GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia

Detecting and Responding to Data Link Layer Attacks

Author: TJ OConnor, terrence.oconnor@usma.edu

Advisor: Joel Esler

Abstract

Attacks against layer two, the 'data link-layer', range from address resolution protocol (ARP) cache poisoning for wired clients to de-authentication of wireless clients. Fairly simple to implement, these attacks can often go unnoticed by intrusion analysts since intrusion detection systems typically look at the network layer and above to detect attacks. This paper examines how packet manipulation tools such as Scapy can be used to examine network traffic for data link layer attacks and proactively respond to attacks against the data link layer. To accompany this paper, I will publish a paper about a light set of tools to implement the detection mechanisms.

1. Introduction

1.1.Intrusion Detection and Prevention Systems

In this paper, we examine techniques for identifying signatures and anomalies associated with attacks against the data link layer on both wired and wireless networks. Methods for signature-based detection and anomaly-based detection are not new. Intrusion detection systems such as SNORT are quite capable of detecting some of the known data link layer attacks and include a mechanism for integrating Intrusion Prevention System (IPS) solutions. This paper does not advocate against the use of these solutions in organizations. What we present can augment your existing capabilities by detecting attacks that may be blind to your IDS.

We present methods for the average user who cannot afford either the time or money to deploy a Wireless Intrusion Detection System (WIDS) or afford to manage IDS sensors behind every layer two device. Imagine the scenario of a user who plugs his laptop into a hotel room connection and begins suspecting malicious activity. The ability to quickly code a script to detect such activity and the presence of a man-in-the-middle (MITM) tool such as ettercap-NG would be extremely helpful. Writing a tool in under thirty lines of code that mitigates the threat of a MITM would be even better. We present both in the following paper.

Before we begin analyzing different coding techniques for identifying and mitigating threats, we must first analyze how the threats against the data link layer uniquely differ from threats against the network, transport and application layers.

1.2. How Does the Data Link Layer Differ?

The data link layer differs from other layers of our protocol design because it is a trusted layer. We can use filtering, access control lists, authentication, and application controls to limit access at the higher layers of our design. In contrast, the data link layer lacks the fine-grain controls to prevent data link layer attacks from occurring.

TJ OConnor, terrence.oconnor@usma.edu

2. Background

2.1. Taxonomy of Data Link Layer Attacks

The following section reviews some of the different methods an attacker may use to attempt to attack the data link layer. In all cases, an adversary attempts to compromise confidentiality, authentication, or availability of information. The attacks succeed for the most part because of the lack of fine-grain controls for the data link layer. While layer 2 is considered a less novel platform for attacks, layer 2 attacks continue to trouble our networked systems. The implementation of each attack is unique. Yeung, Fung, and Wong (2008) enumerated several of the different tools used to implement layer 2 attacks. However, all of the tools rely on the lack of proper authentication during layer 2 communication. For wired data link layer attacks, we examine CAM Table Exhaustion Attacks, ARP Spoofing, DHCP Exhaustion Attacks, and VLAN Hopping. On the wireless data link layer, we will examine the hidden node attack, deauth, and fake access-point attacks against our wireless devices.

2.2. Data Link Layer Attacks on Wired Networks

2.2.1 CAM table exhaustion

At the second layer of the TCP/IP model, a switch delivers Ethernet frames based on the physical medium access control (MAC) address. A content address table (CAM) table maintains a list of the switch ports and the destination MAC addresses by port. This table enables the switch to uniquely deliver information to the intended physical address. By delivering frames based on the MAC address, a switch offers considerable security over a hub. A hub simply broadcasts frames to all ports. An eavesdropper on a hub can listen to the traffic of anyone else connected to the hub.

A CAM Table Exhaustion attack essentially turns a switch into a hub. To succeed, an attacker floods the CAM Table with new MAC-port mappings. When the CAM table fills up beyond the fixed memory, it no longer knows how to deliver based on MAC-port bindings. Therefore, it begins broadcasting Ethernet frames to maintain the flow of traffic. Once the switch begins broadcasting, any connected adversary can hear the traffic that flows through the switch.

TJ OConnor, terrence.oconnor@usma.edu

The *macof* tool, depicted in Figure 1, implements a CAM table exhaustion attack by flooding a switched LAN with random MAC addresses. Dug Song created *macof* as part of the *dsniff* series of tools that attack the data link layer of the TCP/IP model. Doug Song wrote the entire *dsniff* package suite, which includes two other active layer 2 attack tools – arpspoof and and dnspoof and five other passive tools (filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspy). The entire toolkit is available for download at http://www.monkey.org/~dugsong/dsniff/.

