CHAPTER-4

SYSTEM OPTIMIZATION USING SMART SNORT

Snort, an open source intrusion detection system has been used in this work. From the previous chapter it has been concluded that there are still some areas where this tool can be optimized. In this chapter we will try to outline the ways on which efforts have been to bridge the gaps and try to make snort as Smart Snort. It has been tried to customize the snort to work smartly for system optimization.

4.1 Smart Snort

Snort was made to work smartly, when modified with:

4.1.1 Working with Rules

Following considerations have been kept in mind while working with Snort rules to maximize efficiency and speed.

1 Using TCP Flag Tests to speed up Content Rules: As the content rules are tested at last we can take advantage of this fact by using other faster rule options that can detect whether or not the content needs to be checked at all. For instance, most of the time when data is sent from client to server after a TCP session is established, the PSH and ACK TCP flags are set on the packet containing the data. This fact can be taken advantage of by rules that need to test payload content coming from the client to the server with a simple TCP flag test that is far less computationally expensive than the pattern match algorithm. The basic idea is that if the PSH and ACK flags aren't set, there's no need to test the packet payload for the given rule. If the flags are set, the additional computing power required to perform the test is negligible [91].

[93]
2 Content Rules are Case Sensitive: As content rules are case sensitive and that many programs typically use uppercase letters to indicate commands. FTP is a good example of this.

3 Changing the order of the rule: The last rule test that is done (when necessary) is always the content rule option. We can change this order to check the required option first.

4 Snort explains well-known and common vulnerability exploitation attempts, violations of security policy and conditions under which a network packet(s) might be anomalous.

5 Network traffic is checked by the IDS, based on a characteristic database of invading programs. Snort can be adjusted by a rule to give an alert/message or it can take proper action whenever an access in a defined protocol from/to a specific port and from/to specific destination with a content containing a specific string happens. Snort rule has two parts: header and content of data packet. In the rule header, IP address, source and destination port numbers and protocol type, of invading packet is located. Content part contains a string pattern in ASCII, Hex format or combination of two formats. In Snort rule, Hex part is embraced between two ‘|’ signs. For example let us study a rule alert tcp any any ->192.168.1.0/32 111(content:"idc|00 01 86 a5|"; msg: "mountd access"); This rule is due to an invading program that is within a TCP protocol and its source has unknown IP address and port number. The IP address of destination is 192.168.1.0 and its port number is 111. Many of packets in the network have these attributes. Therefore to identify packets of an invader program, data packet content should contain pattern —idc|00 01 86 a5|. This pattern includes characters c, d and i and bytes 00 01 86 a5
6 Work was done with some rules modifications related to protocols like tcp, udp, icmp, ipv4. Wide difference was observed in the alerts generated with standard rule set and modified rule set. Following rules were applied in the present work for pattern matching of the network traffic.

#alert tcp $EXTERNAL_NET 27374 - > $HOME_NET any (msg:"BACKDOOR subseven 22"; flags: A+; content: "|0d0a5b52504c5d30320d0a|"; reference:arachnids,485; reference:url,www.hackfix.org/subseven/; sid:103; classtype:misc-activity; rev:4;)

#alert tcp $EXTERNAL_NET any - > $HOME_NET 21 (msg:"INFO FTP "MKD / " possible warez site"; flags: A+; content:"MKD / "; nocase; depth: 6; classtype:misc-activity; sid:554; rev:3;) caught anonymous ftp server

#alert tcp $HOME_NET any - > $EXTERNAL_NET $HTTP_PORTS (msg:"ET MALWARE Content-loader.com (ownusa.info) Spyware Install"; flow: to_server,established; content:"/fdial2.php?o="; nocase; http_uri; reference:url,doc.emergingthreats.net/bin/view/Main/2003076; classtype:trojan-activity; sid:2003076; rev:4;)

#alert tcp $HOME_NET any - > $HTTP_SERVERS 80 (msg:"WEB-IIS cmd.exe access"; flags: A+; content:"cmd.exe"; nocase; classtype:web-application-attack; sid:1002; rev:2;) caught Code Red infection

