Using the TUN/TAP device driver for network reconnaissance, pivotal gains, and leveraging access to otherwise “restricted” network areas.

“*Space-time is strongly curved near a black hole, so one has to be very careful about definitions. In particular, the radius of a black hole cannot be defined as the distance from the central singularity to the event horizon, because observers trying to measure this would inevitably fall into the singularity*.”
Edward L. Wright

This paper will cover several attack vectors. One may find malfunctions or weaknesses in software & hardware, allowing for the path to be taken. Remember, an attack vector is the path taken to deliver malice.

Reading through old RFC's is something I enjoy, and consider myself to be somewhat of an Internet historian. Indeed the Internet works as you think it should, coincidentally it works backwards too. Society seems to be up in arms over security. Yet, how can you be secure when you are fundamentally flawed?

Prior to using the TUN/TAP interface one must understand the infrastructures in which they are attacking. While performing personal research, I noticed an inherent similarity between MAC (media access control) addresses, in which VLAN1 (virtual local area network) and VLAN2 had only differed by a few bits. Coincidentally, I also noticed the similarity between the MAC of the external network, and the internal addressing of the router/gateway. The similarity once again was only a few bits. Another telling piece of information was the RSSI (received signal strength indicator) Here is an example:

WAN MAC: 00:1e:e1:35:E8:D4
eMTA MAC: 00:1e:e1:35:E8:D3
CM MAC: 00:1e:D1:35:E8:D2
WLAN MAC/VLAN: 00:1e:e1:35:e8:d0
WLAN1/VLAN MAC: 06:1e:e1:35:e8:d0
WLAN2/VLAN MAC: 02:1e:e1:35:e8:d0

WAN 00:1e:e1:35:E8:D4, WLAN 00:1e:e1:35:e8:d0, an WLAN/VLAN WLAN1/VLAN MAC: 06:1e:e1:35:e8:d0

The commonality between “separate” devices screams at you! The exterior octets of the MAC increment 00, 02, 06 …. as well as the interior. Please make note of how these addresses correlate with ports and bridges.

In this example we are looking at a device categorized as Data Over Cable Service Interface Specification or simply DOCSIS. This is evident by the device having an eMTA (Embedded Multimedia Terminal Adapter) MAC address. CM MAC represents the cable modem MAC, WAN (wide area network) MAC represents the external MAC address of the device used for consumer communications to the Internet, WLAN (wireless local area network) MAC shows the wireless address space for an AP (access point), and WLAN1 is another AP. Tying all of this information together, we can begin to gather a deeper understanding of the device. Aiding in our ability to understand the device, one must account for the bridge(s). A network bridge is what allows communication between separate segments. Typically bridges are thought of outside the home/corporate network, yet the modern residential router/gateway, and commercial grade equipment for that matter, contain bridges inside them. This is due to a relationship between wireless interfaces, the switching mechanism/hubs, router, and the flow of data. Trunks are also used in some scenarios. The differences in using a trunk versus a network bridge are vendor dependent. The trunk is in place for the VLAN to carry layer two data to the appropriate device. Really, trunking has no bearing on the example, but will help prepare one for a situation where an unknown may be in place. Because in the example given we are looking at a DOCSIS device we should also address the services performed by the device. Typically, DOCSIS devices are found in homes and small businesses. The devices are multi-purpose, serving as a DHCP (dynamic host configuration protocol) server, DNS (dynamic name service) relay, cable modem, firewall, four port switch, and NAT (network address translator).

One must also understand the methods used by the networking equipment to discover other equipment i.e.: BGP (border gateway protocol), RIP (routing information protocol), MLD (multi listener discovery protocol), Etc... Depending on the device you will have received either an IPv4, or IPv6 address. What information can you discover from the multi-cast packets? Where is the DHCP server located? What is your route? Are you on a class A, B, or C network? These questions all bear much importance, but knowing your network class is an absolute necessity. This is because a class C network is a subset of B, and A, while B is a subset of A. Our goal is to ensure we are in the lowest network block possibly achievable (class A versus C), at which point we will be able to discover all that is around us.

We must take into account how networking devices work, what they are comprised of, and their general purpose. The purpose is fairly self-explanatory, while the relationship between network bridges, DHCP servers, and ports seem to be all but forgotten. Today many think of a switch the same as a router, yet they are very different. A switch delegates frames via MAC (media access control) address, while a router makes its networking decision based upon packet contents.

Now that we have a basic understanding of the network topography, we can begin weaving and worming our way through the network. This means that it's time to roll out the TUN/TAP interface - more specifically, virtualized network taps, and encapsulation via a tunnel, all of which are a part of a package named iproute2. Address resolution poisoning has for a long time been considered a MAC attack. I disagree! Address resolution poisoning or ARP poisoning uses the MAC addresses due to its protocol standard. The attack is simply allowed due to an imbalance in checks and balances. Please reference RFC 903! The following is an excerpt from said RFC.

