Global Information Assurance Certification Paper

Check Point firewalls - rulebase cleanup and performance tuning

GSEC Gold Certification

Author: Barry Anderson, shori@bigpond.net.au

Adviser: John Bambenek

Accepted:

Outline

1. Abstract

Firewall rulebases tend naturally toward disorder over time, and as the size of the ruleset grows, the performance of the firewall starts to suffer. In this paper, a simple procedure for culling unused rules and ordering the rulebase for performance will be presented. The procedure uses open-source software and purpose-built tools (which will be provided) and has been used to cleanup the rulebase of large firewalls at a major financial institution. Anyone interested in improving the performance of their Check Point firewall and/or improving their position come the next audit should read this paper.

2. Introduction

Administrators may come and administrators may go, but firewall rules go on forever (with apologies to Tennyson). Firewall rulebases tend naturally toward disorder over time and, for Check Point firewalls in particular, as the size of the ruleset grows, the performance of the firewall suffers.

To some extent, the assertion that firewall rulebases tend towards disorder is perhaps an inevitable result of the inherent complexity of managing a complex web of interconnections (or the Second Law of Thermodynamics at work), however there are relatively simple steps that we as firewall administrators can take to improve the situation, and in this paper, a procedure for culling unused rules and ordering the rulebase for performance will be presented. The procedure uses a provided framework of database queries and supporting perl scripts (see the appendices) and has been used effectively with free/open source database software on large projects such as cleaning up the rulebase of large firewalls at major financial institutions, as well as to audit a large (complex) rulebase to ensure that a firewall replacement and decommissioning project was not going to go horribly wrong.

The other major benefit of periodically cleaning up the firewall rulebase lies in the administrator's increased ability to respond (truthfully!) to auditors' inevitable questions about

firewall ruleset review.

While there are commercial products with similar functionality – Check Point's Eventia Reporter and Tufin's SecureTrack are two such products - the advantages of a framework and procedure over a product are twofold: firstly, increased flexibility and secondly, decreased cost.

In the interests of full disclosure it must be admitted that there is also a disadvantage to the procedure and associated framework compared to using a commercial product – that's that the increased flexibility comes at the price of some extra work – if you have a great idea for a report you can do on the data you have, the answer is "Sounds fabulous, you should do that", versus "Would you like to submit a Request For Enhancement to Engineering?"

3. Procedure

The Approach

The procedure for performance tuning a Check Point firewall rulebase is fairly simple:

1. move the most used rules to the top of the rulebase – with two important caveats that we'll discuss;

2. where it works to turn off logging, do so;

Cleanup vs Performance Tuning

Before beginning to discuss the above procedure in detail, it is necessary to briefly touch on rulebase cleanup, of which there are two distinct types:

(i) where rules that contain "Any" in the SOURCE, DESTINATION or SERVICE fields are replaced by more explicit rules (in practice, usually performed in haste in the lead-up to an audit, but a useful function to perform periodically regardless);

(ii) where rules are removed from the rulebase (or better yet, disabled):

1) on the basis that they are never used (essentially this is simply the limit case of

the performance tuning approach above); or

2) for other reasons such as the decommissioning of a host or service or the expiry of the period of a defined business need to access said host or service – obviously in the case of expiry, you follow your organization's procedures for determining whether or not access is still required before simply revoking it.

The 7 Steps

1) Obtain management signoff – **in writing**. Seriously. This isn't simply an exercise in CYA, this is your opportunity to be in full communication about what you're planning and perhaps receive useful historical background about the rulebase at the same time.

2) Obtain some database storage – the more, the better. "Which database should I use?" That depends. If your DBA team is willing to give you a few terabytes of SAN-connected database storage on one of your company's databases for use in a rulebase performance tuning/cleanup exercise – great! Otherwise simply use what you do have, and if you don't have anything try either MySQL or PostgreSQL (both freely available Open Source Databases and both well suited to the task at hand) **and** as much storage as you can lay hands on. The reason

for the storage recommendation is that not only will you probably want to end up keeping more and more data, you'll definitely want to speed up your queries by indexing the data you do have. One caveat here: while useful results can be produced with relatively small amounts of data on even a laptop, as the amount of data involved grows server-class storage becomes an important consideration. You do **not** want to be querying a 2 billion row database with even a USB2.0 connection to your hard-drive. Trust me.

3) Create log tables – see Appendix B for the simplest example of an appropriate table creation script.

