ "order by total desc;\n";
   `psql < gquery.sql > $line`;
   print SRC "select  distinct src,service, count (*) as total from
logs\n";
   print SRC"where service = $line\n";
   print SRC"group by src,service\n";
   print SRC"order by total desc;\n";
   `psql < src.sql > $line.src`;
   print DST "select  distinct dst,service, count (*) as total from
logs\n";
   print DST "where service = $line\n";
   print DST "group by dst,service\n";
   print DST "order by total desc;\n";
   `psql < dst.sql > $line.dst`;
}
```

So the output of this file given a list of 80 would be:

80.src (filename)
80.dst (filename)

## *References*

For much of the Perl help: http://www.perlmonks.org
For questions on using Postgres: http://www.postgresql.org/docs/manuals/
For getting Centos: http://www.centos.org
For checkpoint information: http://www.checkpoint.com
For Postgres specific programs and applications: http://pgfoundry.org/