#alert tcp $EXTERNAL_NET any - > $HTTP_SERVERS 80 (msg:"WEB-IIS cmd.exe access"; flags: A+; content:"cmd.exe"; nocase; classtype:web-application-attack; sid:1002; rev:2;) caught Code Red infection

#alert tcp $EXTERNAL_NET any - > $HTTP_SERVERS 80 (msg:"WEB-IIS cmd.exe access"; flags: A+; content:"cmd.exe"; nocase; classtype:web-application-attack; sid:1002; rev:2;) caught Code Red infection
#alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET MALWARE Coolsearch Spyware Install"; flow: to_server,established; content:"coolsearch.biz/united.htm"; nocase; http_header; content:"Host|3a|"; nocase; http_header; reference:url,doc.emergingthreats.net/bin/view/Main/2001479; classtype:trojan-activity; sid:2001479; rev:7;)

#alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET DELETED Corpsespyware.net BlackListed Malicious Domain - google.vc"; flow:to_server,established; content:"Host|3a|"; nocase; http_header; content:"google.vc"; nocase; http_header; reference:url,doc.emergingthreats.net/bin/view/Main/2002765; classtype:trojan-activity; sid:2002765; rev:7;)

#alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET MALWARE Corpsespyware.net Distribution - bos.biz"; flow:to_server,established; content:"Host|3a|"; nocase; http_header; content:"businessopportunityseeker.biz"; nocase; http_header; reference:url,doc.emergingthreats.net/bin/view/Main/2002767; classtype:trojan-activity; sid:2002767; rev:9;)


#alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET MALWARE Couponage Download"; flow: to_server,established; content:".dl_"; nocase; http_uri; content:"couponage.com"; nocase; http_header; reference:url,www3.ca.com/securityadvisor/pest/pest.aspx?id=453090725; reference:url,doc.emergingthreats.net/bin/view/Main/2001453; classtype:policy-violation; sid:2001453; rev:7;)

#alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET
#alert tcp any $HTTP_PORTS -> $HOME_NET any (msg:"ET MALWARE Windows executable sent when remote host claims to send an image"; flow: established, from_server; content:"Content-Type|3a| image"; http_header; file_data; content:"MZ"; within:2; byte_jump:4,58,relative,little; content:"PE|00 00|"; fast_pattern; distance:-64; within:4; reference:url,doc.emergingthreats.net/bin/view/Main/2001683; classtype:trojan-activity; sid:2001683; rev:11;)

#alert tcp any $HTTP_PORTS -> $HOME_NET any (msg:"ET DELETED Windows executable sent when remote host claims to send image, Win32"; flow: established, from_server; content:"Content-Type|3a| image"; file_data; content:"MZ"; within:2; byte_jump:4,58,relative,little; content:"PE|00 00|"; within:4; reference:url,doc.emergingthreats.net/bin/view/Main/2001684; classtype:trojan-activity; sid:2001684; rev:10;)

#alert tcp any !20 -> $HOME_NET !25 (msg:"ET DELETED Possible Windows executable sent when remote host claims to send an image"; flow: established, from_server; content:"Content-Type|3a| image"; content:"|0d 0a|MZ"; within: 12; reference:url,doc.emergingthreats.net/bin/view/Main/2001685; classtype:trojan-activity; sid:2001685; rev:7;)

#alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"ET MALWARE Possible Windows executable sent when remote host claims to send a Text File"; flow: established, from_server; content:"Content-Type|3a| text/plain"; http_header; file_data; content:"MZ"; within:2; byte_jump:4,58,relative,little; content:"PE|00 00|"; fast_pattern; distance:-64; within:4; flowbits:isnotset,ET.Adobe.Site.Download; reference:url,doc.emergingthreats.net/bin/view/Main/2008438; classtype:trojan-activity; sid:2008438; rev:7;)

#alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"ET MALWARE Possible Windows executable sent when remote host claims to send html content"; flow:established,from_server; content:"Content-Type|3a| text/html|0d 0a|"; http_header; file_data; content:"MZ"; within:2; byte_jump:4,58,relative,little; content:"PE|00 00|"; fast_pattern; distance:-64; within:4; reference:url,doc.emergingthreats.net/2009897; classtype:trojan-activity; sid:2009897; rev:6;)

[97]
#alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"COMMUNITY DOS Tcpdump rsvp attack"; ip_proto:46; content:"|00 08 14 01 03 00 00 00|"; reference:cve,2005-1280; reference:cve,2005-1281; reference:bugtraq,13391; classtype:attempted-dos; sid:100000134; rev:1;)
alert udp $EXTERNAL_NET any -> $HOME_NET 1069 (msg:"COMMUNITY DOS Ethereal slimp overflow attempt"; content:"|6C C3 B2 A1 02 00 04 00 00 00 00 00 00 00 00 00 00 FF FF 00 01 00 00 00 56 57 F7|"; reference:cve,2005-3243; reference:url,www.ethereal.com/docs/release-notes/ethereal-0.10.13.html; classtype:attempted-dos; sid:100000175; rev:1;)
alert udp $EXTERNAL_NET !53 <> $HOME_NET !53 (msg:"COMMUNITY DOS Single-Byte UDP Flood"; content:"0"; dsize:1; classtype:attempted-dos; threshold: type threshold, track by_dst, count 200, seconds 60; sid:100000923; rev:1;)

4.1.2 Working with the Automata of Snort

Snort uses the Aho-corasick algorithm for pattern matching. The basic algorithm is given below

**Aho-Corasick Algorithm**

Aho-Corasick is the Multipattern matching algorithm which locates all the occurrence of set of patterns in a text of string. It first creates deterministic finite automata for all the predefined patterns and then by using automaton, it processes a text in a single pass. It consists of constructing a finite state pattern matching automata from the patterns and then using the pattern matching automata to process the text string in a single pass.

**Pre-processing Phase**

Step 1: Automata Construction: Construct finite state automata for the set of predefined patterns (or pattern tree) which are supposed to be found in the text string. The states will be numbered by their names and transitions between the defined states would be represented by the characters existing in the particular
pattern.

Step 2: Failure Function: After constructing automata, failure function of each node is calculated and its corresponding transitions are also required to be mentioned, so the constructed automata would be called as "Automata with failure links. Failure function can be defined as the longest suffix of the string that is also the prefix of some node. The goal of the failure function is to allow the algorithm not to scan any character more than once.

Step 3: Output Function: Lastly in the automata output function for final states has to be calculated for recognizing the pattern string which may be found in the text string. And the resulting automata would be called as “Automata with Output Functions” Output function gives the set of patterns recognized when entering final state.

**Searching Phase**

By using the Aho Corasick Searching Algorithm search the text using the preconstructed Finite State Automata for the set of predefined patterns. The searching phase of Aho corasick is straightforward while scanning the text it walk through automata if any transition found, it get transition, otherwise check the failure function.

---

**Automata Created**

```c
#include <stdio.h>
#include <string.h>
#include <ahocorasick.h>

char* sample_patterns[] = {
    "woodcock",
    "WOODCOCK",
    "trueman",
    "truman",
    "unix",
    "UNIX",
};
#define PATTERN_COUNT (sizeof(sample_patterns))
```

[99]
int main (int argc, char ** argv)
{
    unsigned int i;
    AC_AUTOMATA_t *atm;
    AC_PATTERN_t tmp_pattern;
    AC_TEXT_t tmp_text;
    atm = ac_automata_init ();
    for (i=0; i<PATTERN_COUNT; i++)
    {
        tmp_pattern.astring = sample_patterns[i];
        tmp_pattern.rep.number = i+1; // optional
        tmp_pattern.length = strlen(tmp_pattern.astring);
        ac_automata_add (atm, &tmp_pattern);
    }
    ac_automata_finalize (atm);
    ac_automata_display (atm, 'n');
    printf("Searching: \\
" input_text1); \
    tmp_text.astring = input_text1;
    tmp_text.length = strlen(tmp_text.astring);
    ac_automata_settext (atm, &tmp_text, 0);
    AC_MATCH_t * matchp;
    while ((matchp = ac_automata_findnext(atm)))
    {
        unsigned int j;