4) Load the Check Point log files into the database – see Appendix C for export.pl, the script that uses the Check Point `fwm logexport` command to convert gzipped firewall logs into a form suitable for loading into the database (and bzips them), and Appendix D for load.pl, the script that takes the bzipped text logs, uncompresses them, loads them into the database and recompresses them. At this point building some indexes is highly recommended if you don't want to measure the next step in geological time! Space permitting, build indexes on Src, Dst and Service at a minimum.

5) Query the database to build up an idea of which rules are your most commonly used and which rules are used not at all. You can also use these queries to build up rulesets to replace “Any” rules. See Appendix E for some sample queries, including some queries that are more audit-related in nature.

6) Based on the results of Step 5, reorder the rulebase so that the most commonly used rules are at the top. There are two caveats here; one is based on functionality, the other on performance. The caveats are:

a. Drop rules – your intent here is not to change the functionality of the firewall rulebase. “My drop rule **is** my most frequently hit rule – what should I do?” Obviously though you can’t simply move your drop rule to the top of your rulebase, in the situation where you have specific traffic that accounts for most (or even a large percentage) of your dropped traffic, consider explicit drop rules, which can be placed higher up the rulebase.

b. SecureXL – this is Firewall-1’s acceleration product and (assuming you haven’t had to disable it due to certain *features*)…certain types of rule disable it and (here’s the kicker) it doesn’t get disabled on a per-rule basis – once disabled, it’s disabled for all rules from that point forward in the rulebase, so move your rules

that disable SecureXL towards the bottom of your rulebase. Check Point's `fwaccel stat` command gives you what you need here.

7) **(Optional)** Disable unused rules. **Here Be Dragons.** The performance win is over, this step is really about rulebase maintenance and you should not try this at home unless you are really comfortable about your ability to find new employment, should the need arise!

    a) Obtain signoff **in writing** from the Business Owner of the rule you are about to disable. "But I already got Management Signoff in Step 1." That was from your boss. This is from the person who will come screaming for both of your blood if something goes wrong at this point. "How do I determine the Business Owner of a rule?" The Business Owner should be in the comments field of your rules[1] (or

[1] Experience shows that in large firewall teams what works to have in the Comments field is: (a) the name, initials or username of the administrator making the change, (b) the date, (c) the reference in your organization's change control system, (d) the reference in your work request system if this is distinct from your organization's change control system, and (e) the expiry date (if any) of the rule. Your auditor will love you.

the information's location referenced there) – if the information isn't there ask yourself "Who owns the business data or business process that this rule controls access to?"

b) Disable the rule and move it after the drop rule. Why not to simply remove the rule is so that if after having done your due diligence you still manage to disable the wrong rule you can quickly re-enable it (remembering to move it back above the drop rule). Examples of how this undesirable state of affairs can nonetheless occur are: an ad-hoc business process that hasn't been triggered during the period for which you have logs or a log-lived connection e.g. BGP that hasn't been established during the period for which you have logs.

c) Monitor carefully and perhaps even set up a side-channel with your helpdesk so that you get notified for any issues which even *might* be related (in addition to rather than instead of the usual support teams).

4. Conclusion

These procedures have been used at multiple financial institutions, including both banks and insurance companies, in the course of performing incident handling, performance tuning, audits, rule review and cleanup. A procedure very similar to the procedure described here was used on a High Availability firewall cluster where the active node's CPU was running at 35%: after the four most utilized rules were identified and moved to the top of the rulebase and logging turned off on those rules, CPU utilization dropped to 4%.

5. References

[1] Tennyson, Lord Alfred *The Brook* "Men may come and men may go, but I go on for ever."

