Night Activity Program • Augumented Reality • Android Development

ODROID Magazine

Year Three
Issue #29
May 2016

THE DEEP WEB

- SETUP A TOR RELAY ON YOUR ODROID
- UNDERSTAND WIRELESS CODE INJECTION

- Stream your tunes with Cherry Music
- Make your own smart car with an ODROID-XU4

What we stand for.

We strive to symbolize the edge of technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

Pollin ELECTRONIC

We are now shipping the ODROID-U3 device to EU countries! Come and visit our online store to shop!

Address: Max-Pollin-Straße 1
85104 Pförring Germany

Telephone & Fax
phone: +49 (0) 8403 / 920-920
email: service@pollin.de

Our ODROID products can be found at
http://bit.ly/1tXPXwe

EDITORIAL

Our theme this month is Internet security and anonymity. The Deep Web is a large portion of the Internet that is not indexed by traditional search engines, and requires a custom browser called Tor in order to explore it. Tor provides a secure browsing experience by giving users access to special relays that provide a secure entrance and exit from the Deep Web network. As David describes in his article, an ODROID can be used to set up an inexpensive TOR relay that provides anonymous browsing services. Because so much financial information is transferred via the World Wide Web, it's important to maintain a secure local network. A common technique for maliciously monitoring router traffic is wireless injection, and Adrian details ways to protect a network using Kali Linux, a powerful penetration testing suite.

Tobias also takes a closer look at the Atari Jaguar, Nanik details the Android Support Library for maintaining code compatibility with previous Android versions, and Marian shows us how to monitor a baby's nap remotely. Our DIY projects this month include installing Cherry Music, building a Car PC, and creating augmented reality with an oCAM.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian.
Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815
Hardkernel manufactures the ODROID family of quad-core development boards and the world's first ARM big.LITTLE single board computer.
For information on submitting articles, contact odroidmagazine@gmail.com, or visit http://bit.ly/1ypImXs.
You can join the growing ODROID community with members from over 135 countries at http://forum.odroid.com.
Explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

HARDKERNEL

OUR AMAZING ODROIDIAN STAFF:

Rob Roy, Chief Editor

I'm a computer programmer in San Francisco, CA, designing and building web applications for local clients on my network cluster of ODROIDs. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop prebuilt operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I use my ODROIDs for a variety of purposes, including media center, web server, application development, workstation, and gaming console. You can check out my 100GB collection of ODROID software, prebuilt kernels and OS images at `http://bit.ly/1fsaXQs`.

Bruno Doiche, Senior Art Editor

What does our goofy dude is doing lately? Organizing all his collected music libraries that he got over the last 20 years. We think it is mostly an exercise in dubious taste, specially when we see Bruno going from Death metal to Depeche Mode for babies. But what we can do, right? At least he listen using headphones most of the time!

Besides this task, he still is a sucker against David, that now trounces him on card games using his own ODROID powered XMAGE server. Less users at the same server means that Bruno now loses his games faster than ever.

Tough life.

Manuel Adamuz, Spanish Editor

I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially microcomputers such as the ODROID and Raspberry Pi. I love experimenting with these computers. My wife says I'm crazy because I just think of ODROIDs! My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.

Nicole Scott, Art Editor

Nicole is a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media management, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from web design and programming, Analytics and Adwords, to video editing and DVD authoring, Nicole helps clients with the all aspects of online visibility. Nicole owns anODROID-U2, and a number of ODROID-U3's and looks forward to using the latest technologies for both personal and business endeavors. Nicole's web site can be found at http://www.nicolecscott.com.

James LeFevour, Art Editor

I'm a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored with many aspects that I think most West Coast people take for granted. I live with my lovely wife and our adorable pet rabbit; the latter keeps my books and computer equipment in constant peril, the former consoles me when said peril manifests.

Andrew Ruggeri, Assistant Editor

I am a Biomedical Systems engineer located in New England currently working in the Aerospace industry. An 8-bit 68HC11 microcontroller and assembly code are what got me interested in embedded systems. Nowadays, most projects I do are in C and C++, or high-level languages such as C# and Java. For many projects, I use ODROID boards, but I still try to use 8bit controllers whenever I can (I'm an ATMEL fan). Apart from electronics, I'm an analog analogue photography and film development geek who enjoys trying to speak foreign languages.

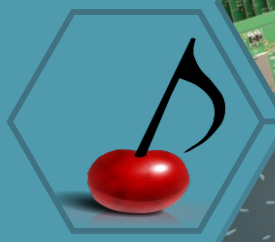
Venkat Bommakanti, Assistant Editor

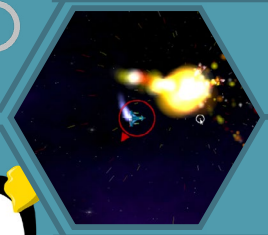
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy learning something new all the time, and try to share my joy and enthusiasm with the community.

Josh Sherman, Assistant Editor

I'm from the New York area, and volunteer my time as a writer and editor for ODROID Magazine. I tinker with computers of all shapes and sizes: tearing apart tablets, turning Raspberry Pis into PlayStations, and experimenting with ODROIDs and other SoCs. I love getting into the nitty gritty in order to learn more, and enjoy teaching others by writing stories and guides about Linux, ARM, and other fun experimental projects.

INDEX





TURNING YOUR ODROID INTO A TOR RELAY

PROTECTING FREEDOM ONE ODROID AT A TIME

by David Gabriel

Although the Internet services of big companies are popular, some find the obtrusiveness invasive, and Tor is a way to create an access relay that is free of prying eyes

Tor is free software that enables access to an open network useful in anonymous communications. The name is derived from the acronym of the original project name, The Onion Router. It protects your privacy by redirecting internet traffic through a network of thousands of relays, and prevents network surveillance and traffic analysis utilities from collecting your data while you navigate. In other words, this makes you "invisible" so that websites do not know your location by your IP address or your Internet Service Provider (ISP). People monitoring your network will not be able to see the websites or resources you access.

All communications inside Tor are encrypted. When data is sent, it is encrypted in the application layer multiple times, and nested like the layers of an onion. Data paths include randomly selected relays. Each relay decrypts a layer of encryption revealing only the next relay and passes the remaining information to it. The process continues until the final relay decrypts the original data and sends it to the destination without revealing the source IP address.

The disadvantage of using Tor is that your effective Internet connectivity will become slower than normal, due to all the encryption and decryption steps and passage through multiple relays. The information transfer speed would be perceivably lower.

Installation

First, ensure the system is updated using the following commands:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Then, install the Tor application and its dependencies with the following command:

```
$ sudo apt-get install tor
```

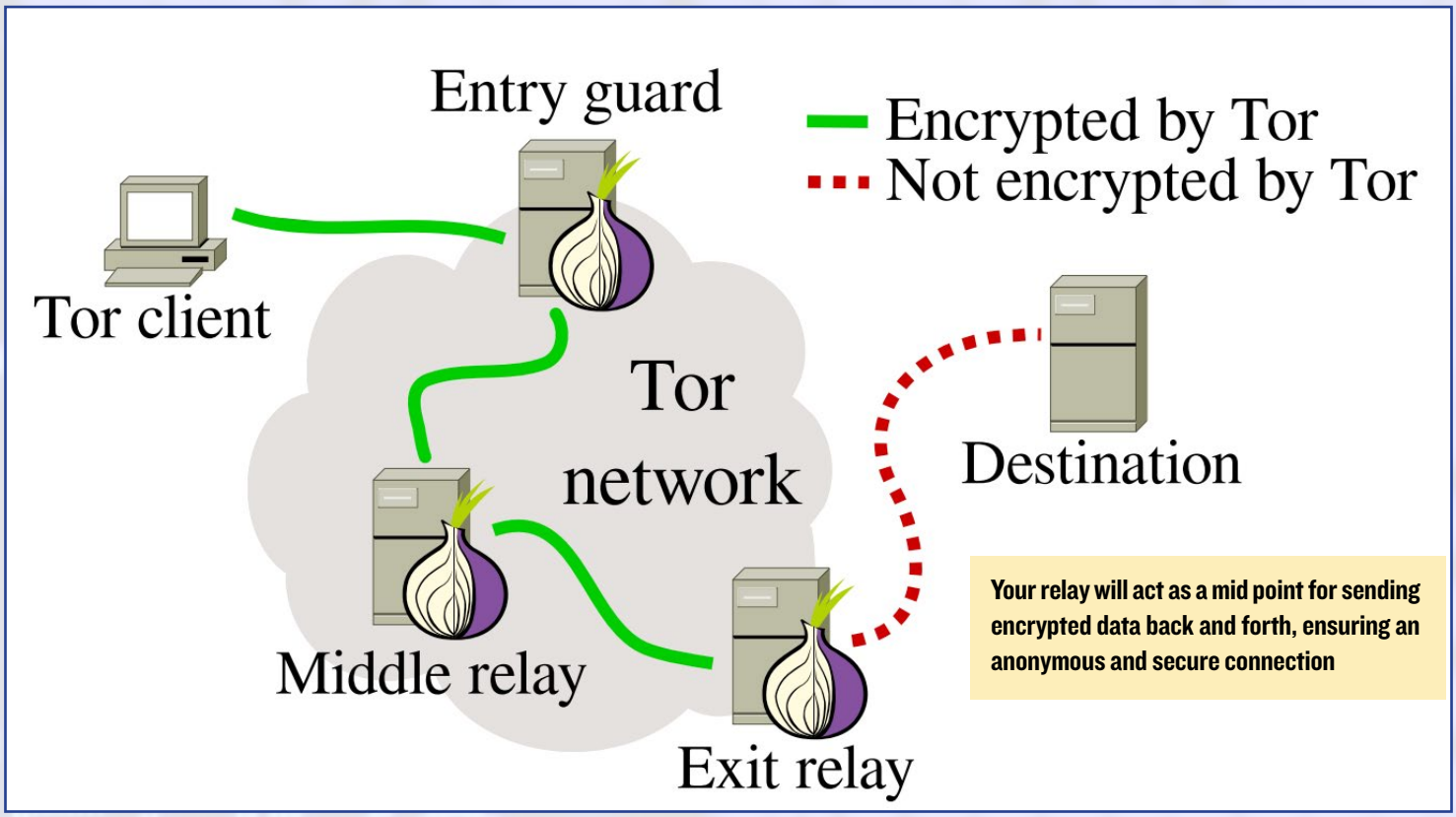
Optionally, you can also install Arm (short-form for: anonymizing relay monitor), which is an application for monitoring and configuring Tor. It works much like the linux utility called top, and can be installed with the following command:

```
$ sudo apt-get install tor-arm
```

