Chapter 4
Post-processor

4.1 System Overview

Because the number of multi-event rules in Snort has grown, the original Detection Engine cannot handle its role efficiently. In this thesis, a novel architecture is proposed to solve this problem. A SoC-based system is a better solution because it incorporates both programmable software and application-specific logic circuit hardware. Figure 4-1 shows a system overview diagram of a SoC-based post-processor [30, 31].

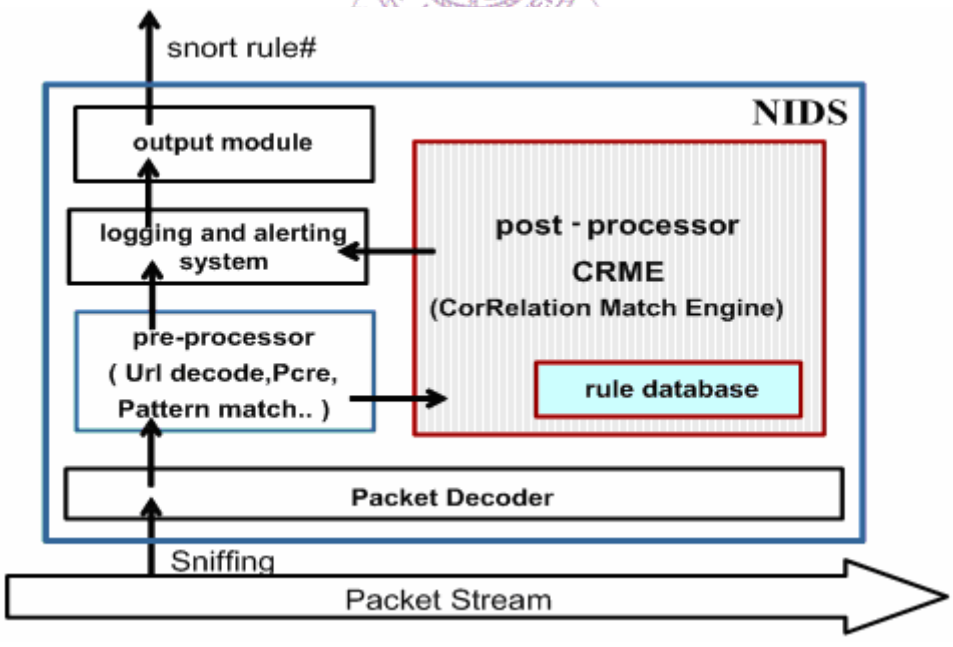


Figure 4-1. The proposed novel NIDS architecture

Two special components are introduced below:

Pre-processor: The pre-processor is responsible for handling content match,

URL filter and regular expression. The pre-processor sends information (EvnetID, EventAddr) when a match condition appears.

Post-processor: Like the original Snort detection engine, the CRME (correlation match engine) is a post-processor which outputs the Snort payload rule ID to the logging and alerting system when it finds an appropriate correlation from pre-processor information.

4.2 Relation between pre / post-processor

Figure 4-2 shows the relationship between the pre-processor and the post-processor. The pre-processor sends information (EventID, EventAddr) when the conditions are matched to the input buffer of the SoC system. The input buffer consists of FIFO (first in first out) memory, which records all events that are triggered by the same packet. The post-processor reads data from the input buffer to execute correlation matching. The rule ID is sent out if the post-processor detects any matches.

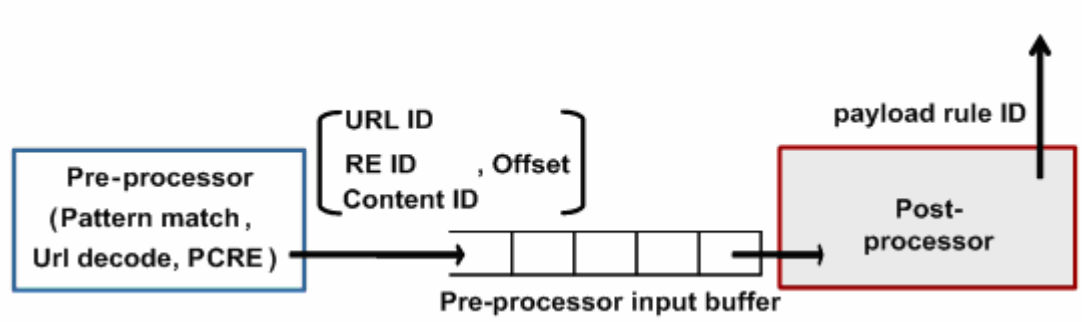


Figure 4-2. The relationship between pre-processor and post-processor

4.3 Post-processor overview

Incorporating the above features, the post-processor system diagram is shown in Figure 4-3 [31], where:

Micro-Processor: The micro-processor is the control center of the post-processor, and is used to maintain the data path and

the control path on the SoC system. The main functions are as follows:

1. Read (EventID, EventAddr) from input buffer
2. Filter out unnecessary and write necessary data to the specific EventTable.
3. Receive information from the UART interface to control the system.
4. Read RuleID from the RuleResultBuffer.

