Journal of Southeast University (English Edition) Vol. 32, No. 1, pp. 21 – 28 Mar. 2016 ISSN 1003—7985

An analysis method of topological relations between Snort rules

Yin Yi[1,2] Wang Yun[1] Takahashi Naohisa[3]

([1]School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)
([2]School of Computer Science and Technology, Nanjing Normal University, Nanjing 210023, China)
([3]Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology, Nagoya 466-8555, Japan)

Abstract: It is difficult to know all the relations between Snort rules. To deal with this problem, the topological relations between Snort rules are classified based on the set theory, and a method for calculating the topological relations between Snort rules is proposed. In the existing methods for analyzing the relations of Snort rules, the relations are usually determined only according to the header information of the Snort rules. Without considering the actions of Snort rules, the proposed method improves upon the existing methods and it can classify and calculate the topological relations between Snort rules according to both headers and options information of Snort rules. In addition, the proposed method is implemented by the functional language Haskell. The experimental results show that the topological relations between Snort rules can be calculated rapidly and effectively. The proposed method also provides an important basis for conflict detection in the succeeding Snort rules.

Key words: intrusion detection system (IDS); Snort rule; functional programming language

DOI: 10.3969/j.issn.1003 – 7985.2016.01.005

The intrusion detection systems(IDS) are very important for network security. The IDS are mainly classified into anomaly-based IDS and signature-based IDS. Most IDS use signature-based detection techniques, such as Snort. Snort will monitor packets on the network and compare them against a pre-defined set of rules, that is, the well-known Snort rules[1]. The effectiveness of security protection provided by Snort mainly depends on the Snort rules. Unfortunately, since it is difficult to know all the relations between Snort rules, managing and configuring Snort rules is a difficult and error prone task for network administrators. Therefore, Snort may experience some problems. For example, when packet P arrives, Snort will compare P with all the rules. If multiple rules match P, all these rules are checked and the most severe alert message is then generated. That is to say, Snort tries comparisons for all of the rules even when packet P never matches some rules and instead match some other rules. The following two rules r_i and r_j try to find the word "HTTP" or "FTP" between the 4th and 7th characters in a payload of the TCP packet. Snort will try to compare a certain packet with r_i and r_j, although a packet cannot match them at the same time.

r_i: alert tcp 192.168.1.0/24 any – > any any (content: "HTTP"; offset: 4; depth: 7; msg: "HTTP matched";)

r_j: alert tcp 192.168.1.0/24 any – > any any (content: "FTP"; offset: 4; depth: 7; msg: "FTP matched";)

Snort may also provide redundant or contradictory alert messages when a set of erroneous and poorly-organized rules is given. For example, in the following two rules, r_t is a redundant rule to r_s, because rules r_s and r_t have the same actions to deal with packets that come from the node with IP address of 192.168.1.117.

r_s: alert tcp 192.168.1.0/24 any – > any any (content: "HTTP"; offset: 4; depth: 7;)

r_t: alert tcp 192.168.1.117 any – > any any (content: "HTTP"; offset: 4; depth: 7;)

The major reason of these problems is that the header and the options of multiple Snort rules overlap, and the coming packets will match these overlapped multiple rules simultaneously. In this case, we can say that there are some relations between these overlapped rules, such as being completely overlapped, partially overlapped, and so on.

If we can know the relations between Snort rules beforehand, it can help administrators to omit the unnecessary comparison of packets with rules and improve the matching speed of Snort. In addition, it also can help administrators to find contradictory or redundant rules, and then decrease conflicts between rules. Therefore, to improve the overall efficiency of Snort, it is very important to analyze the relations between Snort rules.

Research on the analysis of relations of rules has received much attention. For example, some approaches

Received 2015-08-17.

Biography: Yin Yi (1978—), female, doctor, lecturer, yi837@hotmail.com.

Foundation item: The National Natural Science Foundation of China (No. 60973122, 61572256).

Citation: Yin Yi, Wang Yun, Takahashi Naohisa. An analysis method of topological relations between Snort rules[J]. Journal of Southeast University (English Edition), 2016, 32(1): 21 – 28. DOI: 10.3969/j.issn.1003 – 7985.2016.01.005.

focused on the analysis of the firewall rules[2-7]. According to the relations of firewall rules, Al-Shaer et al.[2] identified all kinds of anomalies that could exist in each pair of firewall rules, and presented a set of techniques and algorithms to automatically discover anomalies in a centralized and distributed firewall. Yuan et al[4]. proposed a framework for modeling an individual and distributed firewall, and designed a static analysis algorithm to discover various misconfigurations such as policy violations, inconsistencies and inefficiencies. In our previous works[5-7], we interpreted firewall rules' header fields as range values, and developed a method to analyze the spatial relationships of firewall rules. However, the above methods cannot be used for the analysis of Snort rules' relations, which are different from firewall rules. This is because both the header and options of Snort rules need to be analyzed, while firewall rules only require the analysis of the headers.

