Rule file format for NetL7 compiler

NetLogic Microsystems

May 2009

Contents

Chapter 1

Main Page

The compiler reads one input rule file in the format described below and produces one binary image that can be loaded into the device via dataplane API.One rule file represents one database of rule groups. One binary image represents one database. Multiple binary images can be loaded into the device with multiple load_database calls of dataplane API. The maximum number of simultaneously loaded databases is device dependent. The user can compile any number of databases at any time.

1.1 Compiler Input

One rule file contains one or more rule groups. Each rule group contains one or more rules.

1. General file formatting:

   Empty lines are ignored. Lines starting with '#' character are ignored as comments.

2. Rule group specification:

   Each rule group is specified as a section. Each section starts with a group header and follows by one or more rules. The group header has the following syntax:

   ```
   [group_id]    or    [group_id:group_flags:group_name]
   ```

   - `'`[' and ']' and ':' are literal characters.
   - `group_id` is an integer in the range 1 - 1023. The group id must be unique across all groups that are specified in the file. group_id zero is reserved.
   - `group_flags` and `group_name` are ignored by the compiler.

3. Rule specification:

   The rules for a given group are specified after group header. Each rule is specified on one or two lines.

   If group header is specified as [group_id] each rule looks like:

   ```
   rule_id regex
   rule_modifiers
   ```

   If group header is specified as [group_id:group_flags:group_name] each rule looks like:

```
rule_id rule_flags rule_name regex
rule_modifiers
```

- `rule_id` is an integer in the range 1 - 65535 (16-bit integer). This value along with the group_id is returned to the user when a match occurs. It is desirable for rule_id to be unique within a group. The rule_id zero is reserved.
- `rule_flags` (optional) are four hexadecimal digits (e.g., 1100 or 10FF). If these four digits are PQRS, the P is used to specify the format of the pattern. 1 indicates PCRE syntax and 2 indicates Netscreen syntax. The Q is used to specify if the rule is to be applied in a non-greedy or all matches fashion for overlapping matches. 1 (non-greedy) indicates for a series of overlapping matches only the shortest match should be returned and 0 indicates all overlapping matches for this rule should be returned. Non-overlapping matches are always returned. All other values of P and Q are reserved. The R and S are ignored.
- `rule_name` (optional) contains name of the rule and is ignored. It should not have any spaces in it.
- `regex` is the actual rule/regular expression written in the syntax specified by rule_flags up to a maximum of 16k characters. The regex ends with new line that is not part of the regex. A single group may have rules of different syntax.

See Appendix A for a sample input file in this format.

4. rule_modifiers (optional) (supported by NLS505, NLS1005, NLS1008 and NLS2008 devices):

   The user can modify the default behavior of a rule match. The default behavior is to return the rule_id for every match regardless of match length or offset. The rule_modifiers (if specified) should follow the rule on the next line. The keywords should be separated from other keywords and operands by a space and should be all on one line:

```
[priority #] [min_match_length #] [min_offset|exact_offset|max_offset #] [non_greedy]
[match_once] [trigger_group_id|match_and_trigger_group_id #]
```

   - `priority` - if this keyword is specified, the priority of this rule should follow after a space as a decimal integer from 0-3, where 0 represents the highest priority and 3 represents the lowest. The default priority for a rule is zero (highest priority) if this keyword is not specified. HW allows to set result clamps independently for different priorities. The user can use different result clamps as way to prevent DoS attacks. HW does not sort the results with different priorities, though results for the same flow are guaranted to come in the same order as packets received.
   - `min_match_length` - if this keyword is specified, the minimum match length should follow after a space as a decimal integer from 1-65535. If this keyword is specified, the match must meet the minimum match length criteria to be considered a match. No result will be returned for a match that does not meet the minimum length. For stateful rule groups, this feature works the same for matches contained entirely within a packet and those that span two or more packets - the match length is a function only of the pattern, not the existence (or lack thereof) of a packet boundary anywhere in the match.
   - `min_offset|exact_offset|max_offset` - if either of these keywords is specified, the minimum or maximum or exact allowable offset should follow after a space as a decimal integer from 1-65535. A minimum offset criterion is interpreted as greater than or equal to. A maximum offset criterion is interpreted as less than or equal to. An exact offset criterion is interpreted as equal to. In all cases, the specified offset is relative to start of packet for stateless flows and start of flow for stateful flows. No result will be returned for a match that does not meet the specified criteria.

- `non_greedy` - if this keyword is specified, the rule will be compiled in non-greedy mode: for a series of overlapping matches return the shortest match and start looking for the match again.
- `match_once` - if this keyword is specified, the rule will be compiled in match-once mode: for a series of overlapping matches return the shortest match and never return the matches again for the given rule until the end of the flow (stateful flows) or until the end of the packet (stateless flows), so the rule will report a match only once.
- `trigger_group_id` - see Rule Chaining for a detailed explanation
- `match_and_trigger_group_id` - see Rule Chaining for a detailed explanation

See Appendix B for a sample input file with pattern modifiers.

5. The group header can also be omitted in the rule file, in such case:

   . all rules belong to a single group with group_id = 1

   . rule_modifiers are not supported

   . all rules assumed to be in PCRE format

   . each line (rule specification) should be written as

```
rule_id rule_flags rule_name /regex/
```

   - `rule_id` is an integer in the range 1 - 65535. The rule_id needs to be global within the entire file. Rule_id zero is reserved.
   - `rule_flags` are two hexadecimal digits (e.g. 1E or 0F) and ignored by the compiler.
   - `rule_name` is the name of the rule and ignored. It should not have any spaces.
   - `regex` is the actual rule specified in PCRE format with leading and trailing '/'.