Configuration

Tor can be customized by modifying the Tor configuration file. You can use your favourite text-editor to edit the /etc/tor/torrc file and add the commented (using #) options listed below:

```
Log notice file /var/log/tor/notices.log # Log file
```



Your relay will act as a mid point for sending encrypted data back and forth, ensuring an anonymous and secure connection

```
destination
RunAsDaemon 1 # Start process in background as a daemon
ORPort 9001   # Port to be used by incoming connections
DirPort 9030  # Port to be used by directory connections
ExitPolicy reject *:* # Implies that your relay will be used for
                      # relaying traffic inside the Tor network, but
                      # not for connections to external websites or
                      # other services
Nickname odroid-tor-relay  # Can be anything you like, so people
                           # don't have to refer to your relay by key
RelayBandwidthRate 100 KB  # Throttle traffic to 100KB/s (800Kbps)
RelayBandwidthBurst 200 KB # But allow bursts up to 200KB/s (1600Kbps)
```

If you installed the optional Arm application, you need to include the following configuration lines in the above mentioned file:

```
ControlPort 9051            # Port to be used by controller applica-
tions.
CookieAuthentication 1      # Authentication method to be used by the
                            # controller application
DisableDebuggerAttachment 0 # Required by Arm application to be able to
use
                           # commands like netstat to monitor the network
                            # traffic
```

Then restart Tor for the updated configuration to take effect, using the command:

```
$ sudo service tor restart
```

If all goes fine, you should see an entry in the /varlog/tor/log like so:

```
Jan 15 11:38:53.000 [notice] Tor has successfully opened a circuit.
Looks like client functionality is working.
```

Note that if your network is behind a firewall, you will have to configure it to allow incoming requests on ports, 9030 (for directory service) and 9001 (for relay operation). You may have to refer to the User Guide for your particular firewall, to configure this option. If you have installed Arm, you can start it by using the command:

```
$ sudo arm
```

While there are many options you can configure, the most interesting one is related to the graphics generated for you to monitor all traffic going through you relay. Check the Arm application help option, for more information on how leverage the Arm application.

By default, Tor also supports the Socket Secure protocol (SOCKS), over the default port 9050. You can setup your browser to be a Tor client and redirect all connections through Tor relays protecting your privacy and maintain anonymity. In Firefox, for example, you can go to the Preferences > Advanced > Network > Settings > Change option to manual setup the proxy configuration and add 127.0.0.1 on port 9050 to the SOCKS line and enter OK to confirm.

To check your configuration, visit the Tor project website http://bit.ly/1oh1f82 using a browser. You will notice that the public IP appearing on this page will be different from your real IP. This is the exit node of your request, ensuring that you cannot be traced back for location or personal information. Note that data is encrypted only while it goes through the Tor network. Data will be sent as-is, so anything that was not encrypted from the beginning will continue to remain so, after leaving the exit node.

If you want to disable this SOCKS feature and keep your ODROID only as a relay, add the following line to /etc/tor/torrc file and restart the Tor service:

```
SocksPort 0 # Disable torsocks
```

The Tor client can also be used on other operating systems. Configuration may differ slightly depending on the OS and browser, but the above listed options is a good starting point.

References

http://bit.ly/1cqlVa3
http://bit.ly/1PvVIqy
http://bit.ly/1U9oXqa
http://bit.ly/1U9pgkM
http://bit.ly/19QYR47
http://bit.ly/1MOsQPE
http://bit.ly/1nBUETC

ODROID PRODUCTION

A RETROSPECTIVE FROM HARDKERNEL'S EARLY YEARS

edited by Rob Roy

Have you ever wondered how ODROIDs are made? Although the process has changed over the years, here's an interesting look into how the first line of Hardkernel products, the ODROID-X and the ODROID-Q, were produced in July 2012.

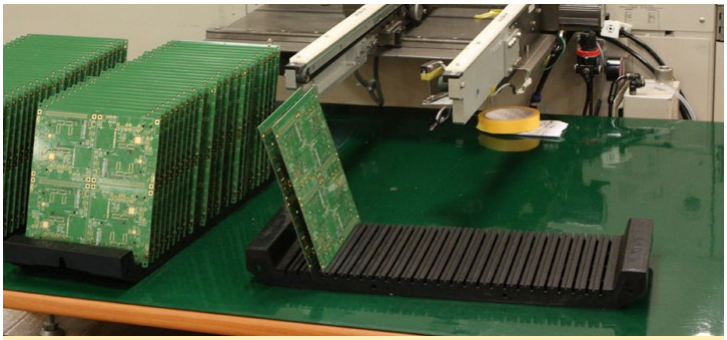


Figure 1 - The first stage of process is a machine called the "Solder cream printer"

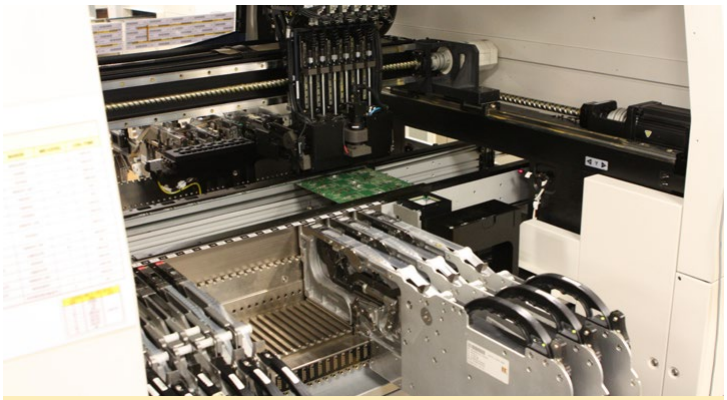


Figure 2 - The Surface Mount Technology (SMT) machine mounts devices on the PCB

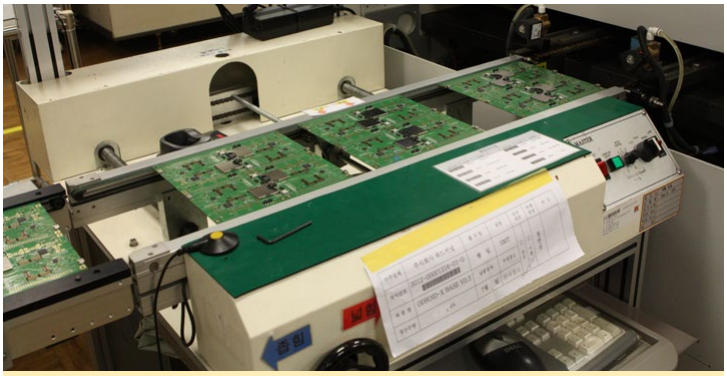


Figure 3 - The boards enter a reflow soldering machine

Figure 4 - SMT soldering is done, and the boards are ready for the inspection process

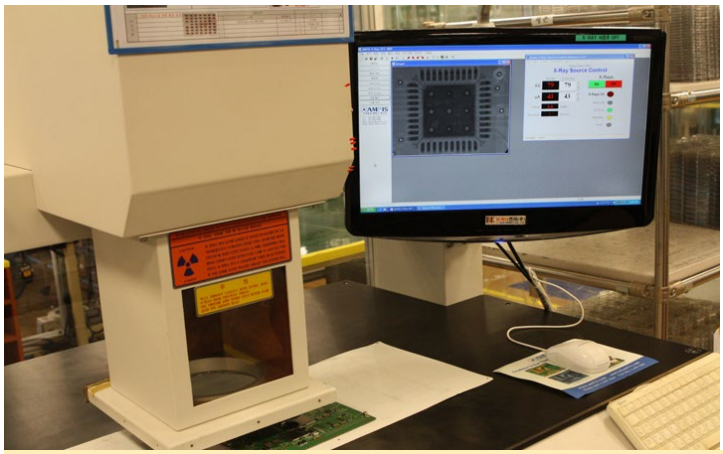


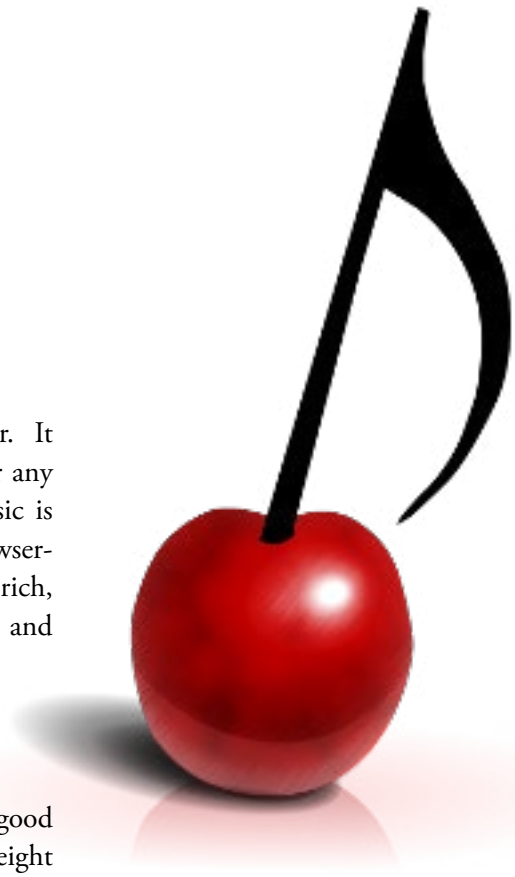
Figure 5 - X-ray inspection to check soldering quality

