International Journal of Electrical and Computer Engineering (IJECE)
Vol. 4, No. 6, December 2014, pp. 848~857
ISSN: 2088-8708

A New Procedure to Detect Low Interaction Honeypots

Eleazar Aguirre-Anaya[1], Gina Gallegos-Garcia[2], Nicolás Solano Luna[3], Luis Alfonso Villa Vargas[4]
[1, 4]Center for Research in Computing
[2, 3]Department of Research and Graduate Studies, Electrical and Mechanical Engineering School
Instituto Politécnico Nacional, Mexico City, Mexico

Article Info

Article history:

Received Sep 21, 2014
Revised Nov 14, 2014
Accepted Nov 22, 2014

Keyword:

Fingerprint
Honeypot Systems
Low interaction
Remote Network Systems
Signatures

ABSTRACT

Honeypots systems are an important piece of the network security infrastructure and can be deployed to accomplish different purposes such as: network sensing, capturing and learning about 0-day exploits, capturing and analyzing of black hat techniques, deterring black hats and data gathering for doing statistical analysis over the Internet traffic, among others. Nevertheless, all honeypots need to look like real systems, due to if a honeypot is unmasked, it loses its value. This paper presents a new procedure to detect low interaction honeypots, through HTTP request, regardless honeypot architecture. It is important to mention that Low Interaction Honeypots network services need to be improved in order to get trustworthy information. Otherwise, it should consider data obtained by low interaction honeypots like inaccurate and unreliable information.

Corresponding Author:

Gina Gallegos-Garcia,
Department of Research and Graduate Studies,
Electrical and Mechanical Engineering School – Instituto Politécnico Nacional.
Av. Sta Ana 1000. Sn. Fco. Culhuacan. Coyoacán. 04430. Mexico City, Mexico.
Email: ggallegosg@ipn.mx

1. INTRODUCTION

Nowadays, honeypots systems are important components in the organization's whole security infrastructure. They can be used to help sense and mitigate security events.

In [1], the author gives the *de facto* definition: 'A honeypot is a security resource whose value lies on being probed, attacked and compromised'. However, if a honeypot is detected, it loses all its value. In other words, if honeypots were susceptible to be detected, the Black hat Community could post a list of known honeypots systems letting others black hats avoid those systems and focus on real systems.

Honeypot systems are used to research over malware propagation and new intrusion techniques used by black hats. They can give the possibility to detect and analyze 0-day exploits or to obtain information related to malware such as: propagation methods or even their source code. Moreover, a honeypot could act like an alarm system because any received connection, from a host inside organizational network, is an unequivocal indication that information security mechanisms have been evaded or there is an insider attacker. This information could be used to design contention methods against malware, to improve network security mechanism, to define new security policies or change some of them. Additionally to that, the managers could take better IT decisions to search about security infrastructure or to deploy new IT services for clients and partners of each organization. However, it is an important task to keep honeypot systems unidentified in order to collect information from the network and reach its goals.

Nowadays, honeypot's remote detection is not an easy task because the detection of uncommon environments depends on the black hat's skills. In example, detecting a decrease in the speed of the returning packets over the network, a limited amount of commands in the service or the operating system, limited

amount of libraries and restricted access to memory or file system areas. Doing this detection implies the interaction between the black hat and honeypot system for a while.

Honeypot systems are able to efficiently emulate a TCP/IP stack and they also can simulate being another Operating System over the network. Besides, honeypot systems are usually deployed behind a NAT capable device and only the services offered by honeypot system can be reached from outside networks. Some techniques for fingerprinting a TCP/IP stacks have been proposed, but they were evaded easily without doing a lot of changes in honeypot code.

Spitzner said that in order to avoid fingerprinting, realism must be developed, blend it with the environment and modify honeypot behavior [1]. However, in case of low interaction honeypots, increasing the realism means to program better network service emulators with more features, and as a consequence, to increase the interaction offered by them.

The reminder of the paper is organized as follows: In Section 2 the background is divided into honeypots and fingerprinting. In Section 3 we detail different schemes used for deploying low interaction honeypots and the different approaches of fingerprinting a remote network system. In Section 4 we detail our proposed solution. Section 5 shows obtained results after testing different Low Interaction Honeypots. Considering our results, Section 6 describes a discussion. In Section 7, Conclusions and Future Work are given. Finally we list References.