Notice in Figure 1 that macof supports setting the IP source and destination address with the –s and –d flags as well as setting the target hardware destination address with the –e flag and the source and destination ports with the –x and –y flags. Finally, the attacker can specify the interface and number of times to send the attack. Dsniff will generate random values for any unspecified values. In this example, we are attacking a switch at hardware address AA:DE:AD:BE:EF:00.

```
root@bt:~# macof -h
Usage: macof [-s src] [-d dst] [-e tha] [-x sport] [-y dport] [-i interface] [-n times]
root@bt:~# macof -s 192.168.1.100 -d 192.168.1.1 -e AA:DE:AD:BE:EF:00
```


Figure 1: *Macof* Flooding a Switch With Random MAC Addresses

TJ OConnor, terrence.oconnor@usma.edu

The CAM Table Exhaustion attack succeeds because there is no authentication when clients broadcast their physical address. Because of this, any user can essentially pretend to be anyone else. In a CAM Table Exhaustion attack, an adversary pretends to be hundreds or thousands of random users. However, an adversary only needs to mimic two addresses to succeed in the next attack.

2.2.2 ARP spoofing

The Address Resolution Protocol (ARP) translates logical Layer 3 addresses (IP Addresses) to layer 2 addresses (physical MAC addresses). On a switched network that relies on the physical address for delivery, clients must maintain an updated table of logical-to-physical address bindings. If unsure of a physical address binding, a client may broadcast an ARP message, asking for the MAC address for a given IP address. Further, clients may broadcast gratuitous ARP messages.

When a client or switch receives a gratuitous ARP message, it updates its ARP table with the new physical-to-logical binding. The next time it has traffic to send, it sends it based on the new physical address located in its table. An adversary can take full advantage of this by broadcasting a gratuitous ARP for any of its neighbor's IP address with the adversary's own physical MAC address. All neighbors in the switched environment will start delivering traffic to the adversary instead of the intended recipient. Thus, the adversary successfully performs a man-in-the-middle attack as depicted in Figure 2.
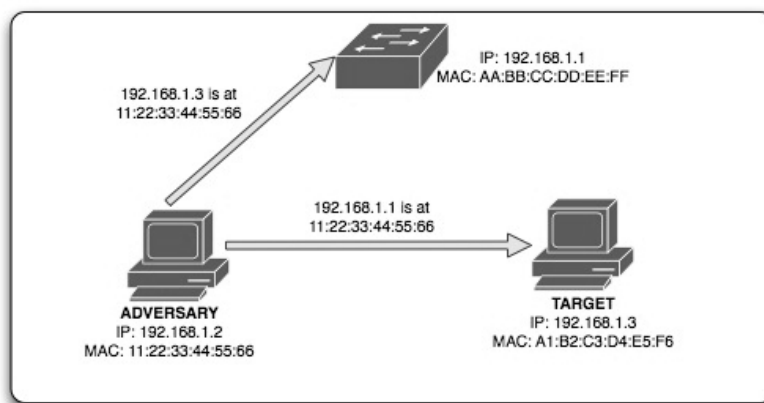

Figure 2: Man-in-The-Middle Attack against a Switched Environment

TJ OConnor, terrence.oconnor@usma.edu

This attack succeeds until the client under attack realizes it is not receiving any traffic and offers a gratuitous ARP itself to the network. The adversary counters with another gratuitous ARP. This tug-of-war between adversary and the client under attack is known as an ARP storm. The ARP storm that follows can consume the network traffic as the two clients' battle back and forth, successfully causing a denial of service to the clients on the switched network.

2.2.3 DHCP starvation attacks

When a client without an Internet protocol (IP) address enters a network, he may choose to contact the DHCP server and request an address. If the network supports DHCP, the server will respond with an address and the lease period of time for the address. This layer two handshake is usually done in an unauthenticated or unencrypted mode.
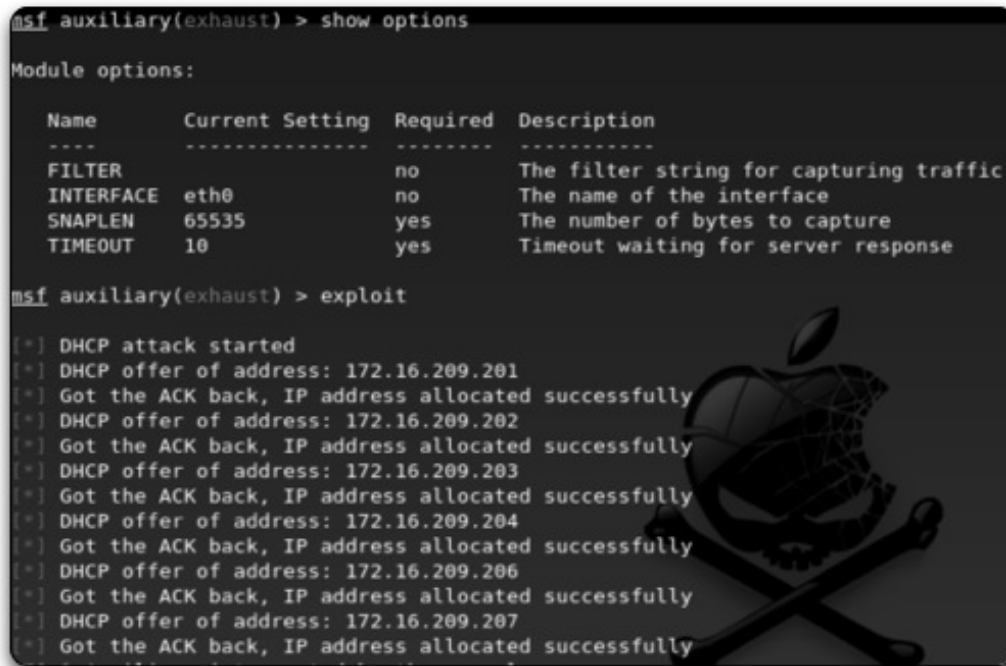
An attacker may wish to take advantage of DHCP by flooding the network with requests for addresses. Presented at Defcon 16 as a DHCP Starvation Attack, this attack consumes the leasable IP address space by the DHCP server. After a successful attack, the DHCP server will not be able to offer addresses to any future clients that join the network. Robin Wood , DigiNinja, created an auxiliary MetaSploit module to perform this specific attack. The entire Metasploit framwork, which is used in other attacks in this paper, is available at http://www.metasploit.com and includes installation instructions at http://www.metasploit.com/framework/support/. The specific DigiNinja module is available for download at http://www.digininja.org/metasploit/dns_dhcp.php. Figure 3 shows the DigiNinja DHCP attack exhausting the possible leased addresses from a server.