"*II. Design Considerations*

The following considerations guided our design of the RARP protocol. A.

ARP and RARP are different operations. ARP assumes that every host knows the mapping between its own hardware address and protocol address(es). Information gathered about other hosts is accumulated in a small cache. All hosts are equal in status; there is no distinction between clients and servers.

On the other hand, RARP requires one or more server hosts to maintain a database of mappings from hardware address to protocol address and respond to requests from client hosts.

B. As mentioned, RARP requires that server hosts maintain large databases. It is undesirable and in some cases impossible to maintain such a database in the kernel of a host's operating system. Thus, most implementations will require some form of interaction with a program outside the kernel.

C. Ease of implementation and minimal impact on existing host software are important. It would be a mistake to design a protocol that required modifications to every host's software, whether or not it intended to participate.

D. It is desirable to allow for the possibility of sharing code with existing software, to minimize overhead and development costs. "

Using the TUN/TAP device driver for network reconnaissance, pivotal gains, and leveraging access to otherwise “restricted” network areas.

RARP was the check to ARP, keeping things in balance. Due to this protocol being dissolved in most network areas, ARP poisoning is able to be performed. It's not a MAC attack - it is simply exploiting the lack of checks and balances of an inherently flawed protocol, because someone thought it a fabulous idea to save money and time by simply eliminating a fundamental authority over the devices on the network! Again, to quote RFC 903, section A:

> *“A. ARP and RARP are different operations. ARP assumes that every host knows the mapping between its own hardware address and protocol address(es). Information gathered about other hosts is accumulated in a small cache.”*

Moving forward, understanding RARP and ARP is important because of its purpose. ARP is a critical foundation to modern networks. Yet its counterpart RARP has been put aside. In modern networks, virtualization is very common. Virtual sockets, programmable, and tailored to suit a curious individual’s needs, are very disturbing. This is due to one’s ability to act as RARP, remember our virtual device is able to be fully tailored to suit one’s needs. Having the information regarding the network's infrastructure, and a general idea regarding the network's communications, we can begin our adventures in offensively looking at the device we have connected to, and any devices connected to it. In order to do so we will need to have root access, or at least be a sudoer to a Linux kernel supplied with iproute2. Suppose you have found yourself an open wireless access point, and after authentication you try to browse the Internet only to be greeted by an ugly old captive portal! Suppose you are at a coffee shop with a public and a private network. Suppose you've cracked a wireless network but it's attached to others. Suppose you are at home, fed up with your ISP, and want explore their back end network. Let us start small, and improve our skill set before becoming too bold.

Easy does it. Example: We have connected to an AP, our internal IP address is 192.168.x.x, yet while wirelessly monitoring the AP, we notice a strangely similar MAC address with near-identical signal strength. Thinking these devices might be interconnected, we start to probe other network blocks; for instance, we are 192.168.x.x in class a C network, and want to see what's going on in 10.x.x.x/8 class A network. Our general ping probes have failed, and NMAP isn't getting the trick done either. Maybe we want to poke around the NAT to see if other hosts are on the network. These are all very possible using iproute2 and the appropriate kernel modifications. The Internet started between two machines (in its most simple form), and simply expanded. In the Internet's expansion, hubs, and switches were developed to allow multiple machines to communicate, and a little while later routers were introduced. This being said, the Internet begins at an end point, and ends at an opposing end. Without communication between machines, there is no Inter-, or Intranet. Knowing this, let us take a step forward and understand a bit of Linux.

At the very heart of every Linux kernel is a router. This means these kernels have the ability to transpose themselves into a routing device, while maintaining the ability to store much more data than a router, or switch, and can be configured on the fly. Understanding this, we can move even further forward into libvirt. Libvirt is the virtualization library for Linux. I found this while playing with an emulator named QEMU. What I found most interesting about QEMU was the ability to boot into any network space. I wanted to know how this is possible, as I did not want to depend on an emulator because of the system resources required. It's not in my case feasible to spin up fifty virtual machines. After a little research the answer was found. The virtual device driver powered by C, named iproute2, which programmatically creates soft taps, tunnels, and more tunnels! People reading this paper should be familiar with the concept of tunneling as VPN's and SSH are tunnels after all. However, tunneling is when one network protocol delivers a different payload protocol via encapsulation. Factually Cisco has a tunnel for just about everything. Tunnels inside the iproute2 package include: IPIP, GRE, L2TP, and SIT. IPIP performs IP in IP encapsulation, GRE (generic routing encapsulation), and L2TP is layer two transport protocol.