2. BACKGROUND

2.1. Honeypots

Honeypots can be classified by their function in: research or produce honeypots and also in low and high interaction honeypots, by commands, libraries and applications they offer. A Honeynet is a special network composes by many systems and a honeypot gateway.

Production honeypots are deployed in organizations with the purpose of giving a set of systems to the black hats, where they can waste their time and computational resources (processor time, memory, network time and bandwidth, among others.), by maintaining the production systems in safe. Usually, for production honeypots, information is fabricated and put it inside the system, in order to confuse the black hats. Examples of such data fabrication are: the creation of fake user's accounts, documents and directories, access to the system, connections to other systems and system logs. We should pay attention to the time of data fabrication and create consistent data; otherwise, a black hat could identify fabricated data, for example, a directory inside the user home with wrong permissions and created information without a previous access of the user, among others. The main functions of this kind of honeypots are to defend the organization by causing deception to black hats. The production honeypots usually are installed over a hardware and software similar to the production servers in organizations. They can be installed over virtual environments too.

Research honeypots are mainly found at the universities and their purpose is to learn more about black hats techniques by offering many systems in a wide variety of configurations. Due to many universities cannot afford new and dedicated computers, honeypots are usually installed on virtual environments or in old hardware computers. Commonly, research honeypots are part of a big deployed Honeynet in different campus and universities.

High interaction honeypots are out of the scope of this paper because they are installed over a real operating system and the services they offer are not emulators. However, it is important to say that high interaction honeypots are deployed over virtual environments and the gathering of information is done over the virtual layer, so, the operating system does not need any modification [2]. On the other hand, low interaction honeypots offer a wide variety of systems and services emulators to black hats, malicious users or malicious software, known as malware. The main advantages of deploying low interaction honeypots are the wide area they can cover, the low risk they represent and the vast variety of services they can emulate. In addition to that, low interaction honeypots could be fingerprinted because they use emulators and have less functionality than real systems with real network services and the interaction they offer is limited.

In [3], authors show some features that only low interaction honeypots have. Most of them can be emulated, in example, by sending pseudorandom traffic to honeypots in order to increase reality or by emulating a few systems to avoid over-heading software. The other features are optional, extensive logging can be covered with the use of a Gateway Honeynet and bandwidth restriction is desirable but also optional and is specified by each organization. Only one feature is inherent of low interaction honeypot and it cannot be changed, they do not implement a full-featured network services set.

2.2. Fingerprinting

As biometric fingerprint, where a specific pattern is extracted and compared against a database, the identification of systems is possible due to the different implementations of communication protocols, network services or specific environments. These different features are collected and then a fingerprint is generated, which include enough features to unequivocally identify a specific system of a set of different systems. Some features that are used to identify systems are specific responses to malformed queries, mistakes in the implementations like misspelled words in error messages, typical behavior like special characters for paths or a specific set of addresses, initial counters or identification numbers, error messages in different anomalous queries, time response or an amount of resources, such as: amount of connections and child processes. In order to the fingerprinting tool loses accuracy, such features can be changed, but some of them are very difficult to change and need a high level programming.

As a general rule, a honeypot should not be detected. But if it is identified, it loses all its value. For production honeypots, the black hats could change their target system and attack a production system. As a consequence, they would be able to obtain valuable information. Moreover, in the case of research honeypots, the possibility to learn about black hat community becomes impossible. To prevent and hinder this possibility, the good practices indicate changes to the default settings. However, the architecture of low interaction honeypots makes this task more difficult. In other words, it is not customizable.

There are two ways to avoid fingerprinting: scrubbing and camouflaging. The first one is the modification of the output in a communication, where the fingerprinting tool cannot determine the identity of the target system. The second one refers to the modification of different expected outputs of other implementations of the protocol, which gives as a result an exact wrong match in the fingerprinting tool. Nevertheless, if the fingerprint sequence is large, the camouflage could be almost as expensive as the redeployment of a different protocol implementation. Definitions of on-line and off-line defenses against the fingerprinting and their features can be found in [4].

In addition, in [4] authors propose minimum set of tests for Nmap, in order to fingerprint an OS without the use of malformed packets, as a consequence a low probability of being a Network Intrusion Detection System (NIDS) is obtained. Due the application fingerprinting uses complete handshake connections, the probability of sensing it or blocking it is low.