Some research focuses on the string matching algorithm of Snort[8-10], thereby improving its detection efficiency. For example, Zhao et al.[9] used a two-step dynamic rule matching algorithm of Snort to improve its detection efficiency. In addition, some research focuses on how to manage the IDS rules[11-13]. For example, Kuang et al.[11] proposed a three-dimensional organization to organize the increasingly massive intrusion detection rules. There was also some research that deals with Snort rules by using hardware[14-16]. For example, Chen et al.[16] proposed a novel two-step pattern decomposition scheme to remove hidden redundancies in the Snort rules database. The above mentioned IDS related works mainly focused on how to examine the string positions of IDS rules, and proposed some methods to control the matching orders of rules and to achieve the goal of high speed string matching. They cannot be used directly for the analysis of relations between Snort rules:

In this paper, in order to decrease the unnecessary comparison of packets with rules, and discover whether there are conflicts between the rules, we propose a method that can calculate the topological relations between Snort rules based on the set theory, which is developed from the spatial relationships calculation method of firewall rules. The major contributions of this paper are that we define topological relations of two Snort rules based on the set theory, propose a method to calculate the topological relations between two rules with string-matching conditions, and use the functional programming language Haskell to implement the proposed methods seriously and concisely. The experimental results show that the topological relations between Snort rules can be calculated rapidly and effectively.

1 Simplified Snort Rules

1.1 Structure of Snort rules

Snort is one popular open-source network IDS that uses a set of rules, i. e. so-called signatures. The following codes illustrate an example of a common Snort rule.

alert udp $HOME_NET any -> any any (msg: "BACKDOOR netthief runtime detection"; content: "|00 00 00 00 00 00 00 82|"; depth: 8; offset: 17; threshold: type limit, track by_src, count 1, seconds 600; metadata: policy security-ips drop; reference: url, www3. ca. com/securityadvisor/virusinfo/virus. aspx? ID = 16078; classtype: trojan-activity; sid: 7760; rev: 2;)

Each Snort rule consists of two logic sections, the rule header and the rule options[1]. The header contains an action and some key fields that are used for comparison with the packet's header. The action determines how to deal with packets when the conditions of header and options are met. The key fields in a rule's header are usually protocol, source IP addresses, destination IP addresses, source port, and destination ports information.

Rule options form the heart of Snort's intrusion detection engine, which contains alert messages and information. According to options, Snort will determine which parts of the packet should be inspected and whether the rule action should be taken. Rule options are divided into four categories: general, payload, non-payload, and post-detection. In this paper, we focus on payload, which are options that all look for data inside the packet payload. The commonly used payloads in Snort rules are content, depth, offset, uricontent, and so on.

1.2 Analysis of Snort rules

A Snort rule consists of header and options. In this paper, a header of a Snort rule is represented as a range-matching condition (represented as RMC), options part is represented as string-matching conditions (represented as SMC) for a packet payload. We simplify Snort rules to a subset of Snort rules which consists of RMC for a packet header and an SMC for a packet payload. For a rule r with an identification number $r.\,\mathrm{sid}$, it is represented as the following triplet:

$$(r.\,\mathrm{sid},\ r.\,\mathrm{RMC},\ r.\,\mathrm{SMC})$$

The $r.\,\mathrm{RMC}$ is a list of range values shown as below:

$$r.\,\mathrm{RMC} = ([a_0, b_0], [a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n])$$

The k-th ($k \in [0, n]$) range value, $[a_k, b_k]$, specifies the lower bound and upper bound values of the k-th key field in a packet header.

In this paper, we use the content option and its two modifiers, depth and offset, as an SMC for a packet payload, for the reason that the content keyword is one of the more important features of Snort. $r.\,\mathrm{SMC}$ is represented as the following three-tuple:

$$r.\,\mathrm{SMC} = (\mathrm{Content}, \mathrm{Offset}, \mathrm{Depth})$$

where Content is a single string, in conjunction with the Depth and Offset values, which is usually used to find a signature in the packet. The offset keyword specifies where to start searching for a pattern within a packet. The depth keyword specifies how far should be searched into a packet for the specified pattern. For example, if the variable $ HOME_NET of the rule shown in Section 1.1 is equal to 192.168.1.0/24, r.sid, r.RMC and r.SMC are represented as follows:

$$r.\text{sid} = 7760$$
$$r.\text{RMC} = ((17,\ 17),\ (192.168.1.0,\ 192.168.1.255),\ (0,\ 65535),\ (0.0.0.0,\ 255.255.255.255),\ (0,\ 65535))$$
$$r.\text{SMC} = (\text{"}\ |00\ 00\ 00\ 00\ 00\ 00\ 00\ 82\ |\ \text{"},\ 8,\ 17)$$

