Kali Linux and Firewalk

Version 0.1, Last Updated: 3 July 2021

This site is dedicated to sharing information about the practice, ideas, concepts and patterns regarding computer security.

Table of Contents

1. Introduction

The motivation behind this paper is to explore using the tool Firewalk that is available with Kali Linux.

From the projects main website, the developer states: "Firewalk is an active reconnaissance network security tool that attempts to determine what layer 4 protocols a given IP forwarding device will pass. Firewalk works by sending out TCP or UDP packets with a TTL one greater than the targeted gateway. If the gateway allows the traffic, it will forward the packets to the next hop where they will expire and elicit an ICMP_TIME_EXCEEDED message. If the gateway hostdoes not allow the traffic, it will likely drop the packets on the floor and we will see no response.

To get the correct IP TTL that will result in expired packets one beyond the gateway we need to ramp up hop-counts. We do this in the same manner that traceroute works. Once we have the gateway hopcount (at that point the scan is said to be `bound`) we can begin our scan.

It is significant to note the fact that the ultimate destination host does not have to be reached. It just needs to be somewhere downstream, on the other side of the gateway, from the scanning host.

…

Firewalk was developed in 1998 out of research we were conducting with traceroute, specifically with contrived port designation.

…

Firewalk was originally designed by Mike D. Schiffman and David Goldsmith and is primarily maintained by Mike D. Schiffman (mike at infonexus dot com)."

source: http://packetfactory.openwall.net/projects/firewalk/

Let's get into the lab's configuration next.

2. Requirements

2.1. Writing Conventions

If you see the following $ symbol on a command line to execute, what that means is that the command is executed as a regular user; meaning an account that does not have administrative privileges. Ignore the leading $ and execute the rest of the command.

```
$ command to execute as a regular user
```

If you see a command line lead with the # symbol, then that means that the command is executed as the root user. This implies you need to elevate to the root user before running the command, e.g. with: `sudo su - root`.

```
# command to execute as the root user
```

2.2. VirtualBox

Go to: https://www.virtualbox.org/wiki/Downloads and download VirtualBox.

The author is running on Ubuntu 18.04, so following to this URL: https://www.virtualbox.org/wiki/Linux_Downloads

For Ubuntu, double click on the .deb file, i.e. virtualbox-5.2_5.2.0-118431-Ubuntu-zesty_amd64.deb, and install VirtualBox on your local workstation.

2.2.1. Clean VirtualBox Networking

This section is here in case you already had virtualbox installed from before. The intent is to clean up the previous networking. If you do not need to do this, skip to Add VirtualBox Networking

Run these two commands from a Terminal:

```
$ VBoxManage list natnetworks
$ VBoxManage list dhcpservers
```

Output (example):

```
NetworkName:    192.168.139-NAT
IP:             192.168.139.1
Network:        192.168.139.0/24
IPv6 Enabled:   No
IPv6 Prefix:    fd17:625c:f037:2::/64
DHCP Enabled:   Yes
Enabled:        Yes
loopback mappings (ipv4)
        127.0.0.1=2


NetworkName:    192.168.139-NAT
Dhcpd IP:       192.168.139.3
LowerIPAddress: 192.168.139.101
UpperIPAddress: 192.168.139.254
NetworkMask:    255.255.255.0
Enabled:        Yes
Global Configuration:
    minLeaseTime:     default
    defaultLeaseTime: default
    maxLeaseTime:     default
    Forced options:   None
    Suppressed opts.: None
        1/legacy: 255.255.255.0
Groups:               None
Individual Configs:   None

NetworkName:    HostInterfaceNetworking-vboxnet0
Dhcpd IP:       172.20.0.3
LowerIPAddress: 172.20.0.101
UpperIPAddress: 172.20.0.254
NetworkMask:    255.255.255.0
Enabled:        Yes
Global Configuration:
    minLeaseTime:     default
    defaultLeaseTime: default
    maxLeaseTime:     default
    Forced options:   None
    Suppressed opts.: None
        1/legacy: 255.255.255.0
Groups:               None
Individual Configs:   None
```

Now, delete ALL of the pre-installed VirtualBox networks (one at a time following the syntax below):

```
VBoxManage natnetwork remove --netname <NetworkName_from_above>
VBoxManage natnetwork remove --netname 192.168.139-NAT
```

Repeat as many times as necessary to delete all of them.

