Firestorm Network Intrusion Detection System

John Leach

john@ecsc.co.uk

Gianni Tedesco

gianni@scaramanga.co.uk

Firestorm Network Intrusion Detection System
by John Leach and Gianni Tedesco

Table of Contents

Chapter 1. Introduction

Firestorm is an extremely high performance network intrusion detection system (NIDS). At the moment it is just a sensor but plans are to include real support for analysis, reporting, remote console and on-the-fly sensor configuration. It is fully pluggable and hence extremely flexible.

At the moment Firestorm is still in early development, but a lot of the features you would expect of a sensor are already there.

This guide aims to help you configure and use the Firestorm intrusion detection system. It is the official and definitive source of Firestorm documentation. Accept no substitutes!

1.1. Installation

1.1.1. How Can I Get Firestorm?

Firestorm source code and pre-compiled binaries are available as free software (under the GNU GPL) and can be downloaded from:

- Source Code:
  `http://www.scaramanga.co.uk/firestorm/vX.Y.Z/firestorm-X.Y.Z.tar.gz`
- Source RPM:
  `http://www.scaramanga.co.uk/firestorm/vX.Y.Z/firestorm-X.Y.Z-1.src.rpm`
- i386 RPM (for Linux on PCs):
  `http://www.scaramanga.co.uk/firestorm/vX.Y.Z/firestorm-X.Y.Z-1.i386.rpm`
- PowerPC RPM (for Linux on power-macintosh)
  `http://www.scaramanga.co.uk/firestorm/vX.Y.Z/firestorm-X.Y.Z-1.ppc.rpm`

1.2. Architecture

1.2.1. Sensor

The sensor component is called ’firestorm-nids’, it sniffs traffic on your network, analyses it (usually using snort signatures) and spools the alerts in extended log (elog) format.

1.2.1.1. Stateful Analysis

Firestorm can analyse state information on the network. Firesstorm can reassemble IP fragments, perform TCP connection tracking, TCP stream reassembly and application layer stateful analysis.

1.2.1.2. Full Application Layer Decode

Firestorm has the capability to fully decode application layer protocols, at the moment HTTP is the only supported protocol but others will follow shortly.

1.2.1.3. Snort Compatibility

Snort rule compatibility is fairly extensive. We aim to track the default signature set that ships with snort 1.9. As far as we know snort 1.8 rules should work also. More information can be found in the *snort.compatibility* document.

1.2.1.4. Rate-Limiting of Alerts

Firestorm features the ability to rate-limit alert output to protect itself from DoS attacks. This feature is rather unique. Built-in alerts (such as state tracking violations) are rate-limited by default and snort rules have two new keywords ’rate’ and ’burst’ for configuring on a rule-by-rule basis.

1.2.1.5. Anomaly Detection

Firestorm has infrastructure for supporting anomaly detection modules, however at this time, no such modules have yet been written.

1.2.2. Extended Logs

Extended logs (or elogs for short) are a new format for transporting alert data. They contain not only the packet data but also the alert information, decode information, state-tracking information, and any other packet meta-data. The advantage of using extended log format is the ability to keep all data in one file.

Firestorm can usually achieve full disk throughput when alerting. In fact, Firestorm can saturate many spindles simultaneously by balancing alerts between multiple spools.

Be aware that the elog format is not yet finalised (and won’t be until version 1.0.0) and is likely to change at any point in time. The files are versioned so you shouldn’t experience data corruption. If you would like to be able to convert from one version or the other, you can pay me to write a conversion tool ;)

1.2.3. Stormwall

Stormwall is as yet unfinished. Its purpose is to monitor alert spools and perform actions when new elog files appear. The Firestorm sensor notifies Stormwall of changes to the spool. This program will facilitate push-style remote logging. Both push and pull logging will be supported by version 1.0.0.

1.2.4. Console

The console is started but is very early in development. It is a GNOME 2 application. By version 0.6.0 it will be fully usable and allow the analyst to search, sort, filter, correlate and extract data from his sensors.

Chapter 2. Firestorm NIDS Sensor

2.1. Configuration File

The firestorm-nids program has one optional command line argument to specify the path to the configuration file. If firestorm-nids is run with no arguments it will default to /etc/firestorm.conf. The Firestorm configuration file is an ASCII text file with UNIX line terminators. If a line is empty of begins with a hash (#) character it is ignored. It consists of zero or more lines of the format:

```
directive <parameters>...
```

Arguments to various directives are usually quite consistent and look something like this:

```
directive string="hello" str='foobar' int=23 size=48K bigsize=12MB
```

The notational convention '<args>...' will be used from now on to denote this style of arguments. All configuration directives are enumerated in detail below.