Figure 6 - Factory lines, where the PCBs move to another soldering machine for various connectors after inspection process, and the go through all the functionality tests

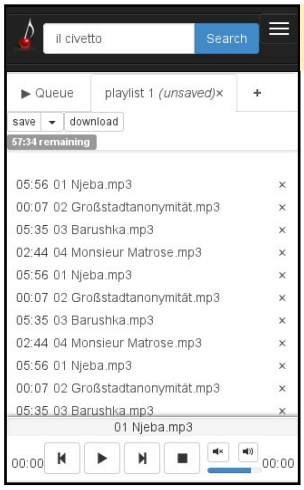
CHERRYMUSIC

YOUR OWN PRIVATE MUSIC STREAM

by @synportack24



CherryMusic is a music streaming server based on CherryPy and jPlayer. It plays music files stored in your PC, smartphone, tablet, ODROID, or any device that has an HTML5-compliant browser installed. CherryMusic is AJAX-based, so it doesn't require page refreshes, making it very fast. The browser-based user interface works on either desktops or mobile devices, and is feature rich, with the ability to create playlists, browse music, setup multiple user-accounts, and much more.



Mobile Webview

A CherryMusic-based system can match any good HTPC system you may come across. It is lightweight and can easily run in the background, without overloading the processors. For the setup described in this article, I used an ODROID-XU4 as a media center. It utilizes a large-capacity hard-drive to store all of the music files. It allows me to stream music to any device in the house, while others are free to watch simultaneously streamed video content using Kodi.

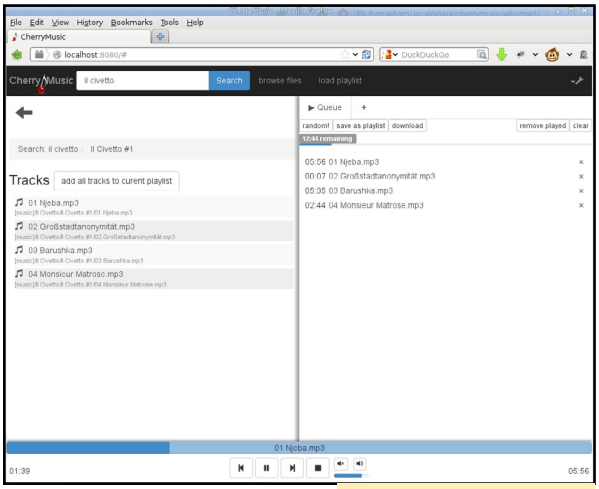
Prerequisites

The following software dependencies need to be met first:

Python (version 2.6 or 3.2)
git
screen

They can be installed using the following command:

```
$ sudo apt-get install python git screen
```



Desktop Webpage

Installation

Setup is easy and straightforward. First, get terminal access (ssh or local) to the device that holds your music files to be streamed. Then, install CherryMusic using the following commands:

```
$ cd ~
$ git clone -b master \
```

```
https://github.com/devsnd/cherrymusic.git
```

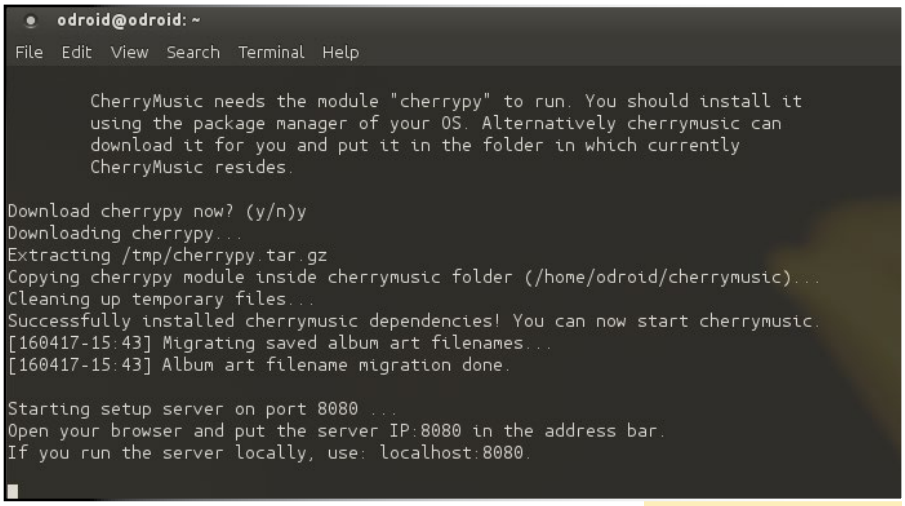
It gets installed in a subdirectory named ~/cherrymusic.

The best way to start CherryMusic is to use the screen utility, which allows you to run CherryMusic in the background. This can be accomplished using the following command:

```
$ screen -mS cherrymusic ~/cherrymusic/cherrymusic --setup --port 8080
```

The CherryPy framework includes a web server, which is used to serve the browser-based user interface. If port 8080 is already in use by another application, you can select another available port by changing the port in above command.

You may be prompted to download cherrypy as well, so select 'Y' for 'yes'. Setup is finished once the you receive a waiting prompt. Your screen should look similar to Figure 3, at which point you should press Ctrl+A+D to ensure that CherryMusic runs in the background.



Installation

Open a web browser, either on the device that has CherryMusic installed, or on a remote networked device and enter the following address. Note that you need to use the appropriate HTTP port if it is not 8080:

```
https://<hostname-of-cherrymusic-device>:8080
```

You can now configure CherryMusic. The "Media base directory" is the directory that contains the music files. Most of the other settings can be left with default values. Click "Save Configuration and start CherryMusic" to move on to create an admin account.

That's it! Next time you connect, you will be asked for username and password. After a successful login, you can stream any and all your music!

Here are some commands for you to experiment with:

Switch to Cherrymusic that was running in the background

```
screen -r cherrymusic
```

Stop CherryMusic from running

```
pkill cherrymusic
```

Further Reading

CherryMusic GitHub
http://bit.ly/1NCqFtb

CherryPy Home Page
http://bit.ly/1NlW5t3

CherryMusic Home Page
http://bit.ly/1r1eeTf

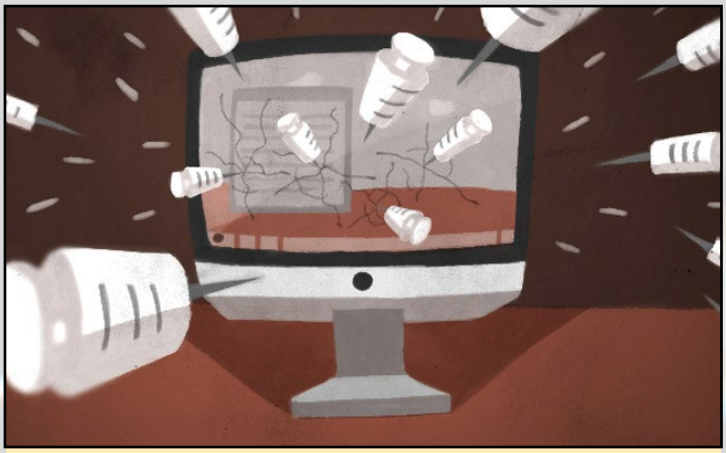
jPlayer Library
http://bit.ly/1VlDyzQ

WIRELESS INJECTION

A HANDS-ON APPROACH TO LEARNING 802.11

by Adrian Popa

In my previous article, we learned how to set all the different ODROID WiFi Modules into monitor mode, which allows us to listen to wireless traffic using Kismet. In this article, we will test traffic injection and analyze open networks which use various protection methods including encryption. We will finish off with some attacks that don't involve breaking the network's encryption. Although this article is technical, its purpose is to familiarize you with how wireless networks work. It is not designed to turn you into a script kiddie who doesn't understand what they're doing. If you have not read last month's ODROID Magazine article about Kismet, avalible at http://bit.ly/20YG7Yg, do so now, since we will be building on past experience. As always, breaking somebody's network without their consent is punishable by law, so only try this against your own networks.



Injection testing can help you keep your network secure from unknown intruders

Preparation

The term "injection" is used to indicate the generation of wireless management traffic based on specially crafted packets that bypass the regular internet stack of your wireless adapter. This means that a program generates packets with whatever fields it needs and sends it to the driver, via the monitor interface, to be sent out even if the packet might not be compliant with the protocol used.

To do our first injection test,, we will need a monitor interface and aireplay-ng, which was covered in the last Kismet article. Optionally, you can specify the channel you want to operate on when creating the mon0 interface:

```
$ sudo airmon-ng start wlan0 6
$ sudo aireplay-ng -9 mon0
```

The program initially sends out broadcast probe requests. These are probe requests which ask any AP listening to respond with a description of itself. Not every AP will respond to this type of request. A list of responding APs is assembled to be used in subsequent steps. If any AP responds, a message is printed on the screen indicating that the card can successfully inject.

