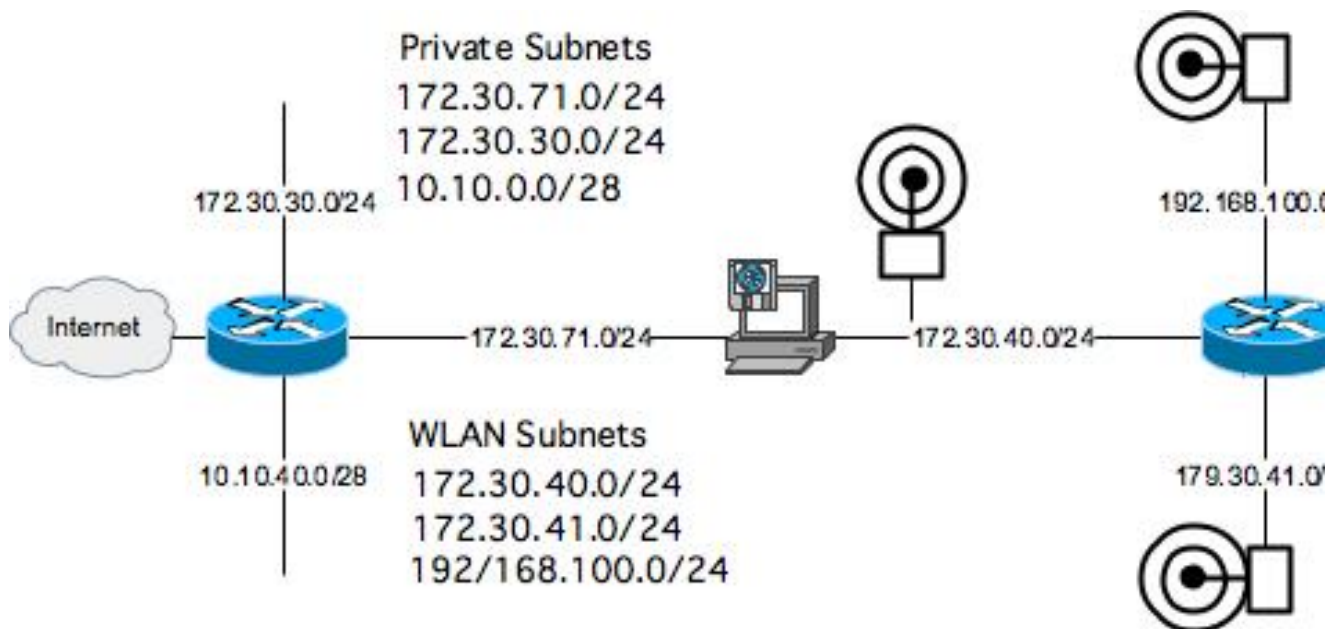


Building A WLAN Proxy Server IP Routing

Michael J. Martin

Picking up where we left off in our last tip, we now find our hero faced with this challenge: How can I configure my multi-interface Linux server so it reaches all of the networks beyond all of its local interfaces (as illustrated in the following diagram)?



Glancing in his trusty network tool-box, he finds two options: implement a static kernel IP route table or run some kind of dynamic IP routing protocol using Zebra, Gated, routed, or Quagga. As dynamic routing protocol configuration is not required for our WLAN solution and is beyond the scope of this article, we won't get into configuring dynamic routing protocols on Linux today. Our hero instead turns to the static IP route option and that's where our story begins...

In last month's article we covered how the `/etc/sysconfig/network` and the `/etc/sysconfig/network-scripts/ifcfg-eth*` scripts configure the server's interfaces during the bootstrap process. And although these scripts set a number of different network variables, they provide only basic IP routing configuration with an option to set the server's default route. In most cases, the default-route is more than adequate, even on a multi-interface server. Typically, additional physical interfaces are added to servers to provide direct server access from hosts, to support applications such as DHCP or network backups. However, there are scenarios where the server may require more than just a default route. One such case is illustrated above, where additional IP networks exist beyond the local segment on more than one server interface. Another case is when the Linux server is configured to operate as an IP Gateway, where the server forwards IP packets through its local interfaces generated by other hosts. Or the server is configured as an application proxy or firewall, where it makes IP service requests on the behalf of other hosts (as is the case with our WLAN proxy server). Red Hat Linux and *NIX provide both command line and boot time tools for managing IP routes.

