HONEYD DETECTION VIA ABNORMAL BEHAVIORS GENERATED BY THE ARPD DAEMON

A. Boulaiche
ReSyd Doctoral School, University of Bejaia, Algeria
K. Adi
Computer Security Research Laboratory, University of Qubec in Outaouais, Canada

Keywords: Security, Honeypot detection, Abnormal behaviours, ARP protocol.

Abstract: In this paper we describe some serious flaws in the software Honeyd that is one of the most popular software of honeypots, these flaws allow an attacker to easily identify the presence and the scope of a deployed honeypot. Hence, we describe in details both the flaws and how they can be used to attack the honeypot. Furthermore, we elaborate a set of possible solutions to fix each of these flaws. Our technique is mainly based on the detection of abnormal behaviors of the honeypot.

1 INTRODUCTION AND RELATED WORK

Honeypots (Spitzner, 2002) have recently received a lot of attention in the research community. They can be used for several purposes; they can provide early warning about new attacks, exploitation trends, attack tools, and evolving attack techniques. Honeypots are more and more deployed within computer networks. However,malicious attackers start to devise techniques for detecting and circumscribing these security tools (Holz and Raynal, 2005), while the honeypot deployments must remain hidden in order to be useful. An attacker who is aware of the location of a monitor can avoid it, mislead it by feeding it erroneous information, or even target the monitor itself. Honeyd (Provos, 2002; Provos, 2003; Provos, 2005) is one of the most popular solutions proposed so far for honeypot technology. It is able to emulate the behavior of most common IP-stack implementations. During its normal operation Honeyd listens on the network interface card (NIC) for incoming ARP requests. If an ARP request is detected, it initiates an ARP request itself. If no response to the own ARP request is given and a rule for the requested IP exists in the configuration file, Honeyd overtakes the IP and starts pre-configured services on the specified ports.

Like any honeypot solution, the stealhtiness of a deployed Honeyd is very important in order for the later to work properly and to avoid exposing itself to specific attacks. Indeed, honeyd can carry risks to a network, and must be handled with care to avoid an attacker to use them to break into a system. Unfortunately, this stealthiness is not guaranteed as we will show in this work. Hence, one of the most effective methods to detect the honeypot is to analyze the content of various generated IP-packets and to compare them afterwards with the content of the same types of packets generated by a normal machine. This comparison allows us to discover some abnormal behaviors of the honeyd technology which leads to its detection.

Honeypots technologies have been extensively studied during the last decade and many Honeypot solutions have been proposed. For instance, the Honeynet Project (Provos, 2005) offers a height-interaction honeypot system. It is composed by a network of honeypots that are controlled by a set of tools collected in the Honeywall gateway that acts as a bridge between the Internet and the network of honeypots. The Honeynet Project is used to gather information about tools, tactics, and intents involved in computer and network attacks. Jiang and Xu (Jiang and Xu, 2004) presented a virtual honeynet system that has a distributed presence and a centralized operation. Anagnostakis *et al.* (Anagnostakis et al., 2005) proposed "Shadow Honeypot", a hybrid architecture that combines the best features of honeypots and anomaly detection. Honeypots attacks and detection techniques have also been studied. For instance, Dornseif *et al.* (Dornseif et al., 2004) have proposed an approach for attacking a honeynet locally. Krawetz (Krawetz, 2004) proposed a commercial anti-honeypot spamming tool: "Send-Safe's Honeypot Hunter". The later attempts to detect "safe" proxies for use with bulk-mailing tools. Xinwen (Fu et al., 2006) proposed a new technique that allows an attacker to remotely fingerprint Honeyd by measuring the latency of the network links emulated by Honeyd.

The reminder of this paper is structured as follows: section 2 gives an overview about Honeyd technology, section 3 describes our experimentations and the discovered flows in the honeyd technology, section 4 proposes some solutions for fixing the discovered flaws. Finally, we provide in section 5 our conclusion.

