A Method to Obtain Signatures from Honeypots Data

Chi-Hung Chi[1], Ming Li[2] (corresponding author) and Dongxi Liu[1]

[1] School of Computing, National University of Singapore, Singapore 117543
{Chich, liudx}@comp.nus.edu.sg

[2] School of Information Science & Technology, East China Normal University
Shanghai 200062, PR. China
ming_lihk@yahoo.com

Abstract. Building intrusion detection model in an automatic and online way is worth discussing for timely detecting new attacks**.** This paper gives a scheme to automatically construct snort rules based on data captured by honeypots on line. Since traffic data to honeypots represent abnormal activities, activity patterns extracted from those data can be used as attack signatures. Packets captured by honeypots are unwelcome, but it appears unnecessary to translate each of them into a signature to use entire payload as activity pattern. In this paper, we present a way based on system specifications of honeypots. It can reflect seriousness level of captured packets. Relying on discussed system specifications, only critical packets are chosen to generate signatures and discriminating values are extracted from packet payload as activity patterns. After formalizing packet structure and syntax of snort rule, we design an algorithm to generate snort rules immediately once it meets critical packets.

1 Introduction

Techniques in an intrusion detection system (IDS) can usually be classified into two. One is anomaly detection and the other misuse detection. Anomaly detection views behaviors deviated significantly from normal profile as attacks, e.g., [6] [7]. Misuse detection systems detect attacks by finding the activities matched with attack signatures, which are drawn from known attacks [3] [4] [5]. This approach is effective to detect known attacks but hard to identify new attacks due to the lack of corresponding signatures.

In order to enable misuse detection systems to identify new attacks adaptively, we explore a method to construct attack signatures from data gathered by honeypots in an automatic and online way. A honeypot is security resource whose value lies in being probed, attacked, or compromised [1]. Generally, honeypots play no role in production systems. Hence, traffic to and from honeypots are suspicious, providing us with opportunities to get pure intrusive packets. Inspired by this feature of honeypots, we can extract patterns of these packets and use them as signatures for misuse detection systems. Here, we choose snort [3] as our target system. Since signatures between

different signature-based IDSs can be mutually translated [8], the present approach can be extended to other misuse detection systems.

It is usually unnecessary to map each packet to a snort rule. E.G., if a scanning packet or ICMP echo request (ping) is captured, it may not provide distinct information to identify abnormal activities as an intruder may fake its source IP address. Generally, choosing which part of the packet payload as signatures is difficult since a successful attack usually involves a sequence of packets. Therefore, instead of constructing a snort rule for each of them, we only maps chosen packets for system specifications of honeypots. Moreover, our method can extract discriminating values from packet payload as activity pattern rather than use the entire payloads as signatures. In the course of building signatures, system specifications are important. They are specified by honeypots administrator and made up of system commanders, system calls, system configuration files and even some machine instructions.

In summary, the contributions in this paper are 1) a new usage of honeypots, which differs from traditional usage; 2) system specifications based way to recognize critical packets and extract discriminating values as activity pattern; and 3) an automatic and online method to generate attack signatures. In the rest of paper, § 2 describes our requirements to the new usage of honeypots, § 3 discusses our method and § 4 concludes the paper.

2 Requirements to Honeypots

There are many types of honeypots. According to interaction level, they are classified into three [1]: low-interaction, medium-interaction and high-interaction honeypots. Low-interaction honeypots emulate some services, medium-interaction ones also emulate services (but they can response attackers' request to some extent) while high-interaction ones are real operating systems and services. This paper concerns with high-interaction, as we need to collect real data coming from intruders instead of simply detecting unauthorized scans or connection attempts.

In a honeynet, production traffic only goes to production network while intrusion traffic goes to all hosts since intruders try to attack as many systems as possible. When intruders use new attack methods, conventional misuse detection systems in production network may be difficult to sense them. However, when they are equipped with honeypots that can update signature bases of misuse detection systems, it will enable misuse detection systems to adapt new attacks quickly. This paper focuses on how honeypots generate snort rules. The issue of how they exchange between honeypots and snorts is not contained in this paper.

Main requirements to honeypots are data control, data capture and data collection [1]. To the honeypots in our scheme, we have two requirements for our purpose:

1) Correspondence between honeypots and servers in production network. This means each honeypot corresponds to a server or several same servers, and vice versa.

2) Security levels between corresponding honeypots and servers. The security level of honeypots should be as secure as the corresponding servers.

3 Generating Signatures Online

By generating signatures online, we mean generating snort rules on line once honeypots capture suspicious packets without the intervention of administrators. To this end, we consider two issues. One is the specific procedure to map a given packet to a snort rule (§ 3.1). The other is the way to choose the packets among those captured by honeypots to map and extract the activity patterns from the payload (§ 3.2).