3. RELATED WORK

3.1. Schemes for Deploying Low Interaction Honeypots

In this section, four schemes for deploying low interaction honeypots sensors are described, the first one is the simplest scheme to configure and maintain them and the fourth one is the most complex scheme. They are described as follows and showed in Figure 1.

The first scheme is the installation and configuration of a low interaction honeypot. Then it is necessary to assign a public IP and connect it to the Internet. This scheme is commonly used to sense and analyzepropagation methods of worms and Internet traffic statistics.

The second scheme includes also the installation and configuration of a packet filtering firewall. Its main function is to redirect specific network traffic to the honeypot. In order to redirect such traffic, the packet filtering firewall analyses network traffic and checks it against a rule set. If features match with a rule, the firewall redirects such packets to the honeypot. Typical rules are: filtering source and destination IP, destination port and flags in the TCP packet header.

The third scheme is similar to the second one with the difference that it includes a network traffic normalizer in the packet filtering firewall, commonly called scrubber. Examples of scrubbers are: the BSD's IP-filter with the enabled option scrub and the use of IP personality or a similar module for IP-Tables in GNU/Linux. Moreover, in order to restrict ingoing and outgoing network traffic, this scheme can also include a honeypot Gateway such as a Honey-wall.

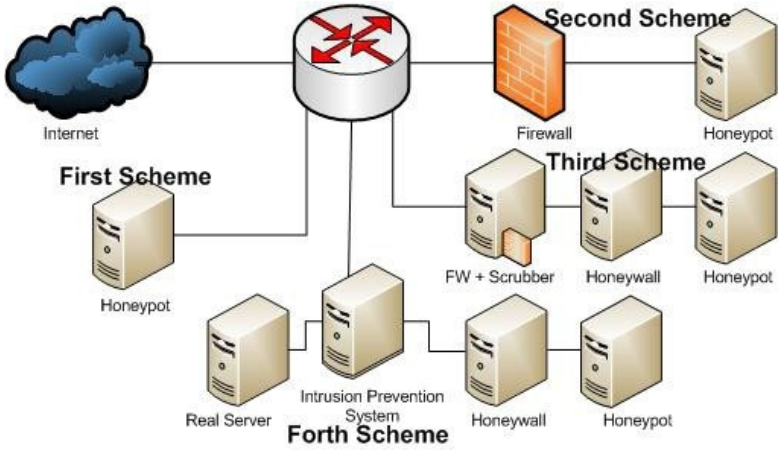

Figure 1. The Four Schemes for Deploying Low Interaction Honeypot.

After a known attack has been identified, the fourth scheme is designed to redirect network traffic. This scheme includes an IPS that senses and redirects network traffic to the honey-net. Such redirection is usually enabled for a previously specified time. This scheme can use a honey-net Gateway too and should be the preferred scheme for production of honeypots.

3.2. Fingerprinting a Remote Network System

There are two methods to remotely identify a system in a network, the passive and the active. The first one uses a network sniffer and then analyses all network traffic passing by the NIC. After that and considering a database, it tries to identify the system. This kind of remote identification will not be considered in this paper, because we do not consider passive identification due to there is no exist an interaction between honeypot and black hat, before such black hat take control of a honeypot. The second one sends specific request over the network and then analyses the responses. After that, it determinates the identity of remote system by comparing it against a fingerprint database. Different approaches to fingerprinting a remote system are explained in the following subsections.

3.2.1. Interactive Fingerprinting

This approach uses a well-known request and as a consequence it is very easy to detect and evade. Actually low interaction honeypots have the same fingerprint and it gives the exact response to the fingerprinting tool. These tools have a module to identify network services and are based on offered banners by remote services.

This approach is useless for all schemes even Honeyd due to itself is able to fool the fingerprinting tool by representing the personality of a honeypot according to the Nmap or Xprobe2 fingerprinting files and by responding the expected values for such tools [2]. In [3], an analysis of time technique to detect low interaction honeypots with good results in local area was proposed, which required sending a lot of packets and was very dependent of network topology.

3.2.2. Statistical Fingerprinting

This approach sends many requests and then applies a statistical analysis over received replies in order to identify the remote system. This approach is very sensitive to changes in network topology and only can be successful in the first scheme in the TCP/IP stack. In the second scheme could be when it is used in network services fingerprinting. In [3], 49 quantitative and qualitative features to fingerprinting TCP/IP stack were proposed. In such proposal the analysis of time is done over the response of ICMP messages. They also demonstrated honeypot systems respond slower than real systems.