2 HONEYD OVERVIEW

Honeyd is a popular open source low interaction honeypot that runs on BSD, Linux and Solaris, although recently ported to Windows. It offers a simple way to emulate services offered by several machines on a single one. The primary purpose of Honeyd is detection, specifically to detect unauthorized activities within your organization. It does this by monitoring all the unused IP addresses in the network. Any attempt connection to an unused IP address is assumed to be unauthorized or malicious activity, so it generates an alert. Recently, honeyd has been used for many purposes. For exemple, it is used to detect,to analyze, and to respond to attacks against organization networked assets, regardless of where those threats come from. Honeyd is generally used in conjunction with another freely available utility called arpd (Song and Provos, 2003) that allows a single host to monitor multiple addresses on a network by responding to ARP requests for unclaimed IP addresses.

Honeyd's architecture consists of several components: a configuration database, a central packet dispatcher, a protocol handlers, a personality engine, and an optional routing component. The configuration database is a sequence of templates, where each one of them represents a system to be emulated. Inside every template, we find the personality of a system, its uptime, its UserID, its GroupID, the state of its ports, and the IP address associated to it. The central packet dispatcher is used to put the outgoing packets in TCP/IP packets, and analyze the received TCP/IP packets according to the associated fingerprint. The personality engine is used by Honeyd for referring to the network stack behavior of a virtual honeypot. The daemon uses the Nmap fingerprint list as a reference. The configured services are script sets. Each script emulates a service of a real system. When the Honeyd daemon receives a packet for one of the virtual honeypots, it is processed by a central packet dispatcher. The dispatcher checks the length of the IP packet and verifies its checksum and then, it queries the configuration database of a honeypot configuration that corresponds to the destination IP address. If no such configuration exists, the default template is used. Then the dispatcher calls the protocol specific handler with the received packet and the corresponding honeypot configuration. For ICMP, it answers with an ICMP ECHO reply packet. For TCP and UDP, the daemon can establish connections to arbitrary services that receive data on standard input system and send their output to standard output system. For more details, see (Provos, 2003).

3 HONEYD ANALYSIS

To make the distinction between the physical machines and the various virtual ones which are created by honeyd, we classified them in three different classes:

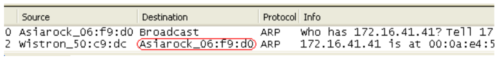
- Class 1: class of real physical machines,
- Class 2: class of virtual machines created by honeyd corresponding to templates which are different from the default one defined in the configuration file. In this class, the number of virtual machines and their IP addresses are fixed in the configuration file.
- Class 3: class of virtual machines created by honeyd corresponding to the default template defined in the configuration file. In this class, the number of virtual machines is dynamic and their addresses are based upon on the free addresses in the physical network.

Throughout this paper we will try to compare the machine behaviors of the last two classes with those of the first class, by trying to highlight some abnormal behaviors. These can lead to the detection of the honeyd technology.

In order to show the flaws of the honeyd technology that allows an attacker to easily identify the presence and scope of a deployed honeypot, we conducted the following experiment. We used Honeyd to setup three other virtual machines (honeypots) on the local area network of our university. The first honeypot is a Linux Suse 8.0, the second honeypot is a Windows XP Professional SP1 Microsoft, and the last honeypot is a Windows 2000. Each of these honeypots runs the services: HTTP, FTP, SMTP, TELNET, Netbios-ns, Netbios-dgm,netbios-ssn, Microsoft - ds. The IP addresses associated to these three machines are successively: 172.16.12.13, 172.16.12.14, and 172.16.12.15. The rest of the range addresses 172.16.12.0/24 ,which is free, is automatically allocated to the third class machines defined in the default template.

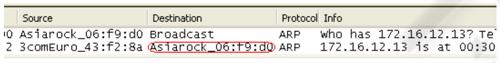
3.1 First Experimentation

In the first test, we captured the incoming and outgoing ARP traffic of our test machine by using the Ethereal (Lamping, 2004) packet sniffer, then we launched three pings commands: the first was intended towards a real machine at 172.16.41.41, the second was intended towards the machine 172.16.12.13 of the second class and the third was intended towards the machine 172.16.12.1 of the third class. During the execution of these three pings, the ARP traffic captured by Ethereal is illustrated in Figures 1, 2 and 3.