3.1 Mapping a Packet to a Snort Rule

To map a given packet to a snort rule, it needs to describe packet structure and the syntax of snort rules formally. We introduce the mapping procedure from a given IP packet to a snort rule in this subsection.

Consider formalizing packet structure. A packet is a stream of raw bits in essence. How to interpret this stream is determined by its structure, which is usually specified as standards. Snort currently can analyze four types of protocols (IP, TCP, UDP and ICMP). Below, we take IP as an example to show how to formalize IP packet structure.

For our purpose, an IP packet structure is described as: <*srcIP*, *dstIP*, *ttl*, *tos*, *fragbits*, *ipoption*, *protocol*, *payload*>, where

--*srcIP* is the source IP address, and *dstIP* the destination IP address;

--*ttl* is the time to live(ttl) filed, and *tos* is the type of service field;

--*fragbits* is the fragment flag filed, including three bits that can be checked, namely, the
reserved (R) bit, more fragments (M) bit and the don't fragment (D) bit;

--*ipoption* is the options field. There eight option types, including strict source routing (ssr),
loose source routing (lsr), IP security option (seq), time stamp (ts), record route (rr), end of list
(eol), no option (nop), and stream identifier (satid).

--*protocol* is the type of transport packet being carried;

--*payload* is the data encapsulated in IP packet.

In the above structure, we omit some fields of less interesting for our research, e.g., check sum field. In addition, suppose *count*(*ipoption*) indicate the count of *ipoption*. Then, *ipoption*[*i*] ($0 \le i < count(ipoption)$) denotes each option's type. If *p* is an IP packet, we use *p.srcIP* to denote *p*'s source address, *p.dstIP* to denote *p*'s destination address, and so on.

The previous syntax of snort rules is the target of signature constructing. If a non-terminal symbol is defined only with a terminal symbol, the non-terminal symbol will

be replaced by the corresponding terminal symbol when generating a rule. Otherwise, we have to determine which terminal symbol should be chosen according to packet content.

Let us discuss mapping procedure. The major work of mapping is to determine values of BNF non-terminal symbols in snort rule syntax. Suppose *p* is an IP packet. Algorithm 1 produces a snort rule from *p*. In addition, option "msg" ":" "Signature from honeypots" ";" is desirable to be included in each generated snort rule so as to facilitate management of rule bases, but we omit this option in *algorithm* 1 for simplicity.

Algorithm 1. *Mapping procedure between a given packet and a snort rule*

INPUT: *an IP packet pt*

OUTPUT: *a snort rule*

1. let ip_patterns = "ttl" ":" "*p.ttl*" ";" "tos" ":" "*p.tos*" ";";
2. let ip_patterns += "fragbits" ":"+*substring*("RDM", *p.fragbits*)+";" ;
3. for *i*=0 to *count*(*p.ipoption*)-1, do
   let ip_patterns += "ipopts" ":" "*p.ipoption*[i]" ";";
4. if *p.protocol*∉{TCP, UDP, ICMP}, then
   <protocol> ::= "ip";
   <rport> ::= "any";
   <options> ::= ip_patterns+"content" ":" "*p. payload*" ";";
5. if *p.protocol*=UDP, then
   <protocol> ::= "udp";
   <rport> ::= "*p.payload.destPort*";
   <options> ::= ip_patterns+"content" ":" "*p. payload.payload*" ";" ;
6. if *p.protocol*=TCP, then
   <protocol> ::= "tcp";
   <rport> ::= "*p.payload.destPort*";
   let tcp_flags= *substring*("12UAPRSF", *p.flags*);
   if tcp_flags= "", then tcp_flags= "0";
   let tcp_patterns = "flags" ":"+ tcp_flags + ";";
   <options> ::= ip_patterns+tcp_patterns+"content" ":"
   "*p. payload.payload*" ";" ;
7. if *p.protocol*=ICMP, then
   <protocol> ::= "icmp";
   <rport> ::= "any";
   let icmp_patterns = "itype" ":" "*p.payload.type*" ";";
   let icmp_patterns += "icode" ":" "*p.payload.code*" ";";
   let icmp_patterns += "icmp_id" ":" "*p.echo_id*" ";";
   let icmp_patterns += "icmp_seq" ":" "*p.echo_seq*" ";";
   <options>::=ip_patterns+icmp_patterns+"content"
   ":" "*p.payload.payload*" ";" ;
8. return;

In the above algorithm, "+" is used to concatenate two terminal symbols in BNF; "" represents an empty string; *substring*(*str*, *indicator*) extracts a substring from *str* according to the indicator. For example, for *substring*("12UAPRSF", *p.flags*), if *p.flags*= 0x55, then substring "2ARF" is generated. It should be noted that *algorithm* 1 simply map the *p*'s payload, which will be improved in the following section.