2.1.1. firestorm_root

This directive tells Firestorm which directory to move to when run. All paths mentioned in the config file are relative to this path. You shouldn't run Firestorm inside some directory which you later hope to unmount ;)

```
firestorm_root /var/firestorm
```

2.1.2. chroot

Firestorm can run inside a so-called "chroot jail", this prevents Firestorm from being able to read files located outside the jail. You will usually want to say yes to this for security reasons. Note that technically it is actually possible for a skilled attacker to break out of a chroot jail, nothing can replace code audit...

```
chroot yes
chroot no
```

2.1.3. capture

The capture directive instructs firestorm-nids where to capture network traffic from. Firestorm is quite unique in the sense that it can capture from a variety of sources and a programmer can quite easily write extensions to capture from new data sources. Only one capture can be used at once.

```
capture type <args>...
```

I will briefly enumerate the capture types currently supported by Firestorm, and the usage of each. Note that although filenames are relative to the firestorm_root, you can still access files outside of the chroot. Also note that none of these plugins set promiscuous mode by themselves. You must enable promisucous mode yourself if you require it. This may be as simple as 'ifconfig ifname up promisc', see your OS vendors documentation for more details.

2.1.3.1. pcap

Most people will want to capture from the live network with libpcap. This plugin has only one option 'if' which is used to specify which interface to listen on. (e.g: if='eth0' or if='any' to listen on all interfaces).

2.1.3.2. pcapfile

Firestorm can also capture from libpcap files, such as those created by tcpdump or ethereal. This plugin also has only one argument 'file' to specify the filename. (e.g: file='./captures/mynetwork.cap').

2.1.3.3. linux

Firestorm has the ability to support high-speed OS specific capture plugins. Use this plugin if you run a recent Linux kernel with mmap() packet socket support. This plugin takes two options 'if' and 'blocks' where 'if' is an interface to listen on and blocks is a number specifying how many blocks to use in the ringbuffer. Generally the higher this number the more memory is used and the less packets will be dropped - you can look in the Firestorm log output to get an idea of how much memory (in kilobytes) it translates to. (e.g: if='any' blocks=128).

2.1.3.4. tcpdump

This plugin is a hi-speed alternative to the pcapfile plugin and doesn't depend on libpcap. This plugin is recommended over and above pcapfile, although your mileage may vary. It takes the same arguments as pcapfile (e.g: file='./myfile.cap').

2.1.3.5. fagrouter

This plugin is for simulating TCP traffic which has been randomly packetized to evade NIDS which do not perform TCP stream reassembly. It's purpose is for testing and validation of application layer decodes running atop the TCP protocol. You can set files containing the client and server sides of the stream and also set a maximum segment size (eg: cl='client' sv='server' maxseg='512').

2.1.4. effective_uid / effective_gid

These directives are ignored unless Firestorm is run as root, their purpose is to prevent Firestorm from actually processing any data while running with a privileged EUID (effective user id). They take precisely one argument which is a numeric UID or GID respectively.

```
effective_uid 303
effective_gid 303
```

2.1.5. load_plugins

Firestorm is totally plugin based and cannot function without loading the required plugins. This directive allows you to specify paths to directories full of plugins to load. Note that you can load plugins from outside the chroot. For each directory specified Firestorm will attempt to load all plugin files contained therein, directories are NOT recursed.

```
load_plugins /usr/lib/firestorm/capture
load_plugins /usr/lib/firestorm/protocols
load_plugins /usr/lib/firestorm/detection
```

2.1.6. load_plugin

This directive is similar to load_plugins except that it loads a single plugin file. Firestorm will fail if the plugin specified cannot be loaded.

```
load_plugin /usr/lib/another-plugin.so
```

2.1.7. preprocessor

The preprocessor directive instructs firestorm-nids to activate any optional modules. Currently the only additional modules available are tcpstream and ipfrag.

```
preprocessor name <args>...
```

2.1.8. logfile

This directive is perhaps a misnomer. It essentially tells Firestorm to daemonize and send all diagnostic messages to a log file. The single argument is the path to the log file. You will nearly always want to set this directive. Note that the log file is overwritten every time that Firestorm is run. It is only intended as a record of what happened last time Firestorm was run for diagnostic purposes. If you do not set this directive, Firestorm will not daemonize and output messages to standard out (screen).

```
logfile firestorm.log
```

2.1.9. output

Firestorm outputs alerts in elog format to a spool directory whose path is specified by the output directive. Firestorm can do automatic log rotation when the logs reach a specified file size, or a certain amount of time has elapsed. Firestorm never rotates empty logs. Log files that have been successfully spooled have names beginning with '@', they are guaranteed to be written to disk, and not-corrupt. The current log file takes the name 'alert.elog'. You should not read from or use this file it is not guaranteed to be in any particular state.