Now, delete ALL of the pre-installed DHCP services:

```
VBoxManage dhcpserver remove --netname <DHCP_Server_NetworkName_from_above>
VBoxManage dhcpserver remove --netname 192.168.139-NAT
```

Repeat as many times as necessary to delete all of them.

2.2.2. Add VirtualBox Networking

Now, add the new VirtualBox networks so the Kali Linux guides work.

```
VBoxManage natnetwork add \
    --netname 192.168.139-NAT \
    --network "192.168.139.0/24" \
    --enable --dhcp on

VBoxManage dhcpserver add \
    --netname 192.168.139-NAT \
    --ip 192.168.139.3 \
    --lowerip 192.168.139.101 \
    --upperip 192.168.139.254 \
    --netmask 255.255.255.0 \
    --enable

VBoxManage hostonlyif create

VBoxManage hostonlyif ipconfig vboxnet0 \
    --ip 172.20.0.1 \
    --netmask 255.255.255.0

VBoxManage dhcpserver add \
    --ifname vboxnet0 \
    --ip 172.20.0.3 \
    --lowerip 172.20.0.101 \
    --upperip 172.20.0.254 \
    --netmask 255.255.255.0

VBoxManage dhcpserver modify \
    --ifname vboxnet0 \
    --enable
```

VirtualBox install complete.

2.3. Vagrant

Go to: https://www.vagrantup.com/downloads.html, follow the appropriate link to your OS and 32 or 64 bit version representing your local workstation. Download.

For Ubuntu, double click on the .deb file, i.e. vagrant_2.0.1_x86_64.deb, and install Vagrant on your local system.

2.4. Kali Linux, a routing device, and our internal target

The author highly recommends to create a directory structure that is easy to navigate and find your code. As an example, you could use something similar to:

```
${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

Go ahead and make this structure with the following command (inside a Terminal):