We implement this attack by starting the metasploit console (*msfconsole)* and then activating the auxiliary dhcp exhaustion module. The next command (*show options*) will list the available options, including the interface to perform the attack on and the filter, snaplen, and timeout for capturing responses. Note, these last three options are a result of DigiNinja using the pcaprub library to forge packets and therefore these options are not important for the actual attack we are performing. Finally, to launch the attack at the metasploit console, we enter *exploit*. After launching the attack, you will notice that the

TJ OConnor, terrence.oconnor@usma.edu

module quickly sends several DHCP requests and receives responses until the DHCP pool is exhausted.

To make matters worse, an adversary can then stand up a rogue DHCP server and begin answering DHCP requests. This will allow the rogue DHCP server to assign IP addresses and gateways, allowing the possibility to man-in-the-middle monitoring of all future traffic.

```
root@bt:~# msfconsole
…
msf> use auxiliary/digininja/dhcp_exhaustion/exhaust
msf> show options
…
msf> exploit
```


Figure 3: *DigiNinja*'s DHCP Exhaustion Attack MetaSploit Module

2.2.4 VLAN hopping attacks

In VLAN Hopping, an attacker generates traffic with a VLAN ID of an end system it cannot normal reach and sends the traffic. Further, the attacker may try to

TJ OConnor, terrence.oconnor@usma.edu

imitate a switch in order to negotiate trunking and send and receive traffic between VLANs. The MetaSploit framework includes a single module to perform VLAN hopping against vulnerable firmware switches. Figure 4 shows this module, with the options to choose the VLAN ID and RMAC for spoofing. Figuerora presented the problems posed by VLAN layer 2 attacks at Defcon 16 (Figueroa, 2007).

To implement this attack, we again start the Metasploit framework with the msfconsole command. Next, we activate the pvstp attack module by enetering the command *use auxiliary/spoof/cisco/pvstp.* To see the available options for this attack, enter *show options.* Finally, we will set the target VLAN ID as 7 (*set VID 7).* After settting any options, we finalize our settings with one final command: *set.* Following that we launch our exploit.

```
root@bt:~# msfconsole
…
msf> use auxiliary/spoof/cisco/pvstp
msf> show options
…
msf>set VID 7
msf>set
msf> exploit
```


Figure 4: MetaSploit Cisco PVSTP Vlan Hopping Attack Module

TJ OConnor, terrence.oconnor@usma.edu

2.3. Data Link Layer Attacks on Wireless Networks

The following section examines some of the attacks unique to Wireless Networks. Examining wireless attacks and defining signatures is not novel. As early as 2003, Josh Wright identified the signatures behind several of these attacks and their tool implementations. However, this paper addresses methods for implementing both detection and mitigation capabilities in scripting languages. Before examining their implementation, this section discusses wireless data link layer attacks including MAC spoofing, the exploitation of the hidden node problem, deauthentication attacks, and the creation of fake access points.