EventTable: Stores several memory device records (EventID, EventOffset). The micro-processor will write these to each EventTable systematically.

RuleTable: Records Snort rules.

Match Engine: A user-defined logic circuit. In this thesis, we will design an effective algorithm and implement the algorithm on such a logic circuit.

RuleResultBuffer: Memory device that records the RuleID which is detected by the ME.

UART: Universal Asynchronous Receiver/Transmitter. This interface in used to control the SoC system.

Other IOs: Flash memory, LCD lights and buttons.

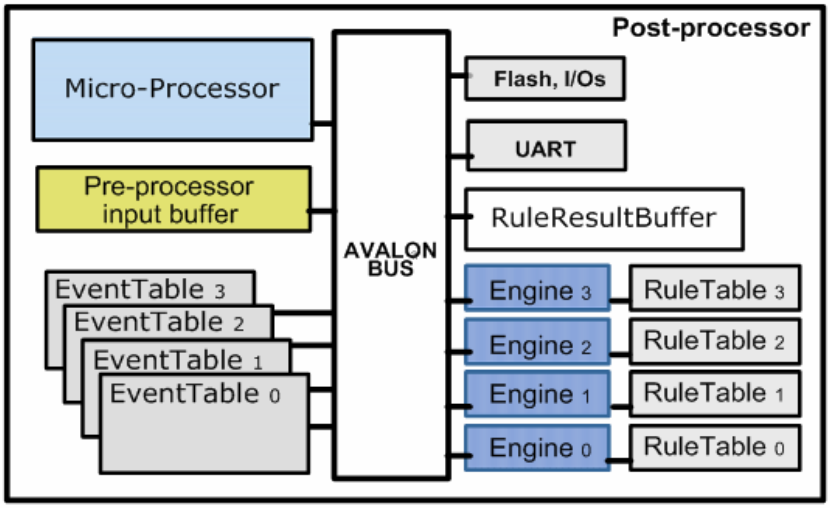


Figure 4-3. The architecture of SoC-based post-processor

4.4 EGF algorithm (event group filter algorithm)

Based on the Snort rule characteristic of interdependence, events following the first event, and in the same group as the first event, only need take action when the first event occurs. A simple algorithm, below, is designed to determine if an extended event needs to be processed by the post-processor.

First, a first event list (FEL) array is defined to record the first event when it appears. Table 4-1 presents the format for an FEL entry, where:

fel_gid: GroupID of the first event.
fel_addr: Address of the first event.
fel_count: Number of extended events which are in the same GroupID as events which have already occurred.

Table 4-1. The form of FEL entry

field	fel_gid	fel_addr	fel_count
length	10	16	6

We devise a simple algorithm to filter out unnecessary events quickly and output the values (FEL#, count, offset) which will determine the event table address where the information will be recorded (Table 4-2).

Table 4-2. The input and output form of EGF algorithm

input/output	field	Descript
Input	EventID	EventID is ID number for this event.
	EventAddr	Location of which event appears.
Output (input is first)	FEL#	Serial number of FEL entry is inserted by the first event.
	count	All zero
	offset	All zero
Output (input is extended)	FEL#	Serial number of FEL entry belongs to extend event.
	count	Amount of extend event already appeared.
	offset	The difference between EventAddr of input and the address of the entry which input event belong to.

The “offset” value returned is the relative address of the first event. Because correlation matching detects the relationship between two events, the relative distance between the two events is more important than their individual absolute address in the payload. The flowchart of the EGF algorithm is shown in Figure 4-4.

