802.11 Network Forensic Analysis

GAWN Gold Certification

Author: Akbar Qureshi, akbarqureshi@gmail.com

Advisor: Carlos Cid Ph.D.

Accepted: January 27, 2009

Table of Contents

Introduction

Theft and leakage of sensitive data pose a great business risk to organizations storing and processing sensitive information. Majority of the times organizations are oblivious to the state of their networks and simply do not have the security controls and processes in place to mitigate against data leakage and data theft.

This paper will demonstrate the detection, extraction and analysis (DEA) of credit card data leakage in an 802.11 network. The DEA process will be used to perform the forensic investigation of sensitive data leakage by both insider and external threats.

Data leakage simulations outlined in this document are fictitious and were conducted in a proof of concept lab. The purpose of the fictitious incidents is to further clarify the DEA process in a network traffic analysis and monitoring solution.

Organizations that are considering implementing or have already implemented an 802.11 network in their environment can benefit by reading this document. They can use it to develop their own solutions and methodologies to mitigate against sensitive data leakage.

Tools

The tools used in the proof of concept lab consist of freeware, opensource and author-developed software. The table below shows the opensource and freeware tools used in the lab.

Tools	Purpose
Snort Version 2.8.0.2 (Build 75) (Snort, 2008)	Snort was used along with customized signatures to inspect and detect credit card data in the packet payloads.
Engage Packet builder (Engage Packet builder, 2008)	Engage packet builder was used to test the snort signatures by injecting data packets with credit card data into the network.
Tcpdump version 3.9.4 libpcap version 0.9.4 (TCPDUMP/LIBPCAP public repository, 2008)	Tcpdump was used in promiscuous mode to capture, analyze and store network traffic.
Honeynet Security Console (Honeynet Security Console, 2008)	Honeynet Security Console is an event analysis tool. HSC can be used to view events generated from Snort, Tcpdump, Firewall, Syslog and Sebek. In the lab HSC was used to view alerts generated from Snort.

Kismet 2007.10.R1 (Kismet, 2008)	Kismet is an 802.11 wireless network detector, sniffer, and IDS system. In the lab Kismet was used as an overlay wireless IDS sensor to detect rogue Access Point association.

The following tools were developed by the author of this document.

Tools / Scripts	**Purpose**
Snort IDS Signatures	Four snort signatures to detect and alert on credit card data flowing in the network in clear text. One snort signature to detect wireless client association with a rogue wireless access point.
15digit.sh	Bash script to extract 15 digit credit card numbers from a raw packet capture file. Please see Appendix "A" for the source code.
16digit.sh	Bash script to extract 16 digit credit card numbers from a raw packet capture file. Please see Appendix "A" for the source code.

The following sections will provide a brief overview, including use and functionality, of the tools, scripts and the IDS signatures used for the wireless forensics analysis of data theft and leakage.

How the IDS signatures detect credit card data

The snort rules have been written using Perl compatible regular expressions (Perle, 2008). The alert will search for the credit card number pattern specified in the regular expression and will match the string "*american express*" and “*visa*” in the payload with the “*nocase*” option .The session option (Cox & Greg, 2004) in the alert is used to capture user data from TCP sessions which will assist in the forensics investigation of the alerts.

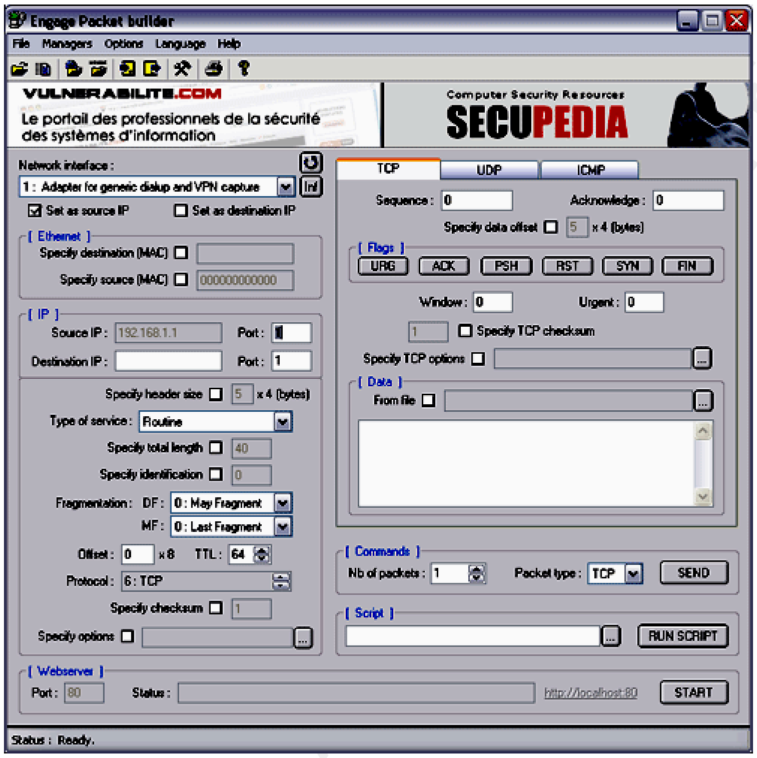
The following snort signatures will detect and alert on 15 and 16 digit credit card numbers in clear text using any source and destination IP address and port numbers. For testing purposes “American Express” has been selected for the 15 digit and “Visa” for the 16 digit credit card numbers.

```
alert tcp any any <> any  any \
(pcre:"/\d{4}(\s|-)?\d{6}(\s|-)?\d{5}/"; msg:"AMERICAN EXPRESS Credit Card \
detected in clear text"; content:"american express"; nocase; session: \
printable; sid:1000040; priority: 1; )

alert udp any any <> any  any \
(pcre:"/\d{4}(\s|-)?\d{6}(\s|-)?\d{5}/"; msg:"AMERICAN EXPRESS Credit \
Card detected in clear text"; content:"american express"; nocase; session: \
printable; sid:1000041; priority: 1; )
```

```
alert tcp any any <>  any any \
(pcre:"/\d{4}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/"; msg:"VISA Credit Card \
detected in clear text"; content:"visa"; session: printable; \
nocase; sid:1000042; priority: 1; )

alert udp any any <>  any any \
(pcre:"/\d{4}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/"; msg:"VISA Credit Card \
detected in clear text"; content:"visa"; session: printable;  \
nocase; sid:1000043; priority: 1; )
```

builder (Engage Packet builder, 2008).

Engage Packet builder (figure 1) is a very powerful packet builder tool for the Windows platform and is available as a free download. The tool requires WinPCAP to be installed. Customized TCP, UDP and ICMP packets with custom hex/ASCII payload can be created and injected. The tool also allows IP and MAC spoofing along with some other very useful features.


Engage Packet builder
File Managers Options Language Help
VULNERABILITE.COM
Le portail des professionnels de la sécurité des systèmes d'information
Computer Security Resources
SECUPEDIA
TCP UDP ICMP
URG ACK PSH RST SYN FIN
SEND
RUN SCRIPT
START

Network interface : 1 : Adapter for generic dialup and VPN capture
☑ Set as source IP ☐ Set as destination IP

[Ethernet]
Specify destination (MAC) ☐
Specify source (MAC) ☐ 000000000000

[IP]
Source IP : 192.168.1.1 Port : 1
Destination IP : Port : 1
Specify header size ☐ 5 x 4 (bytes)
Type of service : Routine
Specify total length ☐ 40
Specify identification ☐ 0
Fragmentation : DF : 0 : May Fragment
MF : 0 : Last Fragment
Offset : 0 x 8 TTL : 64
Protocol : 6 : TCP
Specify checksum ☐ 1
Specify options ☐

Sequence : 0 Acknowledge : 0
Specify data offset ☐ 5 x 4 (bytes)
[Flags]
Window : 0 Urgent : 0
1 ☐ Specify TCP checksum
Specify TCP options ☐
[Data]
From file ☐

[Commands]
Nb of packets : 1 Packet type : TCP
[Script]

[Webserver]
Port : 80 Status : http://localhost:80

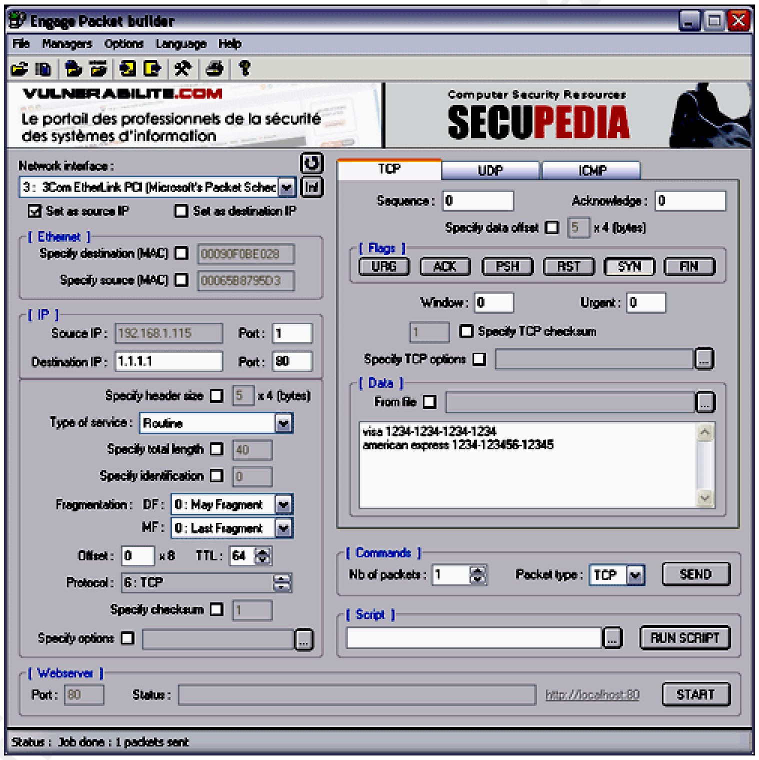
Status : Ready.

Figure 1

When creating IDS signatures it is very important to scope out the purpose. This initial design step will give a better direction when writing or creating IDS signatures. Last thing someone needs is a firework show on several false positives alerts at 3:00 in the morning. Proper tests and sanity checks are crucial to the overall success of the signature.

Fine tuning the IDS with the required signatures is the most time consuming process and is usually resolved by trial and errors checks.

Below (figure 2) is an example on how the snort IDS signature detecting credit card traffic in clear text was tested using Engage Packet builder tool. A source IP of 192.168.1.115 using the SYN flag and a destination IP of 1.1.1.1 using port 80 was used to generate the test alert. The actual alert was triggered by the data in the payload of the injected packet containing 16-digit Visa and 15-digit American Express credit card numbers.