2.3.1 Hidden node attack

Wireless clients share a common medium and therefore maintain a schema that ensures clients share that medium. However, this presents unique challenges on wireless networks as depicted in Figure 5. Here, two wireless nodes, A and C, both have information to transmit. Nodes A and C cannot hear each other but they both can hear B. Before A transmits its data, it broadcasts a Ready-To-Send (RTS) message. Node B, upon receiving the message, acknowledges that the frequency is free and broadcasts a Clear-To-Send (CTS). Node C, hearing the CTS from B, backs off from transmitting. Although Node C did not hear the original CTS from Node A, it knows that Node B is communicating with the hidden Node A.
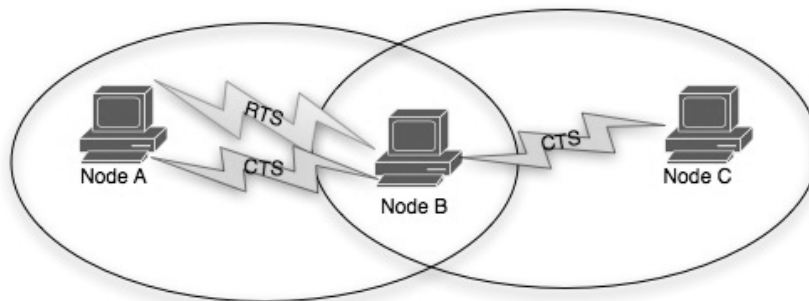
Figure 5: CTS/RTS Signaling Solution for the Hidden Node Problem

Thus, the CTS/RTS message scheme helps prevent collisions on the wireless network medium. However, it also presents a unique opportunity for an attacker. An attacker can

TJ OConnor, terrence.oconnor@usma.edu

abuse the CTS/RTS scheme by flooding the network with RTS or CTS messages. In the case of a CTS Flood, all listening neighbors will assume there is a hidden node that is transmitting and back off. In the case of an RTS flood, neighboring nodes will respond with a CTS, which forces all of their neighbors to back off as well.

Several tools implement this attack. It can be implemented by a programmer in a couple of lines of C code or Python with Scapy. The latest version of MetaSploit even includes a CTS_RTS_Flood as an auxiliary module. Figure 6 shows how an attacker can use MetaSploit to attack neighboring wireless clients. Notice the attacker has the option of either using a CTS or RTS Flood as both will have a negative impact on the ability for 802.11 devices to communicate. To implement this attack, we will use start with the Metasploit framework, importing the *auxiliary/dos/wifi/cts_rts_flood* module and setting the hardware source address as DE:FA:CE:D0:00 and intended target as AA:DE:AD:BE:EF. Following that we finalize our options with *set* and then run the *exploit.*

```
root@bt:~# msfconsole
…
msf> use auxiliary/dos/wifi/cts_rts_flood
msf> show options
…
msf>set ADDR_SRC DE:FA:CE:D0:00
msf>set ADDR_DST AA:DE:AD:BE:EF
msf>set
msf> exploit
```


Figure 6: CTS/RTS Flood MetaSploit Module

TJ OConnor, terrence.oconnor@usma.edu

2.3.2 Deauth attacks

On wireless networks, clients authenticate themselves to the access point (AP). This authentication process can use security protocols such as RADIUS, EAP, or LDAP to authenticate the client. After authentication, the client is associated with the AP. To dissociate from the AP, the client sends a management frame known as a deauthentication frame. The AP, hearing the deauth frame dissociates the client. The client must authenticate to the AP if it wishes to associate with the AP. The deauth frame differs from the authentication process because it requires no security.

The *auxiliary/dos/wifi/deauth* module in Metasploit is one method for spoofing deauth frames. Another options is to use the aircrack-NG suite, which includes a variety of tools for attacking 802.11 networks. The aircrack-NG suite is available for download at http://www.aircrack-ng.org/ and includes documentation on usage.

```
root@bt:~# msfconsole
…
msf> use auxiliary/dos/wifi/deauth
msf> show options
…
msf>set ADDR_BSS AC:CE:55:13:37
msf>set ADDR_SRC AC:CE:55:13:37
msf>set ADDR_DST AA:DE:AD:BE:EF
msf>set
msf> exploit
```

```
msf auxiliary(deauth) > show options

Module options:

    Name          Current Setting    Required  Description
    ----          ---------------    --------  -----------
    ADDR_BSS      00:DE:AD:BE:EF:00  yes       BSSID (e.g 00:DE:AD:BE:EF:00)
    ADDR_DST      00:DE:AD:BE:EF:00  yes       TARGET MAC (e.g 00:DE:AD:BE:EF:00)
    ADDR_SRC      00:DE:AD:BE:EF:00  yes       Source MAC (e.g 00:DE:AD:BE:EF:00)
    CHANNEL       11                 yes       The initial channel
    DRIVER        autodetect         yes       The name of the wireless driver for lorcon
    INTERFACE     wlan0              yes       The name of the wireless interface
    NUM           31337              yes       Number of frames to send

msf auxiliary(deauth) > exploit

[*] Creating Deauth frame with the following attributes:
[*]    DST: 00:DE:AD:BE:EF:00
[*]    SRC: 00:DE:AD:BE:EF:00
[*]    BSSID: 00:DE:AD:BE:EF:00
[*] Sending 31337 frames.....
```

Figure 7: MetaSploit 802.11 Deauthentication Attack Module

TJ OConnor, terrence.oconnor@usma.edu

To implement the Metasploit deauth module, set the BSS address, and source and destination hardware addresses for the attack. After setting all our variables, *set* and then launch the *exploit*.