At the same time, any AP identified via a beacon packet is also added to the list of APs to be processed in subsequent steps. If a specific AP was optionally listed on the command line ,BSSID and SSID, it is also added to the list of APs to be processed. Then, for each AP in the list, 30 directed probe requests are sent out. A directed probe request is addressed to a specific AP. The number of probe responses received plus the percentage is printed on the screen, this indicates if you can communicate with the AP and how well.

```
adrianp@iqp06:~> sudo aireplay-ng -9 mon0
15:25:00  Trying broadcast probe requests...
15:25:00  Injection is working!
15:25:02  Found 9 APs

15:25:02  Trying directed probe requests...
15:25:02  72:55:9C:DF:98:80 - channel: 1 - 'Ro                    881'
15:25:04  Ping (min/avg/max): 5.536ms/76.800ms/130.476ms Power: -38.80
15:25:04  30/30: 100%

15:25:04  72:55:9C:DF:98:81 - channel: 1 - 'Ro                    81'
15:25:07  Ping (min/avg/max): 29.852ms/76.253ms/119.094ms Power: -38.73
15:25:07  30/30: 100%

15:25:07  72:55:9C:DF:98:82 - channel: 1 - 'i          b'
15:25:09  Ping (min/avg/max): 38.904ms/83.900ms/135.193ms Power: -38.73
15:25:09  30/30: 100%

15:25:09  3C:F8:08:43:86:34 - channel: 1 - 'HL          '
15:25:11  Ping (min/avg/max): 15.031ms/70.436ms/136.017ms Power: -67.67
15:25:11  30/30: 100%
```

Figure 1 - An injection test running

Open networks

We are going to take a closer look at how wireless networks operate under normal conditions by setting up an open network and sniffing its traffic. On my router, I setup a test network called "NASA-HQ-Guests", because "FBI-Surveillance-

Van-3" was already taken.

In order to monitor a specific network, you could use either airodump-ng or Kismet, as both tools can do the same job. To list available networks and their clients, run the following command, assuming your monitoring interface is already up:

```
$ sudo airodump-ng mon0
```

You should see a list of networks with their ESSIDs, network name, and BSSIDs, and AP MAC addresses, along with their power, encryption type, and channel. In this case, we want to capture all traffic for the network with the ESSID "NASA-HQ-Guests", which has a corresponding BSSID of "9C:C1:72:3A:5F:E1" and which operates on channel 1:

```
$ sudo airodump-ng --write open-network-NASAHQ --out-
put-format pcap --bssid 9C:C1:72:3A:5F:E1 --channel 1
mon0
```

For my tests, I had my client, my smartphone, connected to the wireless network and ran the following commands:

```
$ ping -c 3 8.8.8.8
$ ping -c 3 www.google.com
$ ping -c 3 www.hardkernel.com
$ wget -p http://www.hardkernel.com/main/main.php
```

The commands do some basic connectivity tests and simulate a browser loading up Hardkernel's main page, assuming it's not cached. Best of all, it's repeatable and generates about 10MB of traffic.

If you take a look inside your packet capture, you might notice one of two possibilities.

Scenario 1: You will see lots of management traffic, but little or no data traffic. This happened to me when I first tested and I struggled for a while to understand the cause. I suspected problems with modulation, interference from neighbors, directional and multipath transmissions, aliens; you name it! After many tests on the ODROID-C1 with Wifi Module 3, I tried the same test on a PC, with the same wifi card, and I was able to miraculously capture traffic. This means the most likely problem is the kernel and driver combination. I have tried to run the mainline kernel on the C1 to redo the tests but, have failed so far. Wifi Module 0 and Wifi Module 4 don't experience the same issues and allow you to capture traffic on the C1 without issues.

Scenario 2: You will see lots of management traffic and some data traffic. The amount of data traffic you see may vary based on your antenna position, relative to the AP and client, your current wireless interference pattern, and possibly other factors as well. Based on some of my tests, the WiFi Module 0 seemed to have fewer lost packets. However, the tests I ran were in the same room as the access-point, so I can't comment on the range you can get.

I saved a packet capture, which is available from github at http://bit.ly/242cdEq. If you analyze the capture in Wireshark, you will find the following:

- **Packets 254, 256 and 258 represent IEEE802.11 Authentication and Association traffic.**
- **Once associated, the client makes DHCP requests to get an IP address: packets 268, 269. The complete DHCP transaction is not captured.**
- **ARP traffic to find out the MAC address of the gateway: packets 431, 435.**
- **ICMP Echo Requests to 8.8.8.8, packet 437. As you can see three pings were issued, but we were only able to capture one request and no replies.**
- **DNS reply for a www.google.com query: packet 484.**
- **ICMP Echo Reply for a ping to Google, packet 486, and the second request, 616, followed shortly by the third request: packets 627-629. You will notice that the requests packets have been retransmitted several times by the MAC layer. This is transparent to the layer 3 protocols, but may introduce additional latency and jitter.**
- **DNS query for www.hardkernel.com, retransmitted 4 times: packets 638-641.**
- **ICMP Echo Request to Hardkernel's IP, two packets out of three: 728, 737, 738.**
- **Finally, a DNS query and response for www.hardkernel.com, this time we ask 8.8.8.8: packets 814, 816.**
- **HTTP traffic, starting with a three way handshake - only two packets captured: packets 818, 824, and an HTTP GET, packet 825.**
- **HTTP traffic with a lot of retransmissions, such as packets: 925, 927, 929, 931.**
- **Dissociation from the WiFi network: packets 27219, 27223.**

To do HTTP analysis in Wireshark, you can either find GET requests and use the "Follow TCP Stream" option, or you can do bulk processing of all HTTP traffic by going into **File -> Export Objects -> HTTP**. Here you can see all the queries made and can potentially extract data, such as images. Unfortunately, if you try to save the site data, you will find out that most of the images are corrupted and text is truncated. This is because the receiver was unable to pick up all the traffic on the wireless medium. This is quite different from capturing packets on Ethernet, where collisions are avoided and the medium is usually reliable. Summary analysis on captured traffic can show you what limitations you might face when trying to capture encrypted traffic, so you'll know what to expect.

If you were to listen to an open network that didn't have active traffic, you will probably still see interesting things from active stations on the network. For instance, you can expect to see some ARP traffic, NetBIOS broadcasts from Windows hosts, UPNP/SSDP multicast packets from DLNA devices such as media players or routers and even multicast DNS (port 5353) from Linux and Apple hosts that advertise their capabilities.

Even if you don't get much data, it can still be useful because you can analyze the user's requests and potentially pick up unencrypted passwords and cookies sent with POST requests. I encourage you to browse my captured data, or try capturing your own on an open network. Remember what can be seen next time you connect to an open network in a restaurant or airport!

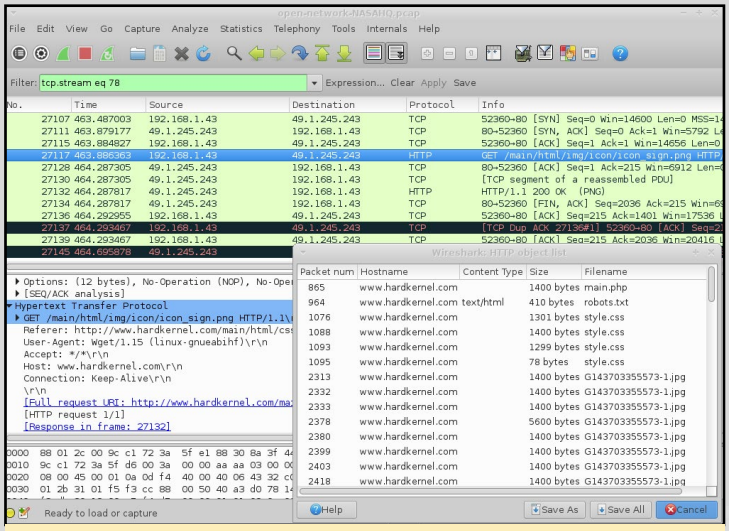
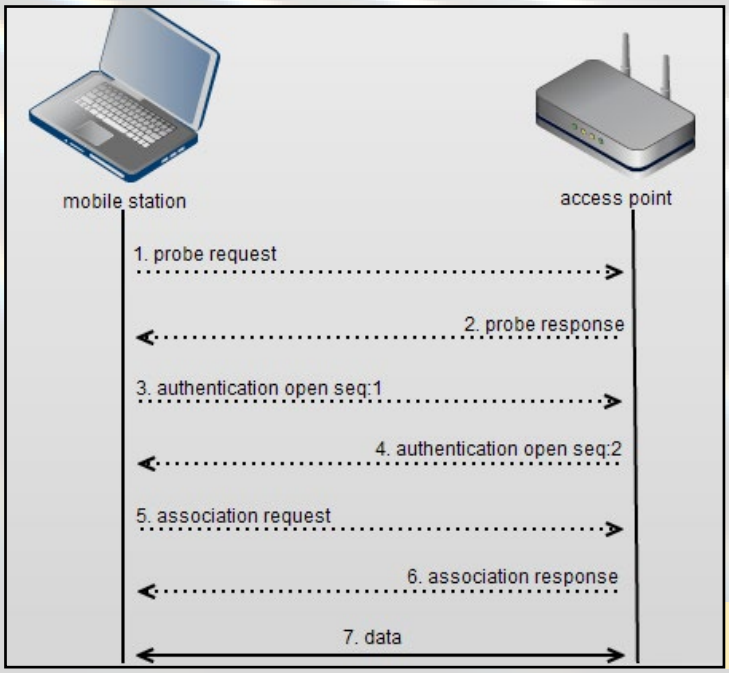


Figure 2 - Open network analysis

How network association works

Connecting to a wireless network involves the following two steps, as shown below:



A mobile station sends probe requests to discover 802.11 networks within its proximity. The probe requests advertise the mobile stations supported data rates and 802.11 capabilities such as 802.11n. Since the probe request is sent from the mobile station to the destination layer-2 address and BSSID of ff:ff:ff:ff:ff:ff, sometimes represented as "Broadcast", all AP's that receive it will respond.