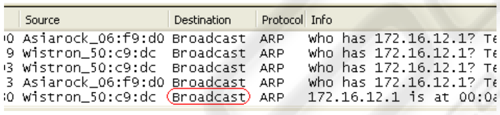
```
  Source             Destination        Protocol  Info
0 Asiarock_06:f9:d0  Broadcast          ARP       Who has 172.16.41.41? Tell 17
2 Wistron_50:c9:dc   Asiarock_06:f9:d0  ARP       172.16.41.41 is at 00:0a:e4:5
```

Figure 1: ARP traffic for the first ping.



```
   Source              Destination        Protocol  Info
'0 Asiarock_06:f9:d0   Broadcast          ARP       Who has 172.16.12.13? Te
 2 3comEuro_43:f2:8a   Asiarock_06:f9:d0  ARP       172.16.12.13 is at 00:30
```

Figure 2: ARP traffic for the second ping.



```
   Source              Destination  Protocol  Info
'0 Asiarock_06:f9:d0   Broadcast    ARP       Who has 172.16.12.1? Te
 9 Wistron_50:c9:dc    Broadcast    ARP       Who has 172.16.12.1? Te
'3 Wistron_50:c9:dc    Broadcast    ARP       Who has 172.16.12.1? Te
 3 Asiarock_06:f9:d0   Broadcast    ARP       Who has 172.16.12.1? Te
:0 Wistron_50:c9:dc    Broadcast    ARP       172.16.12.1 is at 00:0a
```

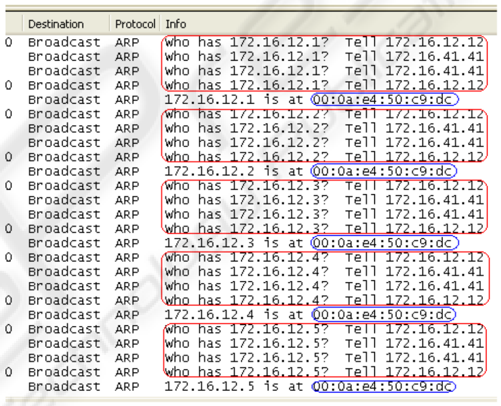
Figure 3: ARP traffic for the third ping

In the three figures, the test machine broadcast an ARP packet carrying the question ”who has $IP_ADDRESS (”who has 172.16.41.41” for the first ping, ”who has 172.16.12.13” for the second ping and ”who has 172.16.12.1” for the third ping) to obtain the MAC address of the destination machine. Then the machine that has this particular IP address answers by an ARP packet containing its MAC address. For the first class machine and the one of the second class, the answer is sent to the test machine which asked this address (see the destination field in the circle of the two first figures). On the other hand, for the third class machine (default virtual machine), the answer is broadcasted to every machine on the network (see the destination field in the circle of the last figure). This broadcast is considered as an abnormal behavior of the honeyd ARP protocol. By exploiting this flaw, an attacker who controls (locally or remotely) a machine on the same LAN as the honeyd can easily detect the presence of the honeyd and its controlled virtual machines.

To resolve this problem it is necessary to modify the source code of the arpd daemon by canceling the ARP answers broadcasting.