1.3 Topological relations of two rules

We define that R represents a set of rules and P_0 represents the whole packets set. For a packet $p \in P_0$ and a rule $r \in R$, we define the following two functions:

$$\text{match\_RMC}(p,\ r) = \begin{cases} \text{True} & \text{if the header of } p \text{ satisfies } r \\ \text{False} & \text{otherwise} \end{cases}$$
$$\text{match\_SMC}(p,\ r) = \begin{cases} \text{True} & \text{if the payload of } p \text{ satisfies } r \\ \text{False} & \text{otherwise} \end{cases}$$

By the above two functions, we use symbols $P\_\text{RMC}(r)$, $P\_\text{SMC}(r)$ and $P(r)$ to represent a set of packets that only match r.RMC, a set of packets that only match r.SMC, and a set of packets that match both r.RMC and r.SMC, respectively, which can be defined as follows:

$$P\_\text{RMC}(r) = \{p \in P_0 \mid \text{match\_RMC}(p, r) = \text{True}\} \quad (1)$$

$$P\_\text{SMC}(r) = \{p \in P_0 \mid \text{match\_SMC}(p, r) = \text{True}\} \quad (2)$$

$$P(r) = \{p \in P_0 \mid \text{match\_RMC}(p, r) = \text{True and match\_SMC}(p, r) = \text{True}\} = P\_\text{RMC}(r) \cap P\_\text{SMC}(r) \quad (3)$$

The topological relations between rules r_i and r_j are represented as $\text{TR}(r_i,\ r_j)$. Using the inclusion relations of $P(r_i)$ and $P(r_j)$, $\text{TR}(r_i,\ r_j) = \{\text{equal, inside, contain, disjoint, overlap}\}$ is defined as

$$\text{TR}(r_i, r_j) = \begin{cases} \text{equal} & \text{when } P(r_i) = P(r_j) \\ \text{inside} & \text{when } P(r_i) \subset P(r_j) \\ \text{contain} & \text{when } P(r_i) \supset P(r_j) \\ \text{disjoint} & \text{when } P(r_i) \cap P(r_j) = \varnothing \\ \text{overlap} & \text{otherwise} \end{cases} \quad (4)$$

We use the symbols $\text{TR\_RMC}(r_i,\ r_j)$ and $\text{TR\_SMC}(r_i,\ r_j)$ to represent the topological relations between the two rules' RMCs and SMCs. They can be defined in the same way as $\text{TR}(r_i,\ r_j)$ by using the inclusion relations of $P\_\text{RMC}(r_i)$ and $P\_\text{RMC}(r_j)$, the inclusion relations of $P\_\text{SMC}(r_i)$ and $P\_\text{SMC}(r_j)$, respectively. The images of all $\text{TR}(r_i,\ r_j)$s for any two rules r_i and r_j in two dimensions (source port and destination IP address) are shown in Fig. 1.

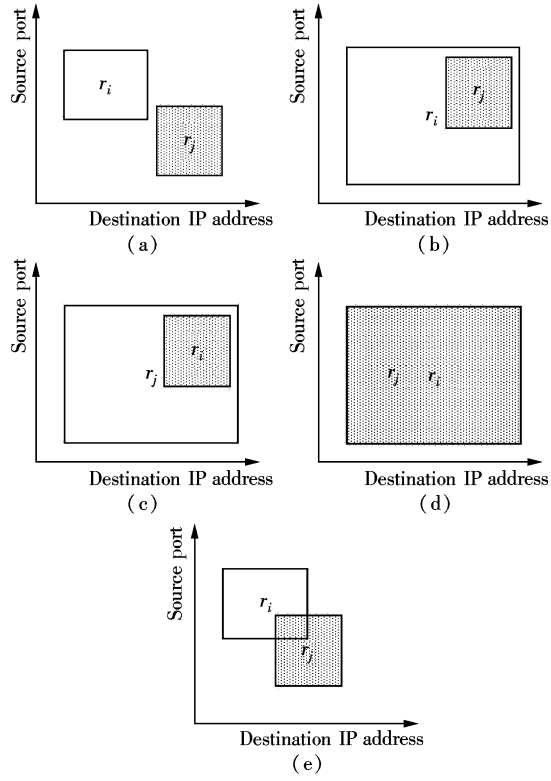

Fig. 1 Images of five topological relations of two rules. (a) $\text{TR}(r_i, r_j)$ = disjoint; (b) $\text{TR}(r_i, r_j)$ = contain; (c) $\text{TR}(r_i, r_j)$ = inside; (d) $\text{TR}(r_i, r_j)$ = equal; (e) $\text{TR}(r_i, r_j)$ = overlap

2 Calculation of Topological Relations

In this section, we will describe how to calculate $\text{TR\_RMC}(r_i,\ r_j)$, $\text{TR\_SMC}(r_i,\ r_j)$, and $\text{TR}(r_i,\ r_j)$.