Knowing that a deauth frame lacks security, an attacker can spoof a deauth frame from a client to the AP. This will disconnect the client at the very least, forcing him to authenticate again. An attacker can implement a Denial of Service attack against the client by flooding the network with deauth frames. A MetaSploit Module, depicted in Figure 7, depicts this attack. However, the adversary may also have an alternative motive for spoofing the deauth frame. The attacker may wish to obverse the client authentication process in order to break the shared wireless key. By brute forcing the key out of an authentication handshake, the attacker can gain access to the network himself.

2.3.3 Fake access point attacks

A tool such as Black Alchemy's FakeAP can generate thousands of counterfeit 802.11 access points by spoofing the 802.11 beacon frame that advertises an AP. An adversary could use such a tool to cause problems with war-driving tools that map wireless networks, or use it during an infrastructure review in order to find rogue access points.

An even more malicious use of a fake AP resides in KaraMetaSploit. This tool combines both the ability to advertise fake APs by Karma and integration with the MetaSploit framework of exploits. Once connected to a fake AP, the MetaSploit engine takes over the launching of an automated attack (autopwn) against the unsuspecting user. This technique is made even more insidious because it does not wait passively for a user to connect but, instead, actually answers all 802.11 probe requests. Security experts from the Hak5 podcast show built a hardware platform the size of a small access point capable of performing this attack.

2.4. Current Methods for Detection and Prevention

Different vendors provide unique security solutions for their layer 2 products. For example, Cisco provides several different proprietary configuration settings such as *port-*

TJ OConnor, terrence.oconnor@usma.edu

security, secure MAC addressing, and *BPDU guards* to prevent attacks against the data link layer. It also offers DAI, which verifies IP-to-MAC address bindings and discards invalid layer 2 packets. These proprietary mechanisms are effective and proven at stopping layer 2 attacks. But what happens when you are not behind a proprietary device with layer 2 security? What about the case of a hotel-wired or wireless network? How can we prevent layer 2 attacks?

A client is fully capable of keeping a list of gratuitous ARP messages it has received or deauthentication frames it has overhead and using that to make a decision if it is under attack. By using a set of scripts, a user can examine traffic to discover signatures for layer 2 attacks. This is not a novel idea. Josh Wright wrote a series of Perl scripts to discover wireless attacks (2003). In one example, he identified the Wellenreiter wireless discovery toolkit that spoofed MAC Addresses of phony wireless clients. Since the tool used a specific MAC address range, Josh could detect Wellenreiter and know when there was an attacker using the toolkit in the vicinity of his machine.

3. Methodology

3.1.Why Roll Your Own Tools?

Tools already exist to detect Layer 2 attacks. In fact, as early as 2004, Valli integrated a Wireless Layer 2 detection capability into the SNORT intrusion detection system. Hsieh, Lo, Lee, and Huang even integrated a response capability to redirect layer 2 attacks and quarantine them to a honeypot (2004). The problem that persists is that a tool must be up and running and in place prior to the attack. Layer 2 attacks are restricted to either locality (by signal strength) or network segment (by switching capability). Therefore, either an intrusion detection system and response capability must be running on the host or on a network asset in range of responding. Placing a full-intrusion prevention system on an individual host has challenges.

This paper and these tools do not attempt to replace capabilities provided by those tools. Instead, this paper suggests ways to write some quick scripts with the capability to detect an attack and respond. Essentially, this paper proposes an incident-handling

TJ OConnor, terrence.oconnor@usma.edu

response to attacks in progress on machines lacking an IDS or IPS capability. To understand what we propose, we must first set up the environment.

3.2. Setting Up Your Environment.

We will establish our development environment to begin rolling our own tools. Python, a high-level language with an abundance of libraries and modules, provides a great starting point. At the time of this article, the Python language has forked with two major releases, 2.x and 3.x. The 3.x code branch does not offer full backward compatibility with the 2.x code branch, and therefore, the abundance of external libraries and modules occasionally do not work with 3.x. Therefore, for this document all the code examples will support the 2.x syntax. For updated information, see http://www.python.org/. Since the majority of the code we will write is limited to 25 lines of a script, we will only use a text editor such as Vim, Notepad++, or Emacs to edit the code. However, full development suites like Eclipse provide an interface to the Python Programming Language for larger projects.

One tool built for the particular problem we are examining is Scapy. A powerful packet-manipulation library, Scapy can analyze several different protocols and display the exact packet or frame. This can prove rather useful as we start examining different frames at the data link layer and their malicious deviations from the standard protocol. With a little effort, Scapy works on a multitude of operating systems, including Windows, Mac OS X, and Linux. For more information, see the developer's site at http://www.secdev.org/projects/scapy/.