Network interface : 3 : 3Com EtherLink PCI (Microsoft's Packet Schec
[✓ Set as source IP] Set as destination IP
Specify destination (MAC) 00090F0BE028
Specify source (MAC) 00065B8795D3
Source IP : 192.168.1.115 Port : 1
Destination IP : 1.1.1.1 Port : 80
Specify header size 5 x 4 (bytes)
Type of service : Routine
Specify total length 40
Specify identification 0
Fragmentation : DF : 0 : May Fragment
MF : 0 : Last Fragment
Offset : 0 x 8 TTL : 64
Protocol : 6 : TCP
Specify checksum 1
Specify options
Sequence : 0 Acknowledge : 0
Specify data offset 5 x 4 (bytes)
Window : 0 Urgent : 0
1 Specify TCP checksum
Specify TCP options
From file

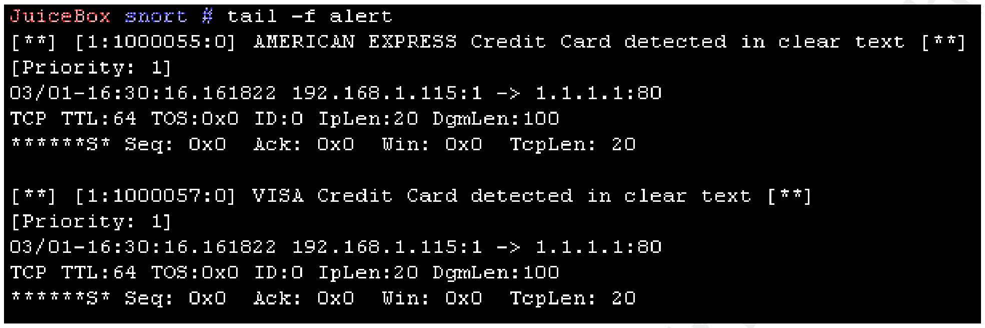
```
visa 1234-1234-1234-1234
american express 1234-123456-12345
```

Nb of packets : 1 Packet type : TCP
Port : 80 Status : http://localhost:80
Status : Job done : 1 packets sent

Figure 2

The following alert (figure 3) was generated as a result of the packet injection in figure 2. The snort alert was logged in the /var/log/snort/alert.

Akbar Qureshi 9



```
JuiceBox snort # tail -f alert
[**] [1:1000055:0] AMERICAN EXPRESS Credit Card detected in clear text [**]
[Priority: 1]
03/01-16:30:16.161822 192.168.1.115:1 -> 1.1.1.1:80
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:100
******S* Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] [1:1000057:0] VISA Credit Card detected in clear text [**]
[Priority: 1]
03/01-16:30:16.161822 192.168.1.115:1 -> 1.1.1.1:80
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:100
******S* Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
```

Figure 3

How the IDS signatures detect Rogue Access Point Association

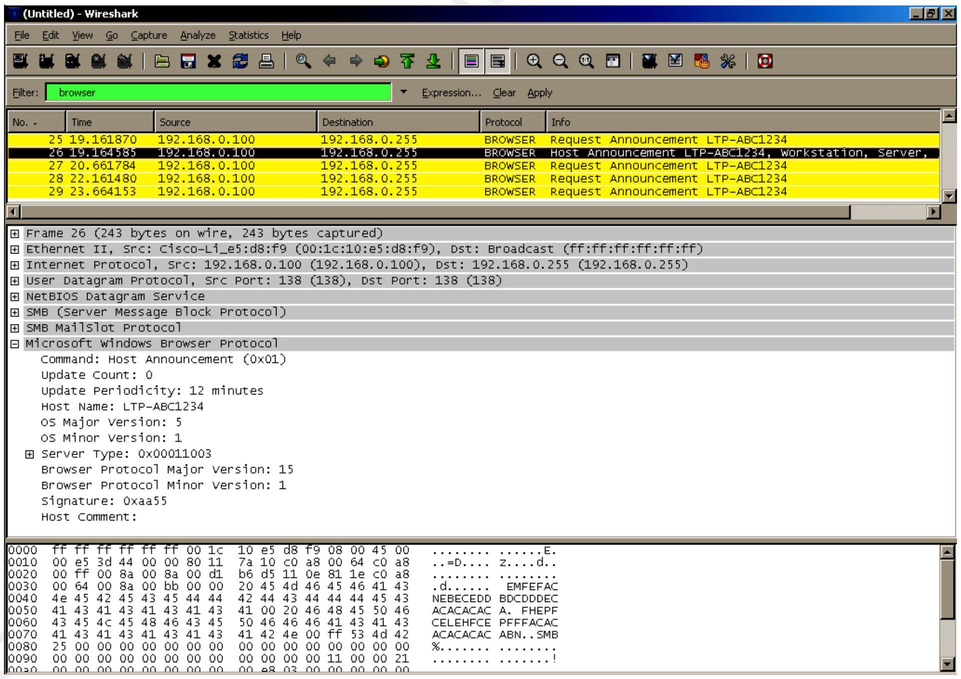
The “Possible Rogue AP Association” snort signature was developed to detect client association and communication with an non-encrypted wireless access point. The signature was developed using Perl regular expression to match a four digit pattern and also including the strings “ltp-abc”. So, any system with hostname ltp-abc1234, ltp-abc4321 etc will trigger an alert.

```
      alert udp any any <>  any any \
      (pcre:"/\d{4}/"; msg:"Possible Rogue AP Association"; content:"ltp-
abc";\
      session: printable; nocase; sid:1000061; priority: 1; )
```

The snort signature alerts on the presence of hostnames present in the Microsoft Windows Browser Protocol broadcasts.

The broadcast messages are usually triggered when the "computer browser" service is enabled on Windows systems. The computer browser service works by dynamically registering NetBIOS names and making the dynamic list available to other systems on the network.

Hence if the client is connected to a non-encrypted network, the snort signature will alert on the presence of hostnames visible in the Browser protocol in clear text. The screen shot (figure 4) shows a packet capture of the browser protocol in action using Wireshark.


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: browser
Expression... Clear Apply

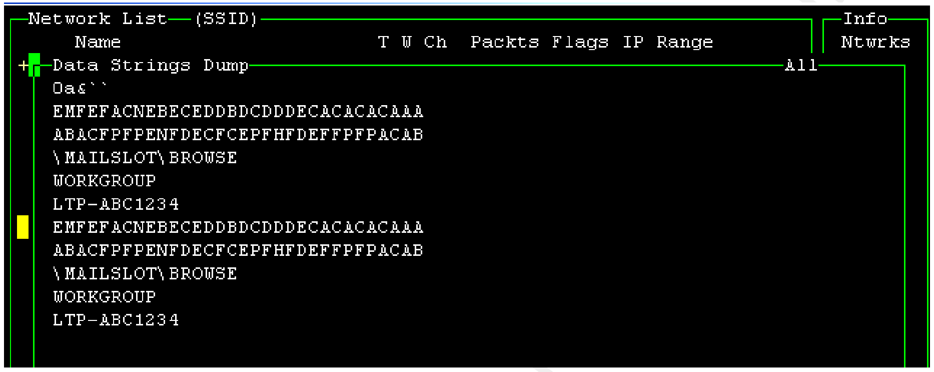
```
No.  Time       Source         Destination    Protocol Info
  25 19.161870  192.168.0.100  192.168.0.255  BROWSER  Request Announcement LTP-ABC1234
  26 19.164585  192.168.0.100  192.168.0.255  BROWSER  Host Announcement LTP-ABC1234, Workstation, Server,
  27 20.661784  192.168.0.100  192.168.0.255  BROWSER  Request Announcement LTP-ABC1234
  28 22.161480  192.168.0.100  192.168.0.255  BROWSER  Request Announcement LTP-ABC1234
  29 23.664153  192.168.0.100  192.168.0.255  BROWSER  Request Announcement LTP-ABC1234
```

```
⊞ Frame 26 (243 bytes on wire, 243 bytes captured)
⊞ Ethernet II, Src: Cisco-Li_e5:d8:f9 (00:1c:10:e5:d8:f9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.255 (192.168.0.255)
⊞ User Datagram Protocol, Src Port: 138 (138), Dst Port: 138 (138)
⊞ NetBIOS Datagram Service
⊞ SMB (Server Message Block Protocol)
⊞ SMB MailSlot Protocol
⊟ Microsoft Windows Browser Protocol
    Command: Host Announcement (0x01)
    Update Count: 0
    Update Periodicity: 12 minutes
    Host Name: LTP-ABC1234
    OS Major Version: 5
    OS Minor Version: 1
  ⊞ Server Type: 0x00011003
    Browser Protocol Major Version: 15
    Browser Protocol Minor Version: 1
    Signature: 0xaa55
    Host Comment:
```

```
0000  ff ff ff ff ff ff 00 1c  10 e5 d8 f9 08 00 45 00   ........ ......E.
0010  00 e5 3d 44 00 00 80 11  7a 10 c0 a8 00 64 c0 a8   ..=D.... z....d..
0020  00 ff 00 8a 00 8a 00 d1  b6 d5 11 0e 81 1e c0 a8   ........ ........
0030  00 64 00 8a 00 bb 00 00  20 45 4d 46 45 46 41 43   .d......  EMFEFAC
0040  4e 45 42 45 43 45 44 44  42 44 43 44 44 44 45 43   NEBECEDD BDCDDDEC
0050  41 43 41 43 41 43 41 43  41 00 20 46 48 45 50 46   ACACACAC A. FHEPF
0060  43 45 4c 45 48 46 43 45  50 46 46 46 41 43 41 43   CELEHFCE PFFFACAC
0070  41 43 41 43 41 43 41 43  41 42 4e 00 ff 53 4d 42   ACACACAC ABN..SMB
0080  25 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   %....... ........
0090  00 00 00 00 00 00 00 00  00 00 00 00 11 00 00 21   ........ .......!
00a0  00 00 00 00 00 00 00 00  00 e8 03 00 00 00 00 00
```

Figure 4

Windows Browser Protocol broadcasts by a Windows system can also be sniffed by running Kismet (figure 5). The system with hostname “lpt-abc1234” was connected to an open network with no encryption enabled.



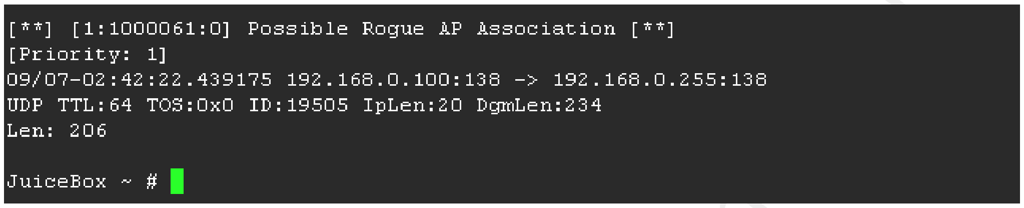
```
Network List (SSID)                                                    Info
   Name                         T W Ch  Packts Flags IP Range          Ntwrks
+ Data Strings Dump                                              All
  Oa&``
  EMFEFACNEBECEDDBDCDDDECACACACAAA
  ABACFPFPENFDECFCEPFHFDEFFPFPACAB
  \MAILSLOT\BROWSE
  WORKGROUP
  LTP-ABC1234
  EMFEFACNEBECEDDBDCDDDECACACACAAA
  ABACFPFPENFDECFCEPFHFDEFFPFPACAB
  \MAILSLOT\BROWSE
  WORKGROUP
  LTP-ABC1234
```

Figure 5

The Browser service messages did not seem to appear when the same system “ltp-abc1234” was configured to connect to a WEP\WPA enabled wireless access point. Let us illustrate the use of the alert in the following scenario.