2.1 Calculation of TR_RMC(r_i, r_j)

The calculation of $\text{TR\_RMC}(r_i,\ r_j)$ is similar to that of spatial relationships in our previous work. The basic idea for calculating the topological relations of RMCs between two rules is divided into two phases. The first phase is the calculation of topological relations of a single key field. Each single key field is represented as a range value. Then, according to the inclusion relations of the set theory, we calculate the topological relations between two rules' RMCs. In the second phase, the topological relations of all single key fields are composed into one topological relation.

2.2 Calculation of TR_SMC(r_i, r_j)

An SMC is a string-matching condition for packet payload, which is represented as a three-tuple (Content, Offset, Depth). The three keywords specify the range of data within which pattern matching should be done. For exam-

ple, when a packet header satisfies the header of the following rule r_0, and if the packet's payload contains the string of "cgi-bin/phf" from the 4-th byte to the next 20 bytes, the rule r_0 will give alert.

r_0: alert tcp any any - > any 80 (content: "cgi-bin/phf"; offset: 4; depth: 20;)

We define TR_SMC(r_i, r_j) to represent the topological relations between two rules' SMCs, which can be determined by the inclusion relations of $P\_SMC(r_i)$ and $P\_SMC(r_j)$. We represent a packet that contains the pattern "ax" from the 1st byte to the next 2 bytes as $P[1-2$, "ax"]. We can find that the packet P only matches r_0. SMC and cannot match r_1. SMC at the same time. In this case, we say that the topological relation between the r_0. SMC and r_1. SMC is disjoint topological relation.

r_0. SMC: ("ax", 1, 2), r_1. SMC: ("ay", 1, 2)

To implement the calculation of TR_SMC(r_i, r_j), we define the following three functions CE_1 to CE_3.

$$CE_1(r_i, r_j) = \begin{cases} \text{True} & \text{if } P\_SMC(r_i) \subseteq P\_SMC(r_j) \\ \text{False} & \text{otherwise} \end{cases}$$

$$CE_2(r_i, r_j) = \begin{cases} \text{True} & \text{if } P\_SMC(r_i) \supseteq P\_SMC(r_j) \\ \text{False} & \text{otherwise} \end{cases}$$

$$CE_3(r_i, r_j) = \begin{cases} \text{True} & \text{if } P\_SMC(r_i) = \{\varnothing\} \text{ or } P\_SMC(r_j) = \{\varnothing\} \\ \text{False} & \text{otherwise} \end{cases} \quad (5)$$

According to Eq. (5), TR_SMC(r_i, r_j)s can be defined as

$$TR\_SMC(r_i, r_j) = \begin{cases} \text{equal} & \text{if } CE_1 = \text{True and } CE_2 = \text{True} \\ \text{inside} & \text{if } CE_1 = \text{True} \\ \text{contain} & \text{if } CE_2 = \text{True} \\ \text{disjoint} & \text{if } CE_3 = \text{True} \\ \text{overlap} & \text{otherwise} \end{cases} \quad (6)$$

That is to say, if we can find the conditions that make three functions CE_1 to CE_3 return True or False, then we can decide TR_SMC(r_i, r_j). To describe the implementation algorithm of CE_1 to CE_3, we first suppose three premises M_1 to M_3.

M_1: SMCs of two rules r_i and r_j are r_i. SMC = (c_i, o_i, d_i) and r_j. SMC = (c_j, o_j, d_j), respectively.

M_2: Variables m, n represent the lengths of two strings c_i and c_j, respectively.

M_3: Variables x, y represent the position ranges of the initial characters of strings c_i and c_j, respectively.

Based on Eq. (6), the implementation algorithm of CE_1 is shown as follows.

Algorithm 1 ((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: True/False.
Initialization int $x = y = 0$, $A = \{\}$, $B = \{\}$;
for each s in [0, $m-1$]
if(exists c_j in n characters from the s-th character of c_i)
Add s to A;
for each x in [o_i, $d_i - m + 1$], y in [o_j, $d_j - n + 1$]
for each s in A
if((not (($y >= o_j$) && ($y <= d_j - n + 1$))) && ($y - x == s$))
Add (x, y) to B;
if($A \neq \{\}$ and $B = \{\}$)
Return True;
else
Return False;

As shown in Algorithm 1, it should satisfy the following conditions D_1 to D_3 when CE_1 returns True.

D_1: $o_i \leqslant x \leqslant d_i - m + 1$ and $o_j \leqslant y \leqslant d_i - n + 1$.

D_2: String c_i contains one or more c_j.

D_3: There are no payloads of packets that match r_i. SMC but do not match r_j. SMC.