At this point we have an understanding of the networking device, we understand how it is communicating to its clients, we have seen the devices similarities in MAC addresses, and now know we have a virtual device driver capable of multiple instances which is treated as a regular device, and fully programmable. LET'S PWN! First you must note the network BSSID (broadcast service set identifier) MAC, and any VLAN MAC addresses similar within 2 octets of the one you are connected to.

```
        BSSID           PWR   Beacons    #Data, #/s  CH  MB    ENC  CIPHER  AUTH   ESSID

 00:1e:e1:35:e8:d0      -64     383           2       0   6   54e  WPA2 CCMP  PSK    HOME-XXXX
 06:1e:e1:35:e8:d0      -62     334           0       0   6   54e             OPN    Provider-wifi
 02:1e:e1:35:e8:d0      -63     299           0       0   6   54e  WPA2 CCMP  PSK   <length:  0>
```

From the 802.11/wireless side of things the similarities start to become clear. The open AP's MAC address differs by 2 octets per VLAN. Note this is an example of a provider who uses proprietary firmware for their devices. Other manufactures externals/internals are different, yet still susceptible to attack once their inner workings are understood. This is why I say 2 octets.
Other manufactures MAC addresses differ by a larger amount of space, while in this scenario it's very minor. If you issue the command: iw wlan0 scan | grep BSS, a list of MAC addresses will be returned within radio range. However for a more clean output, try using airodump-ng which is a part of the aircrack-ng suite of tools. Upon authentication, try issuing the command arp -a as a sudoer or as the root user. Did the MAC returned only differ by a few bytes? Let's refer to the example given earlier.
Have you noticed any MAC addresses similar to the external one you connected to? Let's refer to the example given earlier.

```
                                               ENC    CIPHER    AUTH   ESSID
WAN MAC: 00:1e:e1:35:E8:D4
eMTA MAC: 00:1e:e1:35:E8:D3
CM MAC: 00:1e:D1:35:E8:D2
WLAN MAC/VLAN:  00:1e:e1:35:e8:d0      WPA2   CCMP      PSK    HOME-XXXX
WLAN1/VLAN  MAC: 06:1e:e1:35:e8:d0     OPN                     Provider-wifi
WLAN2/VLAN  MAC: 02:1e:e1:35:e8:d0     WPA2   CCMP      PSK    <length:  0>
```

Note the differences. Externally the viewable difference is in the first 2 octets, while when connected the last 2 octets differ. If you see a MAC address roll though your packet capture device which meets this criteria issue the following command: ip link add link wlan0 (wlan0 represents your wireless device) name tap0 (or whatever you want to call your soft tap) type macvtap . This creates a virtual tap interface. Now an address must be supplied to the device, use the following syntax: ip link set dev tap0 address xx:xx:xx:xx:xx (where xx:xx;ect represents the mac address you have captured and want to clone). Next you want to forward traffic through your kernel so you become a networking device, and minimizing congestion. To do so issue the following command: echo '1' > /proc/sys/net/ipv4/ip_forward. Finally ensure the tap is up with the following command: ip link set dev tap0 up. If you have discovered the correct device address on the network you now should be able to see traffic outside your issued subnet.

Keep going!
Tap the next address and Explore!
To infinity and beyond!
What do YOU want to do?

I was interested in finding out the relationship between the MAC addresses. What does this mean? Earlier I mentioned something called a bridge. A network bridge cannot bridge connections unless it's addressed with a MAC address, because the switch cannot associate an address to pass frames over the bridge. I wanted to know where these bridges were located within the device, and I wanted to traverse these bridges to find out what was behind them. bridge-utils is a package for Linux that allows one to harness the bridge, however it turns out that you can also use the iproute2 package by following syntax:

```
ip link add name br0 type bridge
```

“Bridge interfaces are virtual Ethernet switches. They can be used to relay traffic transparently between Ethernet interfaces, and, increasingly common, as Ethernet switches for virtual machines running inside supervisors. You can assign an IP address to a bridge and it will be visible from all bridge ports. If this command fails, check if "bridge" module is loaded. Add an interface to bridge -

ip link set dev ${interface name} master ${bridge
name} Examples:
ip link set dev eth0 master br0
Interface you added to a bridge becomes a virtual switch port. It operates only on datalink layer and ceases all network layer operation.
Remove interface from bridge
ip link set dev ${interface name} nomaster” http://baturin.org/docs/iproute2/

I hope now with the knowledge of soft taps, bridges, tunnels, and knowing mac addresses within the routers, one can begin to see where inner device pivoting can be performed! All of the MAC addresses are attached to one singular device locally, yet administered virtually via a hypervisor.

