FIREDRL

Language Reference Manual

COMS 4115
Programming Languages and Translators

Marvin J. Rich
e-mail: mjrich@us.ibm.com

1.0 Introduction

The purpose of this manual is to convey the syntax and semantics of the FIREDRL language (*F*irewall *I*ntegrity *R*eview *E*xploit *D*etection *R*eport *L*anguage). Syntax and examples of valid instances of each statement are provided. A BNF format grammar utilize to specify each statement syntax in a concise manner. The semantics associated with each statement is also provided.

2.0 Lexical Conventions

Each statement in the Firedrl language consists of a set of fundamental tokens, where the sequence of tokens uniquely identify a valid statement. This section defines the set of tokens and keywords used to formulate valid Firedrl language statements during lexical analysis. Lexical tokens will be capitalized in order to easily distinguish them in the BNF grammar for language statements.

2.1 Separators:

Blanks (whitespace), tabs, new line, and carriage return are ignored when the language is scanned during lexical analysis. These separators are used to separate tokens, and to provide a measure of flexibility in the input format for readability (can have an arbitrary number of consecutive separator characters between tokens).

BNF Grammar:

WS : (blank | tab | newline | carage_return | newline carage-return)

blank	: ‘ ‘
tab	: ‘\t’
newline	: ‘\n’
carage_return	: ‘\r’

Other single character separators also exist as follows:

SEMI	: ‘ ‘
COMMA	: ‘\t’
COLON	: ‘\n’
LPAREN	: ‘(‘
RPAREN	: ‘)’
LBRKT	: ‘{‘
RBRKT	: ‘}’
SLASH	: ‘/’
POUND	: ‘#’
STAR	: ‘*’
DOT	: ‘.’

2.2 Constants:

Integers are the only numeric constants defined in the Firedrl language. An integer constant is defined as a string of one or more decimal digits from ‘0’ to ‘9’:

BNF Grammar:

```
 INT :  ( DIGIT)+
  DIGIT   : ‘0’ ... ‘9’
```

2.3 Identifiers:

Indentifiers are used to label network objects and local variables in Firedrl. A valid identifier must consist of an initial alphabetic character, followed by zero or more alphabetic, decimal, or ‘_’ characters.

BNF Grammar:

```
 ID:  ALPHA (ALPHA | DIGIT | ‘_’)*
  ALPHA  : (‘a’ .. ‘z’ | ‘A’..’Z’)
```

Examples:

This_is_valid
atemp_name
A
B1_B2

2.3 Operators:

Operators indicate the transformation or comparison of objects or data.. The ‘:=’ operator is used to define host and network objects in Firedrl’s network domain. The ‘@=’ operator is used to assign hosts to a network. The ‘<-’ operator is used to assign packet filtering rules to a host. Regular variable assignments utilize the ‘=’ operator. The standard set of comparison operators are also defined (‘>’,’<‘,’==’, ‘<=’, ‘>=’) . The iteration operator ‘+=’ supports aggregation of host and network attributes within a loop.

Grammar:

```
  DEFINE           :  ‘:=’
  EQ                :  ‘=’
 H_ASSIGN         :  ‘@=’
  R_ASSIGN         :  ‘<-’
  VAR_ASSIGN     :  ‘=’
  CMPR              :  (‘<‘ | ‘>’ | ‘==’ | ‘<=’ | ‘>=’ )
  INCR_ASSIGN  :  ‘+=’
```

2.4 Comments:

Line comments are ignored in Firedrl (treated like white space). A line comment starts with a "//" two character sequence. Upon encountering this character sequence, the remaining line text is ignored, including the initial '//' token. The remaining line text is defined to be any characters up to and including the end of line character(s). Therefore if a comment shares a line with a language statement, it should follow the statement.

Grammar:

```
comment    :  com_token restline
com_token  :  '//'
restline   :  (~('\n' | '\r'))* ('\n' | '\r')
```

Example:

```
 //This is a standalone comment
<language stmt>      // This comment shares line with a statement
```

2.5 Keywords:

The following keywords are reserved, and should not be used as an identifier name:

do   for   if   host   network   attack

3.0 Syntax Notation

A Firedrl language consists of one or more statements. Each statement can be in one of eight possible categories. This section will define the syntax of the statement or statements supported in each of the eight statement categories.

Grammar:

```
lang : (stmt)+
stmt : host       | network      | rule    | loop
       decision   | assignment   | probe   | report
```

3.1 Host Declaration Statement:

A host statement defines one or more hosts that reside in the network configuration. Each host is defined by an identifier and at least one associated host interface address. If a host has more than one host interface address, then each host address must be qualified by the network name it is associated with. A host address consists of a “dot.Int” notation, where one can specify from 1 to 4 blocks of 8 bits. Each 8 bit block is separated by a dot if more than 8 bits are specified. The host interface address only requires enough bits encoded to represent the address right justified in a full 32 bit notation. An implicit cardinality is associated with each hostname such that the language supports looping through hostnames. The cardinality is assigned in the order the hosts are defined.