        printf("@%2ld: ", matchp->position);

        for (j=0; j < matchp->match_num; j++)
        {
            printf("#%ld (%s), ", matchp->patterns[j].rep.number, matchp->patterns[j].astring);
        }
        printf("\n");
    }
    printf("Searching: \\
" input_text2);
    tmp_text.astring = input_text2;
    tmp_text.length = strlen(tmp_text.astring);
ac_automata_settext (atm, &tmp_text, 0);

while ((matchp = ac_automata_findnext(atm)))
{
    unsigned int j;

    printf (@%2ld: ", matchp->position);

    for (j=0; j < matchp->match_num; j++)
        printf (#%ld (%s), ", matchp->patterns[j].rep.number, matchp->patterns[j].astring);

    printf ("n");
}

printf ("Searching: ", input_text3);

tmp_text.astring = input_text3;
tmp_text.length = strlen(tmp_text.astring);
ac_automata_settext (atm, &tmp_text, 1);

while ((matchp = ac_automata_findnext(atm)))
{
    unsigned int j;
    printf (@ %2ld: ", matchp->position);
    for (j=0; j < matchp->match_num; j++)
        printf (#%ld (%s), ", matchp->patterns[j].rep.number, matchp->patterns[j].astring);
    printf ("n");
}

ac_automata_release (atm);
return 0;
}

The time complexity of the algorithm is of O(f(k)+f(q)) in which f(k) is function of length of the pattern and f(q) is the function of the length of the text.

4.1.3 Inline mode Snort with created Automata

Snort was configured in the Inline mode with constructed automata for required
rules to work smartly based on these assumptions

- By default Snort performs multi-pattern matching using Aho-Corasick algorithm for intrusion detection. The library constructs the automata for the entire rule files generated based on the protocol type and rule tree structure (automata) will be constructed for both the header and the options portion occupying more memory. Vast amount of states and transitions will take more time to do the transition.

- In case of smart Snort the finite automata construction for the header portion is completely avoided and the automata is constructed for the options portion of the rule alone. This is further reduced by constructing the automata only for the content portion of the predefined rule set. Figure 4.1 shows the same.

- Main target was in reducing memory consumption by smart Snort IDS.

- Smart Snort does both detection and prevention tasks. Also in this approach, integration of both the misuse and the anomaly detection happens. For implementing the misuse based detection system in smart Snort the default Snort rules were modified.

- Initial pre-processing is done by grouping the rules based on the protocols and storing them separately in four different files for TCP, UDP, ICMP and IP respectively. In the absence of such pre-processing measures packet will be compared against all the rules irrespective of its protocol. This will increase the number of searches needed inducing increased latency to the packet.

- As a packet is trapped in smart Snort, it extracts the protocol field from the packet’s header. Packets are then redirected to the file that contains the rules pertaining to the protocol to which the packet is associated to.
• Smart Snort reduces the number of searches considerably by confining the searches to the protocol to which the packet is associated to.

• With smart Snort, the throughput is in par with a network where there are no IDS/IPS systems are operational.

• Smart Snort will process all the packets which it receives without much packet loss preserving high throughput, and response time.

![Rule Node](image)

**Figure 4.1  Content option of the rule**

### 4.2 Network Performance Evaluation of Security Protocols

Before discussing the results received in this section let us discuss some parameters on which the performance evaluation of the implemented scheme/mode on which work has been done can be checked for any change in the existing scheme of things. Alerts received, packets captured/minute, packet processing rate/min, execution time are some of these parameters. Further rule modifications in respect of the above mentioned parameters are also one of them.