3.2 System Specifications, Critical Packets, Discriminating Values and Building Signature

Using *algorithm* 1, one can map fields of a given packet to the corresponding constructs of a snort rule. Nevertheless, it is still not enough to build accurate and efficient attack signatures. As stated previously, it may be unnecessary to map every captured packet to a snort rule and extracting activity pattern from packet payload is also a difficult task. In this subsection, we address both problems by turning to the knowledge in system specifications of honeypots.

Different definitions of system specifications can be used for different purposes. For example, to describe hardware system for a computer, one can use CPU frequency, memory and hard disk size, network speed as system specifications. Here, we concern with the system specifications of honeypots to characterize the seriousness level of captured packets regarding network security.

Generally, intruders exploit vulnerabilities of programs to obtain necessary privilege to implement attacks. In the course of attacks, in particular for attacks on hosts like R2L attacks in [9], an intruder uses system calls (even some specific machine instructions) to change the execution path and uses system commands to change the system state, or modify system configuration files to leave back doors. For examples, using *WinExec* executes the shell code in buffer overflow attack; copying worm or Trojan programs in malicious code attack. It can be concluded that, among packets captured by honeypots, those containing system calls, commands or configuration files will represent more serious intrusive activities than those without such information.

Therefore, system specifications of a honeypot are defined as a set of system calls, system commands, system configuration files, or even machine instructions. *C* is used to denote system specifications. For example, smd.exe, win.ini, WinExe, dir, cp are all elements of *C* in Microsoft Windows; fork, passwd, ln belong to *C* on unix or linux platform; machine instruction "Jump" is also an element of *C*.

For each honeypot, its administrator should specify its system specifications explicitly. That *C* is empty means that nothing is considered to be serious. In this case, no rules will be generated. On the other hand, if *C* contains all possible objects on honeypots, then almost every captured packet will result in a snort rule. For example, if file index.htm of web server on honeypots is included in *C*, then an unwelcome browser to this file will generate a snort rule, and obviously this rule will cause false positives in production network. Fortunately, an administrator often knows his system very well. In other words, he knows what system specifications are, and which specifications are more important. Therefore, he can give a reasonable system specifications *C* for a honeypot.

A list of probes captured by a honeypot in a 30-day period is given in [12], where some "ordinary" packets (such as the ICMP echo request packets and DNS version query packets) are included. Suppose p is a normal packet and r is the snort rule by using *algorithm* 1. r will match packets in normal production traffic, which will cause false alarms. Therefore, we do not map "ordinary" packet captured by honeypots to a snort rule, and instead, only critical packets are chosen to do so. Below we describe the definition of critical packets.

Definition 1. Suppose $p_1 \rightarrow p_2 \rightarrow p_3 \ldots \rightarrow p_n$ is a series of packets captured by honeypots for an attack. $p_i (1 \le i \le n)$ is a critical packet for the attack, if the following conditions hold:

a) The payload of p_i contains c, $c \in C$;

b) For $\forall p_j (1 \le j \le i)$, p_j is not critical, which are ordinary packets.

In order to implement attacks, ordinary packets are usually used by an intruder to gather information about target hosts. Critical packets help an intruder to get necessary privileges or install backdoor programs. The packets following critical packets represent intruder's activities on honeypots after getting some privileges, such as create directory, modify files, etc. These packets also contain system calls. However, we don't translate them into snort rules because they rely on the critical packets. Since snort rule is per-packet based signature, we only define one critical packet in a successful attack. In fact, if misuse detection system uses state-transition signatures as stated in [4], in which every state represents an occurrence of events, we can choose many critical packets to build such kind of signatures.

For attacks on network, packets may contain no system calls or system commands, such as tear drops attacks and syn flood attack. The goals of these attacks are usually to crash the target systems or make them deny services. Constructing signature based on critical packets may not cover this kind of attacks. However, attacks on network only involve packet headers that have more strict structure. Thus, they have much less variations and new attacks than those on hosts.

In *algorithm* 1, the entire packet payload is used as the argument value of *content* option. Hence, *content* option in the resulting rule contains more than enough bits to characterize activity pattern. It will result in two drawbacks: 1) matching snort rules to network traffic will be low efficient because there are more bits to deal with; and 2) it is possible to make false negatives because finding longer bit sequence exactly in network traffic is more difficult and the change of some redundant bits will cause variants of attacks and lead to miss matching. To avoid these drawbacks, we should identify the representative subsequence of bits in packet payload as activity patterns, called discriminating value of the packet. For a critical packet, only its discriminating value is used as the argument value of *content* options. A formal definition of discriminating value is given below.

Definition 2. Suppose p is a critical packet captured by honeypots. The discriminating value of p is a triple ($serv$, op, c) or a pair ($serv$, c), where

a) $serv$ is the service type;

b) *op* is the type of service operation;

c) $c \in C$ is contained by *p*'s payload.