BNF Grammar:

```
host : “host_def” DEFINE hostname (COMMA hostname)* SEMI
hostname : (ID COLON intfc | ID COLON intfc SLASH ID)
intfc : INT (DOT INT (DOT INT (DOT INT)? )? )?
```

Examples:

```
host_def := HostA:45;
host_def := HostB:2.25, HostR1:2.27/Net1:32/Net2;
```

The first example defines a single host, HostA, with a single interface address of decimal 45. The second example defines two hosts, HostB and HostR1. HostB has a single interface address of two 8 bit integers 2.25. HostR1 has two interfaces defined where the first, 2.27, is associated with network Net1 and the second interface, 32, is associated with network Net2.

3.2 Network Declaration Statement:

A network statement defines one or more networks that comprise the total network configuration. Each network is defined by an identifier and a associated 32 bit network address. The network address consists of a 32 bit “quad-dot” notation followed by a slash and an integer number. The 32 bits are defined in four blocks of 8 bits, separated by dots, with a decimal encode of each of the 8 bits. The most significant bits designate the network address, and the least significant bits are set to zero. The integer value after the network address defines the number of bits, from most significant bit, that make up the network address. If a host has more than one host interface address, then each host address must be qualified by the network name it is associated with. A host address consists of a “dot.Int” notation, where one can specify up to 4 dotted integers (just need to define sufficient integers to represent the host address right justified in a full quad “dot.int” IP address). An implicit cardinality is associated with each network name such that the language supports looping through network names. The cardinality is assigned in the order the network names are defined.

BNF Grammar:

network : “net_def” *DEFINE netwkname (COMMA netwkname)* SEMI*
netwkname : *ID COLON ipaddr SLASH INT*
ipaddr : INT DOT INT DOT INT DOT INT

Example:

```
net_def := Internet:1.73.0.0/16,NetworkA:1.73.2.0/24;
```

This example defines two networks. The network called Internet has a network address of 1.73.x.x since the network is defined for the first 16 bits. The network called NetworkA has a network address of 1.73.2.x since the first 24 bits define the network address.

3.3 Rule Statements:

The functionality of each firewall is defined by rules that are attributed to the host interface. These rules must be defined before they can be assigned to a host interface. There are three types of rules based on the target protocols of TCP, UDP and ICMP respectively. Each rule has fields specific to the protocol that designate what is to be allowed or blocked. A '*' in the source or destination address matches all IP addresses. The source and destination ports can be optionally specify logical comparison of port values. The connect field for TCP rules specifies which control bits are allowed to be set.

BNF Grammar:

```
 rule : "rule"  ID 'DEFINE LPAREN
             (tcp_rule | udp_rule | icmp_rule)
        RPAREN SEMI
tcp_rule     : "TCP"  direction src_addr dest_addr src_port dest_port
                connect action
udp_rule    : "UDP"  direction src_addr dest_addr src_port dest_port action
icmp_rule   : "ICMP"  direction src_addr dest_addr type action
direction    : ("in" | "out")
src_addr     :  "src="   (ipaddr | STAR)
dest_addr   :  "dest=" (ipaddr | STAR)
src_port     :  "src_port="  ( (CMPR)? INT | STAR)
dest_port   :  "src_port="  ( (CMPR)? INT | STAR)
connect      :  INT
type          :  INT
action        :  ("allow" | "block")
```

Examples:

```
rule  Rule1 := TCP   in     *     1..74.2.45  *    >1024   block;
rule  Rule3 := ICMP out   1.74.2.45     *   3   block;
```

Rule1 specifies that incoming TCP packets to ipaddr 1.74.2.45 with port greater than 1024 should be blocked. Rule2 specifies that ICMP type 3 responses from 1.74.2.45 should be blocked.

3.4 Loop Statements:

Loop statements allow for iterations. In addition to looping on a variable, a network and host loop is also defined. The network and host loops take advantage of the implicit cardinality assigned to network and host definitions, to loop through network or host names. A block of statements can be executed under each loop construct.