User John while working within the company facility connects to a non-encrypted external rouge access point. The Computer Browser service in John’s system generates broadcast messages in clear text which are sniffed by the overlay wireless IDS sensor. The overlay wireless IDS sensor at this point triggers an alert (figure 6) due to the presence of the system’s hostname in the browser protocol in clear text. The administrator knows the naming convention standard of the systems in his company i.e. “ltp-abcxxxx”.

The administrator is notified and confirms that one of the systems in his company is communicating with an non-encrypted wireless network and can leak sensitive information in clear text to the malicious party hosting the rouge access point.



```
[**] [1:1000061:0] Possible Rogue AP Association [**]
[Priority: 1]
09/07-02:42:22.439175 192.168.0.100:138 -> 192.168.0.255:138
UDP TTL:64 TOS:0x0 ID:19505 IpLen:20 DgmLen:234
Len: 206

JuiceBox ~ #
```

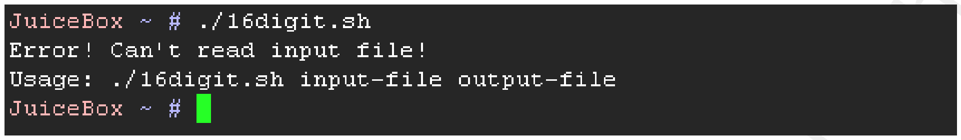
Figure 6

The overlay wireless IDS sensor is blind to company's "abc" encrypted traffic, so in this case gives the administrator more evidence that the alert generated from the overlay IDS sensor was triggered from non-encrypted traffic.

The snort signature was developed to be as simple as possible and at the same time effective in its intended purpose. Although this signature may not be the best way to detect Rogue access point association, as stated before it was developed to keep the signature simple.

How the scripts work

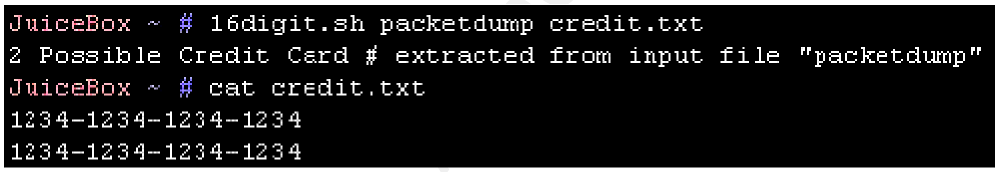
The bash scripts require two arguments as shown in figure 7. The input file can be a raw network capture file, e.g. pcap and the output file is where the credit card data will be extracted. The output file can have any arbitrary name. The script will take any network capture file containing credit card data as input as long as the traffic in the network capture file is not encrypted.



```
JuiceBox ~ # ./16digit.sh
Error! Can't read input file!
Usage: ./16digit.sh input-file output-file
JuiceBox ~ #
```

Figure 7

Below (figure 8) is a sample output on how data is extracted using the 16digit.sh script from a network capture file called *"packetdump"* to an output file called *"credit.txt"*. The command *"cat"* was used to view the output file which contained the credit card numbers.



```
JuiceBox ~ # 16digit.sh packetdump credit.txt
2 Possible Credit Card # extracted from input file "packetdump"
JuiceBox ~ # cat credit.txt
1234-1234-1234-1234
1234-1234-1234-1234
```

Figure 8

Proof of Concept Lab

The primary goal of the Proof of Concept Lab was to test the effectiveness of the tools and techniques for detecting, extracting and analyzing (DEA) different wireless attacks and threats. The lab was setup using the hybrid IDS sensor model solution as shown in the diagram below (figure 9)

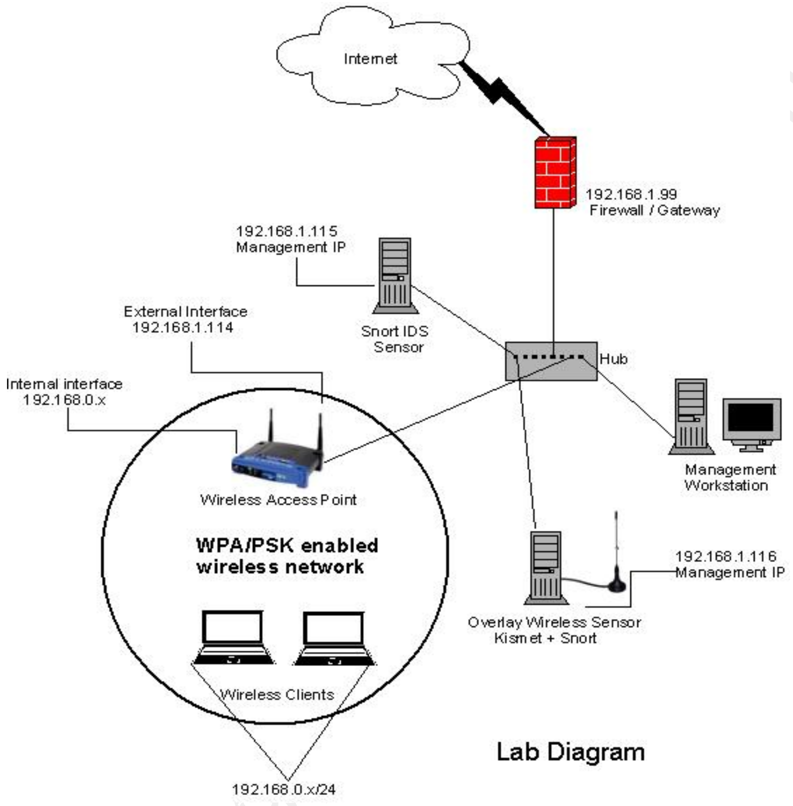


Figure 9

The hybrid deployment sensor solution is a combination of an integrated wired IDS sensor along with an overlay IDS sensor in an 802.11 network.

The differences between the two deployment solutions are described as follows.

1). Wired IDS Integration

In a wired IDS integrated solution, all encrypted wireless network traffic is centrally aggregated and decrypted at the access point and then inspected by the intrusion detection software. In this way wireless network traffic outbound to the

Internet is delivered to the snort IDS sensor unencrypted before it leaves the firewall.

The disadvantage of wired IDS integrated solution is that the sensor is blind to attacks between ad-hoc wireless networks. This is because in a peer to peer wireless setup, the network traffic will not be aggregated at the central access point, but will instead traverse between the local wireless clients. Also in a scenario where the client connects to a rouge access point, the integrated sensor solution will fail to detect and prevent malicious attacks between the client and the rouge access point. If a wireless client is a victim of the Karma tool (Karma, 2008), then all communications between the client and the hacker’s rogue access point will flow undetected by the integrated sensor. This solution only works as long as the network traffic passes in and out from the access point.

2). Overlay Sensor Deployment

In an overlay sensor deployment (figure 10), a dedicated sensor is used to scan the airwaves for malicious wireless attacks. The sensor monitors the wireless network by hopping over all allowable channels within that region or country and scans the entire 802.11 spectrum for wireless threats. Key management is one issue with the overlay sensor deployment solution for decrypting network traffic. For example, the sensor monitoring a WPA enabled network will require the WPA key to decrypt and alert on wireless traffic flowing within that wireless LAN.

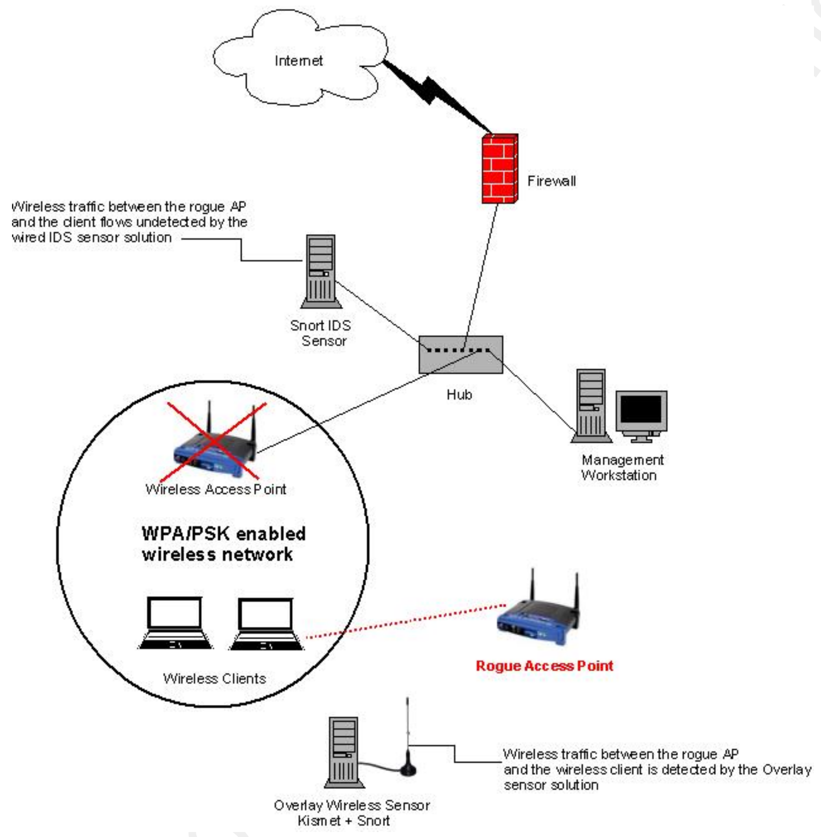

Internet
Firewall
Wireless traffic between the rogue AP and the client flows undetected by the wired IDS sensor solution
Snort IDS Sensor
Hub
Management Workstation
Wireless Access Point
WPA/PSK enabled wireless network
Wireless Clients
Rogue Access Point
Wireless traffic between the rogue AP and the wireless client is detected by the Overlay sensor solution
Overlay Wireless Sensor Kismet + Snort

Figure 10

Note: *Before deploying IDS sensors in an environment, it is very important to conduct a thorough survey of the site and evaluate the requirements. A solid planning will decrease the likelihood of a poor monitoring solution.*

Lab Architecture Components and Configuration

Components	CPU	RAM	DISK	OS
Snort 2.8 / IDS + MySQL 5.0.24a	Pentium 733MHz	256 MB	6 GB	SLAX 6.0
Kismet + Snort (Overlay IDS sensor)	Pentium 1.7 GHz	512 MB	10 GB	SLAX 6.0
Wireless Clients	Pentium	512 MB	15 GB	Windows XP SP

	1.7 GHz			2
Management Workstation	Pentium 733MHz	256 MB	10 GB	Windows XP SP 2
Linksys WRT54GL Wireless AP / Router	200 MHz	16 MB	4 MB	OpenWRT (White Russian 0.9) openwrt-wrt54g-squashfs.bin
Analysis Workstation	Pentium 1.7 GHz	512 MB	10 GB	FreeBSD
3comm Office connect hub				

The “Overlay IDS Sensor” in the lab was basically a Linux system with a Linksys DWL-G520 wireless pci adapter. Snort was configured to log alerts to the default snort alert logging directory (/var/log/snort/) and also to a MySQL database. The MySQL database was managed and accessed by tunneling port 3306 via ssh from the management workstation. The intrusion detection and monitoring solution in the Linux system was configured to utilize the combined power of both kismet and snort. Kismet was configured to stream packets to a FIFO named pipe which was read and processed by snort intrusion detection system. The combined power of kismet and snort, created a very powerful wireless IDS and monitoring solution.