#### 4.2.1 Snort Alerts Received

[**] [1:21874:4] EXPLOIT-KIT Possible exploit kit post compromise activity - StrReverse [**]

[Classification: Successful User Privilege Gain] [Priority: 1]

01/17-12:04:05.012679 96.43.137.99:80 -> 192.168.1.101:39666
tcp TTL:64 TOS:0x0 IP:54666 IpLen:20 DgmLen:16708 DF
**A**** Seq: 0xE6E08027  Ack: 0xCDF03E5D  Win: 0xB880  TcpLen: 32

[**] [1:24676:3] BROWSER-PLUGINS Novell iPrint ActiveX real parameter overflow attempt [**]
Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:04:08.327663 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:54814 IpLen:20 DgmLen:16708 DF

***A**** Seq: 0xE6E55457  Ack: 0xCDF03E5D  Win: 0xAB80  TcpLen: 32
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2011-4187][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-4321]

[**] [1:20716:7] BROWSER-PLUGINS Yahoo! CD Player ActiveX clsid access [**]
Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:04:46.772710 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56021 IpLen:20 DgmLen:16708 DF

***A**** Seq: 0xE708A807  Ack: 0xCDF03E5D  Win: 0xB500  TcpLen: 32

[**] [1:24676:3] BROWSER-PLUGINS Novell iPrint ActiveX real parameter overflow attempt [**]
Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:04:55.079332 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56289 IpLen:20 DgmLen:16708 DF

***A**** Seq: 0xE71088F7  Ack: 0xCDF03E5D  Win: 0xB500  TcpLen: 32
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2011-4187][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-4321]
[**] [1:27223:2] BROWSER-PLUGINS Oracle document capture Actbar2.ocx ActiveX clsid access attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:04:55.402740 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56300 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE710CA07 Ack: 0xCDF03E5D Win: 0xB500 TcpLen: 32
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-3591]

[**] [1:27173:3] BROWSER-PLUGINS Cisco AnyConnect mobility client activex clsid access attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:05:00.131754 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56421 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE7151C17 Ack: 0xCDF03E5D Win: 0xB880 TcpLen: 32
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2011-2040][Xref =>
http://www.securityfocus.com/bid/48081]

[**] [1:20822:7] BROWSER-IE Microsoft Internet Explorer contenteditable corruption attempt malicious string [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:05:02.458542 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56482 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE7172497 Ack: 0xCDF03E5D Win: 0xB880 TcpLen: 32
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2011-1255]

[Classification: A Network Trojan was Detected] [Priority: 1]
01/17-12:05:07.095391 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56604 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE71B76A7 Ack: 0xCDF03E5D Win: 0xB500 TcpLen: 32

[**] [1:20707:7] BROWSER-PLUGINS Dell IT Assistant ActiveX clsid access attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
01/17-12:05:10.407539 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56697 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE71E8367 Ack: 0xCDF03E5D Win: 0xB880 TcpLen: 32
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2011-3397]

[**] [1:26187:3] BROWSER-PLUGINS McAfee Virtual Technician Security Bypass ActiveX clsid attempt [**]
01/17-12:05:11.075239 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56727 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE71F4697 Ack: 0xCDF03E5D Win: 0xB500 TcpLen: 32

[**] [1:24884:2] MALWARE-OTHER Compromised website response - leads to Exploit Kit [**]
[Classification: Misc activity] [Priority: 3]
01/17-12:05:12.756630 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56780 IpLen:20 DgmLen:16720 DF
***A**** Seq: 0xE720CCF7 Ack: 0xCDF03E5D Win: 0x6D00 TcpLen: 44
[Xref => http://stopmalvertising.com/tag/km0ae9gr6m/]

[**] [1:24883:2] MALWARE-OTHER Compromised website response - leads to Exploit Kit [**]
[Classification: Misc activity] [Priority: 3]
01/17-12:05:13.072273 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56790 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE7210E07 Ack: 0xCDF03E5D Win: 0x5700 TcpLen: 32
[Xref => http://stopmalvertising.com/tag/km0ae9gr6m/]