3.2.3. The Fuzzy Approach

The use of fuzzy logic, as in other scenarios [5] [6] gives different advantages.In a honeypots detection processgives the advantage to identify the kind of honeypot is being used. In other words, from a set of possibilities, it is assigned a membership grade, identifying in this way, the major of them. All of this is made in order to get such advantage. Based on the TCP/IP stack, the fingerprinting procedure can be evaded in all schemes, but it is a useful identification mechanism. The main problem of this approach is the

definition of the membership functions for the fuzzy system, which depends on the amount of features to be evaluated. The Xprobe2 tool employs this approachwith the use of ICMP tests. In [7], a detailed description of Xprobe2 and its composition are given.

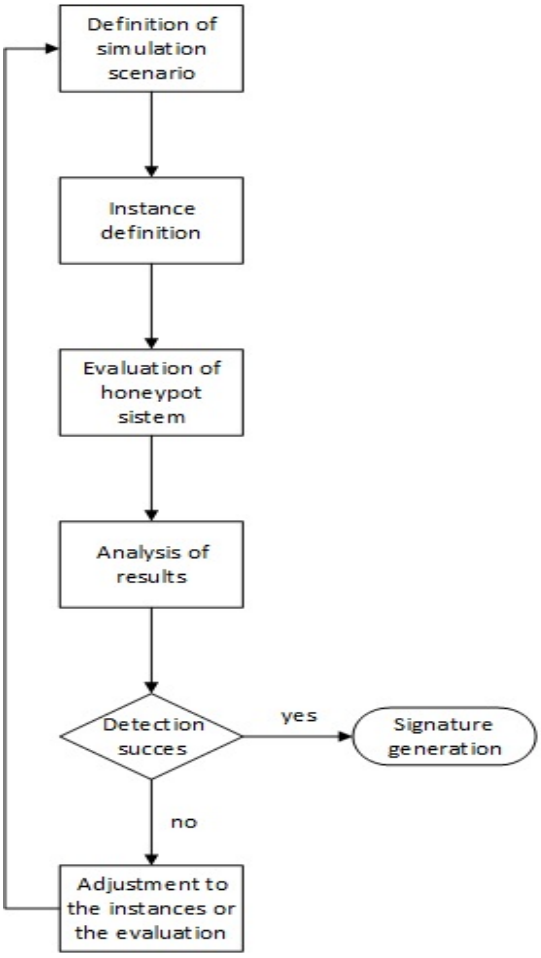

Figure 2. Proposed procedure for generation of low-interaction honeypot fingerprint.

3.2.4. The Network Service Approach

This approach focuses on fingerprinting network services. It covers a small amount of computers because it can only be used against systems that are offering the service. Even this approach seems to be the most limited, it is the best option to fingerprint low interaction honeypots. The reason to use this approach over the other ones is that network services in a honeypot are emulators and they are limited to only respond a small amount of requests. This approach usually employs fuzzy logic in a hierarchy way. An HTTP fingerprinting is preceded by a TCP/IP fingerprinting prove even though Nmap has become less effective now [4]. In [3] authors show results with different features in real services, moreover Honeyd services were given. However, it did not explain how the services are test.

There exist modules to fingerprinting HTTP Server available to Nmap and Xprobe2. This last one has also a module to test HTTP in spite of it is only used to help the identification of the Operating System. In addition to that, there exist many implementations of Honeyd that use different scripts to impersonate as HTTP services. There are other Honeyd scripts services, such as: FTP and Telnet. However, nowadays they have been replaced for newer protocols such a SSH.

4. SOLUTION PROPOSED

4.1. The Procedure

The procedure we use is divided into six stages as can be seen in Figure 2. The first step determines the simulation scenario for generating fingerprints. The second one defines instances that we used in the identification process. The third stage carries out a process between the system evaluation and Honeypot. Once this process is completed, we analyzed obtained results. After that, if such results allow identifying the Honeypot with a percentage of acceptable effectiveness; we proceed to generate the Honeypot fingerprint. Otherwise, we adjust the instances or/and the evaluation process. This stage is repeated until an acceptable percentage of identification is obtained.