For example, if we suppose that r_i. SMC = ("*xyzab*", 1, 7) and r_j. SMC = ("ab", 1, 9), respectively, condition D_1 is equivalent to $1 \leqslant x \leqslant 3$ and $1 \leqslant y \leqslant 8$. Since string c_j = "*ab*" is the substring of string c_i, condition D_2 is also satisfied. The payload of each packet that satisfies r_i. SMC should contain the pattern "*xyzab*", and the pattern should be searched in the range of {12345, 23456, 34567}. The payload of each packet that satisfies r_j. SMC should contain the pattern "ab", and the pattern should be searched in the range of {12, 23, 34, 45, 56, 67, 78, 89}. The search range of characters "ab" in the pattern "*xyzab*" of r_i. SMC should be in {45, 56, 67}, which is the subset of search range of pattern "ab" in r_j. SMC; therefore, the payloads of all the packets match r_i. SMC and r_j. SMC at the same time. That is to say, condition D_3 is satisfied and at this time, we say that the TR_SMC (r_i, r_j) = inside.

Function CE_2 is used to determine whether TR_SMC (r_i, r_j) = contain. The implementation of CE_2 is similar to the implementation of CE_1. We omit the description.

To find the conditions that make function CE_3 return True, we divided them into the following four cases according to the position relations of two strings c_i and c_j.

The first case is that c_i does not overlap c_j at all. Its implementation algorithm is shown as follows:

Algorithm 2 ((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: Set1.
Initialization int $x = y = 0$, Set1 = {};
for each x in [o_i, $d_i - m + 1$], y in [o_j, $d_j - n + 1$]
if(($y >= x + m$) || ($y + n <= x$))

Add (x, y) to Set1;
Return Set1;

The second case is that $m > n$ and c_i includes c_j. Its implementation algorithm is shown as follows:

Algorithm 3 $((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)$
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: Set2.
Initialization int $x = y = 0$, Set2 = { }, A = { };
for each s in $[0, m-1]$
if ((in the n characters from the s-th character in c_i, there exists c_j) && ($s <= m-n$))
Add s to A;
if($A == \{\}$)
Return Set2;
else
{for each x in $[o_i, d_i - m + 1]$, y in $[o_j, d_j - n + 1]$
for each s in A
if ($y == x + s$)
Add (x, y) to Set2;
Return Set2; }

The third case is that the tail of c_i shares the common sub-strings with the forepart of c_j. Its implementation algorithm is shown as follows.

Algorithm 4 $((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)$
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: Set3.
Initialization int $x = y = 0$, Set3 = { }, A = { };
for each s in $[0, m-1]$
if((in the overlapped $m - s$ characters from the s-th character in c_i and the 0-th character in c_j, there are the same substrings) && ($s > m - n$))
Add s to A;
if(A = { })
Return Set3;
else
{for each x in $[o_i, d_i - m + 1]$, y in $[o_j, d_j - n + 1]$
for each s in A
if($y == x + s$)
Add (x, y) to Set3;
Return Set3; }

The final scenario is that the tail of c_j shares the common sub-strings with the forepart of c_i. Its implementation algorithm is shown as follows:

Algorithm 5 $((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)$
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: Set4.
Initialization int $x = y = 0$, Set4 = { }, A = { };
for each s in $[0, m-1]$
if((in the overlapped $s + 1$ characters from the 0-th character in c_i and the $(n - s - 1)$-th character in c_j, there are the same substrings) && ($s < n - 1$))
Add s to A;
if($A == \{\}$)
Return Set4;
else
{for each x in $[o_i, d_i - m + 1]$, y in $[o_j, d_j - n + 1]$
for each s in A
($y + n - 1 == x + s$)
Add (x, y) s to Set4;
Return Set4;
}

Algorithms 2 to 5 describe that when the strings c_i and c_j satisfy the position relations of each case, whether there are (x, y) pairs that make TR_SMC(r_i, r_j) not disjointed. If there are these kinds of pairs, they will be added to Set1 to Set4. For example, when r_i.SMC = ("xyz", 1, 3) and r_j.SMC = ("ab", 5, 6), there is a pair of (x, y) = (4, 5) that makes the conditions of Algorithm 2 be satisfied. At this time, TR_SMC(r_i, r_j) is not the disjoint relation because there will be packet payloads that satisfy both r_i.SMC and r_j.SMC, such as $P[1-6$, "$xyzdab$"]. Therefore, only when Set1 to Set4 are all empty, TR_SMC(r_i, r_j) is the disjoint relation. In addition, we only consider that c_i includes c_j when $m > n$ in the second case. In the following algorithm, we consider how to deal with the case when c_i is included in c_j.