[**] [1:24884:2] MALWARE-OTHER Compromised website response - leads to Exploit Kit [**]
[Classification: Misc activity] [Priority: 3]
01/17-12:05:13.072273 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56790 IpLen:20 DgmLen:16708 DF
***A**** Seq: 0xE7210E07 Ack: 0xCDF03E5D Win: 0x5700 TcpLen: 32
[Xref => http://stopmalvertising.com/tag/km0ae9gr6m/]

[**] [1:24889:3] MALWARE-OTHER Compromised Website response - leads to Exploit Kit [**]
[Classification: Misc activity] [Priority: 3]
01/17-12:05:17.516632 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:56907 IpLen:20 DgmLen:16708 DF
***A*** Seq: 0xE7249CE7 Ack: 0xCDF03E5D Win: 0xB500 TcpLen: 32

[**] [1:27550:1] MALWARE-OTHER Compromised website response - leads to Exploit Kit [**]
[Classification: A Network Trojan was Detected] [Priority: 1]
01/17-12:05:27.869494 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:57177 IpLen:20 DgmLen:16708 DF
***A*** Seq: 0xE72E0437 Ack: 0xCDF03E5D Win: 0xB880 TcpLen: 32

[**] [1:31254:2] MALWARE-CNC Win.Trojan.HAVEX-RAT inbound connection to infected host [**]
[Classification: A Network Trojan was Detected] [Priority: 1]
01/17-12:05:38.451398 96.43.137.99:80 -> 192.168.1.101:39666
TCP TTL:64 TOS:0x0 ID:57495 IpLen:20 DgmLen:16708 DF
***A*** Seq: 0xE7386FC7 Ack: 0xCDF03E5D Win: 0x5700 TcpLen: 32

Figure 4.2 Screen shot of Snort working in the inline mode with >20000 rules
Figure 4.2 explains the snort in the working mode when >20000 rules have been applied for pattern matching of the incoming data packets. When snort applied these rules with smart snort scenario i.e. with modified no of rules and newly created automata for these rules. Automata are basically the rule tree structure which the pattern matching algorithm will follow. In the inline mode with these no of modified rules and the new automata Snort works as an intrusion prevention system. Generally the IPS mode packets are dropped at a faster rate where as the with Smart snort this drop ratio is very less and there is faster packet receiving and processing/min. Results confirm to this. Various data packets captured protocol wise show the same.

4.2.2 Smart Snort results for Network traffic

**Alerts Generated**: Figure 4.3 gives the difference in the alerts generated by the conventional Snort and the smart Snort which contains more number of modified rules as compared to snort, in tabular and graphical form. More number of alerts has been generated with smart snort which applies modified rules.

<table>
<thead>
<tr>
<th>Day of Week</th>
<th>Std. Snort Rules</th>
<th>Improved Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Day</td>
<td>14</td>
<td>52</td>
</tr>
<tr>
<td>2nd Day</td>
<td>10</td>
<td>83</td>
</tr>
<tr>
<td>3rd Day</td>
<td>11</td>
<td>77</td>
</tr>
<tr>
<td>4th Day</td>
<td>9</td>
<td>67</td>
</tr>
<tr>
<td>5th Day</td>
<td>13</td>
<td>88</td>
</tr>
<tr>
<td>6th Day</td>
<td>19</td>
<td>79</td>
</tr>
</tbody>
</table>

**Fig. 4.3** Alerts generated with standard Snort v/s Smart Snort

Smart snort give increase in threat detection with less no of false positives and more accuracy.
TCP packets/min: Figure 4.4 shows the received TCP packets/min for conventional snort and smart snort.

<table>
<thead>
<tr>
<th>TCP</th>
<th>Snort</th>
<th>Smart Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>49.62</td>
<td>101.24</td>
<td></td>
</tr>
<tr>
<td>6.65</td>
<td>117.89</td>
<td></td>
</tr>
<tr>
<td>3.04</td>
<td>155.78</td>
<td></td>
</tr>
<tr>
<td>1.75</td>
<td>7.7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.4 TCP packets/min received with Snort v/s Smart Snort

From the Figure 4.4 it is clear that TCP packets received per min. are quite much more in case of smart Snort as compared to Snort. So throughput is maintained even with more number of rules in case of smart Snort.