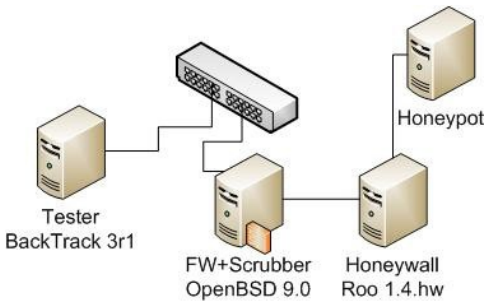

Figure 3. Our testing scenario considers four main elements

Table 1. Specifications of systems used

System	Processor	RAM	Hard Disk	Software
Honeypot	2 x 2.24GHz	2 GB	80 GB	Honeyd
Protection	Pentium 4 2.4GHz	1 GB	1x80GB	OpenBSD 3.8
Honeynet Gateway	Pentium 4 2.4GHz	1 GB	1x40GB	HoneyWall roo-1.4
Referee	Intel Core 2 Duo 1.5GHz	4 GB	100GB	BackTrack 4.2

4.2 Definition of Our Simulation Scenario

The Low Interaction Honeypot systems have a limited range of messages that can respond, as well as the amount of services that emulate.

One of the Low Interaction Honeypots has the greatest number of HTTP services emulators and message is ¨Honeyd¨. Considering that and with the intention to use a simulation scenario as close as the production, we proposed to use a topology consisting of: one Packet Firewall also called Honeywall and one HoneyPot. The Honeywall is a system that captures all requests made in the evaluation process and the Low Interaction Honepot.

Figure 3 shows mentioned scenario and Table 1 summarizes their characteristics. In such Table it is important to mention that we did use a Honeyd for installing Low Interaction Honeypot, moreover we did install Web.sh as network emulators, Apache script 1.3.23, IIS Microsoft emulator IIS/5.0 and y Honeyweb 0.4 that emulates versions of HTTP implementation.

Honeypot System

In the Honeypot we installed the GNU/Linux Debian Operating System, version 6.0 and kernel version 2.6.32 with minimal installation of the system. We updated obsolete programs, and then we downloaded programs for this profile (Honeyd 1.5c version, Honeyweb-0.4, additional scripts for SUSE and Apache Web Server version 2.2.18). Finally we created virtual systems that have an associated IP address and emulators script of HTTP.

Protection System

In the protector we installed the OpenBSD Operating System version 4.8, we configured the IPfilter and the scrub function was enabled.

Honeynet Gateway System

In this system we installed and set up the default HoneyWall root-20090425114542-1.4.hw. We defined two network cards in bridge mode and a third one for administration. We also defined network services offered by the administration interface. Walleye GUI was enabled.

Table 2. Comparison of status codes between Honeyd emulators and Apache server.

Status Code	Httpd Total Codes	Web. Sh	Apache.Sh	Lis.Sh	Honey Web 0.4
Successful (2xx)	8	1	1	1	1
Redirection (3xx)	7	0	0	1	0
Client Error (4xx)	22	1	1	1	5
Server Error (5xx)	9	0	0	0	2

Tester System

In our tester, Backtrack System Version 4.2 was installed and the signatures of the HTTPrint Version 0.301 were updated.For all computer equipment we made a minimum installation of the system and also for configuration of Sebek source code (data capture tool, which captures the activities of attackers on a Honeypot).

4.3 Definition of Instances

During the definition of instances stage we made an evaluation through service exercising techniques to identify emulators of network service of Low Interaction Honeypot System.This kind of evaluation involves a remote tester system that unknowns the architecture and the remote system to be evaluated, which is known as black box evaluation. Therefore the selection of instances contemplates the state codes to the answers given by HoneyPot.

The status codes of HTTP protocol are established in [8] and [9] and they are divided into the following families:

- Informative 1xx indicates a provisional response and is only sent to clients in terms of experimentation.
- Success 2xx indicates that the client request was received, understood and accepted.
- Redirection 3xx refers to further action and is required by the user agent to complete the request.
- 4xx is related to client error that happens when request issued by the client has error.
- Server Error 5xx occurs where the server is unable to perform the request.

As part of definition of instances we compared the states codes found in the source code of different emulators of HTTP to Honeyd against those one found in the implementation of Apache Web Server 2.2.18, the result of such comparison is shown in Table 2.