Let's get started with the command line tools. The Red Hat Linux distribution we are working with has two commands for managing IP routes: The traditional `/sbin/route`, which can be found on just about every *NIX and the newcomer, `/sbin/ip` command, using the "route" sub-command. Both commands add and delete host and network routes along with setting the default route. Here is a breakdown of the command syntax needed to perform the various IP route management functions:

- To display the kernel IP routing table:
`/sbin/route`

Building A WLAN Proxy Server IP Routing

Michael J. Martin

/sbin/ip route

- To set the default route:
`/sbin/route add default gw {ip gateway address in dotted-quad}`
`/sbin/ip route add default via {ip gateway address in dotted-quad}`
- To delete the default route:
`/sbin/route delete default gw {ip gateway address in dotted-quad}`
`/sbin/ip route del default via {ip gateway address in dotted-quad}`
- To add a network route:
`/sbin/route add -net {ip network address in dotted-quad} netmask (subnet mask in dotted-quad) gw {ip gateway address in dotted-quad}`

`/sbin/ip route add {ip network address in dotted-quad/subnet prefix (8-30)} via {ip gateway address in dotted-quad}`
- To delete a network route:
`/sbin/route del -net {ip network address in dotted-quad} (subnet mask in dotted-quad) gw {ip gateway address in dotted-quad}`

`/sbin/ip route del {ip address in dotted-quad/subnet prefix (8-30)} via {ip gateway address in dotted-quad}`
- To add a host route:
`/sbin/route add -host {ip host address in dotted-quad} gw {ip gateway address in dotted-quad}`

`/sbin/ip route add {ip host address in dotted-quad/32 } via {ip gateway address in dotted-quad}`
- To delete a host route:
`/sbin/route del -host {ip address in dotted-quad} gw {ip gateway address in dotted-quad}`

`/sbin/ip route del {ip address in dotted-quad/32} via {ip gateway address in dotted-quad}`

In much the same way that the /etc/sysconfig/network and the /etc/sysconfig/network-scripts/ifcfg-eth* configuration scripts provide the capability to set network interfaces options globally or on a per interfaces basis, both a global and per interface option exists for setting IP routes during the bootstrap process. The global option, /etc/sysconfig/static-routes sets system-wide routes that can be applied ether to a specific or any interface. Here is the command syntax:

```
{phy-interface|any} net {network prefix in dotted-quad} netmask {subnet mask in dotted-quad} gw {gateway IP address in dotted-quad}
```

Here is a syntax example:

```
any net 172.30.30.0 netmask 255.255.255.0 gw 172.30.71.1
any net 10.10.40.0 netmask 255.255.255.240 gw 172.30.71.1
any net 192.168.100.0 netmask 255.255.255.0 gw 172.30.40.1
```

Building A WLAN Proxy Server IP Routing

Michael J. Martin

```
any net 172.30.41.0 netmask 255.255.255.0 gw 172.30.40.1
```

These result in the following kernel routing table (which can be printed using the `/sbin/route` command):

```
root@vulcan sysconfig]# route
Kernel IP routing table
Destination      Gateway         Genmask        Flags Metric Ref    Use Iface
10.10.40.0       172.30.71.1    255.255.255.240 UG      0      0      0 eth0
172.30.30.0       172.30.71.1    255.255.255.0  UG      0      0      0 eth0
172.30.71.0       *              255.255.255.0  U        0      0      0 eth0
172.30.40.0       *              255.255.255.0  U        0      0      0 eth1
172.30.41.0       172.30.40.1    255.255.255.0  UG      0      0      0 eth1
192.168.100.0     172.30.40.1    255.255.255.0  UG      0      0      0 eth1
127.0.0.0         *              255.0.0.0      U        0      0      0 lo
default          172.30.71.1    0.0.0.0        UG      0      0      0 eth0
```

When using this option, it is possible to specify which interface the route should be bound to. I have found the "any" definition is far more flexible and ensures the route will be added, requiring the host to use its interface network address for "closest-match" comparisons for determining the gateway definition.