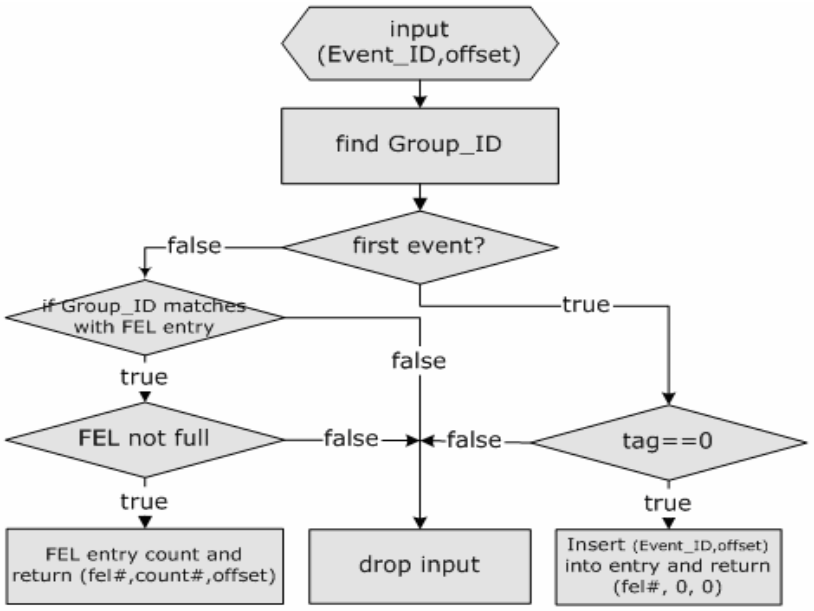


Figure 4-4. The flowchart of EGF algorithm

Table 4-3 (a) to Table 4-3 (c) is a step by step example of the EFG algorithm for the six (EventID, EventAddr) pairs.

Step0:

Initial FEL.

Step1: input (0x 9006, 0x0006)

0x 9006 is first event → insert to FEL#0, return (0x0, 0x0, 0x0)

Table 4-3(a). The content of FEL (step 1)

FEL (first event list)			
Field	fel_gid	fel_addr	fel_count
0	0x6	0x6	0x0

Step2: input (0x 1406, 0x000c)

0x 1406 is extended event and Group ID match FEL#0 → offset=0xc-0x6=0x6;

FEL#0 count one; return (0x0, 0x1, 0x0006)

Table 4-3(b). The content of FEL (step 2)

FEL (first event list)			
Field	fel_gid	fel_addr	fel_count
0	0x6	0x6	0x1

Step3: input (0x 2707, 0x 0013)

0x 2707 is extend event and group ID not match any FEL → drop

Step4: input (0x A007, 0x 0032)

0x A007 is first event → insert to FEL#1, return (0x1, 0x0, 0x0)

Step5: input (0x C008, 0x 0036)

0x C008 is first event; tag =1 and Event_Addr > 32 → drop

Step6: input (0x 2707, 0x 0040)

0x 2707 is extended event and Group ID match FEL#1 →

offset =0x0040-0x0032=0x0008; count FEL#1, return (0x1, 0x1, 0x8)

Table 4-3(c). The content of FEL (step 6)

FEL (first event list)			
field	fel_gid	fel_addr	fel_count
0	0x6	0x6	0x1
1	0x7	0x0032	0x1

4.5 Event Table:

Definition 3:

Let EventBlock [*i*] be the set of all events when GroupID is *i.*

The relationship among input buffer, EGF and micro-processor is depicted in Figure 4-5. After the EGF operation is executed, the (EventID, EventAddr) pair is in the input buffer, and the EGF sends (fel#, count, offset) to the micro-processor to access the corresponding (EventID, Offset) EventTable entry.

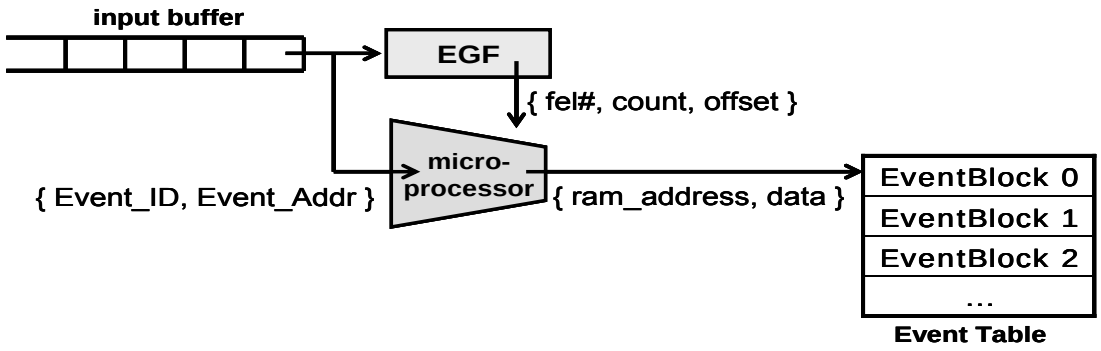


Figure 4-5. Relationship between input buffer, EGF and micro-processor

According to the Snort rule character-group, it is unnecessary to match contents that belong to different groups. The search domain of the original Snort Detection Dngine contains all the events involved in the rule set. After dividing using the first event, each EventBlock only needs to search the RuleGroup which is in the same group (Figure 4-6). This method greatly reduces the amount of effort needed to match and search all the rules in the Rule Group.