3.2 Second Experimentation

During the second test, we launched several ping commands towards five machines of the third class from the test machine 172.16.12.12. The ARP Traffic captured by Ethereal during the execution of these ping commands are illustrated in Figure 4.



```
  Destination  Protocol  Info
0 Broadcast    ARP       Who has 172.16.12.1?  Tell 172.16.12.12
  Broadcast    ARP       Who has 172.16.12.1?  Tell 172.16.41.41
  Broadcast    ARP       Who has 172.16.12.1?  Tell 172.16.41.41
0 Broadcast    ARP       Who has 172.16.12.1?  Tell 172.16.12.12
  Broadcast    ARP       172.16.12.1 is at 00:0a:e4:50:c9:dc
0 Broadcast    ARP       Who has 172.16.12.2?  Tell 172.16.12.12
  Broadcast    ARP       Who has 172.16.12.2?  Tell 172.16.41.41
  Broadcast    ARP       Who has 172.16.12.2?  Tell 172.16.41.41
0 Broadcast    ARP       Who has 172.16.12.2?  Tell 172.16.12.12
  Broadcast    ARP       172.16.12.2 is at 00:0a:e4:50:c9:dc
0 Broadcast    ARP       Who has 172.16.12.3?  Tell 172.16.12.12
  Broadcast    ARP       Who has 172.16.12.3?  Tell 172.16.41.41
  Broadcast    ARP       Who has 172.16.12.3?  Tell 172.16.41.41
0 Broadcast    ARP       Who has 172.16.12.3?  Tell 172.16.12.12
  Broadcast    ARP       172.16.12.3 is at 00:0a:e4:50:c9:dc
0 Broadcast    ARP       Who has 172.16.12.4?  Tell 172.16.12.12
  Broadcast    ARP       Who has 172.16.12.4?  Tell 172.16.41.41
  Broadcast    ARP       Who has 172.16.12.4?  Tell 172.16.41.41
0 Broadcast    ARP       Who has 172.16.12.4?  Tell 172.16.12.12
  Broadcast    ARP       172.16.12.4 is at 00:0a:e4:50:c9:dc
0 Broadcast    ARP       Who has 172.16.12.5?  Tell 172.16.12.12
  Broadcast    ARP       Who has 172.16.12.5?  Tell 172.16.41.41
  Broadcast    ARP       Who has 172.16.12.5?  Tell 172.16.41.41
0 Broadcast    ARP       Who has 172.16.12.5?  Tell 172.16.12.12
  Broadcast    ARP       172.16.12.5 is at 00:0a:e4:50:c9:dc
```

Figure 4: ARP traffic for the pings commands series.

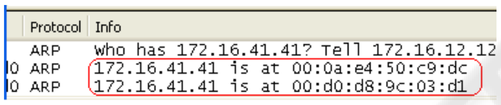
In this figure, we can notice the flaws that can lead to the detection of the presence and the scope of a honeyd deployment:

- the first discovered anomaly concerns the big framed traffic in the figure. In this traffic, we can see that every time the arpd daemon receives an ARP request for a third class machine, it broadcasts an identical request in order to determine if there is a first class machine with this IP address. If it is not the case, it answers the ARP request with the MAC address of the honeyd host. But unfortunately, this behavior can be considered as suspect by an attacker. The later can launch an ARP scan towards all network machines and can deduce addresses representing the honeyd accommodating machine.
- the second discovered anomaly concerns MAC addresses of ARP answers. As shown in Figure 4 (see the small frames), all answered addresses are identical. This comes from the default template of the configuration file concerning the Ethernet card type of the third class machines. This

means that all these various IP addresses have the same MAC address which is an impossible situation in a real system. Such situation is suspect, and offers the possibility for the attackers to identify the presence and scope of a deployed honeypot. The attacker can launch an ARP scan from a machine on the same LAN as the honeyd which he controls (locally or remotely) and extracts IP addresses having the same MAC address. These addresses represent the third class virtual machines (virtual machines of the default template) managed by honeyd.

3.3 Third Experimentation

In the third experimentation, we assigned the 172.16.41.41 address of a first class machine (real machine) to a machine of the second class created by honeyd. Then we restarted honeyd, and finally we launched a ping command toward the machine (172.16.41.41). The ARP traffic captured by Ethereal during the execution of this ping command is illustrated in Figure 5.



```
  Protocol  Info
  ARP       Who has 172.16.41.41?  Tell 172.16.12.12
0 ARP       172.16.41.41 is at 00:0a:e4:50:c9:dc
0 ARP       172.16.41.41 is at 00:d0:d8:9c:03:d1
```

Figure 5: ARP traffic for the address conflict flaw.

As shown in this figure, there is two different answers for the ARP request corresponding to the 172.16.41.41 machine (see the frame in the figure). The arpd daemon answered directly without confirming if this address is already associated to another machine, since it considers that the addresses allocated to the second class machines are manually verified by the administrator. This situation will lead to a MAC address conflict as shown in the figure. This conflict can be easily discovered by attackers sniffing the network.

4 PROPOSED SOLUTIONS

To remedy these discovered flaws, we suggest enhancing the honeyd technology by four new modules and modifying the arpd daemon to make it compatible with these new modules. Each module will be used to patch of one of these flaws.