3.3. Develop Your Own Tools to Detect Wired Layer 2 Attacks

3.3.1 CAM table exhaustion detection

Detecting a CAM Table Exhaustion Attack is a logical starting point. As we have mentioned, a CAM Table Attack attempts to overflow a layer 2 switch's table of MAC to IP translations by flooding it with excessive sources. Thus, to detect it, we would need to be at a logical point to observe it like a span port on the switch or a passive network tap en route to the switch. Figure 8 shows a small Python script to detect a CAM Table

TJ OConnor, terrence.oconnor@usma.edu

Exhaustion Attack. Running this script during a MacOf flood will display a detection message.

```python
import sys
import scapy
import datetime
from scapy.all import *
THRESH = (254/4)
START  = 5

def monitorPackets(p):
        if p.haslayer(IP):
                        hwSrc = p.getlayer(Ether).src
                        if hwSrc not in hwList:
                                hwList.append(hwSrc)
                        delta = datetime.datetime.now() - start
                        if ((delta.seconds > START) and ((len(hwList)/delta.seconds) > THRESH)):
                                print "[*] - Detected CAM Table Attack."

interface = sys.argv[1]
hwList = []
start = datetime.datetime.now()


sniff(iface=interface,prn=monitorPackets)
```

Figure 8: CAM Table Exhaustion Detection Script

To detect this attack, we simply need to observe some threshold number of different MAC addresses during a specified period of time we consider excessive. In our script below, we will use a threshold of 254 addresses in 4 seconds. After sniffing each packet, we will strip off the Ether layer and add the MAC address to an array. If we get more than 254 addresses in 4 seconds, we will print a detection message to the screen.

Certainly there are other methods for detecting this attack. For example, we could maintain a list of hardware addresses that should be associated with this switch and then display detection messages when physical addresses outside of our preapproved list show up on the MAC table.

3.3.2 ARP Spoofing Detection

Next, we will examine a script to detect spoofing of the address resolution protocol (ARP). In ARP Spoofing, an attacker broadcasts gratuitous messages redirecting the logical-to-physical bindings for address tables. As a result, messages get (at least

TJ OConnor, terrence.oconnor@usma.edu

temporarily) redirected to the attacker instead of the victim. Eventually, the victim rebroadcasts his logical-to-physical binding.

We will use this ping-pong effect to detect the ARP binding. If we see an IP address that is now associated with a new MAC address, we will display a detection message to the screen. In this scenario, a user can detect both himself as a victim or another target as a victim.

If the user sees a destination under attack that he communicates with, he can respond by setting a static-ARP binding for that address. Running this tool both detects and negates tools like arpspoof. Figure 9 shows a detection script for ARP spoofing in fewer than thirty lines of Python code.

```
import sys
import scapy
from scapy.all import *
import datetime

global hwTable
global conCnt
hwTable = {}

def monitorPackets(p):
    global hwTable
    if (p.getlayer(ARP).op==2):
        hwSrc=p.getlayer(ARP).hwsrc
        ipSrc=p.getlayer(ARP).psrc
        if ipSrc in hwTable:
            if (hwSrc != hwTable[ipSrc]):
                print "[*] - Detected ARP Conflict for IP: "+ipSrc
                print "[*] - New: "+hwSrc+" Old: "+hwTable[ipSrc]
        hwTable[ipSrc]=hwSrc


interface=sys.argv[1]
sniff(iface=interface,filter="arp",prn=monitorPackets)
```

Figure 9: ARP Spoofing Detection Script

3.3.3 DHCP Exhaustion Detection

Detecting a tool like DigiNinja's DHCP Exhaustion Module is relatively easy and similar to detecting a CAM Table Exhaustion Flood. Essentially, we want to count how many DHCP Requests occur within a specific period of time. If that ratio exceeds our threshold, we consider it an attack against the leasable address space.

To detect a DHCP request message, we must understand the layout of a BOOTP frame as depicted in Figure 10. Notice the packet has a 1-Byte Operations Code that

TJ OConnor, terrence.oconnor@usma.edu

defines the BootP operation. During a request, the client sets the OPCODE to 01. A

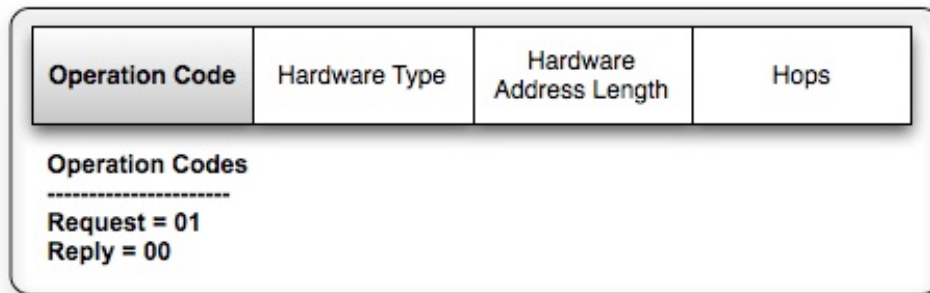DHCP server replies with a DHCP offer and OPCODE value of 00.