The per-interface route configuration option, `/etc/sysconfig/network-scripts/route-eth*`, allows administrators to define the routes for a specific host interface. This approach offers a little more flexibility, at the price of a little extra work, letting you set both network specific routes and the default route (a third method) for a host interface. The only downside of course, is creating a route definition file for each of the server's interfaces. Of course the command syntax for this option is different then that used with `/etc/sysconfig/static-routes` (it is however simpler):

```
{network/prefix} via {gateway IP address in dotted-quad}
```

Here is an example that sets a network route and the default route for the eth0 interface:

```
0.0.0.0/0 via 172.30.71.1
172.30.30.0/24 via 172.30.71.1
10.10.40.0/28 via 172.30.71.1
```

The result is this kernel routing table (printed using the `/sbin/ip route` command):

```
[root@vulcan root]# ip route 10.10.40.0/28 via 172.30.71.1 dev eth0
172.30.71.0/24 dev eth0 scope link
172.30.30.0/24 via 172.30.71.1 dev eth0
172.30.40.0/24 dev eth1 scope link
172.30.41.0/24 via 172.30.40.1 dev eth1
192.168.100.0/24 via 172.30.40.1 dev eth1
127.0.0.0/8 dev lo scope link
default via 172.30.71.1 dev eth0
[root@vulcan root]#
```

As to which approach is better greatly depends on the environment and the number of interfaces on the host, since the default route can be set using `/etc/sysconfig/network` or as part of an interface configuration. From a configuration standpoint, the `/etc/sysconfig/static-route` option is the easiest method if the host has three or more interfaces. One thing to keep in mind about these scripts is that any change made to the interface or routing configuration is only applied to the interface and reloaded.