   See Appendix C for a sample input file in this format.

1.1.1 Rule Chaining

The rule_modifier keywords **match_and_trigger_group_id** and **trigger_group_id** are used for chaining rules. Both the modifiers take a Group id (decimal integer from 1-1023) as value. The value should follow the above keywords with a space in-between. The specified Group id should be a valid id for the database. i.e There should a group with the specified group id in the database. The triggered group can be placed anywhere in the rule file. Triggering multiple groups from a single rule is not allowed.

In case of the above rule modifiers once the rule matches, all the rules in the specified trigger group will be activated. All the activated rules will start matching from the next traffic byte. In case of keyword trigger_group_id there will be no result reported after the rule is matched. Only triggering of the group will be done. In case of keyword match_and_trigger_group_id there will be a result reported and the specified group will be triggered.

The following is a sample file with rule chaining

```
[1]
1 http://
match_and_trigger_group_id 2
2 ftp://
match_and_trigger_group_id 2

[2]
```

```
3 [A-Za-z][A-Za-z0-9]*\.
trigger_group_id 3

[3]
4 netlogic\.com
5 net_logic\.com
```

In the above example upon matching rule 1 or 2 a result will be produced with the corresponding rule id and rule group 2 will be activated. If rule 3 matches after being activated by rules in group 1 there will be no output but the rule group 3 will be activated.

All the rules in the triggered groups will start matching from the next traffic byte. The traffic for the triggered rule should follow immediately the traffic for the triggering rule. In the above example, the traffic

```
http://in.netlogic.com
```

will report a match at offset 7 rule_id 1 and offset 22 rule_id 4 whereas traffic:

```
http://in.snetlogic.com
```

will report a match at offset 7 rule_id 1 only, since all the rules in group id 3 will stop matching at 's'.

Loops are not allowed in the rule chaining. If loops are detected in the chaining, the chaining from the highest group-id group to the lowest group it connects to will be dropped.

In the below example rule file

```
[1]
1 http://
trigger_group_id 2

[2]
2 www\.
trigger_group_id 3
3 localhost

[3]
4 netlogic.com
trigger_group_id 1
```

the trigger from rule 4 will be dropped as it creates a loop.

The head groups in the rule file will be active for all the traffic bytes. The head groups are the groups that are not triggered by any rule. All the triggered groups will become active only if any of the rules that trigger it matches. In case of a loop the loop will be broken by removing some chaining as explained earlier and then the head group will be selected. In the above example group-id 1 is the head group.

The other rule modifiers are not supported when rule chaining modifiers are used

1.2 Compilation Stats, Debug Info, Error Handling

The compiler may print statistical and debug information which is device dependent.

The compiler may produce warning messages when given rule may cause unexpected performance impact or may occupy too much HW memory.

The compiler produces error messages when rule syntax is incorrect or rule cannot be compiled with given constraints.

1.3 Appendix A

Sample input file for multiple rule groups without rule modifiers

```
# my group
[3]
1 abcd
20 ab.*cd

# FTP group section
[1:123F:idp_ftp_grp]
1 1000 xxx abcd
2 2100 ftp-get-filename-2 WinRun\.exe
14 2100 ftp-get-filename-3 \[Addybk\.dll\]
72 1100 ftp-banner-8 (?i)(tRaiToR:\d+\ssErVeR\srEaDy)

# HTTP group
[2:337F:idp_http_post_grp:copies 2]
101 2100 http-post-url-parsed-param-4 .*\[options_identities\.php\]
102 2100 http-post-url-parsed-param-10 \[/s/l/firstping\].*
103 1100 http-post-url-parsed-32 /callaghan/GetInfo.*
104 2101 http-post-url-parsed-9 \u\[/_vti_bin/_vti_aut/fp30reg\.dll\]\u.*
105 2101 http-post-url-parsed-13 .*tbl_(addfield|alter|change|create)\.php
```

1.4 Appendix B

Sample input file with multiple rule groups and rule modifiers

```
# comment
[1]
1 a.*b
2 c.*d.*e
3 ^.{0,100}huh

[2:1000:group_2]
2 1000 pattern_2 a.*b
min_match_length 12
3 1000 pattern_3 abcd
4 1000 pattern_4 abcde
priority 2
# non-greedy a.*b is the following:
5 1100 pattern_5 a.*b
# or the following:
6 1000 xxx a.*b
non_greedy

[3]
10 a.*b
min_match_length 100
11 a.*b
non_greedy
12 a.*b
13 a.*b
match_once

[4:1000:group_4]
105 1000 pattern_105 ^.*acbd
exact_offset 100
```

1.5 Appendix C

Sample input file for a single group

```
00000012 20 av-vir1-2 /\x68\x00\xE9\xAA\xB8\x35\x86\xF5\x6A\x10\x68\x70\x37\x46\xDF\x73\x08\xAB/
00000084 00 av-malw-3 /\x23\xBB/
00004081 00 Trojan-Downloader.Win32.Agent.aht /\x12\x12\x12\x34\xB8\xBA/
```