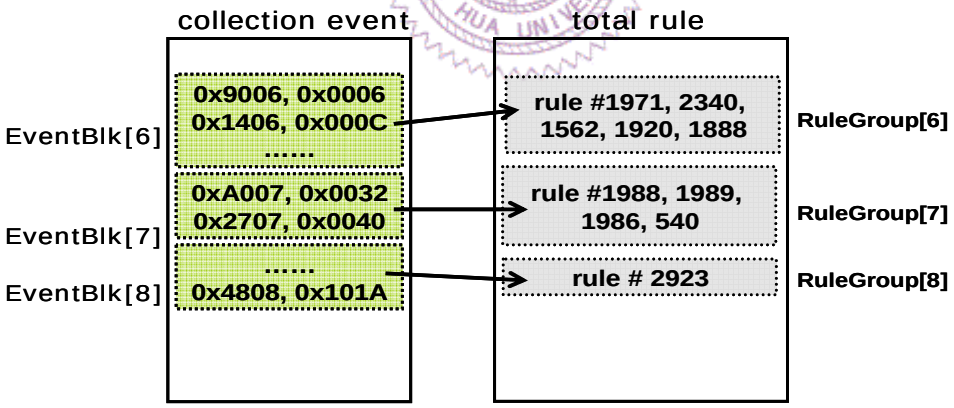


Figure 4-6. Group-based post-processor search domain

Collection:

In order to allocate the group event to the continuous space, the EventTable is partitioned into several blocks. Each FEL entry maps directly to each block. This is depicted in Figure 4-7.

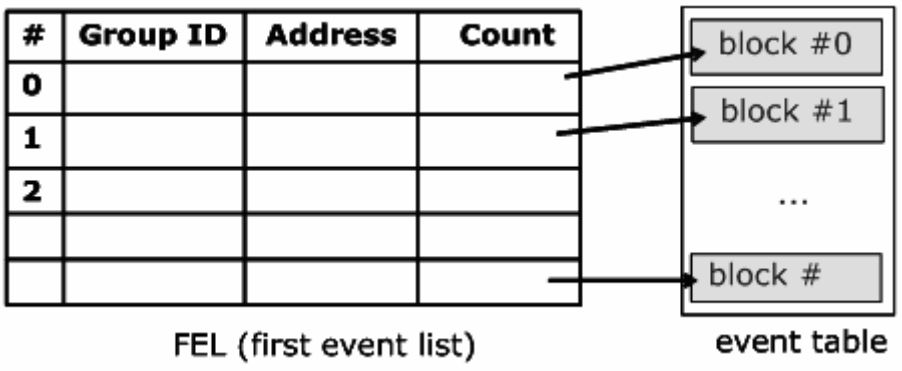


Figure 4-7. The relationship between FEL entry and EventBlock

4.6 Rule Table:

Each rule in the same RuleGroup can have a decision tree to correlate to each event. Like the AC algorithm, the subsequent state depends on the present state and the specific input. Each state has two types: *accept state* and *non-accept state*. Accept state means a rule has been matched.

To construct a unique decision tree, we regulate the longer entries of the sub-tree on the left side. Figure 4-8 is an example based on the rule set shown in Table 4-4.

Table 4-4. The example rule set

Rule ID	Event 1	Event 2	Event 3	Event 4
Rule 1	Pa	Pb; offset:12	Pc	Pd
Rule 2	Pa	Pe; depth:32	Pf	
Rule 3	Pa	Pg; offset:8; depth:32		

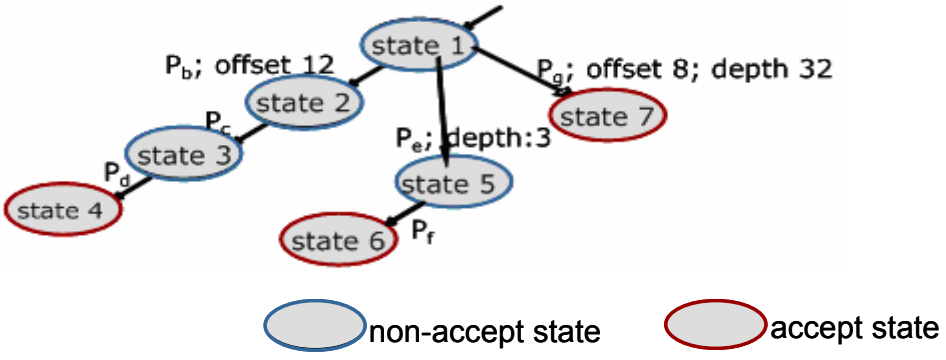


Figure 4-8. The decision tree of the rule set in Table 4-4

Definition 4:

For each state, parameters are defined as follows:

1. StatePrefix:	A subset of the input traverses this state.
2. NextRuleState:	The accept state on the right-side of the present state.
3. NextRulePrefix:	The StatePrefix of the NextRuleState.
4. CommonPrefix:	(StatePrefix) ∩ (NextRulePrefix).
5. StatePrefixSize:	The size of the StatePrefix.
6. CommonPrefixSize:	The size of the CommonPrefix
7. pop:	StatePrefixSize - CommonPrefixSize.
8. level:	All conditions for each state have the same level value.

