DETECTION OF HONEYPOT SYSTEMS IN NETWORK

Martin Teknős
Bachelor Degree Programme (4), FIT BUT
E-mail: xtekno00@stud.fit.vutbr.cz

Supervised by: Maroš Barabas
E-mail: ibarabas@fit.vutbr.cz

Abstract: I am developing a console application in Python for detecting honeypots in network. I am using two techniques (a TCP/IP fingerprinting and a clock skew estimation). I present simplified description of those techniques, of their core functionality in the application and some results of initial experiments.

Keywords: honeypot detection, virtual honeypot, virtual honeynet, clock skew, TCP/IP fingerprinting

1 INTRODUCTION

Being able to detect honeypots is important to malicious users as well as security professionals. Attackers want keep their techniques secret. If an attacker was to perform a zero day attack on a honeypot, it is likely to become public knowledge a lot faster and as a result, value of attackers advantage will be reduced if not expunged. When an attacker recognizes a honeypot, it becomes useless.

Work presented in this paper concentrates on network level detection of virtual honeypots and virtual honeynets. Virtual machines are quite common as honeypots. Physical honeypots are usually undetectable on network level and most of them can be detected only on system level with access to a system.

I am developing a console application in Python for detecting honeypots in network. I am using two techniques (a TCP/IP fingerprinting and a clock skew estimation), which are described in following chapters.

2 TCP/IP FINGERPRINTING

This technique is based on work from [4]. In principle, it is about extracting various quantitative and qualitative data from each TCP/IP connection with a tested machine and using them to classify the machine as a honeypot or a benign system.

Figure 1 shows a simplified core functionality of the application for this technique. The application makes a traffic with a tested machine with help of `nmap` and records it with `tcpdump`. Then it extracts data needed for a classification from recorded packet trace and process them. A result of a data processing is a 51-dimensional normalized vector. Finally, the application classifies the vector with a Support Vector Machine (SVM) using `libsvm` [1].

I managed to get 440 data samples for a local network. A data set contains both benign systems (100 items) and honeypots (340 items). The data set was randomly divided to a set of 150 training data and a set of 290 testing data. Table 1 summarizes performance of classification with the SVM for a testing data set. According to statistics, the application performs well in detecting honeypots, but not so well in identifying regular systems. This is probably due to only 22.7% of benign systems in the data set.

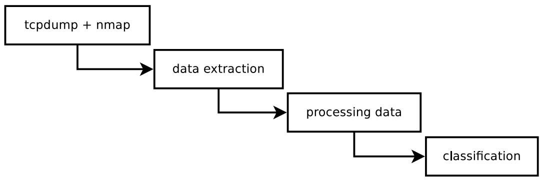

Figure 1: TCP/IP fingerprinting simplified

statistic:	(%)
sensitivity (honeypots correctly classified as honeypots):	100.0
specificity (benign systems correctly classified as benign systems):	39.66
precision (classified as honeypots correctly):	86.84
negative predictive value (classified as regular systems correctly):	100.0
accuracy (overall accuracy):	87.89

Table 1: Statistics for TCP/IP fingerprinting classification with SVM

3 CLOCK SKEW ESTIMATION

This method is based on findings from [2]. It can be used, to determine, whether a set of IP addresses correspond to virtual hosts, e.g., as part of a virtual honeynet. It exploits microscopic deviations in device hardware: clock skews. I am using a TCP timestamps-based fingerprinting technique. A measurer can be any device capable of observing TCP packets from a tested device, assuming that those packets have a TCP timestamps option enabled. When the measurer has obtained a trace of TCP packets from the device, he can estimate a TCP timestamps option clock (TSopt clock) skew. By comparing estimated TSopt clock skews of tested devices, one can determine, if it is virtual honeynet or regular network.

In my experiments, I am using an active approach. That is, the application initiates a TCP flow with the device, assuming that the device is reachable and has an open port. But this technique should work with a passive approach as well. That is, when the measurer passively intercepts packets from the device. Whether they are destined to him, or he is only somewhere in the middle.

