Automatic Generation of Attack Signatures for implementing a new Plugin to Honeyd

Motahareh Dehghan[1] Babak Sadeghiyan[2]
Department of Computer Engineering and Information Technology
Amirkabir University of Technology (AUT), Tehran, Iran
[1] motahareh479@aut.ac.ir [2] basadegh@aut.ac.ir
Paper Reference Number: 07-11-3055
Presenter: Motahareh Dehghan

Abstract

In this paper, we design and implement an automated signature generation system. Current network intrusion detection systems work on misuse detectors, where the packets in the monitored network are compared against a repository of signatures. But, we focus on automatic signature generation from malicious network traffic. Our proposed system inspects honeypot traffic and generates intrusion signatures for unknown traffic.

The signature is based on traffic patterns, using Longest Common Substring (LCS) algorithm. It is noteworthy that our system is a plugin to honeyd - a low interaction honeypot. The system's output is a file containing honeypot intrusion signatures in pseudo-snort format.

Signature generation system has been implemented for Linux Operating System (OS) but due to the common use of Windows OS, we implement for Windows OS, using C programming language.

Keywords: honeypot, honeyd, Intrusion Detection System (IDS), Longest Common Substring (LCS) algorithm, signature

1. Introduction

Today, in order to reduce the effects of network attacks and prevent network intrusion, many security equipment designed and implemented. One of them is honeypot that offers a variety of services and attracts attackers.

In this paper, we obtain patterns from honeypot's traffic on basis of packets sent to multiple hosts from attackers, have approximately similar content. Then from these patterns we generate signatures.

The system's output is a file containing intrusion signatures in pseudo- snort format. As illustrated in Fig. 1, our proposed system is a plugin to honeyd- low interaction honeypot- that designs appropriate responses based on these signatures [3]. Also, these signatures can be used to filter the traffic directed towards the honeypot, in order to reduce the amount of traffic needed to be processed by the honeypot sensors.

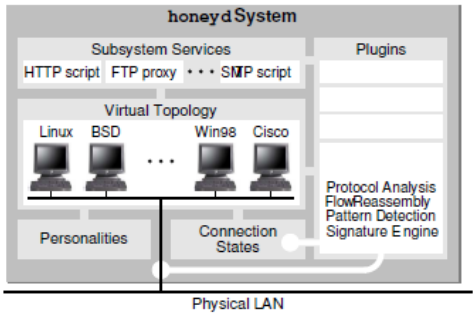

Fig. 1- Honeyd and proposed system

2. Data and materials

2.1. Honeypot

As mentioned in [2], honeypot is a special machine on the network which used as a trap for attackers. Deliberately, operating systems infected by Trojan, backdoor or weak servers are installed on it to interact and engage attackers.

- Types of honeypots[5,6]
  - Honeypots in terms of reality
    - Physical honeypots
      A physical honeypot is a real machine in network which has a particular IP address.
    - Virtual honeypots
      A virtual honeypot is simulated by another machine.
  - Honeypots in terms of interaction by attacker
    - Low interaction honeypots
      A low-interaction honeypot will typically run or emulate a small number of services on a real or emulated operating system.
    - High interaction honeypots
      A high-interaction honeypot is often a real computer running a real operating system.

2.2. Honeyd

Honeyd is an OpenSource low-interaction honeypot implemented for UNIX and Windows Operating Systems (OS) [1]. Every attacker intends to communicate network with useless Internet Protocol (IP) Address, honeyd disconnects this connection and interacts with him. Honeyd is a framework for virtual honeypots which allows thousands of IP addresses communicate with virtual machines. Thus, it should be able to simulate network topology. According to Figure 2, honeyd is a central machine which captures the traffic directed towards the virtual honeypots and simulates appropriate responses [7].