In order to thoroughly check all rules with non-continuous input, the decision tree must add an extra degree “drop”. This means dropping the present input and traversing the present state again. For the same reason, an internal stack is also needed to record the state prefix. When all inputs are traversed, popping several prefixes allows backtracking to the next probable situation. The popping value is determined by subtracting the CommonPrefixSize from the StatePrefixSize. A value less than or equal to zero means that traversing the next probable situation is not required, and the pop value is set to null.

For this reason, if we want to pop several prefixes we need extra memory (FIFO) to store the StatePrefix. This storage is called “TraverseStack” and it records the StatePrefix and state that have been traversed. Results calculated from the rule set in Table 4-4 are shown in Table 4-5 and Figure 4-9.

Table 4-5. Pop value of each state (CRM)

State	StatePrefix	NextRuleState	NextRulePrefix	CommonPrefix	pop
1	Pa	6	Pa, Pe, Pf	nil	nil
2	Pa, Pb	6	Pa, Pe, Pf	Pa	1
3	Pa, Pb, Pe	6	Pa, Pe, Pf	Pa	2
4	Pa, Pb, Pc, Pd	6	Pa, Pe, Pf	Pa	3
5	Pa, Pe	7	Pa, Pg	Pa	1
6	Pa, Pe, Pf	7	Pa, Pg	Pa	2
7	Pa, Pg	nil	nil	nil	nil

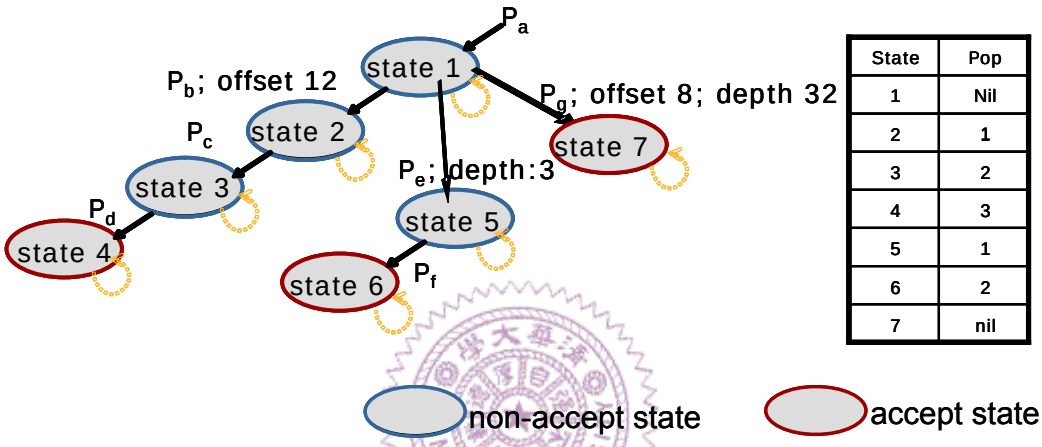


Figure 4-9. The correlation match tree of the rule set in Table 4-4 (CRM); where (Pb, offset: 12), (Pe, depth:32) and (Pg, offset:8; depth:32) are all at the same level.

This “correlation match tree” comprehensively represents the event relationship.

Construct the binary CRM (BCRM) tree:

For efficient hardware design, it is preferable to have a binary CRM search tree. This can be accomplished by converting the correlation tree to a binary search tree. The conversion steps are as follows:

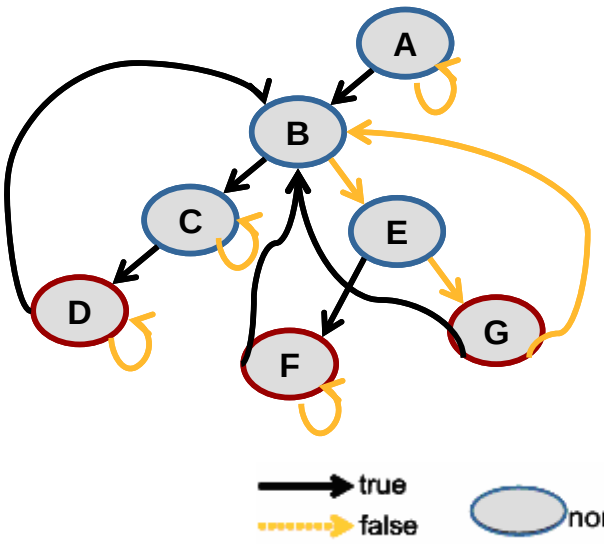
1. Each state of the binary search tree contains one event which consists of an EventID and a relation.
2. For each state, the left son represents the first event in correlation tree, and right son represents the other cases in correlation tree.
3. Each state which is at the same level of the correlation tree contributes to

a circle link list on the binary correlation tree.