- Module for Allocated IP Addresses,
- MAC Address Translation Module,
- Virtual MAC addresses generator,
- IP Addresses Check Module.

The new general structure of the honeyd technology after the integration of these four modules is shown in Figure 6.

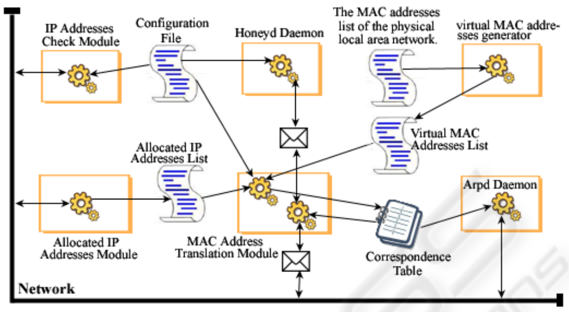

IP Addresses Check Module
Configuration File
Honeyd Daemon
The MAC addresses list of the physical local area network.
virtual MAC addresses generator
Virtual MAC Addresses List
Allocated IP Addresses List
Arpd Daemon
Allocated IP Addresses Module
MAC Address Translation Module
Correspondence Table
Network

Figure 6: New general structure of the honeyd technology.

Hereafter, we describe in details the proposed modules.

4.1 Module for Allocated IP Addresses

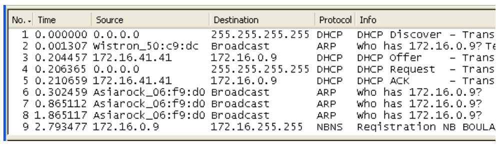
This module is used to resolve the problem of collecting information about free IP addresses belonging to the address-interval controlled by the arpd daemon. The current arpd implementation uses the technique of broadcasting the received ARP requests, but as we saw in our experimentation, this technique leads easily to the detection of the honeyd technology. Hence, to solve this problem we have to introduce a new technique allowing us to obtain the free IP addresses list without awakening attackers suspicions, either by using a passive technique, or by using active techniques without broadcasts.

Indeed, when the LAN uses a DHCP server to allocate dynamically IP addresses to the various machines of the network, the best way to obtain the list of allocated IP addresses is to get it from the DHCP server, either passively by sniffing incoming and outgoing packets toward and from this server, or actively by asking it periodically. And to ensure this operation effectively, the honeyd host machine must be installed on the same network segment as the DHCP server.

To show how it is possible to obtain the list of free IP addresses by using the DHCP packets sniffing technique, we didthe following test:

We installed a DHCP server into a Linux (RHEL4) machine, and from another Windows XP machine we launched the Ethereal sniffer, then from the Windows configuration panel we indicated that the network card must receive dynamically an IP address.

The DHCP traffic captured by Ethereal is shown in the following figure (7):



```
No. Time     Source            Destination     Protocol Info
  1 0.000000 0.0.0.0           255.255.255.255 DHCP     DHCP Discover - Trans
  2 0.001307 Wistron_50:c9:dc  Broadcast       ARP      Who has 172.16.0.9? Te
  3 0.204457 172.16.41.41      172.16.0.9      DHCP     DHCP Offer    - Trans
  4 0.206365 0.0.0.0           255.255.255.255 DHCP     DHCP Request  - Trans
  5 0.210659 172.16.41.41      172.16.0.9      DHCP     DHCP ACK      - Trans
  6 0.302459 Asiarock_06:f9:d0 Broadcast       ARP      Who has 172.16.0.9?
  7 0.865112 Asiarock_06:f9:d0 Broadcast       ARP      Who has 172.16.0.9?
  8 1.865117 Asiarock_06:f9:d0 Broadcast       ARP      Who has 172.16.0.9?
  9 2.793477 172.16.0.9        172.16.255.255  NBNS     Registration NB BOULA
```

Figure 7: DHCP traffic captured by Ethereal.