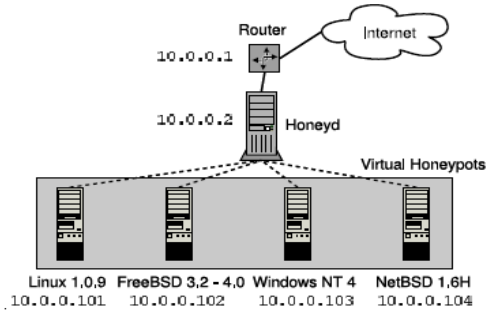

Fig. 2- honeyd as a central machine

2.3. Longest Common Substring of Two Strings [10]

The longest common substring of two strings *S*1, *S*2 will be denoted by *LCS*(*S*1, *S*2). Let *S*1 = *{*contumaciously*}*, *S*2 = *{*contumeliously*}*. Their *LCS*(*S*1, *S*2) is the set formed by strings *{*contum*}* and *{*iously*}*, both with 6 characters each. When there is no common substring, *LCS*(*S*1, *S*2) is *{null}*.

Consider the following example to realize LCS algorithm:

Example. Given three strings S1 = {abac}, S2 = {bada} and S3 = {acd}. The set of terminal symbols will be {$,#}; symbol % is the terminal symbol needed for constructing the suffix tree. The resulting suffix tree T (S1$1S2#S3) = T (abac$bada#acd) is shown in Figure 3.

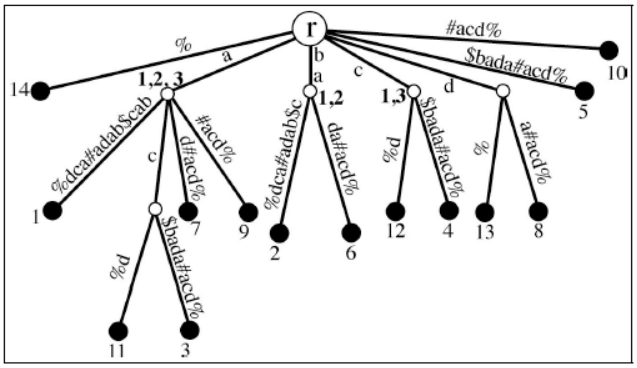

Fig. 3- Generalized suffix tree of a set of strings

In order to solve the LCS problem, we just build the generalized tree of strings S1 and S2, T (S1$S2). When constructing it, mark each internal node with a label 1 or 2, depending on what string the current suffix is from. For example, in Figure 3 there is an internal node marked with three labels, 1, 2, and 3, showing a common substring to S1, S2 and S3, namely, string {a}. The leaves of the subtree rooted at that internal node posses numbers from the 76 Suffix Trees and its Applications three strings S1, S2 and S3. Other internal nodes are just marked with two numbers such as the corresponding to string {ba} or string {c}.

Therefore, a path-label composed of internal nodes marked with both numbers will spell out a common substring to S1 and S2. Finding LCS (S1, S2) is achieved by just outputting the deepest string in the tree whose edges are both marked by 1 and 2.

2.4. Cygwin [4]

Cygwin is a UNIX-compatible environment that runs on Windows systems. It consists of cygwin1.dll, a library that takes POSIX calls and translates them into Win32 calls; a shell (GNU BASH, the shell used on most Linux systems, is the default); an implementation of the X Window System and, of course, GCC.

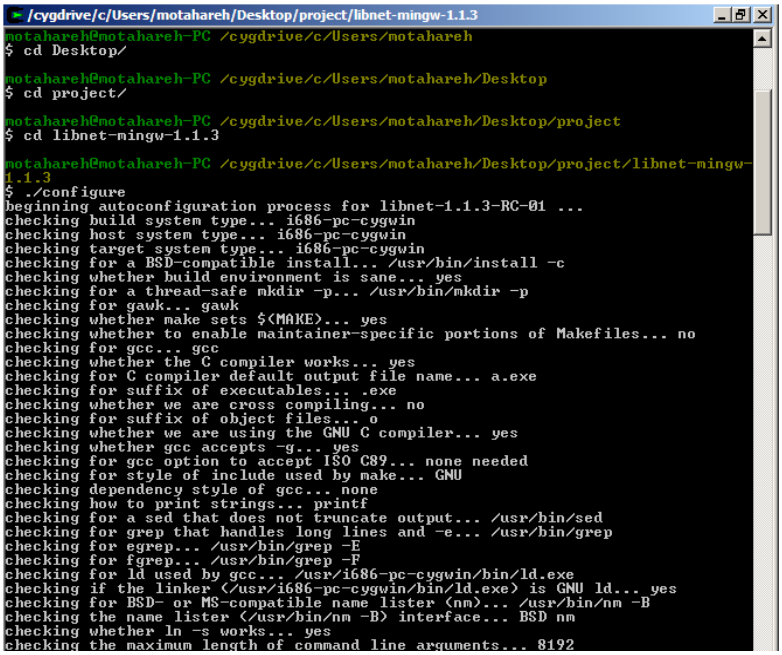

Fig. 4- Cygwin

3. Research Methodology

3.1. System Architecture

The proposed system architecture consists of following parts:

- Local Control Unit
  - Analysis Unit
  - Communication Unit
  - Database
- Known-Attack Filter
- Network Intrusion Prevention System
- Global Control Unit

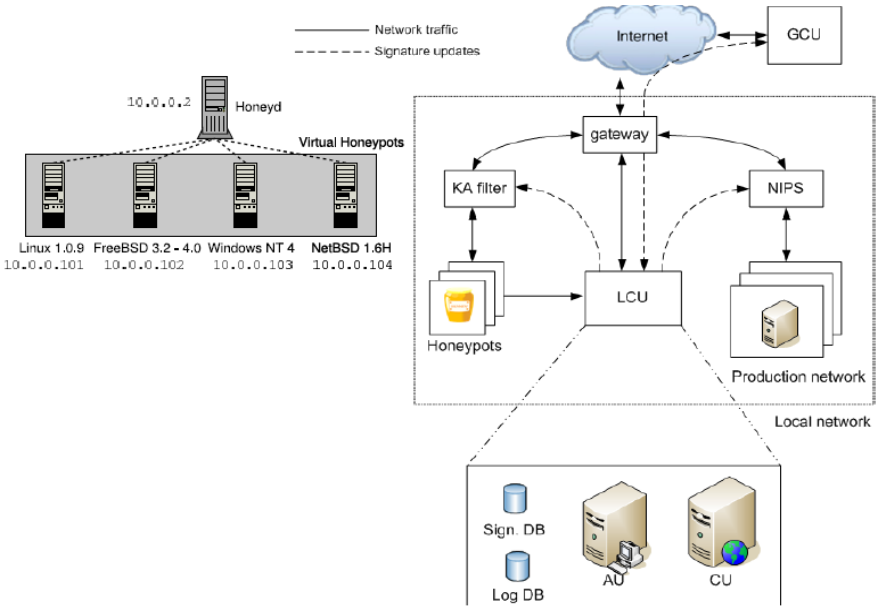

Fig. 5- Proposed System Architecture

❖ **Local Control Unit**

This unit has a simplified version that is only able to receive signature updates from the Global Control Unit (GCU) and use these in NIPS to protect the production network. But in a complicated version, LCU consists of behind units:

- **Analysis Unit - AU**

  The AU's main task is to correlate the incoming honeypot events and create signatures for possible worms. When receiving new events from a honeypot, the following procedure is executed:

  ✓ **Step 1:** The incoming events are stored in the log database and correlated with older events. If a similar chain of events has been received a certain number of times before, it is assumed that the events are caused by a worm and step 2 is carried out. If not, the events are simply stored and the AU returns to idle state.

  ✓ **Step 2:** The network packets causing the same chain of events are compared. If a common substring (larger than a given threshold) is found between these traffic traces, a signature is created.

  ✓ **Step 3:** Before storing the newly generated signature in the database, it is compared with the already existing ones. It can then either be stored directly in the database as a new entry or help to improve one of the older ones.

- **Communication Unit - CU**

  The CU's main purpose is to exchange signatures with the GCU as well as issuing signature updates to the KA filter and NIPS.

- **Databases**

  The signature database is used to store locally generated as well as received signatures. The log database is used to store the logged events along with relevant data.

❖ **Known-Attack Filter – KA Filter**

The main purpose of the KA filter is to look for known attacks (based on the signatures received from the LCU) in the traffic directed towards the honeypots.

❖ **Network Intrusion Prevention System - NIPS**

The NIPS is placed in the system to protect the production network. It can filter traffic that is unwanted based on certain ports as specified by the network administrator, as well as traffic that have been declared malicious as a result of signature updates from the LCU. Similar to the KA filter, it is also possible for the NIPS to report back to the LCU on the activity level of the received signatures.