APs receiving the probe request check to see if the mobile station has at least one common supported data rate. If they have a compatible data rate, a probe response is sent advertising the SSID, wireless network name, supported data rates, encryption types if required, and other 802.11 capabilities of the AP.

A mobile station chooses compatible networks from the probe responses it receives. Additionally, compatibility can also be based on encryption type. Once compatible networks are discovered, the mobile station will attempt low-level 802.11 authentication with compatible APs. Keep in mind that 802.11 authentication is not the same as WPA2 or 802.1X authentication mechanisms, which occur after a mobile station is authenticated and associated. Originally, 802.11 authentication frames were designed for WEP encryption, however this security scheme has been proven to be insecure and therefore deprecated. Due to this, 802.11 authentication frames are open and almost always succeed.

A mobile station sends a low-level 802.11 authentication frame to an AP, setting the authentication to open, and the sequence to 0x0001.

The AP receives the authentication frame and responds to the mobile station with authentication frame set to open indicating a sequence of 0x0002.

If an AP receives any frame other than an authentication or probe request from a mobile station, it is not authenticated. Then, it will respond with a deauthentication frame placing the mobile station into an unauthenticated and unassociated state. The station will have to begin the association process from the low level authentication step. At this point, the mobile station is authenticated but not yet associated. Some 802.11 capabilities allow a mobile station to have low-level authentication to multiple APs. This speeds up the association process when moving between APs, known as "roaming". A mobile station can be 802.11 authenticated to multiple APs. However, it can only be actively associated and transferring data through a single AP at one time.

Once a mobile station determines which AP it would like to associate to, it will send an association request to that AP. The association request contains chosen encryption types if required, and other compatible 802.11 capabilities. If an AP receives a frame from a mobile station that is authenticated but not yet as-

Figure 3 - Authentication/association process

sociated, it will respond with a disassociation frame, which places the mobile into an authenticated but unassociated state.

If the elements in the association request match the capabilities of the AP, the AP will create an Association ID for the mobile station and respond with an association response containing a success message granting network access to the mobile station.

The mobile station is then successfully associated to the AP, and data transfer can begin.

For WPA, WPA2, or 802.1X, additional steps are taken after the association step before data is allowed.

Even after a client is connected to an AP it continues to send probe requests, both broadcasted and also "directed" in order to discover new and potentially better access points in its vicinity. The directed probe requests contain the SSID of access points known to the client in hopes that the access point is nearby. Somebody listening on the wireless medium can use this to get a list of a client's trusted SSIDs and use them for tracking, or to set up "evil twin" access points. A good video on the subject can be found at http://bit.ly/1WLAkVH.

Hidden SSIDs

One way you can try to protect a network is to hide its SSID. Most routers have an option so you can set your SSID to be "non-broadcasting". This means that the router still broadcasts beacon frames, but the frames will contain a blank SSID. Clients that want to connect to the network need to know the SSID in advance and send probe requests. In theory, this is great, since an attacker would have to guess a possibly hard SSID. However, in practice, it offers little protection.

When a client sends a probe request it will broadcast the SSIDs it knows about. This broadcast includes the SSIDs for hidden access points too. Anyone listening can pick it up, since it's unencrypted even on encrypted networks, and can use the SSID information. It's not even too hard to expose hidden networks - airodump-ng does it by default.

Let's set up our network to be non-broadcasting and let's have airodump-ng listen to the traffic, only for open networks in our case. You will notice that hidden networks are displayed typically as "<length: xx>" instead of SSID. As soon as a client is in the neighborhood, it doesn't need to actually connect because it will be leaking directed probe requests, and airodump will show you the network name. It's very easy, since all you have to do is wait.

```
$ sudo airodump-ng -t OPN --channel 1 mon0
```

```
                                Before
CH  1 ][ Elapsed: 12 s ][ 2016-03-07 13:21

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

9C:C1:72:3A:5F:E1  -47  25      127        4    0   1  54e  OPN              <length: 14>
                                After
CH  1 ][ Elapsed: 28 s ][ 2016-03-07 13:21

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

9C:C1:72:3A:5F:E1  -50  29      271       12    1   1  54e  OPN              NASA-HQ-Guests
```

Figure 4 - airodump fills in the SSID when it hears it

You can find out how this works by looking inside the packet capture, as shown in Figure 5. For your convenience, the capture with those three frames is available at http://bit.ly/1U9L2F4.

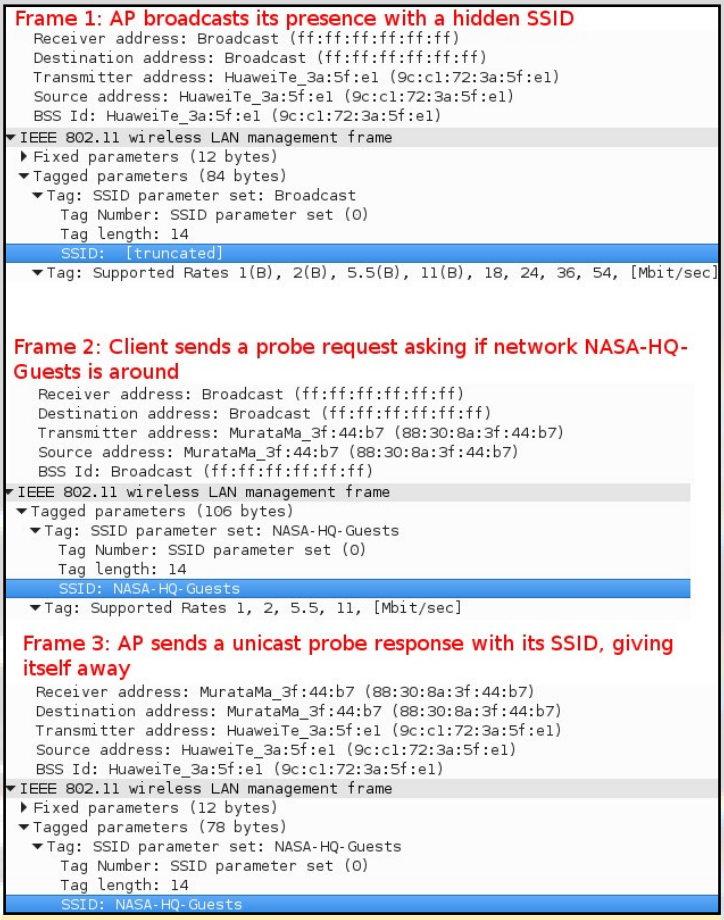


Figure 5 - Packet analysis of hidden SSID connection

MAC access lists

Another popular way of securing networks is by using a MAC access list on the access point. This list specifies which devices can connect to the network. Since the MAC address is unique and doesn't change, this gives the administrator control over which devices can join the network. An attacker would have to find a valid MAC address on that list, and then replace his own MAC to be able to join the network.

If this were a guessing game, the worst case an attacker would need to go through all 2^48 addresses, although probabilistically half would suffice. If it takes 2 seconds to test an address, it would take about 17 million years to go through all of them. If the attacker is not willing to wait that long, he might test with MACs of a particular vendor, such as Apple, Samsung, HTC, etc, in hopes of getting in faster. The first three bytes, 24 bits, of a MAC address represent the vendor ID and a list of vendors can be downloaded from IEEE at http://bit.ly/1Idlxf2.

```
$ wget standards-oui.ieee.org/oui.txt
$ grep '(hex)' oui.txt | grep -i Samsung | wc -l
```

Vendor	Number of OUIs	Max search time (years)
Apple	471	250
Samsung	398	211
LG	127	67
Lenovo	50	26
HTC	29	15
Xiaomi	26	13

Table I - Maximum search time in years for several vendors

Even if it's still a lot of time, scanning one OUI, 24 bits, takes a bit over 6 months. By speeding up the process, using multiple cards, narrowing down which OUI are still relevant, you could carry out an attack that might succeed in a number of months.

Clearly, this is not the way hackers get into your system. Let's explore a faster option, which is listening for a MAC address that already has access. The simplest way is to use Kismet to listen to a network and see which clients are associated with it. If there's little activity ,you will need to wait for a while, but if it's a busy access point, the clients will be revealed instantly. Having a list of connected clients means you have a list of allowed MAC addresses, so your search is over quickly. However, having two devices with the same MAC on the same LAN is serious trouble, so in order not to break the network you can kick the real user off the network by sending him a bunch of deauthentication packets, as we'll see later.