This search algorithm is similar to the search correlation tree algorithm. If the input event matches the binary correlation match tree, we traverse to the left son and take it as the next state, otherwise we drop or go to the right son. When all inputs are traversed, the next rule state is traversed by popping the value from the stack. Each item from the rule set in Table 4-4 is rearranged in Table 4-6 and Figure 4-10.

Table 4-6. Pop value of each state (BCRM)

State	state prefix	next rule state	next rule prefix	common prefix	pop
A	nil	nil	nil	nil	nil
B	Pa	nil	nil	nil	nil
C	Pa, Pb	F	Pa, Pe	Pa	1
D	Pa, Pb, Pc	F	Pa, Pe	Pa	2
E	Pa	nil	nil	nil	nil
F	Pa, Pe	G	Pa	Pa	1
G	Pa	Nil	nil	nil	nil



	event	drop	pop
A	P_a	0	nil
B	P_a;offset:12	0	nil
C	P_c	1	1
D	P_d	1	2
E	P_e;depth:32	0	nil
F	P_f	1	1
G	P_g;offset:8; depth:32	1	nil

Figure 4-10. The binary correlation search tree of rule set Table 4-6 (BCRM)

Although Snort rule relations are complicated, three parameters, ID, start, and range, are extracted by analysis to cover event order and all of the content keywords

(*depth*, *offset*, *within*, *distance*). These fields are introduced below:

name	Description
ID	Present this event's ID.
start	Present this event must appear how many bytes later after the previous event.
range	Present event must appear after "start" in how many bytes. If range is 0, means the distance between present event and previous event is just the "start" value. If range is 255, means present event and previous event do not have to consider the location.

ex:

alert tcp $HOME_NET 139 -> $EXTERNAL_NET any (

msg: "NETBIOS SMB repeated logon failure";…

content: "|ff|SMB"; offset:4; depth:4; content:"|73|"; distance:0; within:1;

content:"|6d0000c0|"; distance:0; within:4;…sid:2923; rev:1;

)

Singular form:

	header	Payload					
Length		4	4	0~1	1	0~4	4
Data	tcp	****	|ff|SMB	*	|73|	****	|6D 00 00 C0|

We assume that,

P_h : |ff|SMB; P_i: |73|; P_j: |6d 00 00 C0|

P_h must occur at the 8th byte of the payload.

P_i must occur after Ph and the distance between them must be 0 or 1 byte.

P_j must occur after Pi the distance between them must be 4 to 8 bytes.

Convert RuleID 2923 to binary correlation tree as follows.

Event1: (P_h, 8, 0); Event2: (P_i, 0, 1); Event3: (P_j, 4, 4)

Table 4-7 is constructed using Table 4-4. *Left* field is the left state serial number, *right* field is the right state serial number, *drop* field is the drop value of the state, *pop* field is the pop value of the state and *RuleID* is the rule ID of the state.

Table 4-7. The BCM table from example Table 4-4

Event ID	Start	depth	true	False	drop	pop	rule ID
Pa	0	255	B	A	0	nil	0
Pb	12	0	C	E	0	nil	0
Pc	0	255	D	C	1	1	0
Pd	0	255	B	D	1	2	Rule 1
Pe	0	32	F	G	0	nil	0
Pf	0	255	B	F	1	1	Rule 2
Pg	8	32	B	B	1	nil	Rule 3

4.7 Binary Correlation Match Algorithm:

To implement a binary correlation match (BCM) operation, three pointers are needed: *prt*, *pet* and *pts*. *Prt* points to input into the EventTable; *prt* points to the present binary correlation tree state and *pts* points to the top of the TraverseStack. Figure 4-11 represents the relationship among the EventTable, RuleTable, TraverseStack and CRME (correlation match engine). They communicate with each other via the shared bus.