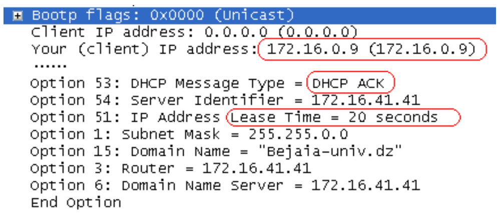
1. To obtain dynamically an IP address, the Windows machine broadcast (general broadcast ”255.255.255.255”) a ”DHCP discover” packet to research a DHCP server which will provide an IP address to it. It uses the source address 0.0.0.0 because it has had no IP address yet.
2. The DHCP sever of the Linux machine, which has the intention of offering the 172.16.0.9 IP address to this client, broadcast an ARP request on this address in order to see if it is available in the network.
3. If the DHCP server does not receive an answer to its ARP request, it offers this address to the client by sending a ”DHCP Offer” to it.
4. The Windows machine answered to the DHCP servers offer in order to confirm the taking of the IP address sent (because it can be taken by several DHCP servers which answer, so there must be a confirmation to the server selected by the client).
5. The DHCP server answers by ”DHCP ACK” to accept or acknowledge the new IP allocation.
6. 7. 8. The client (the Windows machine) do three ARP broadcasts to verify from its side that the 172.16.0.9 IP address is not duplicated on the network.

9. If he does not receive any answer for his ARP broadcasts, he registers his new address.

Indeed, the validity duration of the IP address affected by the DHCP server is limited by the duration defined in the configuration file ”/etc/dhcp.conf (in our case it is of 20 seconds). So, if the client stays connected more than this definite duration, he must renew it periodically, by sending a ”DHCP request”, in order to receive a ”DHCP ACK” from the DHCP server bearing a new duration equals to the previous.

The information list carried in the ”DHCP Offer” packets and the DHCP ACK packets are shown in the following figure (8):

What interests us in all this information captured is the agreement packets (DHCP ACK) sent by the DHCP server, and privately: the given address (the four bytes of the numbers: 16, 17, 18, and 19 of the ”DHCP ACK” packet) and the validity duration (the six bytes of the numbers: 249, 250, 251, 252, 253, 254).



```
⊞ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 172.16.0.9 (172.16.0.9)
  ......
  Option 53: DHCP Message Type = DHCP ACK
  Option 54: Server Identifier = 172.16.41.41
  Option 51: IP Address Lease Time = 20 seconds
  Option 1: Subnet Mask = 255.255.0.0
  Option 15: Domain Name = "Bejaia-univ.dz"
  Option 3: Router = 172.16.41.41
  Option 6: Domain Name Server = 172.16.41.41
  End Option
```

Figure 8: The information lease sent by the DHCP server.

So, our sniffer will be programmed to capture only the DHCP ACK packets, in order to save into a text file all allocated addresses with the validity duration of each of them.

Besides saving this information, our sniffer test periodically the content of a file to extract the list of expired addresses. In order to send them test packets to confirm if they are still active or not. So the general algorithm of our sniffer is:

```
MainProgram AllocIPAddMod
Var dhcp_server_ip : string;
    allocated_addrs_file : TextFileObject;
Begin
 BeginPARALLEL
   Sniffer(eth0, dhcp_server_ip,
   allocated_addrs_file);
 PARALLEL
   Expiry_test(allocated_addrs_file);
 EndPARALLEL
End.

Procedure Sniffer(ethi, dhcp_server_ip,
                  allocated_addrs_file)
Var traf : StringObject ;
    dhcp_pack : DHCPPacketObject ;
Begin
 While true Do
   Capture_traffic(ethi, traf);
   If traf.Src_IP_addr()==dhcp_server_ip Then
     dhcp_pack := traf.Extract_DHCP_packet();
     If dhcp_pack.Type() == "DHCP ACK" Then
         allocated_addrs_file.Insert(
         dhcp_pack.Allocated_addr(),
         dhcp_pack.Validity_duration());
     EndIf
   EndIf
 EndWhile
EndProcedure

Procedure Expiry_test(allocated_addrs_file)

Var expired_ip_list: StringListObject;
    free_expired_ip_list : StringList ;
Begin
  While true Do
    Sleep(T);
    expired_ip_list :=
    allocated_addrs_file.Expired_ip();
    free_expired_ip_list :=
    expired_ip_list.Test_free_expired_ip();
    allocated_addrs_file.Delete(
    free_expired_ip_list);
  EndWhile
EndProcedure
```