[2] Eventia Reporter http://www.checkpoint.com/products/er/

[3] SecureTrack http://www.tufin.com/products_overview.php

[4] MySQL http://www.mysql.com/

[5] PostgreSQL http://www.postgresql.org/

6. Appendix A: database schema – logs table

Field	Type	Null	Key	Default	Extra
Num	text	YES		NULL	
Date	text	YES		NULL	
Time	text	YES		NULL	
Orig	text	YES		NULL	
Type	text	YES		NULL	
Action	text	YES		NULL	
Alert	text	YES		NULL	
i_f_name	text	YES		NULL	
i_f_dir	text	YES		NULL	
Product	text	YES		NULL	
Src	text	YES		NULL	
Dst	text	YES		NULL	
Proto	text	YES		NULL	
Rule	text	YES		NULL	
Service	text	YES		NULL	
s_port	text	YES		NULL	
Agent	text	YES		NULL	
orig_from	text	YES		NULL	
orig_to	text	YES		NULL	
From	text	YES		NULL	
To	text	YES		NULL	
Reason	text	YES		NULL	
cat_server	text	YES		NULL	
Category	text	YES		NULL	
Xlatesrc	text	YES		NULL	
Xlatedst	text	YES		NULL	
NAT_rulenum	text	YES		NULL	
NAT_addtnl_rulenum	text	YES		NULL	
Xlatedport	text	YES		NULL	
Xlatesport	text	YES		NULL	
Icmp_type	text	YES		NULL	
Icmp_code	text	YES		NULL	
th_flags	text	YES		NULL	
message_info	text	YES		NULL	
Message	text	YES		NULL	
DCE_RPC_Interface_UID	text	YES		NULL	
ip_id	text	YES		NULL	
ip_len	text	YES		NULL	
ip_offset	text	YES		NULL	
fragments_dropped	text	YES		NULL	
During_sec	text	YES		NULL	
log_sys_message	text	YES		NULL	
rpc_prog	text	YES		NULL	
User	text	YES		NULL	
TCP_packet_out_of_state	text	YES		NULL	

tcp_flags	text	YES	NULL
res_action	text	YES	NULL
Resource	text	YES	NULL
ICMP	text	YES	NULL
ICMP_Type	text	YES	NULL
ICMP_Code	text	YES	NULL
Attack_Info	text	YES	NULL
Attack	text	YES	NULL
session_id	text	YES	NULL
dns_query	text	YES	NULL
dns_type	text	YES	NULL
Reject_category	text	YES	NULL
Packet_info	text	YES	NULL
Total_logs	text	YES	NULL
Suppressed_logs	text	YES	NULL
Application_Info	text	YES	NULL
Reject_id	text	YES	NULL
sys_message	text	YES	NULL
srcname	text	YES	NULL
Internal_CA	text	YES	NULL
Serial_num	text	YES	NULL
Dn	text	YES	NULL
cp_message	text	YES	NULL
cluster_info	text	YES	NULL
URL_filter_pattern_detected	text	YES	NULL
CP_Condition	text	YES	NULL
StormAgentName	text	YES	NULL
StormAgentAction	text	YES	NULL
System_Alert_message	text	YES	NULL
Object	text	YES	NULL
Event	text	YES	NULL
Parameter	text	YES	NULL
Current_value	text	YES	NULL
Auth_method	text	YES	NULL
Sync_info	text	YES	NULL

Note that this is an extremely simple schema – **everything** is represented as a string.

7. Appendix B: Source Code – create_table.sql

```
CREATE TABLE logs
(
num   text,
date  text,
time  text,
orig  text,
type  text,
action      text,
alert text,
i_f_name,
i_f_dir     text,
product     text,
src   text,
dst   text,
proto text,
rule  text,
service     text,
s_port      text,
agent text,
orig_from   text,
orig_to     text,
from  text,
to    text,
reason      text,
cat_server  text,
category    text,
xlatesrc    text,
xlatedst    text,
NAT_rulenum text,
NAT_addtnl_rulenum text,
xlatedport  text,
xlatesport  text,
icmp_type   text,
icmp_code   text,
th_flags    text,
message_info      text,
message     text,
DCE_RPC_Interface_UID text,
ip_id text  YES,
ip_len      text  YES,
ip_offset   text,
fragments_dropped text,
during_sec  text,
log_sys_message   text,
rpc_prog    text,
user  text,
TCP_packet_out_of_state text,
```

```
tcp_flags   text,
res_action  text,
resource    text,
ICMP  text,
ICMP_Type   text,
ICMP_Code   text,
Attack_Info text,
attack      text,
session_id  text,
dns_query   text,
dns_type    text,
reject_category   text,
Packet_info text,
Total_logs  text,
Suppressed_logs   text,
Application_Info text,
reject_id   text,
sys_message text,
srcname     text,
Internal_CA text,
serial_num  text,
dn    text,
cp_message  text,
cluster_info      text,
URL_filter_pattern_detected   text,
CP_Condition      text,
StormAgentName    text,
StormAgentAction  text,
System_Alert_message    text,
Object      text,
Event text,
Parameter   text,
Current_value     text,
auth_method text,
sync_info   text
)
```