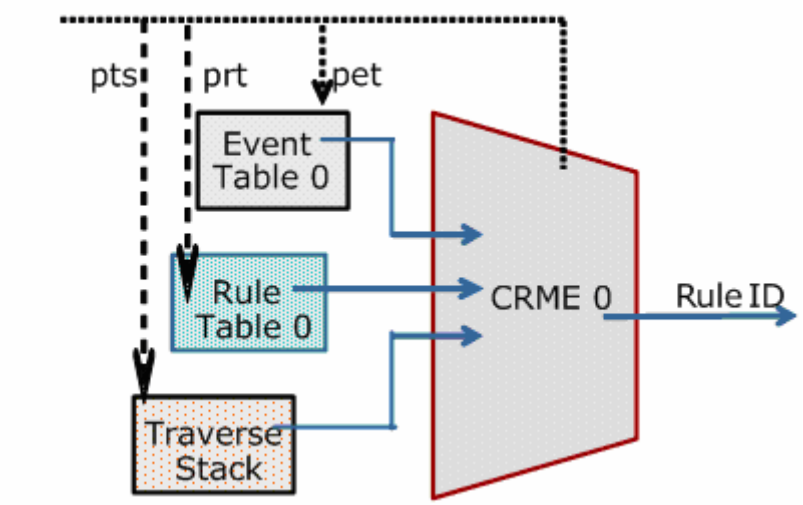


Figure 4-11. The relationship between CRME and memory device

EventTable, RuleTable and TraverseStack input data to the binary correlation match operation. Furthermore, CRME employs *prt*, *pet* and *pts* to control which data

will be used in the next state. Figure 4-12 is a flowchart of the binary correlation match algorithm.

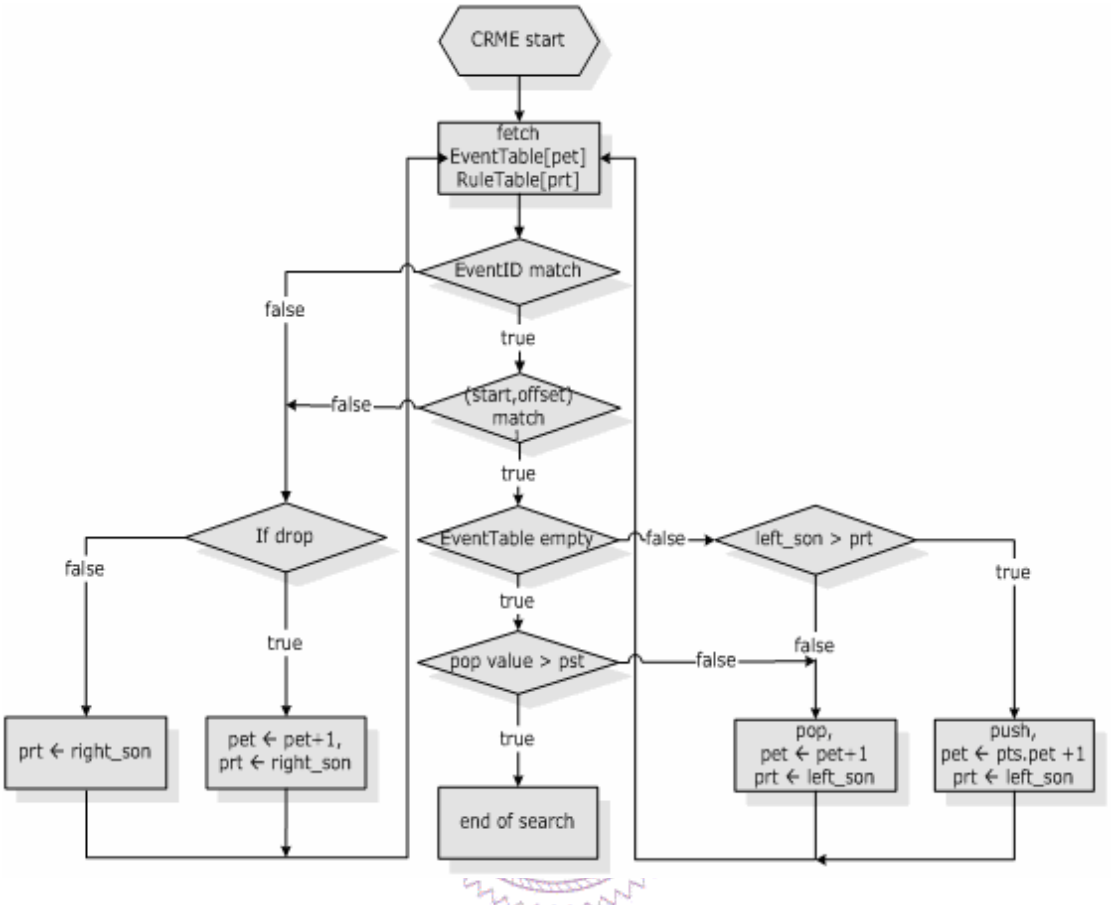


Figure 4-12. The flowchart of Binary CRM algorithm

Table 4-8 illustrates an example of a binary correlation match algorithm. The EventTable is taken from Table 4-9(a),(b).