Building A WLAN Proxy Server IP Routing

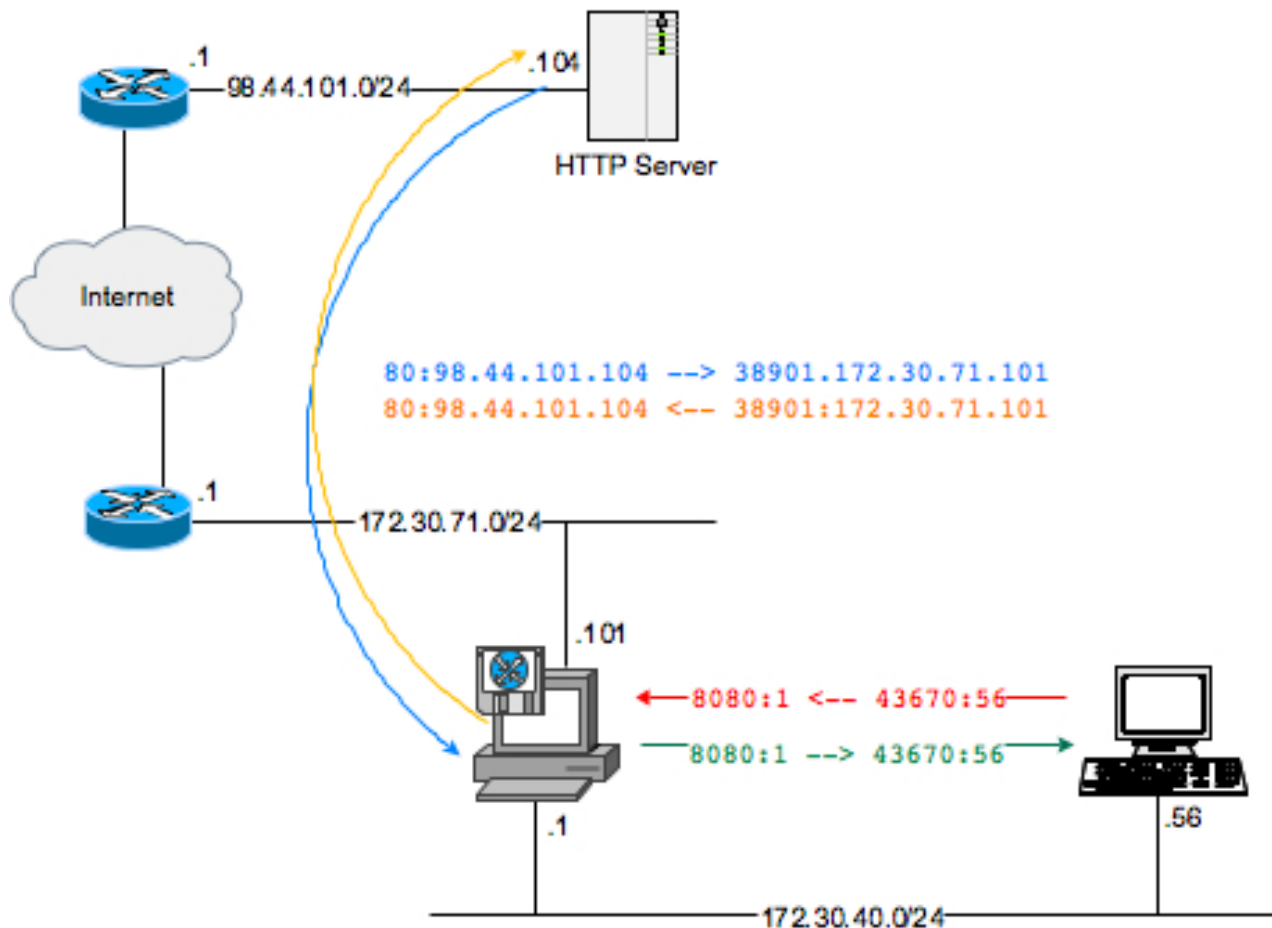
Michael J. Martin

This is opposed to adding a route directly on the command line, which is applied immediately. Reloading the interfaces is easily done using the /etc/rc.d/rc3.d/S10network rc script or /etc/sysconfig/network-scripts/ifdown and ifup scripts. Here are reload syntax examples.

To reload a single interface, the ifup option is the best way to go, as displayed here:

```
root@vulcan network-scripts]# ./ifup eth1
To reload all use the rc script:
root@vulcan rc3.d]# ./S10network start
Setting network parameters:                [ OK ]
Bringing up loopback interface:           [ OK ]
Bringing up interface eth0:               [ OK ]
Bringing up interface eth1:              [ OK ]
root@vulcan rc3.d]#
```

So far, we have been discussing IP routing, from the perspective of having the Linux server operate as an IP relay or proxy server.