Figure 10: Layout of BootP Frame and Operation Codes

Therefore, we can write a small script to detect DHCP requests by looking for frames

containing the BOOTP layer. Figure 11 shows our detection script for excessive DHCP

request. If the Requests exceed our threshold count, we can display a detection message.

Further, we could respond by attempting to detect the MAC address scheme used by the

attacker and blacklist it at the server. Notice this is a very common attack performed in

hotel-area-networks by hackers to restrict access to limited bandwidth access points.

```
import sys
import scapy
from scapy.all import *

global reqCnt
global ofrCnt
reqCnt = 0
ofrCnt = 0

def monitorPackets(p):
    if p.haslayer(BOOTP):
            global reqCnt
            global ofrCnt
            opCode = p.getlayer(BOOTP).op
        if opCode   == 1:
                reqCnt=reqCnt+1
        elif opCode == 2:
                ofrCnt=ofrCnt+1
        print "[*] - "+str(reqCnt)+" Requests, "+str(ofrCnt)+" Offers."

interface=sys.argv[1]
sniff(iface=interface,prn=monitorPackets)
```

Figure 11: DHCP Exhaustion Attack Detection Script

TJ OConnor, terrence.oconnor@usma.edu

3.4. Develop Your Own Tools to Detect Wireless Layer 2 Attacks

To detect wireless layer 2 attacks, we must first examine the frame control contents depicted in Figure 12. For each 802.11 Frame, the Frame Control contains a Type (Management, Control, and Data) and a Subtype. For the Management Type, these Subtypes include Beacons, Disassociation, or Deauth Subtypes. For Control Subtypes, this includes CTS and RTS Subtypes. We will use these types and subtypes to filter frames and detect attacks in the following examples.

Figure 12: 802.11 Frame Control Contents

Figure 13: Layout of 802.11 Frame Structure

3.4.1 Hidden node attack detection

In detecting hidden node attack, we will set a threshold number of CTS or RTS frames that should occur in a given period of time. If we count more CTS frames than should occur, we can display a detection message. Because the CTS/RTS solution to the

TJ OConnor, terrence.oconnor@usma.edu

hidden node problem should use an exponential back-off scheme, even a saturated network should not have excessive CTS/RTS frames.

In the script, depicted in Figure 14, we parse out frames containing the 802.11 header information and control type. Then frames containing the CTS (11) and RTS (12) control subtypes are counted. If we exceed the threshold number for either, the script prints a detection message. Unfortunately, a response capability to this attack is limited because an attacker could easily spoof a legitimate MAC address from the network and use it to broadcast CTS/RTS. Some wireless access points employ anti-CTS/RTS Flooding technology by increasing signal strength. A response to the CTS/RTS most likely will involve physically removing the attacker from the range of the victims.

```python
import sys, scapy, datetime
from scapy.all import *
THRESH =(25/5)
START = 5
global rtsCNT
global ctsCNT

def monitorPackets(p):
    global rtsCNT
    global ctsCNT
    if p.haslayer(Dot11):
        delta=datetime.datetime.now()-start
        if (p.getlayer(Dot11).subtype) == 11:
            rtsCNT = rtsCNT +1
            if ((delta.seconds > START) \\
             and ((rtsCNT/delta.seconds) > THRESH)):
                print "[*] - Detected RTS Flood."
        elif (p.getlayer(Dot11).subtype) == 12:
            ctsCNT = ctsCNT + 1
            if ((delta.seconds > START) \\
             and ((ctsCNT/delta.seconds) > THRESH)):
                print "[*] - Detected CTS Flood."

ctsCNT = 0
rtsCNT = 0
interface=sys.argv[1]
start = datetime.datetime.now()
sniff(iface=interface,prn=monitorPackets)
```

Figure 14: CTS/RTS Flood Detection Script

3.4.2 Deauth flood detection

Robin Wood, DigiNinja, presented a script to detect deauth attacks against wireless networks on the Hak5 podcast in May 2010. Developed separately, our script is

TJ OConnor, terrence.oconnor@usma.edu

very similar since detecting a death attack simply entails observing the Management Type (0) and Deauth Subtype (12). In figure 15, we show our script that detects wireless deauth attacks.