UDP packets/min: Figure 4.5 gives the received UDP packets/min for traditional snort and smart snort

<table>
<thead>
<tr>
<th>UDP Pkts./min</th>
<th>Snort</th>
<th>Smart Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5</td>
<td>26375</td>
<td></td>
</tr>
<tr>
<td>12.91</td>
<td>27647</td>
<td></td>
</tr>
<tr>
<td>14.81</td>
<td>3053.75</td>
<td></td>
</tr>
<tr>
<td>3.42</td>
<td>41961</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5 UDP packets/min received with Snort v/s Smart Snort
UDP packets received per minute as shown in the Figure4.5 are more in case of smart Snort as compared to snort except for one case. The result do prove that the smart Snort works better

**IPv4 Pkts/min:** Figure 4.6 shows the received IPv4 packets/min with conventional snort and smart snort.

<table>
<thead>
<tr>
<th>IPv4 Pkts/min</th>
<th>Snort</th>
<th>Smart Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>233.66</td>
<td>26439</td>
<td></td>
</tr>
<tr>
<td>139.71</td>
<td>27668</td>
<td></td>
</tr>
<tr>
<td>179.16</td>
<td>3066</td>
<td></td>
</tr>
<tr>
<td>20.06</td>
<td>41977.22</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.6 IPv4 packets/min received with Snort v/s Smart Snort**

Packets received per minute in case of smart snort are quite high as compared to snort. So this shows that faster packet processing is there in case of smart snort even with more no of rules applied. As the automata of the snort performs matching of patterns against all options of the rule where as in case of smart snort it does it selectively, say only content part of the option portion of the rule. Moreover pre processing rules also it is done only protocol wise.

**IPv6 packets/min:** Similar is the case with traffic received for IPv6 protocol (as shown in the Figure 4.7, in case of smart snort where it is quite high in comparison to snort. That means all the packets are being entertained and drop ratio is just negligible. Smart Snort will process all the packets which it receives without much packet loss preserving high throughput. Smart snort performs pattern matching at a faster rate as pre processing of rules is done. This help in reducing the number of
searches as it is done based on the protocol

<table>
<thead>
<tr>
<th>IPV6 Pkt/min</th>
<th>Snort</th>
<th>Smart Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.07</td>
<td>25803.34</td>
<td></td>
</tr>
<tr>
<td>0.125</td>
<td>23301.92</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>724.12</td>
<td></td>
</tr>
<tr>
<td>0.12</td>
<td>21320.03</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.7 IPv6 packets/min received with Snort v/s Smart Snort

ICMPv6 packets/min: Figure 4.8 shows the variation of ICMPv6 packets/min for snort and smart snort.

<table>
<thead>
<tr>
<th>ICMP6 Pkts./Min</th>
<th>Snort</th>
<th>Smart Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06</td>
<td>140.99</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>33.28</td>
<td></td>
</tr>
<tr>
<td>0.13</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>0.11</td>
<td>33.58</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.8 ICMPv6 packets/min received with Snort v/s Smart Snort
It depicts the ICMP packets received/min. Same pattern is observed in case of the ICMP packets received where packets received during the runtime of snort are negligible as compared to smart snort. Patterns confirm that there is faster data acquisition and processing. It happens due to the improved automata of the smart Snort, where the finite automata construction is only done for the option portion of the rule only and automata construction for the header portion is not done. Moreover it can be further limiting it to only content part of the rule. So this reduces unnecessary searches, states and transitions and more traffic can be handled by the smart Snort in comparison to snort.

4.2.3 Smart Snort Results for Memory Consumption & Processor Time

Figure 4.9 shows the processor time variation for traditional snort and smart snort.

![Figure 4.9](image)

**Figure 4.9  Percentage processor time variation for Snort v/s Smart Snort**
From the results it is observed that smart snort also out performs the snort as far as processor time with the new automata number of searches have reduced to large extent so the time taken to do those unwanted searches is saved this results in time saving. Time taken in case of smart snort is lesser as compared to the snort for the same no of packets received.