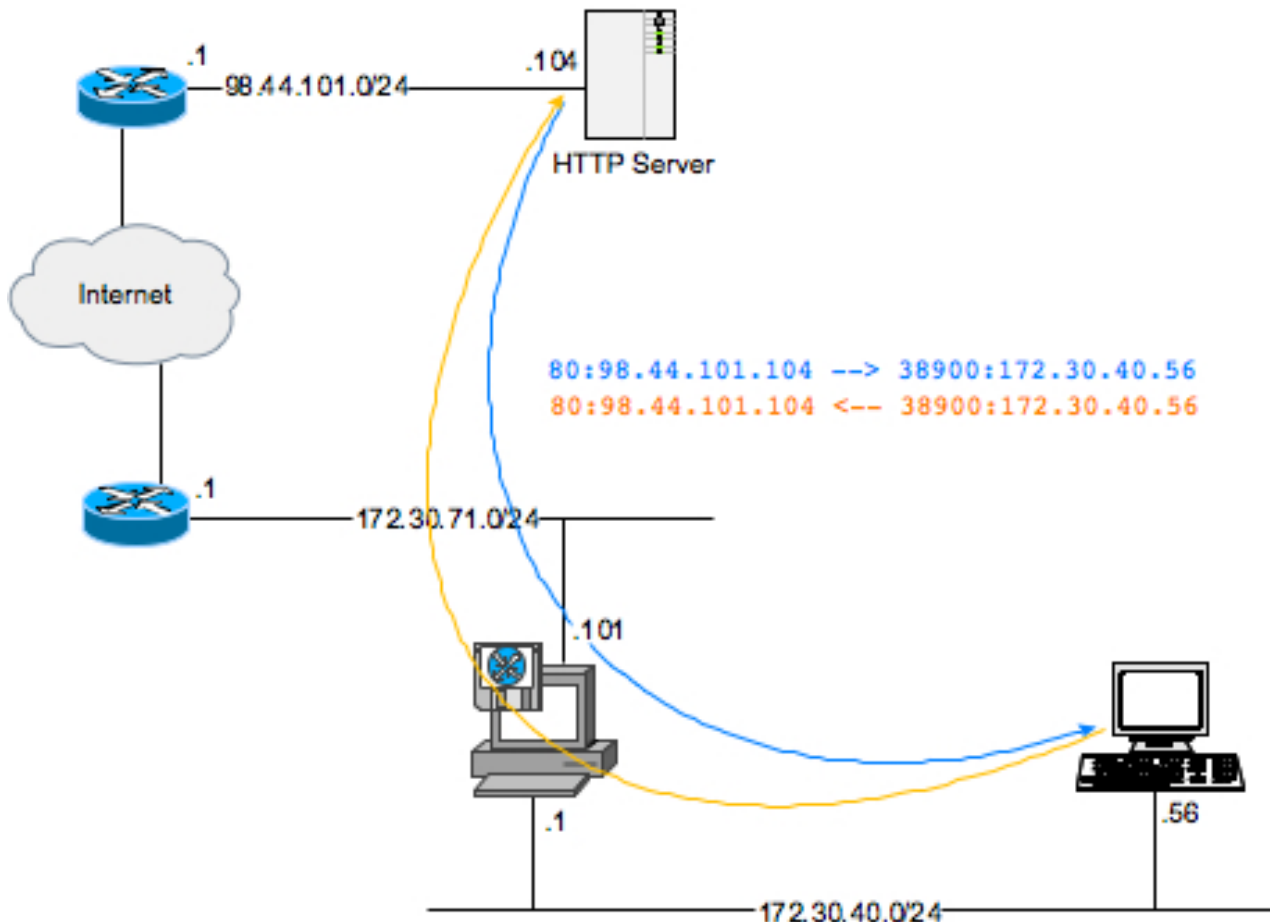


In this configuration, hosts relay application level service requests to the Linux server and the server then makes the service request to the remote host, then relays the reply back to the requesting host. This is the model we are deploying in our WLAN solution. It provides a high level of security; because the "hosts" are only capable of utilizing services for which there are application proxies and it protects the hosts, because they are never directly exposed to the Internet.

Building A WLAN Proxy Server IP Routing

Michael J. Martin

But no discussion on Linux routing would be complete without discussing the capability of a Linux server to forward IP packets generated from other hosts, in other words how to configure the server to operate as an IP router. Some of the first routers were UNIX servers, and even today both with the Linux and BSD platforms and commercial products such as JunOS and Nokia's Voyager operating system's, UNIX continues to be an Ideal platform for IP routers. So looking at the same example, if the Linux host was configured as a router; the host would forward the IP packets to the Linux server and it would in turn forward them onto the next IP gateway.



Now the hosts directly communicate with remote hosts when the Linux server is configured as an IP router. The great thing about being a Linux server is that the server can be configured to operate as both an IP gateway and an application proxy. We will examine this configuration when we cover the Squid Proxy Server configuration options. The important takeaway here is that the server can function both as a router and a server. In order to have the server "forward" IP packets generated from other hosts, a switch in the Linux kernel to permit IP Forwarding must be enabled.

Turning on IP Forwarding is accomplished by modifying the `net.ipv4.ip_forward` option value in the kernel. By default, this capability is disabled. Turning it on requires us to make modifications to files located in the `/proc` file system. In case you are wondering what the `/proc` file system is, it is a directory containing directories and files that represents various aspects of the kernel's operational state. Changes can be made to the kernel's operating state by changing the values stored in the files.

Building A WLAN Proxy Server IP Routing

Michael J. Martin

The first option for enabling IP forwarding sets IP forwarding on for all of the server's interfaces. Using the "echo" shell command, a "0" (disable) or "1" (enable) is directed into the /proc/sys/net/ipv4/ip_forward file.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Sending a "1" enables IP forwarding. This change takes immediate effect. To verify if IP forwarding is enabled or disabled the /bin/cat command can be used to read the /proc/sys/net/ipv4/ip_forward and send the output to the command shell.

```
cat /proc/sys/net/ipv4/ip_forward
```

If a "1" is returned then IP forwarding is enabled, if a "0" is returned the IP forwarding is disabled (the default is disabled). Although, the /proc change takes effect immediately, it only remains in effect until the server is rebooted. To enable IP forwarding at boot, there are two options. The first (which is sort of a hack) is to add echo 1 > /proc/sys/net/ipv4/ip_forward to the /etc/rc.d/rc.local file, which executes custom command at the end of the boot process.