BNF Grammar:

```
loop  :  (net_loop | host_loop | do_loop)
net_loop : "for" "network" LPAREN ID ("to" ID)? RPAREN
           LBRKT (stmt)* RBRKT
host_loop : "for" "host" LPAREN ID ("to" ID)? RPAREN
           LBRKT (stmt)* RBRKT
do_loop : "do" LPAREN ID EQ INT ("to" INT ("by" INT)? )?  RPAREN
           LBRKT (stmt)* RBRKT
```

Examples:

```
for network(NetworkA to NetworkZ) { ..... }        //Loop through network names
for host(host1 to host10) {.....}                   //Loop through host names
do (var = 1 to 5) {.....}                           //Generic variable loop
```

3.5 Decision Statements:

Decision statements allow alternate segments of code to execute based on the outcome of a test predicate. Under a network loop context, a network if statement is defined. Under a host loop context, a host if statement is defined. The network and host if statement allows for unique code to execute, based on the network or host currently iterated on.

BNF Grammar:

```
decision  :  (net_if | host_if | var_if)
net_if : "if" LPAREN "network" CMPR  ID  RPAREN
           LBRKT (stmt)* RBRKT ("else" LBRKT (stmt)* RBRKT)?
host_if : "if" LPAREN "host" CMPR  ID  RPAREN
           LBRKT (stmt)* RBRKT ("else" LBRKT (stmt)* RBRKT)?
var_if : "if" LPAREN ID CMPR  INT  RPAREN
           LBRKT (stmt)* RBRKT ("else" LBRKT (stmt)* RBRKT)?
```

Examples:

```
if (network == NetworkQ ) { ..... }            //if loop network name equals NetworkQ
if (host >= HostZ)   {.....}                   //if loop host name >= HostZ
if (var > 5)  {.....}                          //Regular variable if statement
```

3.6 Assignment Statements:

Assignments are made to networks and hosts. Networks are assigned hosts and hosts are assigned rules. Rules actually apply to interfaces. Therefore if a host has more than one interface, the assignment should be made within a network loop (the associated interface to apply the rule will then be known). Increment assignments (e.g. +=) are also allowed. Types must be compatible in the assignments. A network must be assigned a hostname. A host must be assigned a rule name.

BNF Grammar:

assignment : (*net_assign* | *host_assign* | *incr_add*)
net_assign : *ID H_ASSIGN ID (PLUS ID)* SEMI*
host_assign : *ID R_ASSIGN ID (PLUS ID)* SEMI*
incr_add : *ID INCR_ASSIGN ID SEMI*

Examples:

```
Network1 @= hostB;                  // assign hostB to network Network1
hostB <-  Rule2;                    // assig Rule2 to hostB
for (Network1) { hostc <- Rule3; }  //hostc interface for Network1 assigned Rule3
Network1 += hostC;                  // add hostC to Network1
```

3.7 Probe Statements:

Probe statements initiate packet tests of the network configuration. They work in concert with variable control statements to send packets of various formats to illicit response from firewalls in the network. The three types of probe packets are TCP, UDP and ICMP packets respectively. Variables can be substituted for iteration ranges within loops.

BNF Grammar:

```
probe : (tcp_pkt  | udp_pkt | icmp_pkt )
tcp_pkt  : "snd_tcp" LPAREN "src="    v_ipaddr POUND v_int
                                "dest="   v_ipaddr POUND v_int
                                "cntl="   v_int
                         RPAREN SEMI
udp_pkt  : "snd_udp" LPAREN "src="   v_ipaddr POUND v_int
                                 "dest="  v_ipaddr POUND v_int
                         RPAREN SEMI
icmp_pkt  : "snd_icmp" LPAREN "src="    v_ipaddr
                                   "dest="  v_ipaddr
                                   "type=" v_int
                         RPAREN SEMI
v_ipaddr  : v_int  DOT  v_int  DOT  v_int  DOT  v_int
v_int  : (INT | ID)
```

Examples:

```
snd_tcp(src=1.74.5.3#4756 dest=1.24.5.2#80 cntl=0);      //fixed tcp packet
snd_icmp(src=1.74.5.4 dest=1.75.2.1 type= 4);            //fixed icmp packet
snd_tcp(src=1.74.5.3#2072 dest=1.24.5.C#80 cntl=0);      //variable destination address
snd_icmp(src=1.74.5.4 dest=1.23.2.1 type=D).            //variable type field
```

3.8 Report Statement:

The report statement provides the capability to observe the affects of exploits attempted via probe statements. One can target reports to a specific host or obtain reports for all hosts. The report can be specific to a particular protocol, as well as a specific traffic direction. A special ID called "attack" is used to target reports from the initiator of packets to the network.