8. Appendix C: Source Code – export.pl

```
#!/usr/bin/perl
# Invoke as ./export.pl <node> in the same directory as the gzipped logfiles
# script expects gzipped logfiles to be named:
# <node>__<year>-<month>-<day>_<hour>*.log

$node=shift;
$year=shift;
$month=shift;

foreach $file (glob("${node}__${year}-${month}-*.log.gz")) {
    $logfile = $file;
    $logfile =~ s/.log.gz/.log/;
    $datafile = $logfile;
    $datafile =~ s/.log/.txt/;
    $logfiles = $file;
    $logfiles =~ s/.log/.log\*/;
    $uncomp_logfiles = $logfiles;
    $uncomp_logfiles =~ s/.gz//;
    `time gunzip $logfiles`;
    `time fwm logexport -n -i ./${logfile} -o ./${datafile}`;
    `time gzip $uncomp_logfiles`;
    `time bzip2 $datafile`;
}
```

9. Appendix D: Source Code – load.pl

```
#!/usr/bin/perl
# Invoke as ./load.pl {mysql|postgresql} <node>
# start in the same directory as the bzipped exported logfiles
# load.pl supports both MySQL and PostgreSQL and most importantly
# handles the fact that fields in Check Point FireWall-1 logs appear in
# non-deterministic order;

$database = shift;
if ($database ne "mysql" and $database ne "postgresql") {
die("Script only supports MySQL and PostgreSQL. Giving up");
}

$node=shift;

foreach $file (glob("${node}*.txt.bz2")) {
    $datafile = $file;
    print $datafile, "\n";
    `time bunzip2 $datafile`;
    $datafile =~ s/.bz2$//;
    open( CPLOG, $datafile );
    $header = <CPLOG>;
    chomp( $header );
    $header =~ s/ /_/g;
    $header =~ s/\-/_/g;
    $header =~ s/\//_/g;
    $header =~ s/\://g;
    $header =~ s/Condition/CP_Condition/g;
    open( SQL, ">sql.${datafile}" );

    if ( $database eq “mysql” ) {
        print SQL "LOAD DATA INFILE '",cwd(),"/${datafile}' INTO TABLE logs FIELDS
TERMINATED BY ';' IGNORE 1 LINES (",join(',',split (/;/, $header)), ");";
    }
    elsif ( $database eq “postgresql” ) {
        print SQL 'COPY logs("', join('","',split (/;/, $header)), '") FROM ', "'”, $file,
"'", " DELIMITER ';' CSV HEADER\n";
    }
    else {
        die “Unsupported database engine”;
    }

    close SQL;

    if ( $database eq “mysql” ) {
        `time mysql audit < sql.${datafile}`;
```

```
    }
    elsif ( $database eq “postgresql” ) {
        `time psql audit < sql.${datafile}`;
    }
    else {
        print “Unsupported database engine.\n”;
    }
    `time bzip2 $datafile`;
}
```

Appendix E: Source Code - sample sql queries

Here's an example script that creates new database tables for further analysis. We're putting the ftp, https and telnet traffic from each day in July into a separate table. We're not interested in traffic to isc.sans.org. Notice that strictly speaking we've cheated, looking up isc.sans.org in advance, however this isn't necessary as we can build either build database user-defined functions to do this work for us, or, more sensibly, do this work in our script.

```
#!/usr/bin/perl

$database = shift;
if ($database ne "mysql" and $database ne "postgresql") {
die("Script only supports MySQL and PostgreSQL. Giving up");
}

open( SQL, ">sql" );
foreach $protocol_iterator ( "ftp", "https", "telnet" ) {
    foreach $date_iterator ( 1..31 ) {
        $datestring = sprintf("200807%02d", $date_iterator);
        print SQL "select * into ${protocol_iterator}_logs_${datestring} from logs where
date like '${date_iterator}Jul2008' and service like '", $protocol_iterator, "%' and
action like 'accept' and dst not in ('65.173.218.96', '65.173.218.95');\n"
    }
}
close SQL;

if ( $database eq "mysql" ) {
    `mysql audit < sql`;
}
elsif ( $database eq "postgresql" ) {
    `psql audit < sql`;
}
else {
    print "Unsupported database engine.\n";
}
```

Here's an example query that counts SMTP connections over the period for which we have logs. **All of them**. Unless you know what you're doing (and own the machine you're doing it on, which doesn't run any busy production-like databases) this is almost certainly a horrible mistake!

```
#!/usr/bin/perl

$database = shift;
if ($database ne "mysql" and $database ne "postgresql") {
die("Script only supports MySQL and PostgreSQL. Giving up");
}

open( SQL, ">sql" );
print SQL "select count(*) from logs where service like 'smtp%' and action like 'accept';
close SQL;

if ( $database eq “mysql” ) {
    `mysql audit < sql`;
}
elsif ( $database eq “postgresql” ) {
    `psql audit < sql`;
}
else {
    print “Unsupported database engine.\n”;
}
```