However, knowing what an allowed MAC is and having a device with that MAC are two different things. Luckily, you can temporarily change the MAC address of a wifi adapter, either through GUI, NetworkManager, or through macchanger:

```
$ sudo apt-get install macchanger
$ ifconfig wlan0
$ sudo macchanger --mac 88:30:8a:3f:44:b7 wlan0
$ ifconfig wlan0
```

```
root@iqp06:~# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 7c:dd:90:73:5a:3c
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@iqp06:~# macchanger --mac 88:30:8a:3f:44:b7 wlan0
Current MAC:   7c:dd:90:73:5a:3c (Shenzhen Ogemray Technology Co., Ltd.)
Permanent MAC: 7c:dd:90:73:5a:3c (Shenzhen Ogemray Technology Co., Ltd.)
New MAC:       88:30:8a:3f:44:b7 (Murata Manufactuaring Co.,Ltd.)
root@iqp06:~# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 88:30:8a:3f:44:b7
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 6 - Changing your MAC address

Up to this point, we've explored mostly theoretical concepts of wireless networks. These will help you become a better network engineer, but you're not a "haxtor" yet. Now that the theory is behind us we can get to the "133t" part.

Deauthentication packets

There's a simple and guaranteed way to make trouble in your wireless neighborhood; start flooding the network with deauthentication packets. These frames signal to the access point that a client has left the network. There is no encryption and no protection mechanism around the frames, so anyone with a powerful enough transmitter can spoof these packets and cause the clients to be disconnected. There are some legitimate cases where this should be done:

Wireless intrusion prevention systems (WIPS) routinely send fake deauthentication packets in order to disconnect clients connected to other APs. This is to enforce a set of APs in a specific geographical area for certain businesses or government agencies, as described at http://bit.ly/1T5lNi6.

Kicking the mother in law or kid off the network. This probably can be done easier through your AP's management, but it's technically not illegal to do so with deauthentication packets.

There are also several illegitimate cases for use of deauthentication packets:

- **To reveal hidden SSIDs**
- **To capture WPA and WPA2 handshakes by forcing clients to reconnect**
- **Perform Denial of Service attacks**

To send a deauthentication packet you need only aireplay-ng:

```
$ sudo aireplay-ng -0 10 -a 9C:C1:72:3A:5F:E1\
 -c 88:30:8A:3F:44:B7 mon0
```

The parameters are as follows:

-0 : means deauthentication
10 : how many tries to make. Each try sends 64 packets to the client and 64 packets to the AP. Setting it to zero will keep sending deauthentication packets forever.
-a : is the MAC address of the access point to which the client is connected.
-c : is the MAC address of the client. If you omit it, aireplay will send broadcast frames to disconnect all clients connected to that AP. This is how you do a Denial of Service attack.

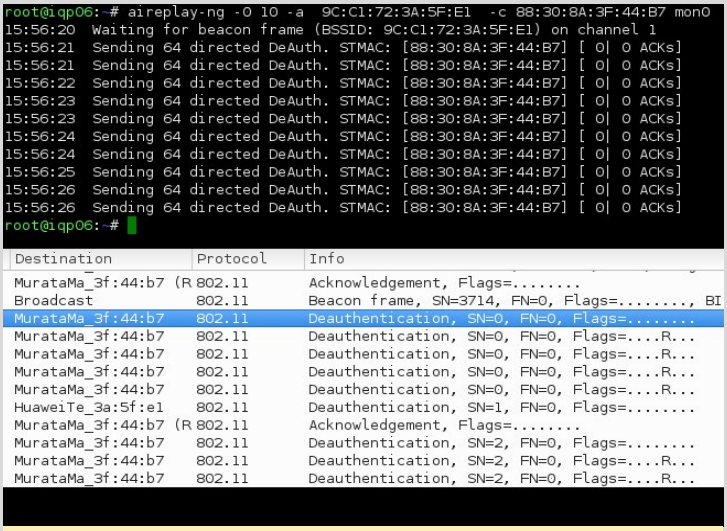


Figure 7 - Deauthentication attack

While aireplay is a powerful tool, there are even better tools available. For instance, if you install mdk3, described at http://bit.ly/1psIKUw, you have the ability to disconnect all wireless clients from all visible access-points, or do other types of attacks, upsetting all your neighbors at once. To install it, you have to download and compile it yourself:

```
$ git clone \
https://github.com/wi-fi-analyzer/mdk3-master
$ cd mdk3-master
$ make
$ sudo make install
```

To disconnect every client that is connected to any access-point on channel 6, you can run the following command:

```
$ sudo mdk3 mon0 d -c 6
```

But wait, there's more! Mdk3 can also emit broadcasts for fake access-points. For instance, I've announced fake access points with WEP encryption on channel 11 that have random names. If this doesn't crash or slow down your neighbor's computers, it will at least baffle them when trying to connect to a new network. Note that the AP broadcast feature didn't work correctly with the ODROID Wifi Module 3, but worked great with the other two wifi modules.

```
$ sudo mdk3 mon0 b -c 11 -h 11 -w -g
```

Visiting Hardkernel's HQ

So far we've played with listening to wifi networks, kicking people off and testing transmission range. Now, let's try something more audacious; let's broadcast specific access-points and influence nearby devices to think they are in a different place.

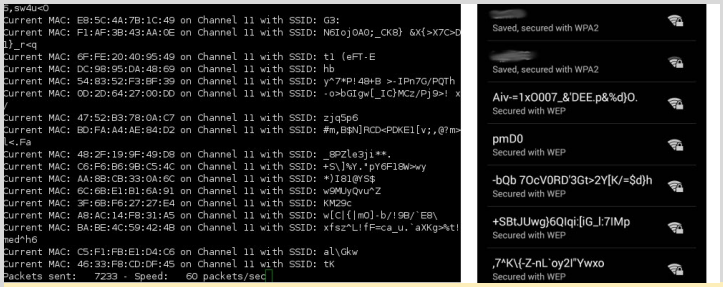


Figure 8 - So many networks to choose from!

We're going to visit Hardkernel's Headquarters in South Korea, so buckle up!

Network based geo-location on mobile devices works by collecting the MAC addresses of your nearby access points and sending the data to a location provider service, which is typically Google if you're on Android. The location service looks up the list of MAC addresses in an internal database and reports back an approximate location. It's funny that network location doesn't rely on IP addresses at all. Maybe because you could be connecting to the Internet through a series of tunnels and the end point could be far away geographically. What we're going to do now is generate enough access-points using mdk3 to fool the mobile device.

To extract access point information we can use wigle.net, available at http://wigle.net/search. Fill in the coordinates and click Search, which is available only for registered users. You can copy and paste the results in a text file. I tried two approaches, but the small area worked for me:

Wide area:
320 access points: `http://bit.ly/1NHtIjX`
Small area:
45 access points: `http://bit.ly/217LBAa`

You can process this file and extract only BSSID and SSID:

```
$ cat hardkernel-small.txt | grep 'infra'\
 | sed 's/infra.*//' | sed -r 's/^map\s+//'\
 | sed -r 's/\t/ /' > hardkernel-small-ssid.txt
```

In order for the attack to be successful, the victim needs to have network location active, set to WiFi - Battery Saving, and not have other location sources available. This means that GPS must be off, or have no GPS satellites in view. If the device is in Airplane Mode, it helps to keep the lock for longer, but it isn't mandatory. Wifi needs to be on and the victim needs to be able to access the Internet in order to communicate with the network location service. Your typical victim can be a tablet without a 3G data connection.

To start broadcasting the networks use mdk3, use the following command:

```
$ sudo mdk3 mon0 b -v\
 hardkernel-small-ssid.txt -g -t
```

You should see the networks appearing in your network list, but if you wait and wait some more… nothing seems to happen. I had Google Maps open on my phone and it stubbornly refused to move. This is because the new networks were conflicting, in terms of location, with the current networks in my area and the location service was confused. In this "confused" state, the location service preferred to keep my current location where it originally was. If you don't have many wireless networks around you, you can skip the next step.

In order to silence the networks around me we need something more drastic - a faraday cage. This is a metallic cage that shields out electromagnetic waves and can cut off wireless networks and 3G data. How do you build a faraday cage out of household items? I tried with tin foil but it wasn't any good. I needed something stronger, like a microwave oven. If I put my phone inside the microwave oven, with the power disconnected, and close the door, I could see that the network signal of available wifi hotspots would significantly decrease.

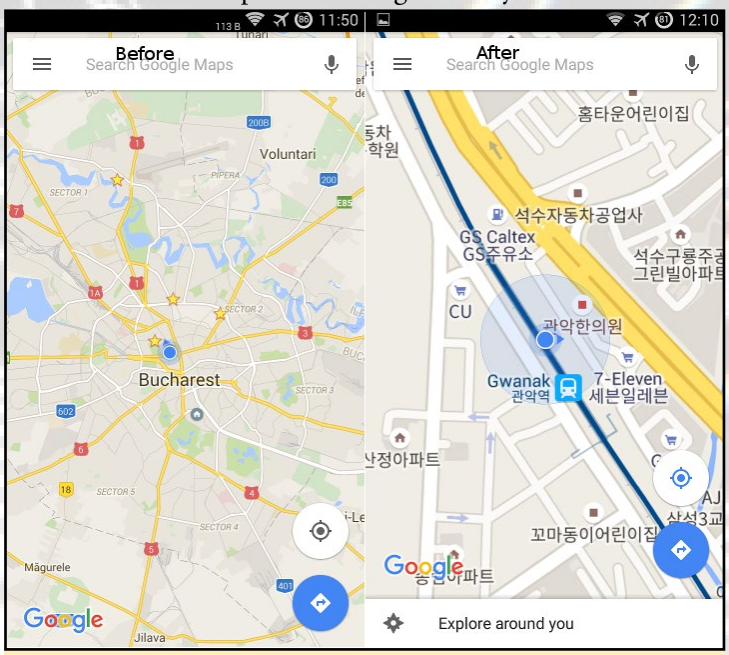


Figure 9 - Messing with location

So, grab your ODROID, power it from a power bank, start mdk3 and pop it into the microwave together with the victim's phone. Sound like a horror story? Don't be alarmed, but make sure that you've unplugged the microwave oven!

I left the phone and ODROID inside for a few minutes and nothing happened. It still had the same location on the map. However, when I had given up and opened the oven door, the location jumped off to Hardkernel's HQ. So, what happened? It turns out the wireless data connection I was using had failed as well, the signal couldn't get through the oven, and the phone couldn't contact the location provider. However, while it was in the microwave, it had collected enough Korean access points that it firmly believed to be in the other location. More interestingly, the phone kept its location even when taken out of the faraday cage. It could still see the same access points and was a bit confused about the other ones around it, but preferred to keep the location until the ODROID stopped broadcasting.

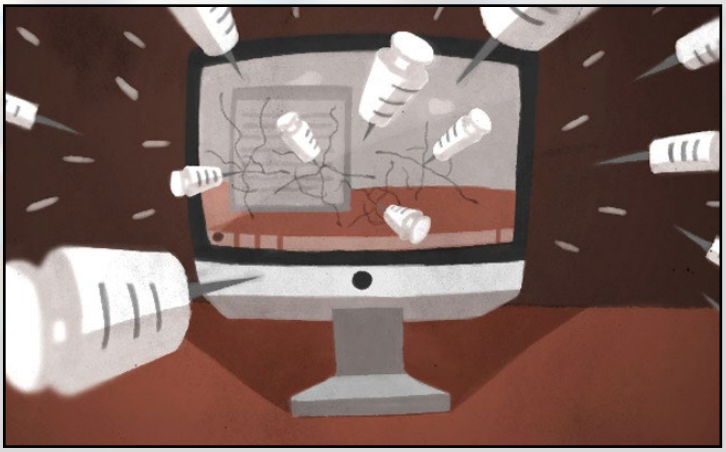
Conclusion

In light of the tests we've performed, we can see that 802.11 is far from secure even when using strong AES encryption. An attacker can easily disconnect any target, and with sufficient disconnect attempts can force the target to connect to a different access-point. The attacker can now perform man-in-the-middle attacks against the victim, without the victim knowing. One method of defence against some of these attacks is the 802.11w standard (http://bit.ly/1rrivQh) which adds protected management frames, but it's not widely supported by all clients.

In terms of privacy, wireless networks are terrible. Every device will broadcast its known networks every few seconds, even if you are already connected. To get around this issue, either delete the known networks from your wireless configuration or keep wifi off when not needed.

Hidden networks and MAC access lists are only effective if the network you're trying to secure is only rarely accessed. For example, the wifi in your vacation house could benefit from these settings. Otherwise they can be considered broken, since they can be circumvented with little effort.

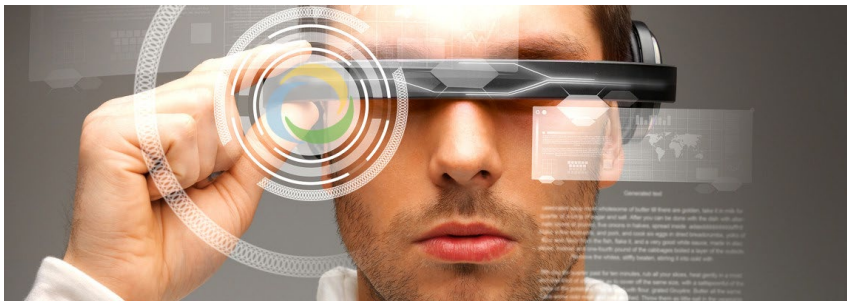
Services based on Wireless hotspots, such as location, are unreliable and can be easily spoofed. For instance, if your target's phone has programs like Tasker that run certain actions based on location data, such as unlocking the phone when it's near a specific AP, you could use a wireless attack to open up new attack avenues. For questions, comments, and further discussion, please visit the support thread at http://bit.ly/1r7wLNv.



AUGMENTED REALITY

USING THE OCAM AND ODROID-XU4

by nowdac@withrobot.com



Augmented reality, or AR, is a part of the virtual reality, VR, which have become a buzzword these days. Simply speaking, AR overlays artificial objects or information over an image of the real world. Although it has been around for a very long time, AR becomes popular with the recent wide use of mobile devices.

In this tutorial, example code will be shown, built and executed. The application will put an interesting animated character over a special marker.

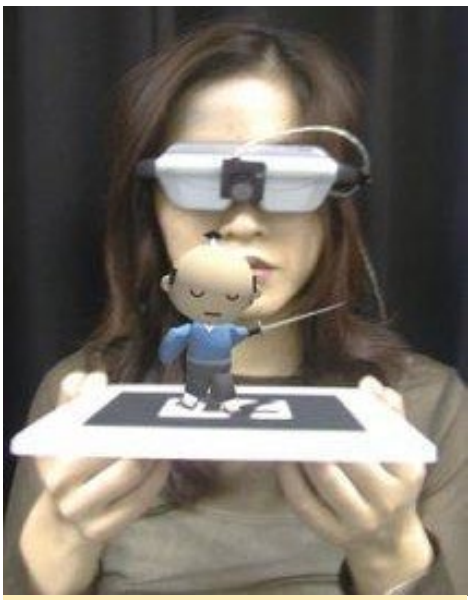


Figure 1 - Example setup of AR with VR (Source: ARToolKit)

Operation Principle

To run our AR demo, we need a video frame that contains the special marker. The program detects the marker and estimates the position and the orientation of it. Using this data, we can put an artificial character at the correct location with proper posture.

For the special marker, we will use ArUco which is a part of the OpenCV library. For further details about ArUCO, please refer the relevant documents at http://goo.gl/Ao6hBg. To make an animated character, we will use Ogre3D

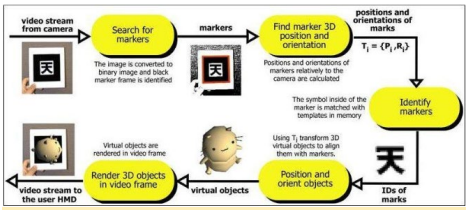


Figure 2 - Operation principle of AR (Source: ARToolkit)

which is an open source graphic rendering engine. You can find more information about Ogre3D at http://goo.gl/k1bMO4 .

Since various libraries are used in this application, it may seem a little complicated. However, when you see that cute little character running in AR, you will agree that it was worth the trouble.

Setup

We need the following items to run the sample application:

- ODROID-XU4
- oCam
- Printer for printing ArUco marker board
- Libraries for OpenCV, ArUco, and Ogre3D

Start by entering the following command, which makes sure that you have the latest package list:

```
$ sudo apt-get update
```

Prepare a "Project" directory to handle the sample code for ArUco and Orge3D by entering the following commands:

```
$ cd ~
$ mkdir Project
$ cd Project
```

Net, install latest version of OpenCV, which is 2.4.9:

```
$ sudo apt-get install libopencv-dev
```

Build

To build ArUco, we need to install the libraries using the following command:

```
$ sudo apt-get install cmake build-essential\
 libicu-dev freeglut3 freeglut3-dev\
 Libgstreamer0.10-dev\
 libgstreamer-plugins-base0.10-dev libxine2-dev
```

We are now ready to download and uncompress the ArUco source code. To do so, type the following commands:

```
$ cd ~/Project
$ mkdir aruco && cd aruco
$ wget http://sourceforge.net/
projects/aruco/\
files/1.3.0/aruco-1.3.0.tgz
$ tar xzvf aruco-1.3.0.tgz
$ cd aruco-1.3.0
```

Finally we can build ArUco:

```
$ mkdir build && cd build
$ cmake ..
$ make -j 4
$ sudo make install
```

The argument after the "make" command, "-j 4" is used to create 4 build jobs. With 8 cores available on the ODROID-XU4, this will help the build process finish much faster.

Test

Once the build step has finished successfully, it is time to test ArUco. For this test, we will need a ArUco marker board, oCam, and the intrinsic parameters of the camera.

To create an ArUco marker board, we can use these commands:

```
$ cd utils
$ ./aruco_create_board 6:4 board.
png board.yml
```

The arguments for aruco_create_board are:

```
aruco_create_board
  [cols:rows]
  [board image filename]
  [board information filename]
```

The board information file, "board.yml", contains the information about the markers, such as IDs and corner positions. By printing "board.png", we can get an ArUco marker board, as shown in Figure 3.

If you know how to calibrate the camera, you can run the calibration and use the intrinsic parameters of OpenCV XML/YAML format. If you are not familiar with camera calibration, in any text editor create a "camera.yml" and fill it with the following content.

```
%YAML:1.0
```

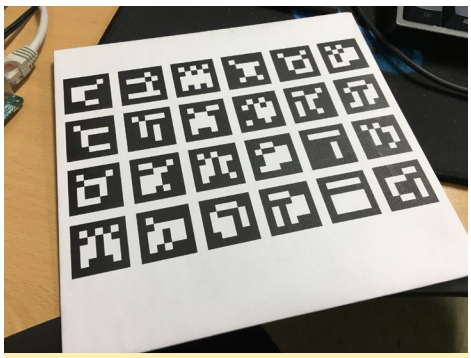


Figure 3 - Printed ArUco board of 6 x 4 makers

```
image_width: 640
image_height: 480
camera_matrix: !!opencv-matrix
    rows: 3
    cols: 3
    dt: d
    data: [ 500., 0., 320.,
                0., 500., 240.,
                0., 0., 1.  ]
distortion_coefficients: !!opencv-
matrix
    rows: 1
    cols: 4
    dt: d
    data: [ 0., 0., 0., 0.  ]
```

You may want to use the GUI editor, such as Pluma, which works similarly to Notepad for Windows.