Algorithm 6 $((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)$
Input: (c_i, o_i, d_i), (c_j, o_j, d_j), m, n.
Output: True/False.
Initialization Set1 = { }, Set2 = { }, Set3 = { }, Set4 = { }
Set1 = Case1((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
if($m >= n$)
Set2 = Case2((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
else
Set2 = Case2((c_j, o_j, d_j), (c_i, o_i, d_i), n, m)
Set3 = Case3((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
Set4 = Case4((c_i, o_i, d_i), (c_j, o_j, d_j), m, n)
if(Set1 = $\varnothing$ && Set2 = $\varnothing$ && Set3 = $\varnothing$ && Set4 = $\varnothing$)
Return True;
else
Return False;

2.3 Calculation of TR(r_i, r_j)

The calculation of TR(r_i, r_j) only needs to compose TR_RMC(r_i, r_j) and TR_SMC(r_i, r_j). For example, if TR_RMC(r_i, r_j) and TR_SMC(r_i, r_j) are equal or inside relations, based on the set theory and Eqs. (1) to (4), TR(r_i, r_j) can be calculated as follows:

$$
\begin{aligned}
&(\text{TR\_RMC}(r_i, r_j) = \text{Equal or Inside}) \wedge \\
&\quad (\text{TR\_SMC}(r_i, r_j) = \text{Equal or Inside}) \Leftrightarrow \\
&\quad (P\_\text{RMC}(r_i) \subseteq P\_\text{RMC}(r_j)) \wedge \\
&\quad (P\_\text{SMC}(r_i) \subseteq P\_\text{SMC}(r_j)) \Leftrightarrow \\
&\quad (P\_\text{RMC}(r_i) \cap P\_\text{SMC}(r_i)) \subseteq (P\_\text{RMC}(r_j) \cap \\
&\quad P\_\text{SMC}(r_j) \Leftrightarrow P(r_i) \subseteq P(r_j) \Leftrightarrow \\
&\quad \text{TR}(r_i, r_j) = \text{Equal or Inside}
\end{aligned}
$$

The implementation algorithm of two topological relations' composition is shown as follows:

Algorithm 7 (rel_1, rel_2)

Input: rel_1, rel_2.

Output: TR(r_i, r_j).

```
Initialization {
   Boolean flag1 = flag2 = flag3 = 0;
   Relation rel1, rel2;
}
if (((rel1 = = Equal) || (rel1 = = Inside)) &&
 ((rel2 = = Equal) || (rel2 = = Inside)))
       flag1 = 1;
else if (((rel1 = = Equal) || (rel1 = = Contain)) &&
      ((rel2 = = Equal) || (rel2 = = Contain)))
       flag2 = 1;
else if ((rel1 = = Disjoint) || (rel2 = = Disjoint))
       flag3 = 1;
if (flag1 && flag2)      TR(ri, rj) = Equal;
else if (flag1)          TR(ri, rj) = Inside;
else if (flag2)          TR(ri, rj) = Contain;
else if (flag3)          TR(ri, rj) = Disjoint;
else TR(ri, rj) = Overlap;
Return TR(ri, rj);
```

3 Experiments and Discussion

We developed a prototype system for our proposal on Intel Core i5-2400 processor with 4GB of RAM. The composition of the prototype system is shown in Fig. 2. The inputs of the prototype sytem are RMCs and SMCs of n Snort rules. The extraction of RMCs and SMCs according to n Snort rules is a manual operation in this paper. The functions for calculating TR_RMC(r_i, r_j), TR_SMC(r_i, r_j), and TR(r_i, r_j) are implemented by the Haskell programming language.

In this paper, we only implemented the topological relations calculation of rules, which have the form of k range matching key fields and one string matching content. Therefore, not all the Snort rules are the analysis objects of our proposed method, such as the Snort rules that have two or more content options, the rules that have uricontentoptions, and so on.

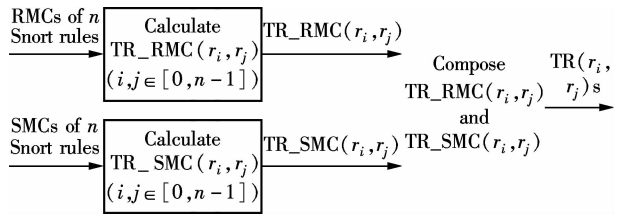

Fig. 2 Prototype system

To validate the applicable scope of our proposed method, we divided each Snort rule file into two parts: analyzable rules and not analyzable rules. The results are shown in Fig. 3. According to Fig. 3, we analyzed twelve Snort rule files, where their total numbers of rules range from approximately 80 to 400, and the percentage of analysis objects is shown in Tab. 1. From the results, we can see that only rpc. rules have no objective rules, and other rule files have at least over 50% objective rules. Therefore, our proposed method can be applied to most Snort rule files to calculate the topological relations between rules.