“*A hypervisor or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines. A computer on which a hypervisor is running one or more virtual machines is defined as a host machine. Each virtual machine is called a guest machine*.” http://en.wikipedia.org/wiki/Hypervisor

Or, they may be administered by the router/switching device itself. In my case I am granted an administration panel with little configuration, allowing me to perform basic configuration such as port forwarding, apply simplistic firewall configurations that are per-installed, very minor DHCP configurations, MAC filtering, and content blocking. In order to traverse the bridge one must know the bridges MAC address. Start off by creating a soft tap of the bridge, and start sniffing traffic. You should be able to gain some information regarding what is on the opposing side of the bridge. At this point you need to make a decision. Either you want to be ninja, and simply exfiltrate data off the bridge, or you want to cross it with the intentions of going deeper down the rabbit hole.

Tinkering, and research leads to believe the CPE is using MPLS. According to Wikipedia, “*Multi protocol Label Switching (MPLS) is a mechanism in high-performance telecommunications networks that directs data from one network node to the next based on short path labels rather than long network addresses, avoiding complex lookups in a routing table*.” We can exploit this by definition! In MPLS, traffic is passed between nodes, but what talks to the node? A hypervisor!!

In a term I pulled out of thin air, dubbed Network Asphyxiation, one can discover the hypervisor's MAC address! What I have discovered is by placing network taps in place with localized legitimate MAC addresses, and assigning them to multi-cast and send ARP traffic. The goal in doing so was simply to observe the device's behavior. More and more internal traffic spewed, all from local MAC addresses, obviously causing serious network congestion, but not to the point of halting all service. The Virtual Manager checks up on the device, revealing its own MAC address and corresponding IP address. We now have opened up another path of communication within the device! What's really interesting is tapping the Hypervisor or Virtual Manager's MAC address, because this takes your providers ability to administrate your device away from them. :) Perhaps you may be inclined to throw a tunnel off that tap and sniff around. Asphyxiation of a device is used to discover any controllers, exploiting the mechanism relationship with other devices which administrate the node.

Another especially interesting attack vector lays in router behind router setups. One is able to wirelessly sniff out the router behind the other router based on radio signal sensitivity. Once located, the router behind router can be cloned with the tap interface, and upon authentication with the initial router be switched to address in the CAM (content addressable memory) table of the router behind the one you've initially authenticated with. Once successfully completed this attack vector will allow for eavesdropping, data exfiltration, and since you have an address allocated in the CAM table you do not have to make your own route to other machines on the other network block. This is inherently dangerous. However, once again this information is proprietary to the provider. Although in other research ventures I have noticed these techniques also affect larger commercial grade equipment. Specifically when pivoting through the router and switch. Network asphyxiation only pertains to managed networking equipment.

Once a VPN (virtual private network) is able to be located on a network, its contents can be viewed quite easily. To simply understand a VPN is to understand a tunnel. Therefore, to gain access to its contents one simply tunnels the tunnel. I found this out upon accessing an open access point. Upon acquiring a DHCP leased address, I began sniffing traffic, to find all traffic encapsulated, and being routed to an IP address obviously not belonging to the internal router. I have not tried to view the contents of a VPN running on a PC, whilst thinking about the concept; I am led to believe the action would probably result in denial of service. This is due to upon setting up the tap interface a ping packet is thrown, resulting in a release in DHCP, but wait, this would depend again on the networks topography, and how the devices are maintained, yet another point a pentester should look at.

Since all the traffic was being routed to the same address, and being I was assigned said address upon joining the network. I thought it only logical to tap the MAC of the access point, and construct a tunnel assigned the IP of the VPN, while not setting a value for the port. This circumvented the VPN, allowing me access to the contents of the tunnel!

Also, a deauth packet isn't needed to bump a client off an 802.11 network. You can simply clone its MAC address, and run a fake auth attack with Aircrack-ng. The client will get bumped out, and per Windows user friendly ways automatically re-authenticate with the network. TAADAAA you've got hash.

References

http://publib.boulder.ibm.com/infocenter/tivihelp/v20r1/topic/com.ibm.tivoli.tpm.net.doc/network/cnet_trunking.html

http://en.wikipedia.org/wiki/Tunneling_protocol

http://www.lartc.org/lartc.html

http://seravo.fi/2012/virtualized-bridged-networking-with-macvtap

http://backreference.org/2014/03/20/some-notes-on-macvlanmacvtap/

http://baturin.org/docs/iproute2/

http://en.wikipedia.org/wiki/Hypervisor

http://en.wikipedia.org/wiki/Category:Tunneling_protocols

https://www.nanog.org/meetings/nanog49/presentations/Sunday/mpls-nanog49.pdf