4.2 MAC Address Translation Module

This module fixes the third flaw. Its mechanism is similar to the NAT technology (Network Address Translation). This module contains a virtual MAC address list generated by the MAC address generator module. Addresses are similar to MAC address supplied by VMWare (Website1, 2008). From this list and the free address list supplied by the previous module AllocIPAddCMod as well as addresses of the configuration file ”honeyd.conf”, it generates a correspondence table where each entry corresponds to an IP address of a virtual machine managed by honeyd with the associated virtual MAC address. By using this table, the arpd daemon will answer ARP requests for virtual IP address by sending the corresponding virtual MAC address. The proxy is configured to listen to the network in promiscuous mode, and as soon as it receives a frame, it verifies its destination MAC address. If this address has an entry in the correspondence table then the later is replaced by the MAC address of the honeyd machine and the frame sent to it. The frames generated by honeyd are sent at first towards the proxy. The later will change their MAC source addresses by the virtual MAC addresses according to IP source addresses of packets carried in these frames, and sends them then toward their final destination. So the general algorithm of this module is:

```
MainProgram MACAddTransMod
Import honeyd.conf, allocated_addrs_file,
       virtual_MAC_addrs_file ;
Var corresp_table : ObjectTable; Begin
  corresp_table.Initialization(honeyd.conf,
  allocated_addrs_file,
  virtual_MAC_addrs_file);
 BeginPARALLEL
  While true Do
    Sleep(T);
    corresp_table.Update(allocated_addrs_file,
    virtual_MAC_addrs_file);
  EndWhile
 PARALLEL
  Manage_traffic(corresp_table);
 EndPARALLEL
End.

Procedure Manage_traffic(corresp_table)
Var traf : StringObject ;
Begin
  BeginPARALLEL // Traffic intended
                // towards honeyd.
    While true Do
      Capture_traffic(eth0, traf) ;
      If Belong_to(traf.Dest_ip_addr(),
       corresp_table) Then
        traf.Change_dest_MAC_addr(
             honeyd_MAC_add);
        Send_to_honeyd(traf) ;
      EndIf
    EndWhile
  PARALLEL // Outgoing traffic of honeyd.
    While true Do
      Receive_honeyd_outbound_traffic(traf);
      MAC_Addr:=corresp_table.Extract_MAC_addr(
                traf.Src_IP_addr());
      traf.Change_src_MAC_addr(MAC_Addr);
      Send_to_network(traf);
    EndWhile
  EndPARALLEL
EndProcedure
```

4.3 Virtual MAC Addresses Generator

As its name implies, this module is built to generate a list of unused virtual MAC addresses in the honeyds LAN.This list will then be used by the MAC Addresses Translation Module to generate the correspondence table.

Furthermore, the MAC addresses conflict arises only in the local area networks (the sources and addressees machines have a direct physical link among them), so the creation of the virtual MAC addresses list is relatively easy. It is enough to generate in a unpredictable way a big number of virtual MAC addresses of various usual builders, and delete from it afterward all the addresses already existing in our physical local area network, by a simple comparison with a file containing the list of all MAC addresses of our physical local area network.

4.4 IP Addresses Check Module

This module is conceived to fix the addresses conflict problems like those noticed in the last experimentation. Every start up of honeyd, this module verifies the IP addresses allocated to the various templates in the configuration file in order to test if there is no real machine which took one of these addresses, by using the same technique used with the DHCP mechanism. As soon as it discovers that one of these addresses is already associated with another real machine, it shows an error message to the administrator and does not start until the issue is resolved. To facilitate the administrator task, this mechanism must be able to look for the free addresses in order to suggest them with the message. So the general algorithm of this module is:

```
MainProgram IPAddCheckMod
Import allocated_addrs_file, honeyd.conf;
Var honeypots_ip_list,
    erreur_ip_list : ListObject;
Begin
  honeypots_ip_list[] :=
  Extract_ip(honeyd.conf);
  For i=1 to honeypots_ip_list[].length() Do
   If Allocated(honeypots_ip_list[i]) Then
      erreur_ip_list.Insert(
      honeypots_ip_list[i]);
   EndIf
  EndFor
  If Not Empty(erreur_ip_list) Then
    free_ip := Extract_free_ip(
    allocated_addrs_file);
    Alert_administrator(alert_msg,
    erreur_ip_list);
    Offer_to_administrator(free_ip);
  EndIf
End.
```