Again, we can specify a threshold of deauth frames that we consider to be a flood. Another options is to list hardware addresses of our machines and watch for deauth messages against them specifically. Detection of a significant deauth attack to disrupt service might suggest that we should consider replacing our wireless infrastructure with a RADIUS compliant solution that could use a management frame protection mechanism such as a shared secret to prevent spoofing management frames such as the deauth frame.

```python
import sys, scapy, datetime
from scapy.all import *
THRESH =(25/5)
START = 5
global deauthCNT

def monitorPackets(p):
    global deauthCNT
    if p.haslayer(Dot11):
        type = p.getlayer(Dot11).type
        subtype = p.getlayer(Dot11).subtype
        if ((type==0) and (subtype==12)):
            deauthCNT = deauthCNT + 1
            delta = datetime.datetime.now()-start
            if ((delta.seconds > START) and ((deauthCNT/delta.seconds) > THRESH)):
                print "[*] - Detected Death Attack: "+str(deauthCNT)+" Dauth Frames."

deauthCNT = 0
interface=sys.argv[1]
start = datetime.datetime.now()
sniff(iface=interface,prn=monitorPackets)
```

Figure 15: 802.11 Deauth Attack Detection Script

3.4.3 Fake access point detection

Various methods exist to detect fake access points. For example, Josh Wright wrote a series of Perl scripts to lookup the OUI offset of the MAC address in a database to ensure it was a registered OUI and not a spoofed MAC. Additionally, commercial wireless intrusion detection systems look at the signal strength. We propose one method for detecting the fake access tool included with MetaSploit.

The FakeAP module in MetaSploit broadcasts phony wireless access points. To do this, it creates an 802.11 Management frame with a subtype of 8 (Beacon). This frame

TJ OConnor, terrence.oconnor@usma.edu

subtype includes a timestamp for clients to sync with the access point. Logically, this timestamp should grow incrementally. However, the fakeAP tool spoofs random timestamp information. Therefore, it is easy to detect.

The script depicted in Figure 16 removes the BSSID from each beacon along with the timestamp. It creates an array of timestamps for each BSSID and ensures they are growing. If the timestamps are out of order past a certain threshold, it displays a message indicating that the FakeAP tool has been detected.

```python
Import sys, scapy, datetime
from scapy.all import *
from sets import Set

THRESH = 5
global ssidDict
global ssidCnt

def monitorPackets(p):
    global ssidDict
    global ssidCnt
    if p.haslayer(Dot11):
        if (p.getlayer(Dot11).subtype==8):
            ssid = p.getlayer(Dot11).info
            bssid = p.getlayer(Dot11).addr2
            stamp = str(p.getlayer(Dot11).timestamp)
            if bssid not in ssidDict:
                ssidDict[bssid] = []
                ssidCnt[bssid]=0
            elif (long(stamp) < long(ssidDict[bssid][len(ssidDict[bssid])-1])):
                ssidCnt[bssid]=ssidCnt[bssid]+1
                if (ssidCnt[bssid] > THRESH):
                    print "[*] - Detected fakeAP for: "+ssid
            ssidDict[bssid].append(stamp)

interface=sys.argv[1]
ssidDict = {}
ssidCnt = {}
start = datetime.datetime.now()
sniff(iface=interface,prn=monitorPackets)
```

Figure 16: Fake Access Point Detection Script

4. Conclusions

In this paper, we examined techniques for identifying signatures and anomalies associated with attacks against the data link layer on both wired and wireless networks. We presented methods and scripts for the average user who can afford neither the time nor the money to deploy a Wireless Intrusion Detection System (WIDS) or afford to manage IDS sensors behind every layer two device. The idea of using scripts to detect

TJ OConnor, terrence.oconnor@usma.edu

attacks solves the problem of a user with degraded service at a hotel area network or a professor visiting a conference overseas that wants to determine if he is being arpspoofed. However, we do not advocate using scripts as a long-term solution to intrusion detection needs or intrusion prevention solutions. Certainly, it does not scale to enterprise. Writing scripts solves the off-by-one problem and can assist with users detecting novel attacks or identifying existing attacks against their layer 2 infrastructure.

We have demonstrated that these scripts, typically less than 30 lines, can easily detect wireless and wired attacks. This includes detecting attacks such as arpspoofing, dhcp exhaustion, cam table exhaustion, deauth attacks, fake access points, and hidden node attacks. The responses to these attacks can be integrated into scripts as well, including setting static ARP tables, counter-broadcasting gratuitous ARP and notifying users of fake access points.

5. References

Figueroa, M. (2007). *VLAN layer 2 attacks: their Relevance and their kryptonite. Poster session presented at Defcon 16*, Las Vegas, NV.

Hsieh, W., Lo C., Lee J., & Huang, L. (2004). The implementation of a proactive wireless intrusion detection system. *Computer and Information Technology*, 581-586.

Valli, Crag. (2004, November). Wireless SNORT – *A WIDS in progress*. Proceedings of the 2nd Australian Computer, Network and Information Forensics Conference, Perth, Western Australia.

Wright, J. (2003). *Detecting wireless LAN MAC address spoofing.* Retrieved June 29, 2010, from http://www.ecsl.cs.sunysb.edu/~fanglu/wlan_spoof_detection.htm

TJ OConnor, terrence.oconnor@usma.edu

Yeung, K., Fung, D., & Wong, K. (2008). Tools for attacking layer 2 network infrastructure. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008, II*.

TJ OConnor, terrence.oconnor@usma.edu