BNF Grammar:

```
report : "report"  "target=" (STAR | "attack" | ID)
                   "protocol=" (STAR | "TCP" | "UDP" | "ICMP")
                   "direction=" (STAR | "snd" | "rcv" )
          SEMI
```

Examples:

```
report  target=* protocol=TCP  direction= *;            //report all TCP activity on all hosts
report  target=HostA protocol=ICMP direction=rcv;  //report ICMP pkts received on HostA
```

4.0 BNF Summary

```
lang : (stmt)+
stmt : host         | network      | rule     | loop
       decision     | assignment   | probe    | report

 host : "host_def"  DEFINE hostname (COMMA hostname)* SEMI
 hostname :  (ID COLON intfc  |  ID COLON intfc SLASH ID)
 intfc :  INT  (DOT INT (DOT INT (DOT INT)? )? )?

 network : "net_def"  DEFINE netwkname (COMMA netwkname)* SEMI
 netwkname :  ID COLON ipaddr SLASH INT
 ipaddr :  INT DOT INT DOT INT DOT INT

 rule : "rule"  ID 'DEFINE LPAREN
           (tcp_rule  |  udp_rule  |  icmp_rule)
        RPAREN SEMI
tcp_rule      : "TCP"  direction src_addr dest_addr src_port dest_port
                 connect action
udp_rule     : "UDP"  direction src_addr dest_addr src_port dest_port action
icmp_rule    : "ICMP"  direction src_addr dest_addr type action
direction     : ("in"  |  "out")
src_addr      :  "src="   (ipaddr  |  STAR)
dest_addr    :  "dest=" (ipaddr  |  STAR)
src_port      :  "src_port="  ( (CMPR)? INT  |  STAR)
dest_port    :  "src_port="  ( (CMPR)? INT  |  STAR)
connect       :   INT
type            :  INT
action         :   ("allow"  |  "block")

 loop  :  (net_loop  |  host_loop  |  do_loop)
 net_loop : "for" "network" LPAREN ID ("to" ID)? RPAREN
            LBRKT (stmt)* RBRKT
 host_loop : "for" "host" LPAREN ID ("to" ID)? RPAREN
            LBRKT (stmt)* RBRKT
 do_loop : "do" LPAREN ID EQ INT ("to" INT ("by" INT)? )?  RPAREN
            LBRKT (stmt)* RBRKT
```

```
decision  :  (net_if | host_if | var_if)
net_if : “if” LPAREN “network” CMPR  ID  RPAREN
               LBRKT (stmt)* RBRKT (“else” LBRKT (stmt)* RBRKT)?
host_if : “if” LPAREN “host” CMPR  ID  RPAREN
               LBRKT (stmt)* RBRKT (“else” LBRKT (stmt)* RBRKT)?

var_if : “if” LPAREN ID CMPR  INT  RPAREN
               LBRKT (stmt)* RBRKT (“else” LBRKT (stmt)* RBRKT)?

assignment  :  (net_assign | host_assign | incr_add )
net_assign  : ID H_ASSIGN ID (PLUS ID)* SEMI
host_assign  : ID R_ASSIGN ID (PLUS ID)* SEMI
incr_add  : ID INCR_ASSIGN ID SEMI

probe :  (tcp_pkt  | udp_pkt | icmp_pkt )
tcp_pkt  : “snd_tcp” LPAREN “src=”    v_ipaddr POUND v_int
                                      “dest=”  v_ipaddr POUND v_int
                                      “cntl=”  v_int
                            RPAREN SEMI
udp_pkt  : “snd_udp” LPAREN “src=”  v_ipaddr POUND v_int
                                       “dest=” v_ipaddr POUND v_int
                            RPAREN SEMI
icmp_pkt  : “snd_icmp” LPAREN “src=”   v_ipaddr
                                        “dest=” v_ipaddr
                                        “type=” v_int
                            RPAREN SEMI
v_ipaddr  : v_int  DOT  v_int  DOT  v_int  DOT  v_int
v_int  : (INT | ID)

report : “report” “target=” (STAR | “attack” | ID)
                      “protocol=” (STAR | “TCP” | “UDP” | “ICMP”)
                      “direction=” (STAR | “snd” | “rcv” )
            SEMI
```

```
ID:  ALPHA (ALPHA | DIGIT | ‘_’)*
ALPHA  : (‘a’ .. ‘z’ | ‘A’..’Z’)

INT :  ( DIGIT)+
DIGIT   : ‘0’ … ‘9’

DEFINE         :   ‘:=’
EQ              :   ‘=’
H_ASSIGN   :  ‘@=’
R_ASSIGN    :  ‘<-’
VAR_ASSIGN    :  ‘=’
CMPR                  :   (‘<‘ | ‘>’ | ‘==’ | ‘<=’ | ‘>=’  )
INCR_ASSIGN  :  ‘+=’


SEMI           :  ‘  ‘
COMMA       :  ‘\t’
COLON       : ‘\n’
LPAREN     :    ‘(‘
RPAREN     :    ‘)’
LBRKT        :    ‘{‘
RBRKT        :    ‘}’
SLASH        :    ‘/’
POUND       :    ‘#’
STAR           :    ‘*’
DOT             :    ‘.’
```