The second option is to change the net.ipv4.ip_forward definition in the /etc/sysctl.conf file. The /sbin/sysctl command is used to set kernel run attributes at boot. The /etc/sysctl.conf file stores the custom attributes settings, the changes are stored in the /proc file system.

```
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values, 0 is disabled, 1 is enabled. See sysctl(8) and
# sysctl.conf(5) for more details.
# Controls IP packet forwarding net.ipv4.ip_forward = 0
```

To enable IP forwarding using /sbin/sysctl, change the net.ipv4.ip_forward value from "0" to "1" and reboot the server. IP forwarding will be from then on enabled. There may be some cases where you will not want to enable IP forwarding on every server interfaces. Luckily, it is also possible to set IP forwarding on a per interface basis. To do this first get a listing of the available interfaces:

```
[root@vulcan eth0]# ls /proc/sys/net/ipv4/conf/
all default eth0 eth1 eth2 lo
[root@vulcan eth0]
#
```

Next, disable IP forwarding on all the interfaces (this is more of a best practice measure):

```
echo 0 > /proc/sys/net/ipv4/conf/all/forwarding
```

Now, let's say you want to enable IP forwarding only between eth0 and eth1. To enable this you would use the following:

```
echo 1 > /proc/sys/net/ipv4/conf/eth0/forwarding
echo 1 > /proc/sys/net/ipv4/conf/eth1/forwarding
```

Once the changes have been made, verify by "catting" the /proc files:

Building A WLAN Proxy Server IP Routing

Michael J. Martin

```
[root@vulcan eth0]# cat /proc/sys/net/ipv4/conf/eth0/forwarding 1
[root@vulcan eth0]# cat /proc/sys/net/ipv4/conf/eth1/forwarding 1
[root@vulcan eth0]#
```

Once IP forwarding has been enabled, depending on how you want the server/router to behave, there are a few other options that should be enabled/disabled. These options, like IP forwarding, can be set for all interfaces or on a per interface basis.

- To enable arp proxy for all or per interface:
`echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp`
`echo 1 > /proc/sys/net/ipv4/conf/eth*/proxy_arp`
- To disable arp proxy for all or per interface (this is default):
`echo 0 > /proc/sys/net/ipv4/conf/all/proxy_arp`
`echo 0 > /proc/sys/net/ipv4/conf/eth*/proxy_arp`
- To enable ICMP redirects for all or per interface (this is default):
`echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects`
`echo 1 > /proc/sys/net/ipv4/conf/eth*/accept_redirects`
- To disable ICMP redirects for all or per interface:
`echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects`
`echo 0 > /proc/sys/net/ipv4/conf/eth*/accept_redirects`
- To enable support for IP source routing on all or per interface:
`echo 1 > /proc/sys/net/ipv4/conf/all/accept_source_route`
`echo 1 > /proc/sys/net/ipv4/conf/eth*/accept_source_route`
- To disable support for IP source routing on all or per interface (this is default):
`echo 0 > /proc/sys/net/ipv4/conf/all/accept_source_route`
`echo 0 > /proc/sys/net/ipv4/conf/eth*/accept_source_route`
- To enable support for ICMP redirects on all or per interface (this is default):
`echo 1 > /proc/sys/net/ipv4/conf/all/send_redirects`
`echo 1 > /proc/sys/net/ipv4/conf/eth*/send_redirects`
- To disable support for ICMP redirects on all or per interface (this is default):
`echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects`
`echo 0 > /proc/sys/net/ipv4/conf/eth*/send_redirects`

When configuring the server to forward packets. It's good idea to disable sending and receiving ICMP, arp-proxy and IP source routing from a security perspective. For these types of specializations, I recommend loading them via rc.local or in a script called by rc.local.

Well that does it for Linux and IP static routing, leaving us with Linux and 802.11q VLAN interfaces. But that exciting topic is for next month. Meantime, hope you found this information useful and if you are interested in having dynamic routing protocol configuration converted, let me know. As always questions, comments and complaints are always welcome. Stay tuned.