```
$ mkdir -p ${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

From a Terminal, change directory to:

```
$ cd ${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

2.4.1. Vagrantfile

Inside of the kali-linux-vm directory, populate a new file with the exact name, “Vagrantfile”. Case matters, uppercase the “V”. This file will contain both virtual machines for Kali Linux as well as setting up the other two virtual machines.

We are aggregating all virtual machines into one file in order to save time. The coolness here is setting up the variables at the top of the Vagrantfile mimicing shell scripting inside of a virtual machine (passed in with provision: shell). I tested using: `apt-get update && apt-get upgrade -y`, but opted to take it out since it took over 45 minutes on my slower (old) hardware. See comment about downloading this file immediately preceding the code block.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

$os_update = <<SCRIPT
apt-get update
SCRIPT

VAGRANTFILE_API_VERSION = "2"


Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.define "router-vagrant" do |conf|

        conf.vm.box = "ubuntu/xenial64"

        conf.vm.hostname = "router-vagrant"

        # For Linux systems with the Wireless network, uncomment the line:
        conf.vm.network "public_network", bridge: "wlo1", auto_config: true
        conf.vm.network "private_network", ip: "192.168.139.5"

        # For macbook/OSx systems, uncomment the line and comment out the Linux Wireless network:
        #conf.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true
        #conf.vm.network "private_network", ip: "192.168.32.5"

        conf.vm.provider "virtualbox" do |vb|
            vb.memory = "1024"
            vb.cpus = "2"
            vb.gui = false
            vb.customize ["modifyvm", :id, "--vram", "32"]
            vb.customize ["modifyvm", :id, "--accelerate3d", "off"]
            vb.customize ["modifyvm", :id, "--ostype", "Ubuntu_64"]
            vb.customize ["modifyvm", :id, "--boot1", "dvd"]
            vb.customize ["modifyvm", :id, "--boot2", "disk"]
            vb.customize ["modifyvm", :id, "--audio", "none"]
            vb.customize ["modifyvm", :id, "--clipboard", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--draganddrop", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--paravirtprovider", "kvm"]
        end
        conf.vm.provision "shell", inline: $os_update
        conf.vm.provision :shell, path: "router.sh"
    end


    config.vm.define "target-vagrant" do |conf|

        conf.vm.box = "ubuntu/xenial64"

        conf.vm.hostname = "target-vagrant"

        # For Linux systems with the Wireless network, uncomment the line:
        conf.vm.network "private_network", ip: "192.168.139.10"

        # For macbook/OSx systems, uncomment the line and comment out the Linux Wireless network:
        #conf.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true
        #conf.vm.network "private_network", ip: "192.168.32.5"

        conf.vm.provider "virtualbox" do |vb|
            vb.memory = "1024"
            vb.cpus = "2"
            vb.gui = false
            vb.customize ["modifyvm", :id, "--vram", "32"]
            vb.customize ["modifyvm", :id, "--accelerate3d", "off"]
            vb.customize ["modifyvm", :id, "--ostype", "Ubuntu_64"]
            vb.customize ["modifyvm", :id, "--boot1", "dvd"]
            vb.customize ["modifyvm", :id, "--boot2", "disk"]
            vb.customize ["modifyvm", :id, "--audio", "none"]
            vb.customize ["modifyvm", :id, "--clipboard", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--draganddrop", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--paravirtprovider", "kvm"]
        end
        conf.vm.provision "shell", inline: $os_update
        conf.vm.provision :shell, path: "target.sh"
    end


    config.vm.define "kali-linux-vagrant" do |conf|
        conf.vm.box = "kalilinux/rolling"

        # For Linux systems with the Wireless network, uncomment the line:
        conf.vm.network "public_network", bridge: "wlo1", auto_config: true

        # For macbook/OSx systems, uncomment the line and comment out the Linux Wireless network:
        #conf.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true

        conf.vm.hostname = "kali-linux-vagrant"
        conf.vm.provider "virtualbox" do |vb|
            vb.gui = true
            vb.memory = "2048"
            vb.cpus = "2"
            vb.customize ["modifyvm", :id, "--vram", "32"]
            vb.customize ["modifyvm", :id, "--accelerate3d", "off"]
            vb.customize ["modifyvm", :id, "--ostype", "Debian_64"]
            vb.customize ["modifyvm", :id, "--boot1", "dvd"]
            vb.customize ["modifyvm", :id, "--boot2", "disk"]
            vb.customize ["modifyvm", :id, "--audio", "none"]
            vb.customize ["modifyvm", :id, "--clipboard", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--draganddrop", "hosttoguest"]
            vb.customize ["modifyvm", :id, "--paravirtprovider", "kvm"]
        end
        conf.vm.provision "shell", inline: $os_update
    end

end
```

Save and write this file.

You can also download from:

```
$ curl -o Vagrantfile  http://securityhardening.com/files/Vagrantfile_20210705.txt
```

2.4.2. router.sh

Inside of the kali-linux-vm directory, populate a new file with the exact name, `router.sh`. Case matters, all lowercase. See comment about downloading this file immediately preceding the code block. `router.sh` (include the shebang in your file: the first line with `#!/usr/bin/env bash`):

```
#!/usr/bin/env bash

sed -i 's/^#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/' /etc/sysctl.conf

sysctl -p

ufw disable

exit 0
```

2.4.3. target.sh

Inside of the kali-linux-vm directory, populate a new file with the exact name, `target.sh`. Case matters, all lowercase. See comment about downloading this file immediately preceding the code block. `target.sh` (include the shebang in your file: the first line with `#!/usr/bin/env bash`):

```
#!/usr/bin/env bash

ufw disable

apt update
apt install apache2 -y
Firewalk was originally designed by Mike D. Schiffman and David Goldsmith and is primarily maintained by Mike D.
Schiffman (mike at infonexus dot com).
systemctl enable apache2


cat <<-'EOF' > /var/www/html/index.html
<html>
    <head>
        <title>Welcome to your Target!</title>
    </head>
    <body>
        <h1>Success! Your Target host is working!</h1>
    </body>
</html>
EOF

chown root:root /var/www/html/index.html
chmod 0644      /var/www/html/index.html

systemctl start apache2

exit 0
```

Save and write these files.