A simplified active approach, that I used in my experiments with the application, is in figure 2. Firstly, the application finds opened TCP port(s) with `nmap`. Secondly, it starts capturing TCP packets from the device with `tcpdump` and opening and closing TCP connections on found port(s). This is done in random intervals of specified length (e.g., 0–120 seconds). After a specified time, the application stops connecting and capturing TCP packets. Then it extracts time when the measurer received a packet and a TCP timestamp option TSval for each packet. Finally, it tries to estimate the clock skew for the device. A clock skew estimation uses a linear programming solution from [2] using `PuLP` [3].

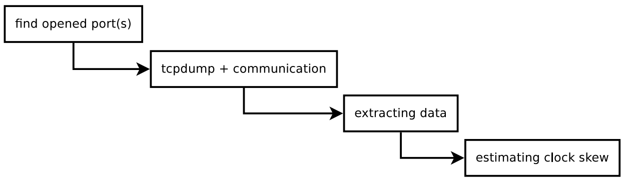

Figure 2: Clock skew estimation simplified

In most of my experiments, I used a PC with 3.3GHz Intel Core i5-2500K CPU, openSUSE 11.4 OS and `ntpd` enabled. For honeynet installation, I used a notebook with 2.10GHz Intel Core 2 Duo T8100 CPU. On this notebook, openSUSE 11.4 had `ntpd` enabled. Data for some experiments, that I did, are in table 2. A first group is for openSUSE 11.4 as a host operating system and 3 guest operating systems installed on the host using a Virtualbox. Maximum difference between estimated clock skews was 305 nanoseconds. In a second group is Windows 7 Professional as a host operating system and 3 guest operating systems under a VMware Player. Maximum difference between estimated clock skews for a second group was 310 nanoseconds. Maximum differences for both groups was too small, to be from different physical devices. It is highly improbable, that a group of physical devices would correlate so nicely. Those experiments suggest, that if the measurer is able to capture packets from the measured device (with above premises) for long enough period of time, he can successfully differentiate a virtual honeynet from a regular network using this application. The application also supports ICMP-based technique [2] for clock skew estimation.

Operating system	Hardware	TSopt clock [Hz]	Skew est. [ns]
openSUSE 11.4 (Linux kernel 3.2.7-10)	real	1000	-65.1
Debian 6.0.4 (Linux kernel 2.6.32-5)	VirtualBox	250	-132.4
Mint 12 (Linux kernel 3.0.0-12)	VirtualBox	250	-168.7
Windows XP Professional SP3	VirtualBox	10	136.2
Windows 7 Professional	real	100	-10029.2
Debian 6.0.4 (Linux kernel 2.6.32-5)	VMware	250	-10016.7
Debian 6.0.4 (Linux kernel 2.6.32-5)	VMware	250	-10014.0
Windows XP Professional SP3	VMware	10	-10323.6

Table 2: TCP timestamps option clock skew estimations for openSUSE 11.4 with VirtualBox and Windows 7 Professional with VMware Player as virtual honeynets.

4 CONCLUSION

Both techniques performed well in the application. But there must be done more testing. Especially for the TCP/IP fingerprinting. It should be tested in the Internet environment as well. In future work, I will add functionality for a service exercising technique, for detecting low-interaction honeypots.

5 ACKNOWLEDGEMENT

This article was created within the research plan of the Security-Oriented Research in Information (MSM0021630528). This work was supported by the Operational Programme Research and Development for Innovations under the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the project Advanced Secured, Reliable and Adaptive IT (FIT-S-11-1).

REFERENCES

[1] Chang, C. C.; Lin, C. J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, volume 2, 2011: p. 27:1–27:27, software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] Kohno, T.; Broido, A.; Claffy, K. C.: Remote Physical Device Fingerprinting. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2339-0, p. 211–225.

[3] Mitchell, S.; Roy, J.: pulp-or - puLP: An LP modeler in Python - Google Project Hosting. 2011 (accessed March 4, 2012), `http://code.google.com/p/pulp-or/`.

[4] Mukkamala, S.; Yendrapalli, K.; Basnet, R.; etal.: Detection of Virtual Environments and Low Interaction Honeypots. In *Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC*, june 2007, p. 92 –98.