❖ **Global Control Unit - GCU**

The GCU serves as a central signature storage and distribution unit. It receives signature updates from the distributed LCUs and is able to correlate received data from different locations to compose improved signatures. Based on the received data, it issues periodic updates to the LCUs. As the GCU is a potential single point-of-failure and the effects can be catastrophic if it is compromised, the requirements regarding security are strict. All communication between the GCU and LCUs should be authenticated and encrypted in order to avoid forged signature updates.

3.2. Implementation

The implementation is based on traffic patterns, using LCS algorithm. Our system output is a file containing honeyd intrusion signatures in pseudo- snort format to filter

unwanted production network traffic. Also the proposed architecture introduces the use of a Known-Attack (KA) filter. The main purpose of this filter is to remove known attacks from the traffic directed towards the honeypots. This filter reduces the amount of traffic needed to be processed by the honeypot sensors.

Signature generation system implemented in Linux OS, but due to the common use of Windows OS, our implementation is in Windows OS, using C programming language.

According to [1, 8, 11], for implementing the system we use below items:

1. Winpcap to capture packets
2. Cygwin compiler
3. Libevent library
4. WpdPack
5. Libdnet library
6. Libstree library
7. Python

4. Results and Analysis

Signatures are periodically reported to an output module which implements the actual logging of the signature records. At the moment, there are modules that convert the signature records into pseudo-Snort format and a module that dumps the signature strings to a file. Our proposed system generated 53 signatures during a roughly 18-hour period and the signatures format is as follow:

alert tcp 61.0.0.0/8 any → 129.241.196.0/24 80 (msg: "Hello!!! "; flags: PA+; flow: established; content: "GET http://lookfreebies.com/prx1.php HTTP/1.0|0D 0A|Accept: */*|0D 0A|Accept-Language: en-us|0D 0A|User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)|0D 0A|Host: lookfreebies.com|0D 0A|Connection: Keep-Alive|0D 0A 0D|";)

As mentioned in [9], rule options are:

- The direction operator → indicates the orientation, or direction of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source.
  - ✓ The generated signature has TCP protocol that the source IP address is 61.0.0.0/8 and the destination IP address is 129.241.196.0/24 and its port is 80.
- The **msg** rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert.
- The **content** keyword is one of the more important features of Snort. It allows the user to set rules that search for specific content in the packet payload and trigger response based on that data.
  - The **flags** keyword is used to check if specific TCP flag bits are present.

5. Conclusion

In Summary, an automated signature generation system for Windows OS designed and implemented. This system considered as honeyd plugin. Our tests show the proposed system is particularly good at generating attack signatures.

Acknowledgements

I would like to thank Mr. Erfan Khosravian for valuable feedbacks and Comments.

References

[1] Baumann, Reto. "*Honeyd – A low involvement Honeypot in Action*". GCIA, GSEC, CCNA. 2003.

[2] Grønland, Vidar Ajaxon. "*Building IDS rules by means of a honeypot*". Department of Computer Science and Media Technology Gjøvik University College, 2006.
[3] Kreibich, Christian; Crowcroft, Jon. "*Honeycomb. Creating Intrusion Detection Signatures Using Honeypots*". University of Cambridge Computer Laboratory, JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom. 16. January 2006.
[4] Moody, George. "*An Introduction To Cygwin*". Harvard-MIT Division of Health Sciences and Technology.
[5] Noordin , Yusuff, Mohamed. "*HONEYPOTS REVEALED*". IT Security Officer. Specialist Dip. Info Security, MA. Internet Security Mgmt. (Ongoing)
[6] O'Shea, Claire. "*Intrusion Detection with honeypots*". COMP 290. Spring 2005.
[7] Provos, Niels. "*Honeyd- A Virtual Honeypot Daemon*" . Center for Information Technology Integration, University of Michigan. 2003.
[8] Provos, Niels; Mathewson, Nick. "*Libevent – an event notification library*". 12 Oct 2011. URL: http://libevent.org/
[9] Roesch ,Martin; Green, Chris. "*SNORT Users Manual 2.8.5*". October 22, 2009.
[10] Sung, Wing-Kin; Melvin, Zhang Zhiyong. "*Combinatorial methods in bioinformatics. Suffix Tree and Suffix Array*". September 2,2005.
[11] Van Rossum, Guido; "*Introduction to Python*". LinuxWorld, New York City. January 2002.