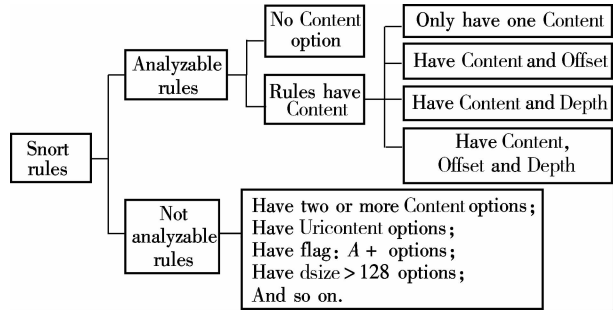

Fig. 3 Classification of Snort rules

Tab. 1 Percentage of analyzable Snort rules

Snort rule files	Number of Snort rules	Number of not analyzable rules	Number of analyzable rules		Percentage of not analyzable rules/%	Percentage of analyzable rules/%
			No Content option	Have Content option		
Exploit	235	97	16	122	41	59
Ftp	83	19	0	64	23	77
Icmp-info	93	0	79	14	0	100
Oracle	325	14	0	311	4	96
Policy	75	35	0	40	47	53
Rpc	167	167	0	0	100	0
Smtp	97	46	0	51	47	53
Specific-threats	149	34	0	115	23	77
Sql	91	9	0	82	10	90
Web-cgi	370	15	304	51	4	96
Web-iis	143	16	108	19	11	89
Web-php	145	20	95	30	14	86

To evaluate the efficiency and performance of our prototype system, we first select approximately 10 to 140 rules from five commonly used Snort rule files, such as exploit. rules, ftp. rules, and so on. Then, we calculate the topological relations between any two rules pairs of each rule file, which have no duplicate pairs, and evalu-

ate their time and memory consumptions. For example, if a rule file only has five rules, their topological relations of any no duplicate rule pairs are calculated, and the experimental results are shown in Fig. 4. The triple of (1324, 1325, inside) shown in Fig. 4 demonstrates that two Snort rules with sid 1324 and 1325 have the inside topological relation.

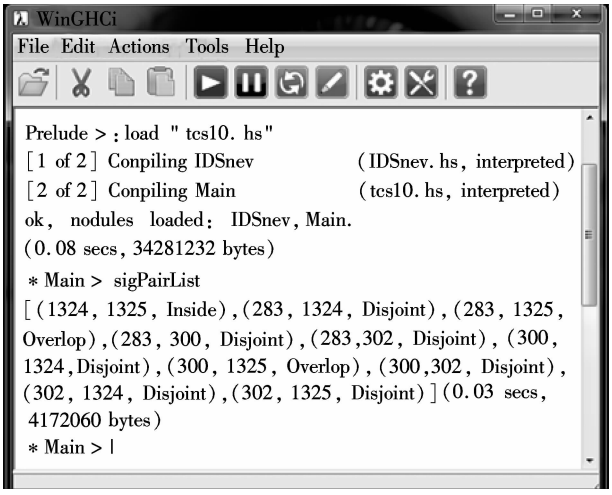

Fig. 4 Topological relations of five rules

The detailed experimental results are shown in Tab. 2 and Tab. 3. Tab. 2 shows the time consumptions when 10 rules are increased per time, and Tab. 3 shows the corresponding memory consumptions. From these results, we can find that the topological relations between Snort rules can be calculated with reasonable time and space requirements when the numbers of rules range from approximately 10 to 140. Therefore, in small sets of Snort rules, our proposed method can be used.

Tab. 2 Time consumption of each rule file s

Number of rules in each file	Exploit	Ftp	Policy	Specific-threats	Sql
10	0.08	0.11	0.09	0.08	0.08
20	0.09	0.09	0.08	0.08	0.09
30	0.09	0.09	0.09	0.09	0.11
40	0.12	0.12	0.12	0.14	0.12
50	0.12	0.14		0.14	0.14
60	0.16	0.14		0.14	0.14
65	0.16	0.16		0.16	0.16
82	0.16			0.16	0.16
90	0.16			0.17	
100	0.17			0.17	
110	0.20			0.20	
115	0.23			0.22	
134	0.25				

The analysis method of topological relations between Snort rules is essential before conflict detection. Although the proposed method cannot be used to detect the conflicts of Snort rules directly, they can be determined by the analysis results and actions of the Snort rules.

Tab. 3 Memory consumption of each rule file MB

Number of rules in each file	Exploit	Ftp	Policy	Specific-threats	Sql
10	35.67	36.84	35.65	36.63	35.59
20	42.12	42.18	42.13	43.61	42.13
30	48.61	48.13	48.59	50.59	49.10
40	54.64	54.63	55.12	57.58	55.61
50	61.07	61.09		65.03	62.06
60	67.59	67.58		71.54	68.57
65	74.06	70.58		78.51	75.51
82	80.58			85.55	83.61
90	87.05			92.51	
100	94.07			99.02	
110	100.55			105.45	
115	107.03			109.01	
134	118.02				

4 Conclusion

In this paper, we propose a method to calculate the topological relations between Snort rules. We use the Haskell programming language to implement our proposed method. Experimental results clarify that the topological relations between Snort rules can be calculated with reasonable time and space consumptions.

Our future research will focus on developing an extensional function to deal with all kinds of Snort rules. We will also further improve the system so that it can deal with a large number of Snort rules.