To enable kismet to write data to a named pipe, the following line in kismet.conf was commented out (# sign was removed).

```
#fifo=/tmp/kismet_dump
```

Kismet when started creates the named pipe “kismet_dump” in the tmp directory and blocks the kismet_dump file until its read by another application or process. In our case snort was used to read the kismet_dump file.

```
snort –r /tmp/kismet_dump –c /etc/snort/snort.conf
```

The overlay IDS sensor was managed by using a second network interface card. As best practice multiple network cards should be used, for example a minimum of two network cards, one for monitoring and the second for management access. The network monitoring system should not be managed using the network card configured for capturing data, as this may contaminate the capture files with management traffic data.

The linksys access point in the lab was also used as a network traffic recorder. Most data leak prevention or content inspection solutions record and dump network traffic in real time. The recorded traffic can be used for forensics investigations where traffic or sessions can be reconstructed for forensic analysis. Tcpdump was installed on the access point by running “*ipkg install tcpdump*” and used on the access point to record all network traffic to a “pcap” file. The flash size on the Linksys AP running the third party firmware “OpenWRT” was only 4MB and was not sufficient to store the pcap file. To overcome this barrier SHFS “Secure Shell Filesystem” was installed on the access point and was used to mount a remote file system on the wireless AP using SSHv2 (figure 11). The SSH server was running on the snort IDS system.

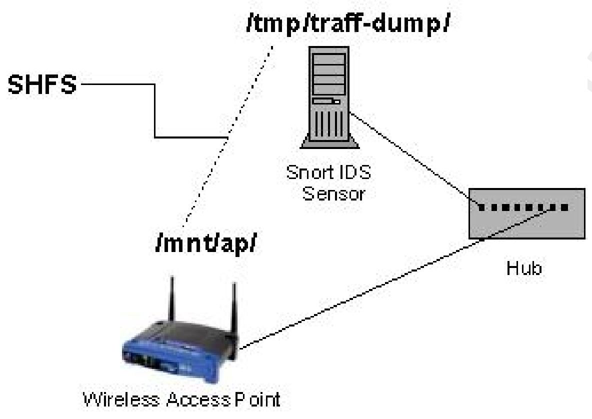


Figure 11

A directory called “traff-dump” was created on the snort IDS sensor system for storing the “pcap” file. The snort IDS system was connected to the same hub as the access point. The reason for creating the directory on the snort IDS sensor system was to centralize both snort alert data and the captured data from tcpdump running on the access point in one location. This created two usable sources of network based evidence to be used in an event of a forensics investigation. The benefit of using both the sources in a forensics investigation will be demonstrated in the data theft simulation section of this document.

The remote filesystem mount “How to” is covered in Appendix B.

Note: *It is very important that all the components involved in network monitoring have their system clocks synched with a central NTP server. This will avoid the disaster of mismatch in time stamps in event logs from different logging systems.*

Data Theft Simulations

The following data theft simulations will illustrate credit card data theft and leakage. The 802.11 network monitoring solution will detect and alert on credit card data in clear text flowing across the network. The detection, extraction and analysis (DEA) process will be used in the simulations.

The data theft simulation covers data leakage by an insider threat and data theft by an external threat.

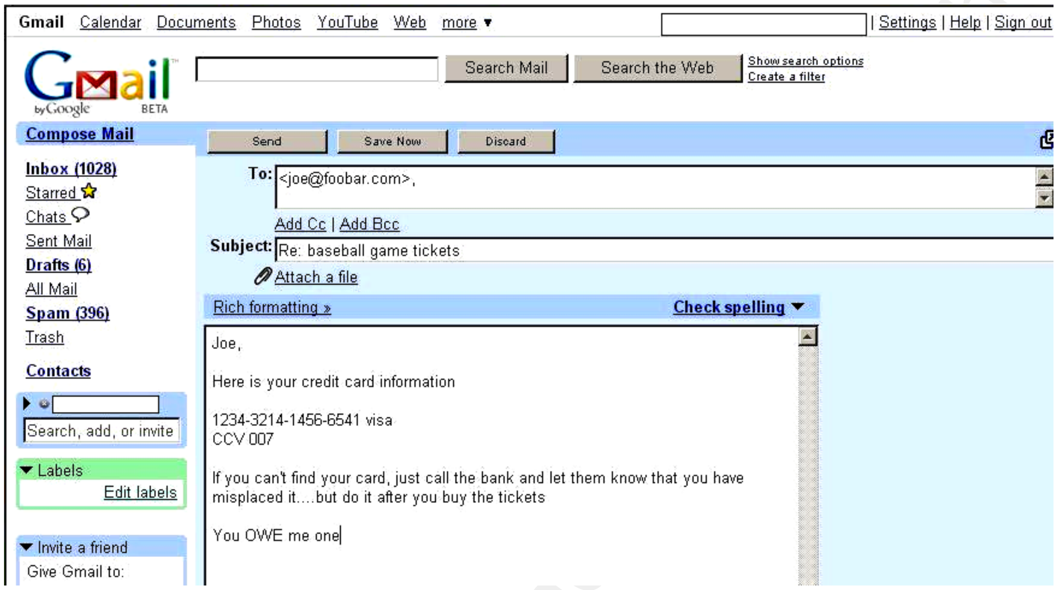
Simulation 1 – Data leak by Insider Threat

Bob is employed as an account manager for a fictitious retail bank called “XYZ Bank”. XYZ Bank recently deployed a wireless LAN in their branch to increase mobility and productivity for their office staff. For Bob, this meant that he can now take his laptop and use it anywhere in the branch office, for example in the conference room, lunch room or in the bank’s reception area.

Scenario

Bob is scheduled for a meeting in the conference room with a new client. The meeting is about to start in fifteen minutes when his best friend “Joe” calls him on his cell phone. Joe is panicking on the phone as he can’t seem to find his credit card which he needs to make a last minute ticket purchase for a baseball game. Bob being a good friend and feeling sorry for his buddy tells him that he can pull up his account information and email him the credit card details to make the necessary purchase.

Bob composes the following email (Figure 12) using his gmail account, as he knows that the company emails are being monitored. Although he does not know that the company has a data leak prevention solution capable of detecting credit card data leakage.



To: <joe@foobar.com>,

Subject: Re: baseball game tickets

Joe,

Here is your credit card information

1234-3214-1456-6541 visa
CCV 007

If you can't find your card, just call the bank and let them know that you have misplaced it....but do it after you buy the tickets

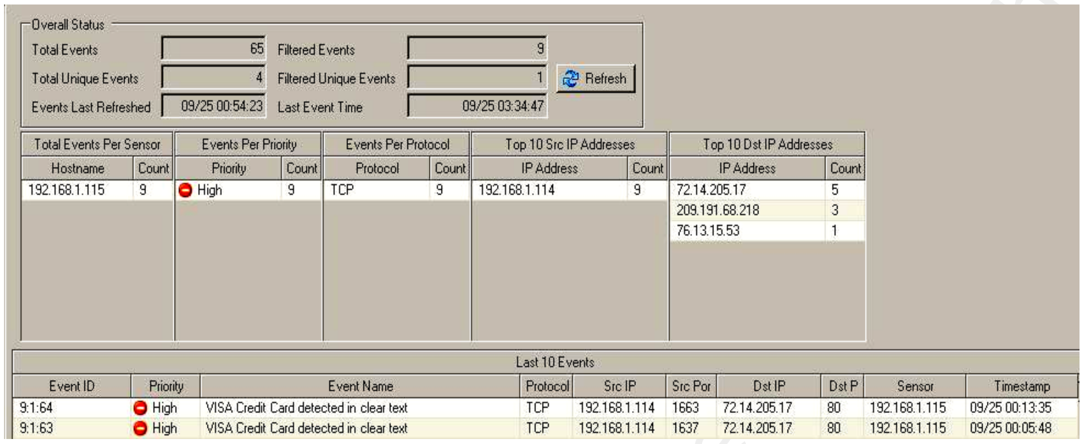
You OWE me one

Figure 12

Detection

This is the detection stage of the “Detection, Extraction and Analysis” process.