Table 4-8. The content of EventTable

#	EventID	Offset
0	P_a	0x 4
1	P_b	0x 8
2	P_b	0x 10
3	P_c	0x 20
4	P_g	0x 30
5	P_e	0x 34
6	P_f	0x 3c
7	P_d	0x 48
8	nil	nil

Table 4-9(a). Binary CRM step-by-step operation

#	present state	input	next state	detection	action	prefix
1	A	(P_a, 0)	B	match	*prt* ←left son *pet* ← pet+1 push (Pa, 0)	P_a,0
2	B	(P_b, 8)	E	address-0<start → mismatch	*prt* ← right son	P_a, 0
3	E	(P_b, 8)	G	mismatch	*prt* ← right son	P_a, 0
4	G	(P_b, 8)	B	mismatch, drop=1	*prt* ← right son *pet* ←*pet*+1	P_a, 0
5	B	(P_b, 16)	C	match	*pet* ←*pet*+1 *prt*← left son	P_a, 0 P_b, 16
6	C	(P_c, 32)	D	match	*pet*←*pet*+1 *prt*← left son	P_a, 0 P_b, 16 P_c, 32
7	D	(P_g, 48)	D	mismatch drop	*prt*← right son *pet* ←*pet*+1	P_a, 0 P_b, 16 P_c, 32
8	D	(P_e, 52)	D	mismatch drop	*prt*← right son *pet* ←*pet*+1	P_a, 0 P_b, 16 P_c, 32
9	D	(P_f, 60)	D	mismatch drop	*prt*← right son *pet* ←*pet*+1	P_a, 0 P_b, 16 P_c, 32
10	D	(P_d, 72)	B	match rule	pop two entries *prt* ← go *pet* ← last pop *pet* return Rule1	P_a, 0
11	B	(P_d,32)	E	mismatch	*prt*← right son	P_a, 0
12	E	(P_c, 32)	G	mismatch	*prt*← right son	P_a, 0
13	G	(P_c, 32)	B	mismatch, drop	*prt*← right son *pet*← *pet*+1	P_a, 0
14	B	(P_g, 48)	E	mismatch	*prt* ← right son	P_a, 0
15	E	(P_g, 48)	G	mismatch	*prt* ← right son	P_a, 0
16	G	(P_g, 48)	B	(0x48-0)>(0x32+ 0x8), → mismatch, drop	*prt*← right son *pet*← *pet*+1	P_a, 0

Table 4-9 (b). Binary CRM step-by-step operation

17	B	(P_e, 52)	B	mismatch, drop	*prt*← right son *pet* ←*pet*+1	P_a ,0
18	E	(P_e, 52)	G	0x52 > 0x32 → mismatch	*prt*← right son	P_a, 0
19	G	(P_e, 52)	B	mismatch, drop	*prt*← right son *pet*← *pet*+1	P_a, 0
20	B	(P_f, 60)	E	mismatch	*prt*← right son	P_a, 0
21	E	(P_f, 60)	G	mismatch	*prt*← right son	P_a, 0
22	G	(P_f, 60)	B	mismatch, drop	*prt* ← right son *pet*←*pet*+1	P_a, 0
23	B	(P_d, 72)	E	mismatch	*prt*←failure	P_a, 0
24	E	(P_d, 72)	G	mismatch	*prt*←failure	P_a, 0
25	G	(P_d, 72)		mismatch, end of input, pop==7 → end of search	end of search	

In this example, all possible rules are perfectly checked in only 25 operations.

Because each RuleGroup is independent and the EventTable also stands alone, binary correlation match operations are partitioned by GroupID. Figure 4-13 is an example - the system was partitioned into four independent sub-systems. [23]

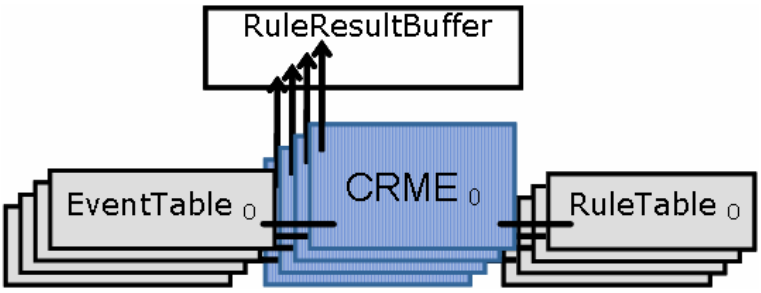


Figure 4-13. The relationship between each BCRM engine