References

[1] Snort Team. Snort users manual 2.9.7 [EB/OL]. (2014)[2015-06-17]. http://manual.snort.org/.

[2] Al-Shaer E, Hamed H, Boutaba R, et al. Conflict classification and analysis of distributed firewall policies [J]. *IEEE Journal on Selected Areas in Communication*, 2005, **23**(10): 2069 - 2084.

[3] Gouda M G, Liu X Y A. Firewall design: Consistency, completeness, and compactness [C]//*Proceedings of the 24th International Conference on Distributed Computing Systems*. Washington, DC, USA, 2004: 320 - 327.

[4] Yuan L, Mai J, Su Z, et al. Fireman: A toolkit for firewall modeling and analysis [C]//*Proceedings of the* 2006 *IEEE Symposium on Security and Privacy*. Washington, DC, USA, 2006: 199 - 213.

[5] Yin Y, Xu J D, Takahashi N. Verifying consistency between security policy and firewall policy by using a constraint satisfaction problem server [C]//2011 *International Conference on Future Wireless Networks and Information Systems*. Macao, China, 2011: 135 - 145.

[6] Yin Y, Katayama Y, Takahashi N. Detection of conflicts caused by a combination of filters based on spatial relationships [J]. *IPSJ Journal*, 2008, **16**(9): 142 - 156.

[7] Thanasegaran S, Yin Y, Tateiwa Y, et al. A topology-based conflict detection system for firewall policies using bit-vector-based spatial calculus [J]. *International Journal of Communications, Network and System Science*, 2011, **4**(11): 683 - 695.

[8] Al-Mamory S O, Hamid A, Abdul-Razak A, et al. String matching enhancement for Snort IDS [C]//2010 5*th International Conference on Computer Sciences and Convergence Information Technology*. Seoul, Korea, 2010: 1020 - 1023.

[9] Zhao K, Chu J F, Che X L, et al. Improvement on rules matching algorithm of Snort based on dynamic adjustment [C]//2*nd International Conference on Anti-counterfeiting, Security and Identification*. Guiyang, China, 2008: 285 - 287.

[10] Meng Q D, Zhang X L, Lü D W. Research on detection speed improvement of Snort [C]//*International Conference on Internet Technology and Applications*. Wuhan, China, 2010: 1 - 5.

[11] Kuang J, Mei L K, Bian J L. An innovative implement in organizing complicated and massive intrusion detection rules of IDS [C]//*IEEE* 2*nd International Conference on Cloud Computing and Intelligent Systems*. Hangzhou, China, 2012: 1328 - 1332.

[12] Kang B J, Kim H S, Yang J S, et al. Rule indexing for efficient intrusion detection systems [C]//*Lecture Notes in Computer Science*, 2011, **7115**: 136 - 141.

[13] Stakhanova N, Ghorbani A A. Managing intrusion detection rule sets [C]//*Proceedings of the Third European Workshop on System Security*. Paris, France, 2010: 29 - 35.

[14] Cho Y H, Mangione-Smith W H. Programmable hardware for deep packet filtering on a large signature set [C]//*Workshop on Architectural Support for Security and Anti-Virus*. Boston, USA, 2004: 1 - 9.

[15] Hutchings B L, Franklin R, Carver D. Assisting network intrusion detection with reconfigurable hardware [C]//*Proceedings of the* 10*th Annual IEEE Symposium on Field-Programmable Custon Computing Machines*. Washington, DC, USA, 2002: 111 - 120.

[16] Chen H, Summerville D H, Chen Y. Two-stage decomposition of Snort rules towards efficient hardware implementation [C]//7*th International Workshop on Design of Reliable Communication Networks*. Washington, DC, USA, 2009: 359 - 366.

一种 Snort 规则间拓扑关系的分析方法

殷 奕[1,2] 汪 芸[1] Takahashi Naohisa[3]

([1] 东南大学计算机科学与工程学院, 南京 211189)

([2] 南京师范大学计算机科学与技术学院, 南京 210023)

([3]Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology, Nagoya 466-8555, Japan)

摘要:针对 Snort 规则间的相互关系难以把握的问题,基于集合理论对 Snort 规则间的拓扑关系进行了分类,并提出了 Snort 规则间拓扑关系的计算方法. 在已有的 Snort 规则相互关系分析方法中,通常只根据 Snort 规则的头部信息来决定整条规则之间的相互关系. 所提方法在不考虑 Snort 规则动作的情况下,对已有的方法进行了改进,能够同时根据 Snort 规则的头部信息和选项部分的取值来分类和计算整条 Snort 规则之间的拓扑关系. 另外,使用函数式编程语言 Haskell 实现了所提方法. 实验结果表明,该方法能够快速有效地计算出 Snort 规则间的拓扑关系,并且能为后续的 Snort 规则间的冲突检测提供重要的依据.

关键词:入侵检测系统;Snort 规则;函数式编程语言

中图分类号:TP393. 08