If you have issues with copying and pasting the above files because code blocks in PDFs always copy correctly [NOT!], you could use curl, i.e. Make sure the router.sh file ends up in the same directory as the Vagrantfile.

```
$ curl -o router.sh  http://securityhardening.com/files/router_sh_20210705.txt
$ curl -o target.sh  http://securityhardening.com/files/target_sh_20210705.txt
```

From a Terminal, change directory to:

```
$ cd ${HOME}/Source_Code/Education/vagrant-machines/kali-linux-vm/
```

Then run (inside the directory kali-linux-vm):

```
$ vagrant up
```

This will download the appropriate images and start the virtual machines. Once running, through the VirtuaBox GUI, login as root. Password is “toor”, root backwards. Edit the following file: `/etc/ssh/sshd_config`

And change the line: `#PermitRootLogin prohibit-password` To: `PermitRootLogin yes` Meaning strip the comment out on the beginning of the line and alter `prohibit-password` to `yes`.

Then restart the ssh daemon:

```
# kill -HUP $(pgrep sshd)
```

Notice, you are on a Bridged adapter, this will open the instance to allow root to ssh in with the most unsecure password in the world. Only make this change (allowing root to login via SSH) if you require root SSH access. You can change the root user’s password, which is highly recommended.

For the other instances, I would first run ‘vagrant status’ to capture the name that vagrant is using for the running instance.

```
# vagrant status
```

Choose

```
Current machine states:
kali-linux-vagrant running (virtualbox)
router-vagrant running (virtualbox)
target-vagrant running (virtualbox)
```

This environment represents multiple VMs. The VMs are all listed above with their current state. For more information about a specific VM, run `vagrant status NAME`.

From there, log into the router instance with:

```
$ vagrant ssh router-vagrant
```

And then get the current IP address.

```
$ ip a
```

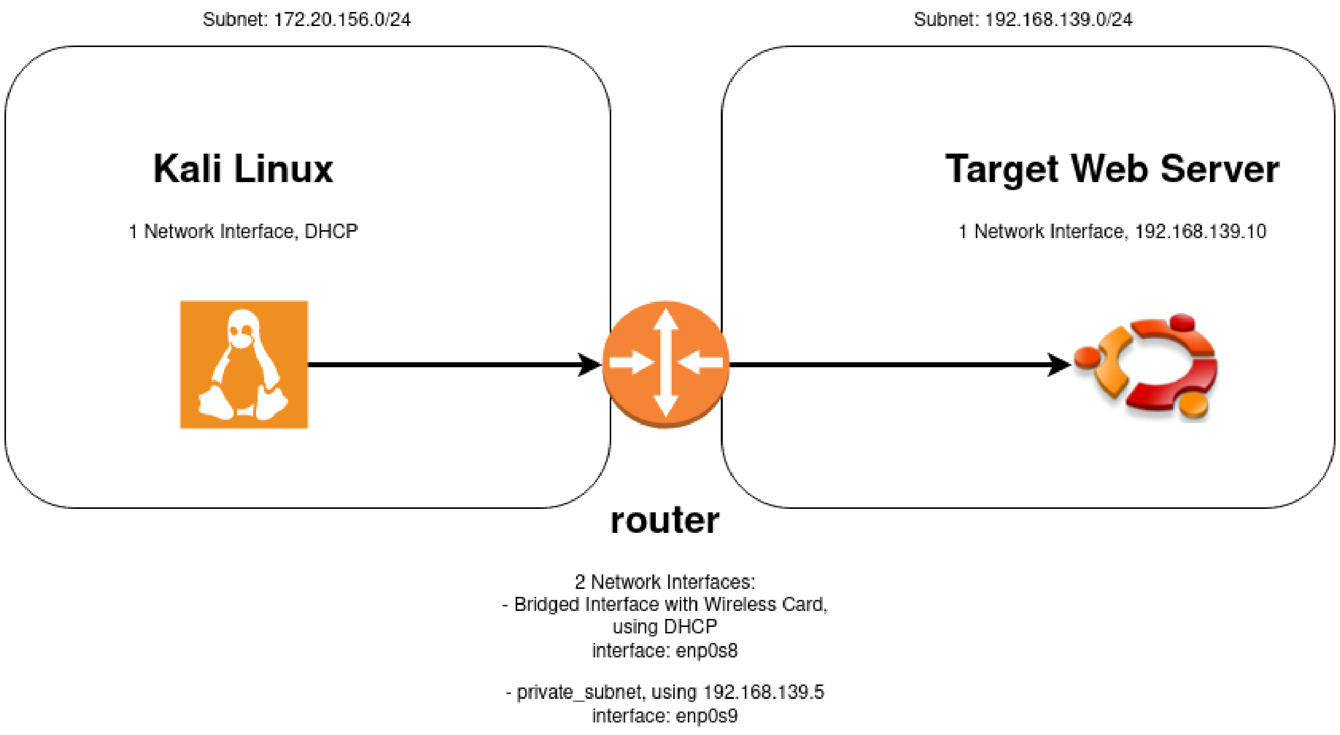
Choose the second network adapter, it should look like:

```
ubuntu@router-vagrant:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 02:95:cf:22:ea:76 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::95:cfff:fe22:ea76/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d9:50:a4 brd ff:ff:ff:ff:ff:ff
    inet 172.20.156.72/24 brd 172.20.156.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed9:50a4/64 scope link
       valid_lft forever preferred_lft forever
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:e2:6c:de brd ff:ff:ff:ff:ff:ff
    inet 192.168.139.5/24 brd 192.168.139.255 scope global enp0s9
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee2:6cde/64 scope link
       valid_lft forever preferred_lft forever
```