The transfer of credit card data in the outbound email generates an alert in Honeynet Security Console (figure 13)



Overall Status

Total Events	65	Filtered Events	9
Total Unique Events	4	Filtered Unique Events	1
Events Last Refreshed	09/25 00:54:23	Last Event Time	09/25 03:34:47

Total Events Per Sensor		Events Per Priority		Events Per Protocol		Top 10 Src IP Addresses		Top 10 Dst IP Addresses	
Hostname	Count	Priority	Count	Protocol	Count	IP Address	Count	IP Address	Count
192.168.1.115	9	High	9	TCP	9	192.168.1.114	9	72.14.205.17	5
								209.191.68.218	3
								76.13.15.53	1

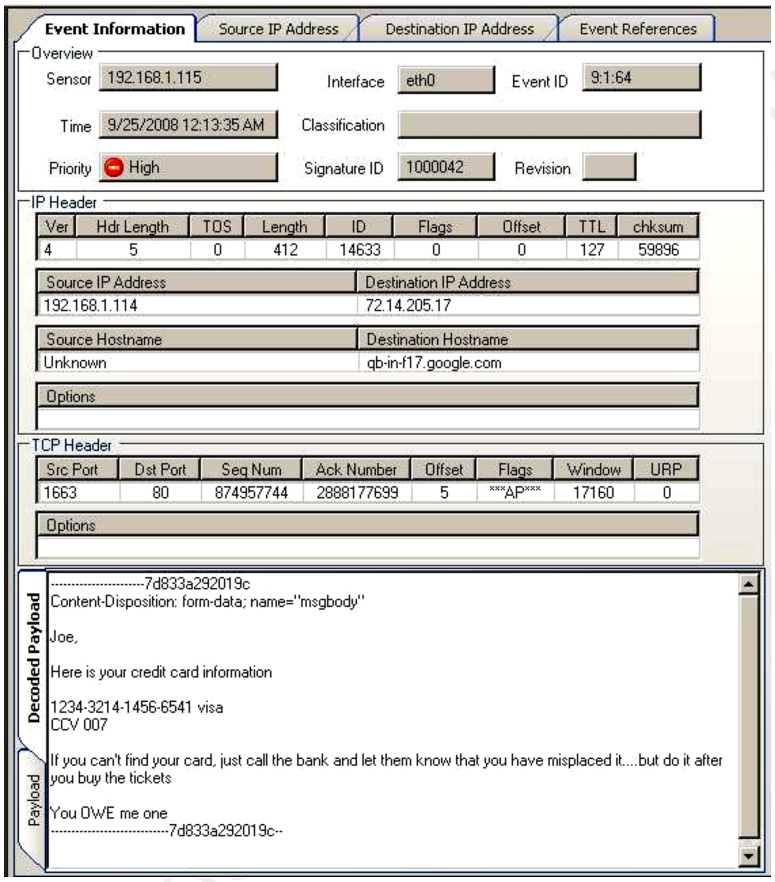
Last 10 Events

Event ID	Priority	Event Name	Protocol	Src IP	Src Por	Dst IP	Dst P	Sensor	Timestamp
9:1:64	High	VISA Credit Card detected in clear text	TCP	192.168.1.114	1663	72.14.205.17	80	192.168.1.115	09/25 00:13:35
9:1:63	High	VISA Credit Card detected in clear text	TCP	192.168.1.114	1637	72.14.205.17	80	192.168.1.115	09/25 00:05:48

Figure 13

The decoded ASCII payload (figure 14) clearly shows the email sent by Bob to Joe with credit card information. The following session capture is a proof of the communication that occurred between Bob and Joe.

Note*: The IDS alert was triggered by sending an email to the fictitious email address joe@foobar.com with fake credit card data using Gmail using HTTP not HTTPS.*



Overview

Sensor	192.168.1.115	Interface	eth0	Event ID	9:1:64
Time	9/25/2008 12:13:35 AM	Classification			
Priority	High	Signature ID	1000042	Revision	

IP Header

Ver	Hdr Length	TOS	Length	ID	Flags	Offset	TTL	chksum
4	5	0	412	14633	0	0	127	59896

Source IP Address	Destination IP Address
192.168.1.114	72.14.205.17

Source Hostname	Destination Hostname
Unknown	qb-in-f17.google.com

Options

TCP Header

Src Port	Dst Port	Seq Num	Ack Number	Offset	Flags	Window	URP
1663	80	874957744	2888177699	5	***AP***	17160	0

Options

```
---------------------7d833a292019c
Content-Disposition: form-data; name="msgbody"

Joe,

Here is your credit card information

1234-3214-1456-6541 visa
CCV 007

If you can't find your card, just call the bank and let them know that you have misplaced it....but do it after
you buy the tickets

You OWE me one
---------------------------7d833a292019c--
```

Figure 14

The next stage will be the extraction process where we will identify and extract data of interest for the incident response and forensics investigation.

Extraction

The network evidence will be moved from the Snort IDS sensor and the wireless access point to the analysis system.

We will copy the evidence from the Snort IDS sensor to the evidence analysis system using secure copy (SCP).

The data of interest on the snort IDS system are the snort logs in /var/log/snort/ and the aircapture.pcap file in the /tmp/traff-dump/ directory.

The following methodology was used to copy the evidence from the snort IDS sensor. The methodology also included techniques to secure and preserve the network evidence for the forensic investigation.

1) Hashes were created remotely from the analysis station against all the evidence files before the evidence was copied from the snort IDS sensor to analysis workstation.

SSH was used from the analysis workstation to log on to the snort IDS sensor (192.168.1.115). Sha256deep was used to compute hashes of all original files and results of the hashes were stored locally on the analysis workstation.

```
     ssh aqureshi@192.168.1.115 "sha256deep -r /var/log/snort/" >
/evidence/orig-hashes/snort.sha256

     ssh aqureshi@192.168.1.115 "sha256deep -r /tmp/traff-
dump/aircapture.pcap" > /evidence/orig- hashes/aircapture.pcap.sha256
```

```
     cat /evidence/orig-hashes/snort.sha256
     e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

```
/var/log/snort/PLACER
     2023fe111664c8e71dd719a9fb8539bd5186a75be7abfa070d0c43f742e10090
/var/log/snort/alert
     f3cd527b88e067affce1455ef59503d0a9b9a357489409a3ba60226586ab4f28
/var/log/snort/snort.log.1222315810
     4be3d41b336bd82046360bde3a4388c09d04e11bb8db18eda7a5a172186042c6
/var/log/snort/192.168.1.114/SESSION:1663-80
     c597ec6afab0d89d4463bace2e2f7f7567a0f331779686ecdb21e9a3e6620f91
/var/log/snort/192.168.1.114/SESSION:1688-80
     5ce59094c7a7e473704b5900a9a28e1a3224d85a2de31663b41bf3f574c2cd56
/var/log/snort/1201670737
```

Hash of the aircapture.pcap files using sha256.

```
     cat /evidence/orig-hashes/aircapture.pcap.sha256
5ba9f8e7a95922f56011eea62373ebb43b891ee4dcc0c6e128b6b28a8cbb899d
/tmp/traff-dump/aircapture.pcap
```

2) After generating the hashes, the evidence was copied from the snort IDS sensor to the analysis workstation using secure copy (SCP).All evidence was copied under the "/evidence/data/" directory in the analysis workstation.

a) Snort logs were copied from the snort IDS sensor using SCP from the analysis workstation.

```
     scp -r aqureshi@192.168.1.115:/var/log/snort/
/evidence/data/
```

b) The aircapture.pcap file was copied from the Snort IDS sensor using SCP.

```
     scp -r aqureshi@192.168.1.115:/tmp/traff-dump/aircapture.pcap
/evidence/data/
```

3) The permissions on the copied data were changed to read-only to prevent accidental modification of the evidence.

a)   chmod –R 444 /evidence/data/snort/

b)   chmod 444 /evidence/data/aircapture.pcap

Now that the evidence is securely copied and protected from any accidental act that may contaminate the evidence, we will move to the analysis stage of the incident.

Analysis

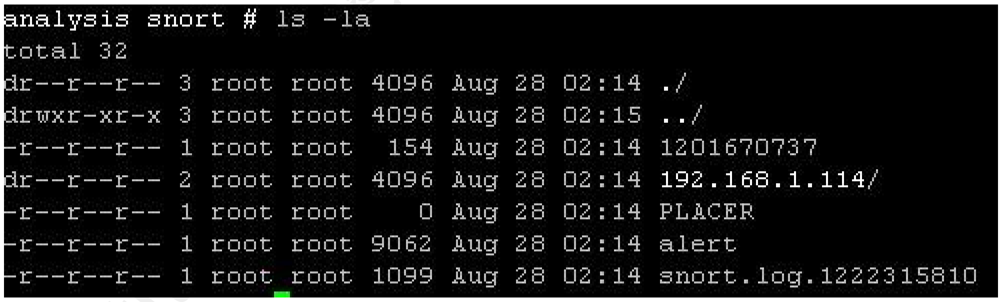
The decoded section in the honeynet security console clearly shows the contents of the email in clear text, including the credit card number. Other useful information available on the honeynet security console are timestamps, source/destination ports, source/destination ip addresses, tcp flags etc. The only information not available is the real source IP address which would be the internal mapped address of the offending system.

The source IP address of 192.168.1.114 belongs to the external interface of the wireless access point. Since the wireless access point is configured to do network address translation (NAT) we know that the real IP address of Bob’s system is not 192.168.1.114.

We will need to look at additional sources of information to identify the source IP address of Bob’s system.

Let us use the two sources of information available for our forensics investigation. Source number one are snort logs and source number two is the “aircapture.pcap” file which was produced by running tcpdump directly from the wireless access point.

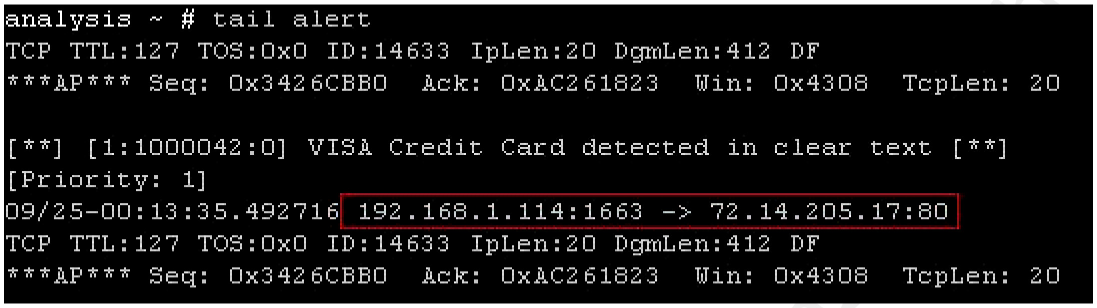
The captured data contained in the snort logs only shows data captured from the external interface of the wireless access point. This can be seen in the screen shot below (figure 15).



```
analysis snort # ls -la
total 32
dr--r--r-- 3 root root 4096 Aug 28 02:14 ./
drwxr-xr-x 3 root root 4096 Aug 28 02:15 ../
-r--r--r-- 1 root root  154 Aug 28 02:14 1201670737
dr--r--r-- 2 root root 4096 Aug 28 02:14 192.168.1.114/
-r--r--r-- 1 root root    0 Aug 28 02:14 PLACER
-r--r--r-- 1 root root 9062 Aug 28 02:14 alert
-r--r--r-- 1 root root 1099 Aug 28 02:14 snort.log.1222315810
```

Figure 15

The alert file is viewed and only the communication between the external interface of the wireless AP and the remote host is available (figure 16). The alert data in the honeynet security console can also be found under /var/log/snort/alert file.

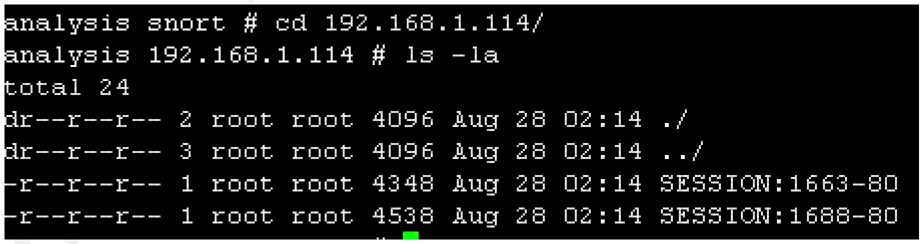


```
analysis ~ # tail alert
TCP TTL:127 TOS:0x0 ID:14633 IpLen:20 DgmLen:412 DF
***AP*** Seq: 0x3426CBB0  Ack: 0xAC261823  Win: 0x4308  TcpLen: 20

[**] [1:1000042:0] VISA Credit Card detected in clear text [**]
[Priority: 1]
09/25-00:13:35.492716 192.168.1.114:1663 -> 72.14.205.17:80
TCP TTL:127 TOS:0x0 ID:14633 IpLen:20 DgmLen:412 DF
***AP*** Seq: 0x3426CBB0  Ack: 0xAC261823  Win: 0x4308  TcpLen: 20
```

Figure 16

The snort signatures were written using the "session" option. The session option dumps application layer information for the generated alert in /var/log/snort/ directory.

The directory "192.168.1.114/" contains the captured session information, specific to the email transfer of credit card data in clear text. An "ls -la" command in the 192.168.1.114 directory shows the following sessions captured in the screen shot below (figure 17).