The arguments for this directive are pretty self explanatory from the example below. One thing to take note of is that if the 'minutes' parameter is set to 0, Firestorm won't rotate by time and if the 'size' directive is set to 0, Firestorm won't rotate by size. If both are set Firestorm rotates on whichever comes first. The buf parameter is more thoroughly explained in Section 2.2.3>.

```
output dir='log' minutes=60 size=512K stormwall=none buf=16k
```

2.1.10. signatures

The signatures directive tells Firestorm where to load signatures from, you may load as many signature files as you wish. Currently the only supported signature format is snort. You can't put signatures, variables or anything snort related directly in to Firestorm.conf.

```
signatures snort ./snort-rules/classification.config
signatures snort ./snort-rules/shellcode.rules
```

2.2. Advanced Configuration

2.2.1. IP De-fragmentation

This plugin has two roles, the first is the de-fragmentation of fragmented IP datagrams and the second is to statefully detect IP fragmentation based attacks (such as teardop, boink, etc.). To enable this module you will need to add the 'ipfrag' preprocesser using a configuration line similar to the following:

```
preprocessor ipfrag mem_hi=4096k mem_lo=3072k minttl=1 timeout=30
```

2.2.1.1. mem_hi / mem_lo

In order to prevent denial-of-service attacks, the ipfrag module requires specifying an upper boundary on the amount of memory used to remember fragmented packets. Firestorm will allocate up to 'mem_hi' bytes of memory, when this threshold is hit, Firestorm will prune the oldest packets until only 'mem_lo' bytes are used. This strategy is identical to that used in the BSD/Linux derived IP stacks. The default values are 1MB and 768KB respectively. These are very conservative values, on a hot network you may want to use much more memory.

2.2.1.2. minttl

If Firestorm is not sniffing directly from the same network as your protected hosts then it is possible for an attacker to send IP fragments with low TTLs which the IDS will see but the target system will not (as the packet gets dropped when the TTL expires). Setting this value will make ipfrag ignore packets with TTLs lower than the value. You will usually want to keep this as 0 (the default).

2.2.1.3. timeout

The ipfrag modules can timeout fragment reassembly after a given amount of time (in seconds). For best security you should set this value to be the same as your target OS. The default is 60 seconds, Linux is 30 seconds by default (/proc/sys/net/ipv4/ipfrag_time).

2.2.2. TCP Stateful Inspection and Stream Reassembly

Firestorm can perform stateful inspection of TCP packets to avoid DoS attacks such as stick and snot. When enabled Firestorm tracks TCP sessions and maintains state information (including support for window tracking, PAWS, window-scaling, and many other TCP protocol options).

TCP stream reassembly is still a work in progress so regard it as an experimental feature.

```
preprocessor tcpstream num_streams=1k minttl=1 reassemble=yes num_flows=1k
```

2.2.2.1. num_streams

There is an upper bound on the number of connections tracked to prevent denial-of-service attacks. When the limit is reached the oldest connections are evicted from the connection tracking table. The default is 2048 which is very conservative. On a hot network this should be much higher.

2.2.2.2. minttl

Sets the minimum ttl value for which Firestorm will examine TCP segments. See the discussion in Section 2.2.1> for more information. By default this value is 0, don’t fiddle with it unless you understand what your are doing.

2.2.2.3. reassemble

Enables/disables TCP stream reassembly. Can be either ’yes’ or ’no’. You should probably keep this enabled at all times, the overhead is pretty minimal.

2.2.2.4. num_flows

The tcpstream module also keeps track of application layer state information (termed ’flows’). This setting controls the upper bound on application layer state objects. Set to the same as num_streams for guaranteed 100% coverage. It is set to 1024 by default.

2.2.3. High Performance Alert Spooling

The output directives in your firestorm.conf can make a massive impact on performance if applied well. You should be able to log almost anything and not worry about destroying sensor performance. The Firestorm analysts approach should be to log anything that might matter and just use the console to filter away the chaff. This goal is a long way off yet, but one component is already in place, world-class alert spooling performance. To configure this correctly you must first pay attention to the ’buf’ parameter.

The buf parameter sets how large the output buffer should be in bytes. The higher this value, the higher the throughput of alerts, and the less susceptible to denial-of-service attacks you become. The cost of a high value however is reliability, there is a longer period of time where the logs are not written to disk and are lost if Firestorm crashes or is killed forcefully. However, the relatively small detriment to reliability can be amortised by setting log rotation size or time limit.

With Firestorm it is possible to scale your alerting throughput right up to ’enterprise level’ with very little hardware investment. Simply place one alert spool on each hard disk in your sensor - alerts will be balanced between them meaning that you only need add a couple of fast SCSI disks to log alerts at full gigabit rate.