The test network used for this paper uses 172.20.156.0/24 as the network range [shown here in section 3]. Therefore, the adapter, enp0s8 is what we are looking for. The IP to use as a target is 172.20.156.72. Write down your value.

3. Firewalk

The lab will be using the following setup:


Subnet: 172.20.156.0/24
Subnet: 192.168.139.0/24
Kali Linux
1 Network Interface, DHCP
Target Web Server
1 Network Interface, 192.168.139.10
router
2 Network Interfaces:
- Bridged Interface with Wireless Card, using DHCP
interface: enp0s8
- private_subnet, using 192.168.139.5
interface: enp0s9

If you have not done so, go ahead and run `vagrant up` from inside of your directory containing the Vagrantfile, router.sh and target.sh files.

Log into Kali Linux.

username:root

password:toor

From what I found this year, the software certificate has expired.

Run the following commands as the root user to update the Kali linux software certificate:

```
cd /root/

curl -O https://archive.kali.org/archive-key.asc

apt-key add ./archive-key.asc
```

Firewalk does not come installed as default, so we will need to install it as the root user.

Run the following commands as the root user:

```
apt update

apt install -y firewalk
```

To scan our target host, we are going to run the following command as the root user:

```
firewalk -S80  -i eth1 -n -pTCP 172.20.156.72 192.168.139.10
```

Let’s stop and talk about the above command.

The `-S80` is going to hit port 80.

The `-i eth1` will go out the network interface named, "eth1".

The `-n` will not perform a DNS lookup on each IP found.

The `-pTCP` will use the TCP protocol.

The first ip listed above is the router’s base IP, i.e. 172.20.156.72 [don’t forget we are using a Virtual Machine to route for this lab].

The second ip listed above is the target’s IP address, i.e. 192.168.139.10.

Example Output of command:

```
root@kali:~# firewalk -S80  -i eth1 -n -pTCP 172.20.156.72 192.168.139.10
Firewalk 5.0 [gateway ACL scanner]
Firewalk state initialization completed successfully.
TCP-based scan.
Ramping phase source port: 53, destination port: 33434
Hotfoot through 172.20.156.72 using 192.168.139.10 as a metric.
Ramping Phase:
 1 (TTL  1): expired [172.20.156.1]
 2 (TTL  2): expired [172.20.10.1]
 3 (TTL  3): *no response*
 4 (TTL  4): *no response*
 5 (TTL  5): *no response*
 6 (TTL  6): *no response*
 7 (TTL  7): *no response*
 8 (TTL  8): *no response*
 9 (TTL  9): *no response*
10 (TTL 10): *no response*
11 (TTL 11): *no response*
12 (TTL 12): *no response*
13 (TTL 13): *no response*
14 (TTL 14): *no response*
15 (TTL 15): *no response*
16 (TTL 16): *no response*
17 (TTL 17): *no response*
18 (TTL 18): *no response*
19 (TTL 19): *no response*
20 (TTL 20): *no response*
21 (TTL 21): *no response*
22 (TTL 22): *no response*
23 (TTL 23): *no response*
24 (TTL 24): *no response*
25 (TTL 25): *no response*
Scan aborted: hopcount exceeded.

Total packets sent:                25
Total packet errors:               0
Total packets caught               4
Total packets caught of interest   2
Total ports scanned                0
Total ports open:                  0
Total ports unknown:               0
```