```
analysis snort # cd 192.168.1.114/
analysis 192.168.1.114 # ls -la
total 24
dr--r--r-- 2 root root 4096 Aug 28 02:14 ./
dr--r--r-- 3 root root 4096 Aug 28 02:14 ../
-r--r--r-- 1 root root 4348 Aug 28 02:14 SESSION:1663-80
-r--r--r-- 1 root root 4538 Aug 28 02:14 SESSION:1688-80
```

Figure 17

The session files are named by source and destination port numbers (figure 18).

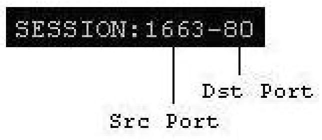
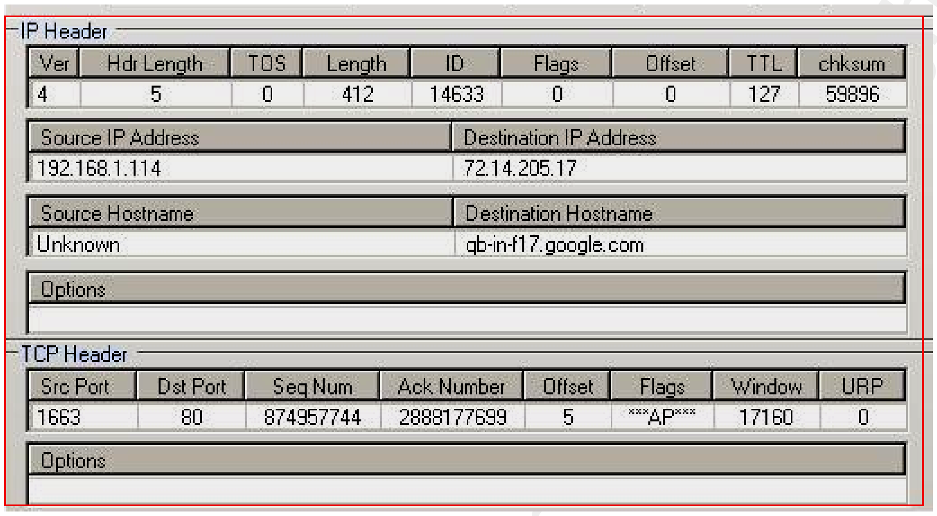


Figure 18

At this point, the information available is still not sufficient to reveal the source IP address of Bob’s system. This does not mean that the information gathered so far is useless and should be discarded. All the valuable session and packet information will be used to process the “aircapture.pcap” file, which is the second source of evidence. We will use tcpdump to read the “airecapture.pcap” file and will also use filters to assist in identifying and focusing only on the relevant session information.

The session data in the alert provides us with sufficient information in creating the required filters. The IP header fields will be used to create the necessary filters to process the “airecapture.pcap” file. For example the source and destination IP and ports numbers, tcp flags, timestamps, data length, identification etc can be used as filters in tcpdump.

The IPv4 header information can be easily retrieved from the honeynet security console window (figure 19).



IP Header

Ver	Hdr Length	TOS	Length	ID	Flags	Offset	TTL	chksum
4	5	0	412	14633	0	0	127	59896

Source IP Address	Destination IP Address
192.168.1.114	72.14.205.17

Source Hostname	Destination Hostname
Unknown	qb-in-f17.google.com

Options

TCP Header

Src Port	Dst Port	Seq Num	Ack Number	Offset	Flags	Window	URP
1663	80	874957744	2888177699	5	***AP***	17160	0

Options

Figure 19

Tcpdump was used with the known source and destination port numbers to filter data from the “aircapture.pcap” file. Tcpdump was executed in the following manner.

Command:

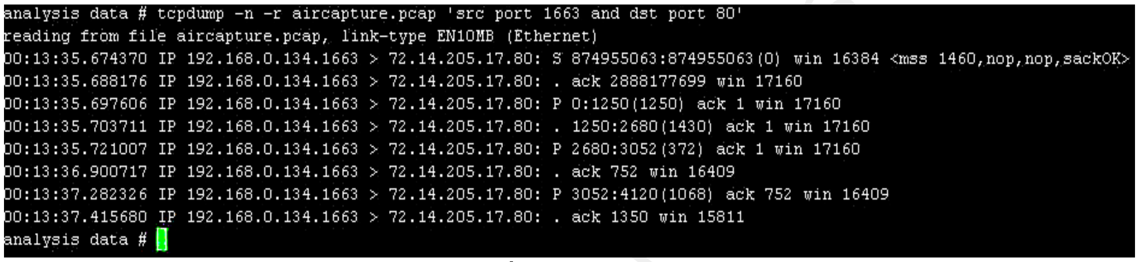
tcpdump -n -r aircapture.pcap 'src port 1663 and dst port 80'

Options:

```
-n = disable name resolution
-r = read the dump file
```

Finally, the “aircature.pcap” file revealed the source IP address of Bob’s system. We can see in the screen shot (Figure 20) that IP address 192.168.0.134 actually communicated using source port of 1663 to the remote host 72.14.205.17 at port 80.

The IP address 192.168.0.134 was mapped to the external interface of the wireless access point. This is a basic concept of “many-to-one” in network address translation where internal private addresses are mapped to one external IP address.



```
analysis data # tcpdump -n -r aircapture.pcap 'src port 1663 and dst port 80'
reading from file aircapture.pcap, link-type EN10MB (Ethernet)
00:13:35.674370 IP 192.168.0.134.1663 > 72.14.205.17.80: S 874955063:874955063(0) win 16384 <mss 1460,nop,nop,sackOK>
00:13:35.688176 IP 192.168.0.134.1663 > 72.14.205.17.80: . ack 2888177699 win 17160
00:13:35.697606 IP 192.168.0.134.1663 > 72.14.205.17.80: P 0:1250(1250) ack 1 win 17160
00:13:35.703711 IP 192.168.0.134.1663 > 72.14.205.17.80: . 1250:2680(1430) ack 1 win 17160
00:13:35.721007 IP 192.168.0.134.1663 > 72.14.205.17.80: P 2680:3052(372) ack 1 win 17160
00:13:36.900717 IP 192.168.0.134.1663 > 72.14.205.17.80: . ack 752 win 16409
00:13:37.282326 IP 192.168.0.134.1663 > 72.14.205.17.80: P 3052:4120(1068) ack 752 win 16409
00:13:37.415680 IP 192.168.0.134.1663 > 72.14.205.17.80: . ack 1350 win 15811
analysis data #
```

Figure 20

All that is left at this point is to match the network session data to confirm the findings. This means that we will look at network session specific information between Bob’s system and the wireless access point and between the wireless access point and the remote host.

The IP identification field in the IPv4 header of the communication was used to identify and confirm the email transmission. In IPv4, the IP identification field is always unique to the source and destination endpoints and for the time the IP datagram is active in the network communication.

The IP identification (IP ID) for the packet that generated the alert was 14633. Tcpdump was used to only process packets with ID 14633.

Tcpdump was executed with the following options

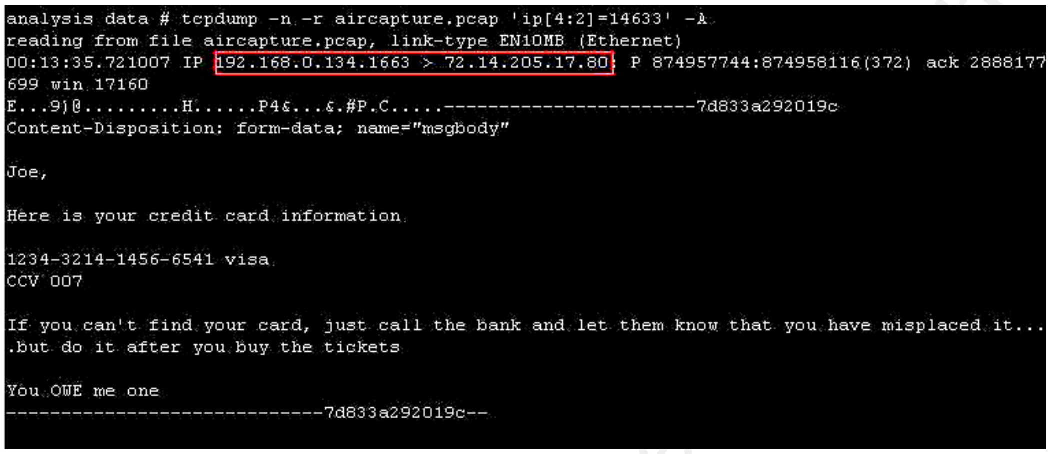
Command:

```
tcpdump -n -r aircapture.pcap 'ip[4:2]=14633' –A
```

Options:

```
     -n = disable name resolution
     -r = read the dump file
     -A = Print each packet (minus its link level header) in
ASCII.
     Ip[4:2]=14633 = Protocol [byte count: offset]=IPID
```

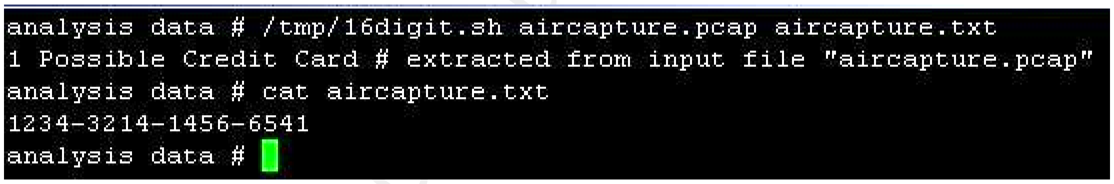
The result from running tcpdump with IPID filter clearly shows that the source IP address 192.168.0.134 was involved in leaking credit card data via email (figure 21). Filtering and matching both the snort logs and the “airecapture.pcap” file with IPID of 14633 further verifies that it was indeed Bob’s system that communicated with the remote host.