4.5 Modified ARPD Daemon

As we said first, the mechanism of arpd daemon is totally changed. It is no more based on the send out of its own broadcast ARP requests to be sure that an IP address is truly unused before claiming it with an ARP reply. It rather works in conjunction with the correspondence table supplied by the ”MACAddTransMod” module containing all the free IP addresses with the virtual MAC address. So, when it receives an ARP request for such an IP address, the deamon verifies directly in the correspondence table if there is an entry for this address or not and if it is the case, it extracts the virtual MAC address corresponding to this address from the correspondence table, then it loads it in the answered packet and send it to the requester. So, the general algorithm of the arpd daemon is:

```
MainProgram ARPd
Import corresp_table ;
Begin
  While true Do
    Capture_ARP_Requests(eth0, arp_req);
    If Belong_to(arp_req.ip_addr(),
    corresp_table) Then
      MAC_Addr:=
      corresp_table.Extract_MAC_addr(
      arp_req.ip_addr());
      arp_rep := prepare_ARP_reply(MAC_Addr);
      Send_reply(arp_rep, arp_req.src());
    EndIf
  EndWhile
End.
```

To our knowledge, there will be no possibility in the future to exploit the flaws discovered in this article with the implementation of those three modules and their integration with the current honeyd technology. This hardens honeyd defences against attacks intended to detect honeypots.

5 CONCLUSIONS

In this paper, we have presented various techniques that can be used by potential attackers to detect the presence of any Honeyd-based honeypot. They can subsequently modify their attack to either avoid the honeypot or send misleading traffic to the honeypot to hide their actions. We have made different experimental tests that allow us to highlight suspect behaviors of honeyd technology, and demonstrate how an attacker can exploit easily these flaws to detect the presence of this technology. Furthermore, we have presented some possible solutions to solve theses problems.

So, with this work, we can say that the honeyd technology really acquired more strength against detection attacks. But naturally, the flaws discovered in this article are not the only ones. Therere certainly other flaws which awaits us to be discovered. Only with hard work can we find them before they are exploited by malicious hackers.

REFERENCES

Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., and Keromytis, A. D. (2005). Detecting targeted attacks using shadow honeypots. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*.

Dornseif, M., Holz, T., and Klein, C. (2004). Nosebreak – attacking honeynets. In *Proceedings of the 5th IEEE Information Assurance Workshop*.

Fu, X., Yu, W., Cheng, D., Tan, X., Streff, K., and Graham, S. (2006). On recognizing virtual honeypots and countermeasures. *DASC, IEEE Computer Society*.

Holz, T. and Raynal, F. (2005). Detecting honeypots and other suspicious environments. In *Systems, Man and Cybernetics (SMC) Information Assurance Workshop*.

Jiang, X. and Xu, D. (2004). Collapsar: A vm-based architecture for network attack detention center. In *Proceedings of 13th USENIX Security Symposium*.

Krawetz, N. (2004). Anti-honeypot technology. *IEEE Security and Privacy*, 2.1.

Lamping, U. (2004). *Ethereal Developer's Guide: 18189 for Ethereal 0.10.14*. The Free Software Foundation.

Provos, N. (2002). *OpenBSD System Manager's Manual*.

Provos, N. (2003). Honeyd : A virtual honeypot daemon (extended abstract). In *Security Workshop in networking System, Hamburg*.

Provos, N. (2005). *Honeyd Project*. http://www.honeyd.org. Documentation and tools for general "honeyd" users.

Song, D. and Provos, N. (2003). *arpd*. http://www.honeyd.org/tools.php. Documentation and tools for general "honeyd" users.

Spitzner, L. (2002). *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing, Boston.

Website1 (2008). *Vmware homepage*. http://www.vmware.com. Documentation and tools for general "vmware" users.