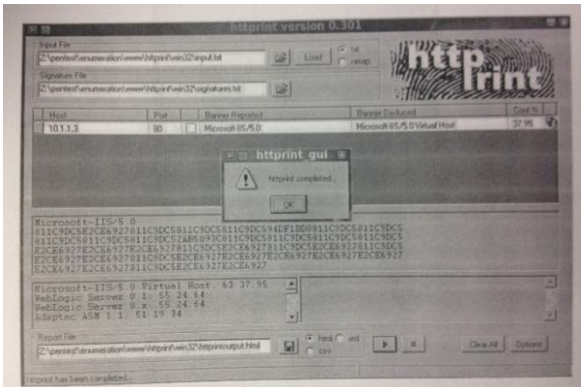

Figure 4. The identification process that is made by the HTTPrint is fooled by Honeyd

4.4 Evaluation of Low Interaction Honeypots

As we mentioned before, our simulation scenario depicted in Figure 3, has a remote computer that is responsible for the evaluation and the interaction with architecture of Low-Interaction Honeypot. In addition to that we propose requests injection of type HTTP, which are made by tester system. Such requests are made with the aim that Honeypot answers them according to the characteristics of each emulator has.

In the tester system we activate the Whireshark tool to capture the network traffic that is exchanged between the tester system and the Honeyd architecture. In addition to that we did run HTTPrint tool with the intention of observing what remote system was identified during evaluation.As is possible to see in Figure 4, fingerprinting identification process that is made by the HTTPrint, is fooled by Honeyd because identifying emulators like they were HTTP servers. After that, we did proceed to analyze information of stored flows and requests made by the tester system.

5. RESULTS

The first parameter we consider in our analysis is the number of status codes implemented in Honeyd HTTP scripts. As we did mention before they are defined [8-10] and as a result of pcap files analysis, we observe that HTTPrint tool makes 23 requests to define the kind of remote system to be identified.

The analysis of the number of status codes implemented in Honeyd HTTP scripts and in a real HTTP server shows us a significant difference between them. Table 2 summarizes the number of status codes offered by Apache httpd 2.2.18 (HTTPD), by the Honeyd scripts web.sh (WEB), by apache.sh (APACHE), by iis.sh (IIS) from Honeyd scripts for SUSE and Windows and finally HoneyWeb 0.4 (HWEB). As is shown in Figure 5, after execution of HTTPrint against Honeyd scripts, the new signatures were obtained. Such signatures were written in the signatures.txt file with the name of correspondent emulator. Then we repeat previous tests and for all of them we did get a 100% of identification.

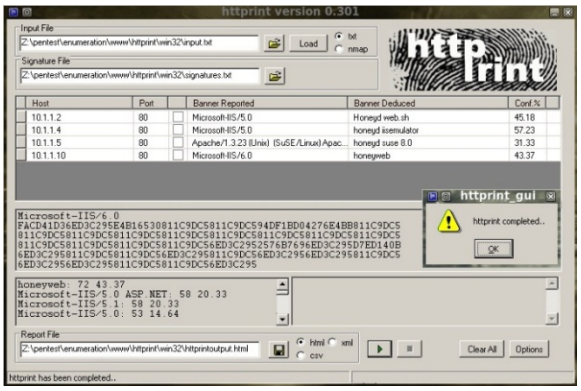

Figure 5. Successful detection of Honeyd HTTP scripts after re-running HTTPrint tool.

```
## 25/04/2011
## contributed by Nicolas Solano: nsolano0900@ipn.mx
Honeyd web.sh
811C9DC5E2CE6927811C9DC5811C9DC5811C9DC594DF1BD0811C9DC5811C9DC5
811C9DC5811C9DC5811C9DC52AB5093C811C9DC5811C9DC5811C9DC5811C9DC5
E2CE6927E2CE6927E2CE6927811C9DC5E2CE6927811C9DC5E2CE6927811C9DC5
E2CE6927E2CE6927811C9DC5E2CE6927E2CE6927E2CE6927E2CE6927E2CE6927
E2CE6927E2CE6927811C9DC5E2CE6927E2CE6927

## 25/04/2011
## contributed by Nicolas Solano: nsolano0900@ipn.mx
honeyweb
FACD41D36ED3C295E4B16530811C9DC5811C9DC594DF1BD04276E4BB811C9DC5
811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5
811C9DC5811C9DC5811C9DC56ED3C2952576B7696ED3C295D7ED140B
6ED3C295811C9DC5811C9DC56ED3C295811C9DC56ED3C2956ED3C295811C9DC5
6ED3C2956ED3C295811C9DC5811C9DC56ED3C295

## 26/04/2011
## contributed by Nicolas Solano: nsolano0900@ipn.mx
honeyd suse 8.0
811C9DC5811C9DC5811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5
811C9DC5811C9DC5811C9DC5E2CE6923811C9DC5811C9DC5811C9DC5
811C9DC5811C9DC5811C9DC5811C9DC56ED3C295811C9DC56ED3C295
811C9DC5811C9DC5811C9DC5E2CE6923E2CE6923

## 26/04/2011
## contributed by Nicolas Solano: nsolano0900@ipn.mx
honeyd iisemulator
811C9DC5E2CE6923811C9DC5811C9DC5811C9DC594DF1BD04276E4BB72A2BDCD
0D7645B5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5
E2CE6923E2CE6923E2CE6923811C9DC568D17AAE811C9DC5E2CE6923811C9DC5
E2CE6923E2CE6923811C9DC5E2CE6923E2CE69236ED3C295E2CE69236ED3C295
E2CE6923E2CE6923811C9DC568D17AAE68D17AAE
```