If the service is based on TCP or UDP, then discriminating values will take the triple form, otherwise the pair form. For example, discriminating value for an HTTP packet can be (HTTP, GET, cmd.exe); discriminating value for the attack based on buffer overflow on IP protocol software can be (IP, *WinExec*). In the first case, field *serv* can be characterized by the destination port uniquely, such as 80 for HTTP and 23 for TELNET; field *op* can be determined by interpreting packet payload according to the packet format of service *serv*. In the second case, field *serv* can be determined by using protocol name, i.e. IP or ICMP. In the both cases, field *c* can be gotten by looking up each element of *C* and one of the found elements can be used as *c*. Because *p.payload* maybe contains several elements of *C*, we use a heuristic method to choose the most distinct one. Suppose *n* elements c_i ($1 \le i \le n$) have been found and $pos(c_i)$ is the position of c_i in *p.payload*. Then c_i with minimum $pos(c_i)$ is chosen as *c*.

Combining critical packets with discriminating values, *algorithm* 1 can be improved to be *algorithm* 2. For simplicity, only the modifications are listed.

Algorithm 2. *Building snort rules with critical packets and discriminating values*

INPUT: *critical packet p from a honeypot, System Specifications C*

OUTPUT: *a snort rule or null*

1. $Sp = \Phi$;
2. for $\forall c \in C$, if *found*(*c*, *p.payload.payload*), then
   $Sp = Sp \cup \{(c, pos(c))\}$;
3. if $Sp = \Phi$, then
   return *null*;
4. Let $c = c'$, where $(c', pos(c')) \in Sp$ and $pos(c') = min(\{pos(c'') \mid (c'', pos(c'')) \in Sp\})$;

……

5. if *p.protocol* $\notin$ {TCP, UDP, ICMP}, then
   ……
   <options> ::= ip_patterns+"content" ":" "*c*" ";";
6. if *p.protocol*=UDP, then
   ……
   Let *op* be operation type of service in *p.payload.payload*;
   <options> ::= ip_patterns+"content" ":" "*op*" ";" + "content" ":" "*c*" ";" ;
7. if *p.protocol*=TCP, then
   ……
   Let *op* be operation type of service in *p.payload.payload*;
   <options> ::= ip_patterns+"content" ":" "*op*" ";" + "content" ":" "*c*" ";" ;
8. if *p.protocol*=ICMP, then
   ……
   <options>::=ip_patterns+icmp_patterns+"content" ":" "*c*" ";" ;
9. return;

Compared with *algorithm* 1, *algorithm* 2 has two differences. The first one is to check whether *p* is a critical packet, and if not, it will not generate snort rules for *p*, and if yes, *c* in discriminating values will be calculated. The second is to use *c* and *op* as the argument value of *content* option. As a result, *algorithm* 2 can produce more compact and flexible snort rules to identify the variants of attacks. In addition, we ignore the details to interpret the operation type *op*.

The above explanations imply that the present method is not simply to translate each captured packet to a snort rule. Owing to the limit space, cases to show the application of the present method is not given.

4 Conclusions and Acknowledgements

A usage of honeypots for on-line building snort rules from the data captured by honeypots has been discussed. We have analyzed the requirements to honeypots with respect to assuring honeypots to generate useful signatures for detecting attacks in production network. System specifications used to recognize critical packets and extract discriminating values as activity pattern have been explained. Algorithms for automatic and online generation of attack signatures have been derived. This research is under a grant for the project Pervasive Virtual Community in Cyberspace（R-252-000-079-112), Singapore. The paper is in part sponsored by SRF for ROCS, State Education Ministry, PRC.

References

1. L. Spitzner, Honeypots: *Tracking Hackers*, Addison-Wesley, 2002.
2. Honeynet Project, Know Your Enemy, Honeynets, http://project.honeynet.org/papers/honeynet/.
3. M. Roesch, Snort-lightweight intrusion detection for networks, *1999 USENIX*, 1999.
4. K. Ilgun and et al, *IEEE T. on Software Eng.*, 21 (3), 1995, 181-199.
5. V. Paxson, *Computer Networks*, 31 (23/24), 1999, 2435-2463.
6. M. Li, An approach to reliably identifying signs of DDOS flood attacks based on LRD traffic pattern recognition, to appear on *Computer & Security*, 2004.
7. R. A. Kemmerer and G. Vigna, *Supplement to Computer*, 35 (4), 2002, 27-30.
8. S. T. Eckmann, *Proc., RAID 2001, LNCS 2212*, 2001, 69-84.
9. K. Kendall, *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems*, Master Thesis, MIT, 1999.
10. M. Roesch and C. Green, Snort users manual, http://www.snort.org/docs/ SnortUsers-Manual.pdf
11. http://project.honeynet.org/papers/enemy/probed.txt.