```
analysis data # tcpdump -n -r aircapture.pcap 'ip[4:2]=14633' -A
reading from file aircapture.pcap, link-type EN10MB (Ethernet)
00:13:35.721007 IP 192.168.0.134.1663 > 72.14.205.17.80: P 874957744:874958116(372) ack 2888177
699 win 17160
E...9)@.........H......P4&...&.#P.C.....------------------------7d833a292019c
Content-Disposition: form-data; name="msgbody"

Joe,

Here is your credit card information

1234-3214-1456-6541 visa
CCV 007

If you can't find your card, just call the bank and let them know that you have misplaced it...
.but do it after you buy the tickets

You OWE me one
-----------------------------7d833a292019c--
```

Figure 21

The “16digit.sh” script was used to extract the credit card information from the capture file (figure 22)



```
analysis data # /tmp/16digit.sh aircapture.pcap aircapture.txt
1 Possible Credit Card # extracted from input file "aircapture.pcap"
analysis data # cat aircapture.txt
1234-3214-1456-6541
analysis data #
```

Figure 22

Extracting only the credit card data is beneficial in cases, for example where the financial fraud department is only interested to know how many and what kind of credit cards were leaked or stolen. The Investigator can quickly provide the financial fraud department with the required information and can continue on focusing on the investigation. It will be cumbersome to go through large volumes of captured data manually, where the theft or leak of credit cards can be in hundreds or thousands.

Bob’s email to Joe was an act of sensitive data leak. Bob did not steal Joe’s credit card information for purchasing goods or for other self beneficial purposes. He just helped is best friend.

Simulation 2 – Risks from Rogue Wireless Access Points

This simulation presents the risks from the presence of rouge wireless access points and how rogue access points can be used to steal corporate data from unwary wireless users. This simulation is more of a general overview of how various forms of sensitive information e.g. financial, passwords, corporate proprietary information etc can be stolen by a hacker.

The wireless monitoring solution in this demonstration will alert on client associations to rogue wireless access points, therefore minimizing the risks of clients creating a backdoor access to their own corporate network.

Scenario

Alice works as a financial accountant for a bank called ABC bank. ABC bank has no wireless network in place but has deployed wireless sensors to detect rouge access points using the overlay monitoring deployment solution.

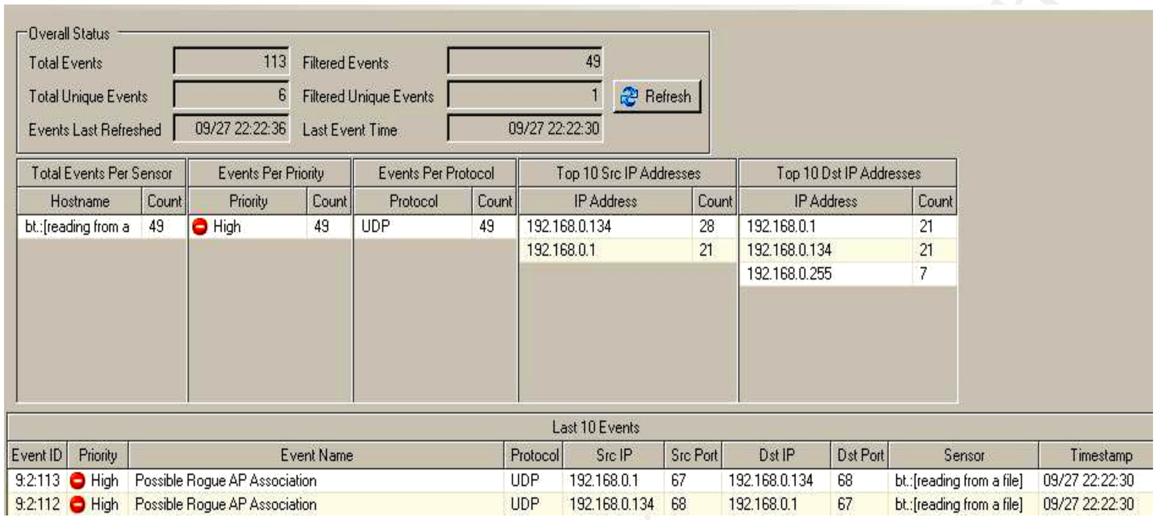
Note: *The overlay wireless sensor in the Lab to demonstrate the fictitious incident was a combination of both kismet and snort running on the same system. Kismet was configured to stream all the sniffed data to a named pipe, which was read by Snort.*

Alice is connected to the corporate network using the wired port in her cube. The hostname of her laptop is *“LTP ABC1234”*.Her laptop has an integrated wireless adapter which is always enabled both at home and at work. The wireless card in her laptop is constantly broadcasting the SSID of her trusted home network to which she recently connected with.

A hacker who just happens to live across the corporate office in an apartment decides to use his newly purchased high gain antenna for a wireless night out. The hacker starts the attack by launching KARMA and waits for clients within the range of the antenna to associate with his system.

Alice’s laptop which is configured to automatically connect to access points falls within the range of the hacker’s antenna and connects to his system.

The following alert (figure 23) is generated when Alice’s system connects to the Rogue Access Point. As mentioned before, the snort signature detects the rogue access point association by inspecting the airwaves for system hostnames in clear text. The hostname announcements are triggered by Microsoft’s Browser protocol.



Overall Status

Total Events	113	Filtered Events	49
Total Unique Events	6	Filtered Unique Events	1
Events Last Refreshed	09/27 22:22:36	Last Event Time	09/27 22:22:30

Refresh

Total Events Per Sensor

Hostname	Count
bt.:[reading from a	49

Events Per Priority

Priority	Count
High	49

Events Per Protocol

Protocol	Count
UDP	49

Top 10 Src IP Addresses

IP Address	Count
192.168.0.134	28
192.168.0.1	21

Top 10 Dst IP Addresses

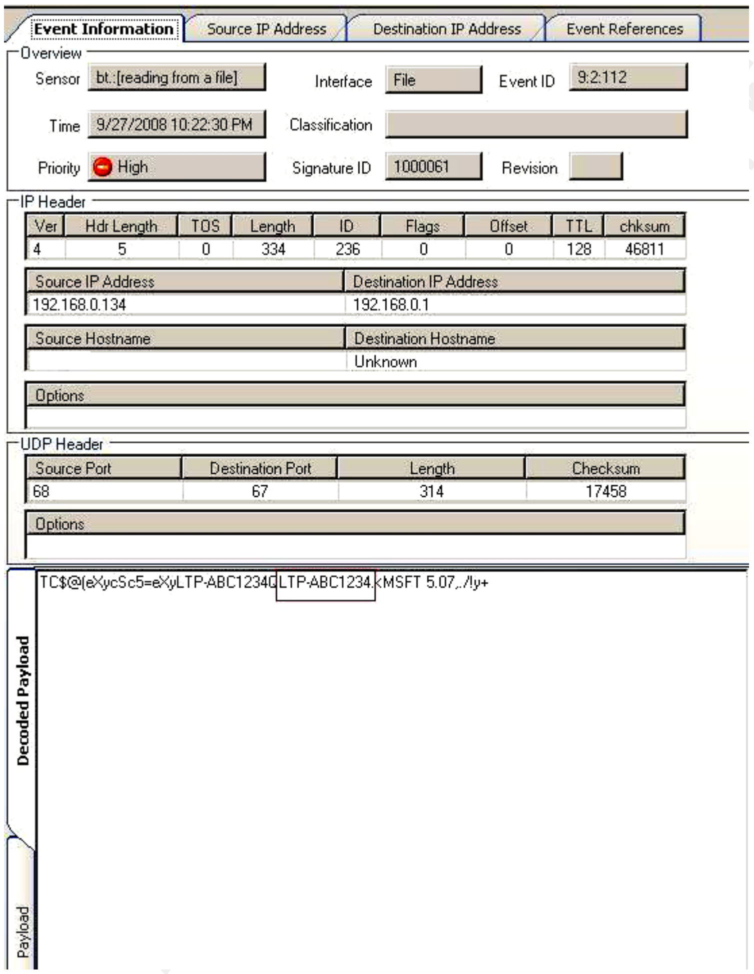
IP Address	Count
192.168.0.1	21
192.168.0.134	21
192.168.0.255	7

Last 10 Events

Event ID	Priority	Event Name	Protocol	Src IP	Src Port	Dst IP	Dst Port	Sensor	Timestamp
9:2:113	High	Possible Rogue AP Association	UDP	192.168.0.1	67	192.168.0.134	68	bt.:[reading from a file]	09/27 22:22:30
9:2:112	High	Possible Rogue AP Association	UDP	192.168.0.134	68	192.168.0.1	67	bt.:[reading from a file]	09/27 22:22:30

Figure 23

The hostname “LTP-ABC1234” is visible in the decoded payload section of the honeynet security console (figure 24).



Overview

Sensor	bt.:[reading from a file]	Interface	File	Event ID	9:2:112
Time	9/27/2008 10:22:30 PM	Classification			
Priority	High	Signature ID	1000061	Revision	

IP Header

Ver	Hdr Length	TOS	Length	ID	Flags	Offset	TTL	chksum
4	5	0	334	236	0	0	128	46811

Source IP Address	Destination IP Address
192.168.0.134	192.168.0.1

Source Hostname	Destination Hostname
	Unknown

Options

UDP Header

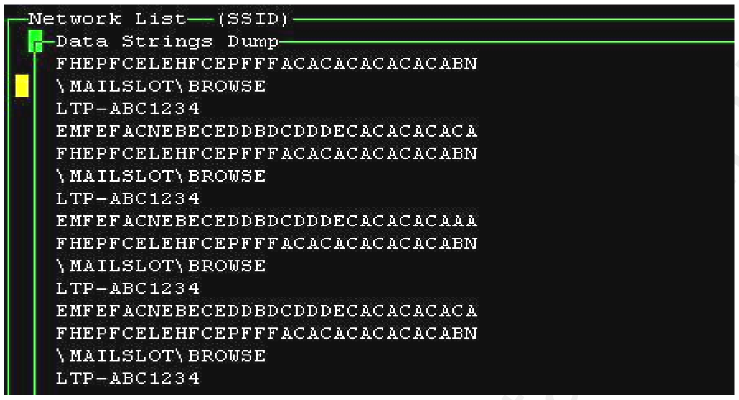
Source Port	Destination Port	Length	Checksum
68	67	314	17458

Options