Figure 6. New obtained signatures of Honeyd HTTP scripts.

Moreover, even if HoneyWeb is able to emulate 4 versions of IIS, 8 versions of Apache and 3 versions of Netscape Enterprise, a single signature allows detecting all versions. After that, the new signatures were added to previous ones and re-run HTTPrint with all Honeyd scripts successfully detected. It is presented in Figure 6. In addition, due to the fingerprinting service is made over a valid connection and the huge amount of possible queries, this method is more difficult to be identified by an IDS. As a consequence, it is only important to make a modification in the query, such as: do the request GET / HTTP/1.1, increase

the field of the version or subversion or change a single character in the GET /antidisestablishmentarianism HTTP/1.0 query. In Table 3 the replies to HTTPrint are presented.

6. DISCUSSION

From obtained results we can state the following important points:

- The HTTP emulators for Honeyd answer very differently to HTTPrint requests.
- Web.sh emulators, apache.sh and honeyweb.sh do not send any answer to requests Htprint.
- The apache.sh emulators, iis.sh and honeyweb send status codes of success when they should send client error codes.
- The web.sh and web.sh emulators send status codes of error when they should send status codes of server error.
- The difference between HTTP emulators for Honeyd and a real HTTP server is very strong due to these latter implements, in a more complex way, status codes in the protocol standard and not just a portion.
- Being the same HTTP emulators for Honeyd, the same answer for HTTPrint requests is presented, regardless of the distribution of the Operating System or their respective architecture.

Table 3. Results of the analysis of stored flow.

HTTPRINT QUERY	HTTPD	WEB	APACHE	IIS	HWEB
garbage	501	-	-	400	-
GET / HTTP/1.0	200	404	200	200	200
GET / HTTP/1.0 (*)	200	404	200	200	200
*OPTIONS * HTTP/1.0*	200	404	501	400	200
OPTIONS / HTTP/1.0	200	404	501	400	200
GET /antidisestablishmentarianism HTTP/1.0	404	404	400	302	200
PUT / HTTP/1.0	405	404	501	400	-
JUNKMETHOD / HTTP/1.0	501	404	501	400	-
GET JUNK /1.0	200	404	501	400	-
get / http/1.0	501	404	501	400	-
POST / HTTP/1.0	200	404	501	400	200
GET /cgi-bin/ HTTP/1.0	403	404	400	302	200
GET/scripts/ HTTP/1.0	404	404	400	302	200
GET / HTTP/0.8	200	404	501	400	200
GET / HTTP/0.9	200	404	501	400	-
GET / HTTP/1.1 Connection: close	200	404	200	200	200
GET / HTTP/1.2 Connection: close	200	404	501	400	200
GET / HTTP/1.1 (**)	400	404	200	200	200
GET / HTTP/1.2 (**)	400	404	501	400	200
GET / HTTP/3.0	200	404	501	400	200
GET /. asmx HTTP/1.0	404	404	400	302	-
GET /../../ HTTP/1.0	400	404	400	302	200

7. CONCLUSIONS AND FUTURE WORK

The fingerprinting of TCP/IP stack is useless against common low interaction honeypots due to the amount of characteristics and limited responses of specific request. Nevertheless, low interaction honeypots are susceptible to fingerprint of network services. It is because of the differences between a real service and emulator scripts.