Full Options

```
Usage : firewalk [options] target_gateway metric
           [-d 0 - 65535] destination port to use (ramping phase)
           [-h] program help
           [-i device] interface
           [-n] do not resolve IP addresses into hostnames
           [-p TCP | UDP] firewalk protocol
           [-r] strict RFC adherence
           [-S x - y, z] port range to scan
           [-s 0 - 65535] source port
           [-T 1 - 1000] packet read timeout in ms
           [-t 1 - 25] IP time to live
           [-v] program version
           [-x 1 - 8] expire vector
```

Let’s understand this more.

"Firewalk is a network-auditing tool… It attempts determines what transport protocols a given gateway will let through. The firewalk scan works by sending out TCP or UDP packets with an IP TTL one greater then the targeted gateway. If the gateway allows the traffic, it will forward the packets to the next hop where they will expire and elicit a TTL exceeded in transit message. If the gateway host does not allow the traffic, it will likely drop the packets on the floor and we will see no response. By sending probes in a successive manner and recording which ones answer and which ones don’t, the access list on the gateway can be determined.

To work its magic, firewalk has two phases, a network discovery phase, and a scanning phase. Initially, to get the correct IP TTL (that will result in expired packets one beyond the gateway) we need to ‘ramp up’ hop counts. We do TTL ramping in the same manner that traceroute works, sending packets out with successively incremented IP TTLs, towards the destination host. Once we know the gateway hopcount (at that point the scan is ‘bound’) we can move onto the next phase, the actual scan.

The actual scan is simple. Firewalk sends out TCP or UDP packets and sets a timeout; if it receives a response before the timer expires, the port is considered open, if it doesn’t, the port is considered closed."

source: http://packetfactory.openwall.net/projects/firewalk/firewalk-final.pdf

4. Conclusion

This was an interesting paper to write. It had the author scratching his head at several points about how to configure a reasonable lab to showcase the technology. Not super difficult, but still interesting in what felt like a time travel back to 1998.

From the developers original paper, the author found this blurbage interesting:

"RISK MITIGATION

The easiest solution to this problem is to disallow ICMP TTL Exceeded messages from leaving an internal network. This will also have the affect of breaking valid uses of traceroute and may inhibit remote diagnostics of an internal network problem. Another defense against Firewalking is the use of some form of proxy server. Network Address Translation (NAT) or any proxy server (both application level and circuit level) can prevent Firewalk from probing behind them. While network based intrusion detection tools could detect certain attacks; it is possible to develop a version of Firewalk that would generate packets that would look like valid packets for each service that it is scanning. Currently, Firewalk only fills in the packet header and does not insert any data into a packet. A more sophisticated version could emulate various services in an attempt to masquerade as valid traffic and randomize the order and times that it scans services."

source: http://packetfactory.openwall.net/projects/firewalk/firewalk-final.pdf

The Author's interpretation of the results of this tool is that they are interesting, but most likely not going to be off any major value because most Linux systems come pre-configured where the ICMP TTL Exceeded is protected via the base configuration of the system and kernel [meaning from the kernel tunable parameters]. The Author remembers a colleague talking very positively about this tool in 2003. That time has come and gone and most devices should protect against this type of attack. Still, the author feels it is worth knowing about the tool and its usage [in the back of security professional's heads] since a routing/firewall device could be misconfigured and this tool would be an excellent way to test said for misconfigurations dealing with ICMP TTL Exceeded messsages.

5. Appendix

References

https://www.kali.org/tools/firewalk/

http://packetfactory.openwall.net/projects/firewalk/

http://packetfactory.openwall.net/projects/firewalk/firewalk-final.pdf