```
TC$@(eXycSc5=eXyLTP-ABC1234QLTP-ABC1234.kMSFT 5.07,./!y+
```

Figure 24

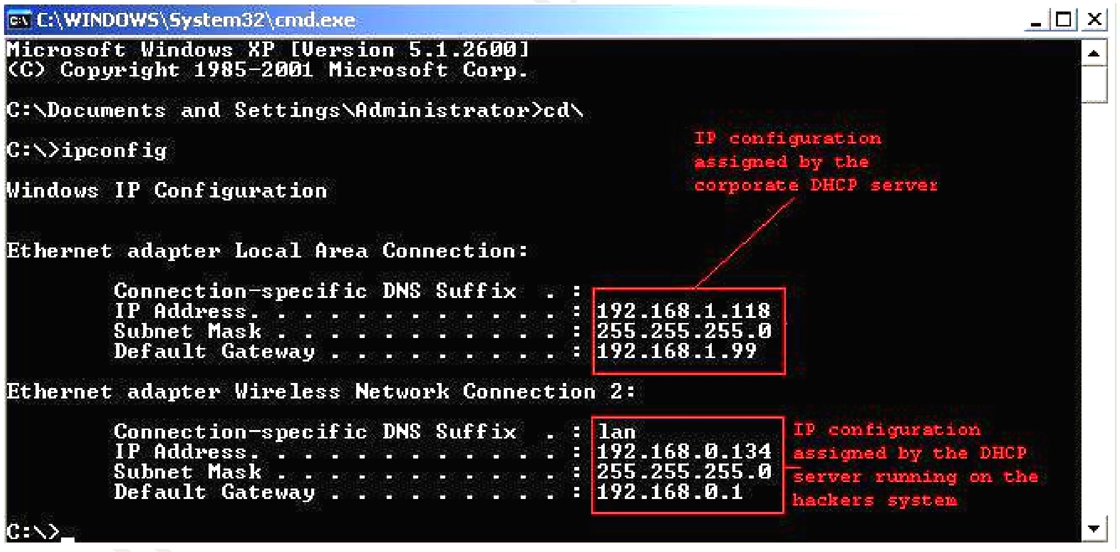
Since kismet and snort were running side by side, the clear text hostname announcements from the system “ltp-abc1234” can also be seen by using kismet (figure 25).



```
Network List—(SSID)
 Data Strings Dump
  FHEPFCELEHFCEPFFFACACACACACACABN
  \MAILSLOT\BROWSE
  LTP-ABC1234
  EMFEFACNEBECEDDBDCDDDECACACACACA
  FHEPFCELEHFCEPFFFACACACACACACABN
  \MAILSLOT\BROWSE
  LTP-ABC1234
  EMFEFACNEBECEDDBDCDDDECACACACAAA
  FHEPFCELEHFCEPFFFACACACACACACABN
  \MAILSLOT\BROWSE
  LTP-ABC1234
  EMFEFACNEBECEDDBDCDDDECACACACACA
  FHEPFCELEHFCEPFFFACACACACACACABN
  \MAILSLOT\BROWSE
  LTP-ABC1234
```

Figure 25

The following screen shot (Figure 26) shows Alice's IP configuration after associating with the Hacker's system.



```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd\

C:\>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.1.118
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.99

Ethernet adapter Wireless Network Connection 2:

        Connection-specific DNS Suffix  . : lan
        IP Address. . . . . . . . . . . . : 192.168.0.134
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.0.1

C:\>_
```

IP configuration assigned by the corporate DHCP server
IP configuration assigned by the DHCP server running on the hackers system

Figure 26

In the absence of an overlay monitoring solution this attack may have gone completely unnoticed. The user whose system is connected to the rogue access point is oblivious to the fact that the hacker is using the wireless interface on the user's system as a bridge to his or her company’s wired network.

From this point the hacker can access other systems on the network and use the compromised system as a jump point to launch or compromise other systems on the network. The hacker can now steal information from the company and also transfer information out of the company using backdoors and Trojans.

Exploiting client side vulnerabilities like the one just demonstrated shows how vulnerable users are, whether at home or at work.

Conclusion

The above simulations are just few examples on how an 802.11 network forensics solution can provide a proactive solution in mitigating sensitive data leakage and data theft. It is not the author’s intent to push for the use of free tools for setting up an 802.11 network forensics solution. Enterprises should plan and assess their requirements before deploying a wireless network forensics solution, whether opensource or commercial.

The main point of this entire document was to emphasize the main importance and the benefits of having an 802.11 network monitoring and content inspection solution. Companies are embracing wireless technology both for convenience and for cost. They have to realize that the frequency of data leakage and theft is constantly increasing and costing companies millions in lawsuits. There is a great need for profiling user activities in regards to who is doing what and what are they sending out of the company.

In today’s networks we depend strongly on technology to protect us from external or internal threats, and a solid network monitoring and forensics solution can help enterprises build a strong counter intelligence program.

References

1) Cox, K., & Greg, C. (2004). Snort and IDS tools . O'Reilly.

2) The Honeynet Project. (2004). Know Your Enemy (2nd ed.). Addison-Wesley Professional.

3) Perle. Retrieved March ,5,2008, Web site: http://perldoc.perl.org/perlre.html

4) RemoteFileSystemHowTo. Retrieved September,20,2008, Web site: http://wiki.openwrt.org/RemoteFileSystemHowTo

5) Kismet. Retrieved July,05,2008, Web site: http://www.kismetwireless.net/documentation.shtml

6) OpenWRT. Web site: http://openwrt.org/

7) Karma. Retrieved September 25, 2008, from Karma Web site: http://blog.trailofbits.com/karma/

8) Honeynet Security Console . Retrieved March 20, 2008, from activeworx Web site: http://www.activeworx.org/

9) Engage Packet builder. Retrieved Feburary 10, 2008, from Engage Security Web site: http://www.engagesecurity.com/products/engagepacketbuilder/

10) Snort. Retrieved January 5, 2008, from Snort - the de facto standard for intrusion detection/prevention Web site: http://www.snort.org/

11) TCPDUMP/LIBPCAP public repository. Retrieved March, 20 2008, from TCPDUMP/LIBPCAP Web site: http://www.tcpdump.org/

Appendix A

1) Bash script for extracting 15 digit credit numbers

```
#!/bin/bash
# Author : Akbar Qureshi
# Email: akbarqureshi@gmail.com
#
# Copyright (C) 2008 Akbar Qureshi
# All Rights reserved
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
  #
  #########################
  # PURPOSE OF THE PROGRAM#
  #########################
  # This Script will extract 15 digit Credit Card numbers.
  # Forensic Investigators can use this program to extract
  # credit card data.
  #
  #####################
  #    The SCRIPT     #
  #####################


      E_FILE_ACCESS=70
      E_WRONG_ARGS=71

      if [ ! -r "$1" ]     # Is the input file readable?
      then
       echo "Error! Can't read input file!"
       echo "Usage: $0 input-file output-file"

       exit $E_FILE_ACCESS

     fi                   # Will exit with same error

     exec < $1            # Will read from input file.


     exec > $2            # Will write to output file.

     strings $1|grep -P -o [0-9]{4}-[0-9]{6}-[0-9]{5}  # Regular Expressions

     exec 1>&2 2>&-

     echo  `cat $2|wc -l` Possible Credit Card '#' extracted from input file
  \"$1\"

     exit 0
```

2) Bash script for extracting 16 digit credit numbers

```
#!/bin/bash
# Author : Akbar Qureshi
# Email: akbarqureshi@gmail.com
#
# Copyright (C) 2008 Akbar Qureshi
# All Rights reserved
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
#########################
# PURPOSE OF THE PROGRAM#
#########################
# This Script will extract 15 digit Credit Card numbers.
# Forensic Investigators can use this program to extract
# credit card data.
#
#####################
#    The SCRIPT     #
  #####################

      E_FILE_ACCESS=70
      E_WRONG_ARGS=71

      if [ ! -r "$1" ]     # Is the input file readable?
      then
       echo "Error! Can't read input file!"
       echo "Usage: $0 input-file output-file"
       exit $E_FILE_ACCESS

     fi                   # Will exit with same error

     exec < $1            # Will read from input file.


     exec > $2            # Will write to output file.

     strings $1|grep -P -o [0-9]{4}-[0-9]{4}-[0-9]{4}-[0-9]{4}  # Regular
  Expressions

     exec 1>&2 2>&-

     echo  `cat $2|wc -l` Possible Credit Card '#' extracted from input file
  \"$1\"

     exit 0
```

Appendix B

The following steps were performed to mount the file system

1. The "traff-dump" directory was created on the snort IDS system using the following command

```
mkdir /tmp/traff-dump
```

2. The mount point on the access point was created by running the following command

```
mkdir /mnt/ap
```

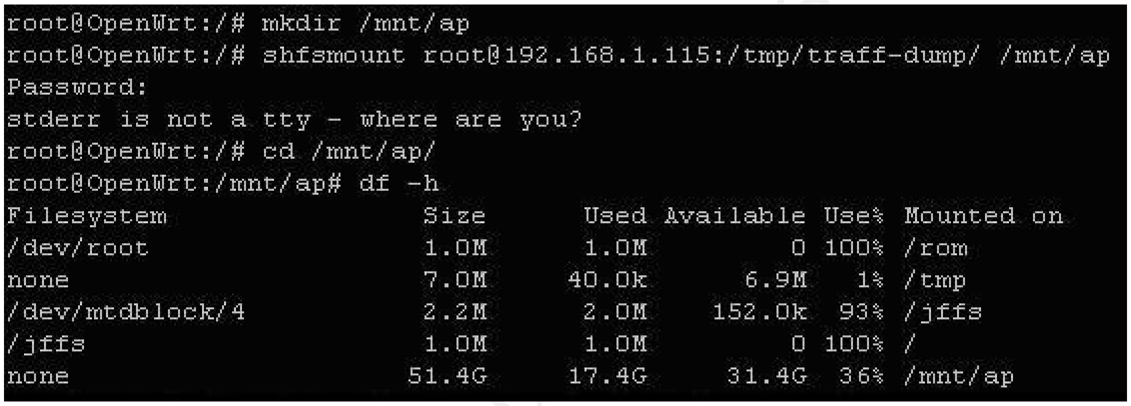
3. The following command was executed on the linksys access point to mount the remote directory "*/tmp/traff-dump/*" to the mount point */mnt/ap/* on the access point.

```
shfsmount root@192.168.1.115:/tmp/traff-dump/ /mnt/ap/
```

4.tcpdump was started and the traffic capture file was written to the mounted directory

```
tcpdump –n –s0 –w /mnt/ap/aircapture.pcap
```

The above mentioned steps are show in the screen shot below.



```
root@OpenWrt:/# mkdir /mnt/ap
root@OpenWrt:/# shfsmount root@192.168.1.115:/tmp/traff-dump/ /mnt/ap
Password:
stderr is not a tty - where are you?
root@OpenWrt:/# cd /mnt/ap/
root@OpenWrt:/mnt/ap# df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                 1.0M      1.0M         0 100% /rom
none                      7.0M     40.0k      6.9M   1% /tmp
/dev/mtdblock/4           2.2M      2.0M    152.0k  93% /jffs
/jffs                     1.0M      1.0M         0 100% /
none                     51.4G     17.4G     31.4G  36% /mnt/ap
```