Fingerprinting of network services is successful because of the amount of available options in the construction of queries, making the fingerprinting tools hard to be detected. Moreover, the attack could be made in a long period of time, reducing in this way, a possible identification of the tool. In addition it is very concerning that there are tools ready to detect low interaction honeypots such as: HTTPrintthat is a way to generate new fingerprints and add them to signatures files. In addition to that, the use of fuzzy logic in fingerprinting tools increases success rate of identification. Moreover, nowadays, as we have found, low interaction honeypots need to be improved in order to ensure their correct functionality. Otherwise, they should not be deployed as research honeypots. Unfortunately, recent activity in some low interaction honeypots sensors projects is null.

As future work, a research on detecting high interaction honeypots will be done, in addition to the creation of a fingerprinting tool of low interaction honeypots in order to be used in different network services. Finally, it is important to mention that another challenge would be to design a fingerprinting tool that is able to identity what kind of honeypot scheme is being used.

REFERENCES

[1] Spitzner L. *The Honeynet Project: Trapping the Hackers*. IEEE Security and Privacy, Vol 1, No. 2. 2003. Pp: 15-23.
[2] Provos N, Holz T. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. 1st Edition, Addison Wesley Professional.2007.Pp: 19-69.
[3] Mukkamala S, Yendrapalli K, Basnet R, Shankarapani M.K. Sung A. H.*Detection of Virtual Environments and Low Interaction Honeypots*.In *Proc*. IEEE Workshop on Information Assurance and Security.2007. Pp: 92-98.
[4] Greenwald Lloyd G, Thomas Tavaris J. *Toward Undetected Operating System Fingerprinting*. In *Proc*. First USENIX Worshop of Offensive Technologies.2006. Article No. 6.
[5] Godbole Vaibhav. *Performance Analysis of Clustering Protocol Using Fuzzy Logic for Wireless Sensor Network.* International Journal of Artificial Intelligence.Vol. 1. No. 3. 2012. Pp: 103-111.
[6] Hamzah Mustafa I, Abdall Turki Y. *Mobile Robot Navigation using Fuzzy logic and Wavelet Network.* International Journal of Robotics and Automation.Vol. 3. No. 3. 2014. Pp: 191-200.
[7] Yarochkin F.V, Arkin O, Kydyraliev M, Shih-Yao D. *Xprobe2++: Low Volume Remote Network Information Gathering*. In *Proc*. IEEE/IFIP International Conference on Dependable Systems & Networks.2009.Pp: 205-210.
[8] Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T. *Hypertext transfer Protocol -- HTTP/1.1. RFC 2616*, 1999.Pp: 39-41.
[9] Khare R, Lawrence S, *Upgrading to TLS Within HTTP/1.1. RFC 2817.*2000. Pp: 8.
[10] Nielsen H, Leach P, Lawrence S. *An HTTP Extension Framework. RFC 2774.*2000. Pp: 8-13.

BIOGRAPHIES OF AUTHORS

Eleazar Aguirre-Anayaholds a Ph.D. degree on Communications and Electronics. He is professor at the Center for Research in Computing the National Polytechnic Institute of Mexico. He has been involved as information security specialist in consulting projects for public and private organizations; Aguirre-Anaya also has published papers on research journals and conferences and has served as thesis advisor for several graduate students on information security topics. His main research topics are network security, honeynets and secure infrastructures.

Gina Gallegos-Garciareceived a MS Degree and Ph. D from the National Polytechnic Institute of Mexico in 2005 and 2011 respectively. She is currently Professor of Graduated Section of Mechanical and Electrical Engineering School and belongs to the National System of Researchers. During the summer of 2011 she performed a postdoctoral research at Yale University in the United States of America. Her areas of interest include The Electronic Voting, the Secure Cryptographic Application Design, Information Systems and Cryptography, Software Engineering.

Nicolas Solano Luna received a ME Degree from the National Polytechnic Institute of Mexico in 2012. He is currently working in the Federal Economical Competence Commission as a computer forensics investigator. His areas of interest include network security, computer forensics and mobile devices security.

Luis Alfonso Villa Vargasholds a Ph.D. degree on Informatics. He is the principal at the Center for Research in Computing the National Polytechnic Institute of Mexico. He has been involved as information security specialist in consulting projects for public and private organizations.Villa Vargas also has published papers on research journals and conferences. He performed a postdoctoral research at the Massachusetts Institute of Technology (MIT).