**Memory Consumption:** Figure 4.10 shows the percentage memory consumption with snort and smart snort.

| Comparison between Snort and Smart Snort in terms of Memory Consumption. |
|------------------|------------------|
| Smart Snort | Snort |
| 17.8 | 25.8 |
| 21.4 | 30.54 |
| 24 | 35 |
| 21.1 | 29.1 |
| 24 | 35 |
| 23.3 | 30.3 |
| 22.8 | 27.4 |
| 22.8 | 27.4 |
| 19.2 | 29.2 |
| 28.4 | 31.4 |
| 26.4 | 35.4 |

**Figure 4.10** %age memory consumed with Snort v/s Smart Snort

It presents the memory consumption variation for smart and smart Snort, which is lesser. This due to the special automata created for the smart snort. As automata is constructed only for content portion of the rule’s option portion because we know that network intrusion attempt can be checked by checking the content part of the rule and all other fields of the rule like header part is avoided as data processing is only protocol wise.
4.2.4 Smart Snort results for Execution time

Figure 4.11 shows the runtime for packet processing for inside tcpdump dataset from DARPA.

<table>
<thead>
<tr>
<th>Total packets</th>
<th>snort</th>
<th>smart snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011149</td>
<td>5.741</td>
<td>4.386</td>
</tr>
<tr>
<td>1563069</td>
<td>11.293</td>
<td>10.912</td>
</tr>
<tr>
<td>1362422</td>
<td>9.153</td>
<td>8.374</td>
</tr>
<tr>
<td>1753377</td>
<td>11.456</td>
<td>10.753</td>
</tr>
<tr>
<td>1362422</td>
<td>8.936</td>
<td>7.591</td>
</tr>
</tbody>
</table>

**Figure 4.11  Packet Processing time (sec) for inside tcpdump dataset**

Here we analyze that the smart snort depicts lesser execution time than snort for the DARPA dataset taken from DARPA Intrusion Detection system Evaluation Datasets. The data was taken for inside tcpdump format and given to snort and smart snort.

Figure 4.12 shows the runtime for packet processing in seconds for outside tcpdump dataset.

<table>
<thead>
<tr>
<th>Total Packets</th>
<th>Snort</th>
<th>smart snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1337777</td>
<td>10.47271</td>
<td>9.4581</td>
</tr>
<tr>
<td>1535894</td>
<td>10.8945</td>
<td>9.3961</td>
</tr>
<tr>
<td>888139</td>
<td>4.791</td>
<td>3.9513</td>
</tr>
<tr>
<td>1412645</td>
<td>9.23988</td>
<td>8.6329</td>
</tr>
<tr>
<td>1252412</td>
<td>8.21449</td>
<td>7.75313</td>
</tr>
</tbody>
</table>

**Figure 4.12  Packet Processing time (sec) for outside tcpdump dataset**
It is found that Smart Snort shows less execution time for packet processing for outside tcpdump dataset from DARPA Intrusion detection system Evaluation dataset.

4.3 Conclusion

Conventionally when configured in inline mode, normally there is too much packets dropping where as this is not n case of smart Snort. This is also visible from the number of packets received per/minute for various protocols. Because initial pre-processing is done by grouping the rules based on the protocols and storing them separately in four different files for TCP, UDP, ICMP and IP respectively. As a packet is trapped by smart Snort, it extracts the protocol field from the header of the packet. These are then redirected to the file that contains the rules associated to the protocol to which the packet is associated to. So this reduces the memory consumption which is visible from the results. By reducing the number of searches restricted due to the improved rule set and automata creation for pattern matching reduces time required to do the searches. This way it can search more packets and reducing the drop ratio and increased throughput. From the results it can be concluded that smart Snort has performed better than the snort on various parameters such as throughput, time taken to execute processing of the received data and memory consumption. Further runtime for a standard data set was noted down for snort and smart Snort, it is less in case of smart Snort confirming that it has faster packet processing.