```
$ pluma camera.yml
```

You can find further information about camera calibration at http://goo.gl/j3j0Xn.

Run

Once the oCam is connected to ODROID-XU4, we are ready to test the ArUco using the following command:

```
$ ./aruco_test live camera.yml
```

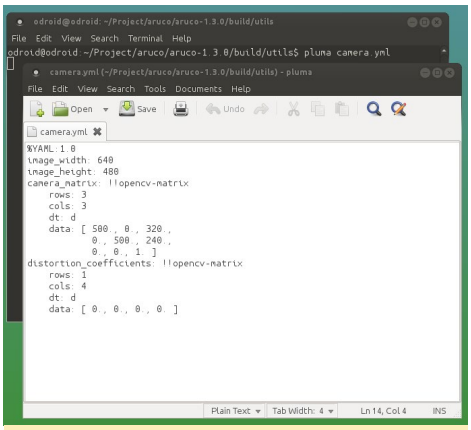


Figure 4 - Creation of camera.yml using the Pluma editor

```
0.025
```

When viewing the ArUco marker board with oCam, the program will detect the markers and shows their IDs and positions in 3D.

The next step is to check if we can detect the board:

```
$ ./aruco_test_board live board.
yml camera.yml 0.025
```

Here, you can see that we specified "board.yml" and "camera.yml" as the parameters to detect the board.

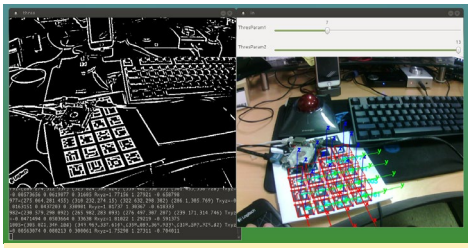
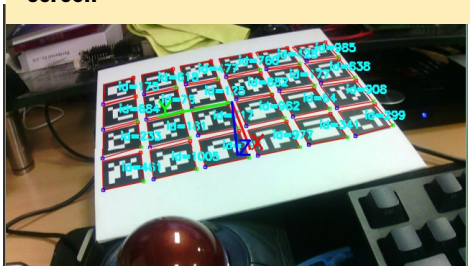


Figure 5 - ArUco marker test program screen

Just like the ArUco marker test program, this ArUco board test program detects the board and shows the posture of

Figure 6 - ArUco board test program screen



the board in 3D as well.

We are almost to the point where we can add our animated green creature to the board. To use Ogre, we need to install the library:

To get the source code containing ArUco and Ogre, use the following commands:

```
$ cd ~/Project
$ mkdir ogre-test && cd ogre-test
$ wget https://sourceforge.net/
projects/aruco/\
  files/ogre-test/ogre-test-
0.0.3.tgz
$ tar xzvf ogre-test-0.0.3.tgz
$ cd ogre-test-0.0.3
```

Unfortunately, this publicly available source code has a few errors. You will need to open up the three source, "*.cpp" files and edit them in order to correct the errors. The fix is the same for each file, and the position where the patch needs to be applied appears next to the file name.

aruco_test_ogre.cpp (line 164)
aruco_test_board_ogre.cpp (line 161)
aruco_test_board_ogre_mask.cpp (line 179)

For the fix, change the following line:

```
if (argv[1]==”live”) TheVideoCap-
turer.open(0);
```

To this:

```
if (string(argv[1]).
compare(“live”)==0) TheVideoCap-
turer.open(0);
```

We also need to correct "CMakeLists.txt" at line 41 by adding "boost_system" as shown below:

```
set (REQUIRED_LIBRARIES ${OpenCV_
LIBS} ${aruco_LIBS} ${OGRE_LI-
BRARY} ${OIS_LIBRARY} boost_sys-
tem)
```

After these 4 corrections, we are ready to build the edited code:

```
$ cmake -DCMAKE_BUILD_
TYPE=Release
$ make -j 4
```

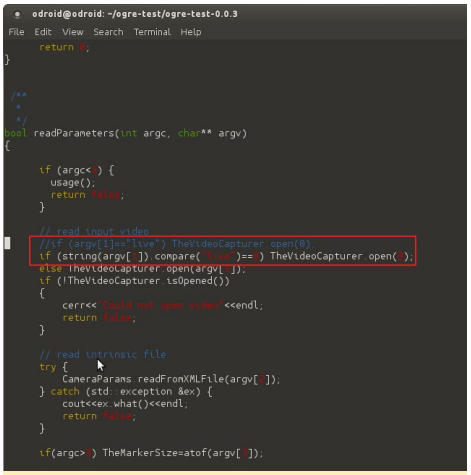


Figure 7 - Source code correction

We now have two more things to do. The first one is to modify the path to Ogre library defined in "plugins.cfg" at line 4 to be suitable to our ODROID environment.

Change from this:

```
PluginFolder=/usr/lib/i386-linux-
gnu/OGRE-1.7.4
```

To this:

```
PluginFolder=/usr/lib/arm-linux-
gnueabihf/OGRE-1.8.0
```

The last thing is to prepare the board parameters and the camera intrinsic parameters. For this, we can just use the files already prepared for the detection tests.

Copy the files "board.yml" and "camera.yml" using the following commands.

```
$ cp ~/Project/aruco/aruco-1.3.0/
build/utils/board.yml
$ cp ~/Project/aruco/aruco-1.3.0/
build/utils/camera.yml
```

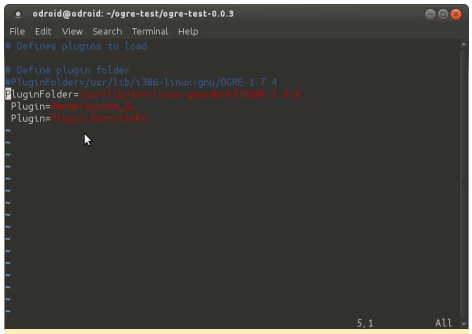


Figure 8 - Modification of the path to Ogre library

Finally, we are ready to start our AR demo.

```
$ ./aruco_test_board_ogre live\
 camera.yml board.yml 0.025
```

Accept all the default settings and click "Accept". Viewing the ArUco marker board with the oCam, we can finally meet our little creature, Sinbad!

Figure 9 - Launcher to run the AR demo

Please refer the video at http://goo.gl/PYRQpw to see this AR demo in action using an oCam and an ODROID-XU4.

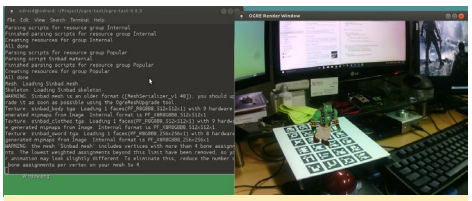


Figure 10 - AR character, Sinbad

LINUX GAMING

ATARI JAGUAR ON ODROID-XU3/XU4

by Tobias Schaaf

Our art editor, Bruno, worked as a teenager on a game store and had the dubious privilege of being able to play games with a Jaguar but not having to actually purchase it

Recently, I did a lot of work on my images, especially on the emulators. I tried to improve the performance and did a lot of testing. In the process of testing, I spoke with @ptitSeb from the OpenPandora forums, and we talked about the Virtual Jaguar core of the libretro project, which is quite slow on all ODROID devices, even on the powerful ODROID-XU3/XU4. He said that they use the Virtual Jaguar standalone emulator to run Atari Jaguar games on OpenPandora, which is a single core ARM board with a Cortex-A8 1GHz CPU). According to him, it should work much better on a powerful board such as the ODROID-XU3/XU4 than it already does on OpenPandora. We checked the code together, and with some patches made by @ptitSeb for OpenPandora device, we were able to port the Virtual Jaguar standalone emulator to ODROID using Qt5 in order to enable OpenGL ES acceleration. After that, I did a lot of testing to see what you can actually do with the emulator and how well it runs on ODROIDs, which is what I will discuss in this article.

Introduction

The Atari Jaguar was the last console from Atari, and was marketed as a 64 bit console to compete with existing 16 and 32 bit consoles, such as the SNES, Sega Genesis, or 3DO. It was first released in November 1993 in North America, and later in Europe and Japan. It had multiple processors for different tasks and was quite fast (13.295MHz) as compared to a SNES (3.58MHz) or Sega Genesis (7.6Mhz). Later, it got a CD and VR headset as an add-on. Although not a bad console, it was already discontinued by 1996, which was not even 3 years after its first appearance. One of the reasons for the low sales might be the weird looking controller, which was not well-designed.

Games

The design of the Atari Jaguar with different chips for different tasks made it hard to develop games for the console because of the fact that Atari wasn't really pushing the development, since only 67 licensed titles were released for the Atari Jaguar. The Jaguar CD add-on got another 15 titles which adds up to a total of 82 games, most of which were released by Atari themselves. Although there are only a few licensed games for the console, in the late 1990s, the console was put under public domain and declared an open platform, and some new games were developed as homebrew games. Even today, new games pop up every now and then, like Alice's Mom's Rescue for the Jaguar CD, which was released for the Atari Jaguar in 2015.

2D Platformer Alice's Mom's Rescue released in 2015 for Atari Jaguar CD

The Atari Jaguar Controller was not well designed, and was so bad that even if you were kidnapped by it for a long time, you wouldn't develop Stockholm Syndrome

Although the Atari Jaguar library is quite limited, it has some very good titles, such as Raiden, Rayman, Pitfall and Cannon Fodder, which are also known from other consoles.

Rayman for the Atari Jaguar vs Rayman for the Playstation I

The Atari Jaguar CD added more games to the library which were improved with movie cutscenes, better graphics and better sound. Still, there were only a few titles, so the Atari Jaguar CD could not compete with the Playstation 1 or Sega Saturn.

ODROID support

So how good is the Atari Jaguar supported on ODROIDs? I'm afraid it's actually not that well supported. There is a libretro core for Virtual Jaguar which can be used for retroarch, but I found it to be very slow and only a few games actually work with it at all. Some games run nearly full speed on an XU3/XU4 like Cannon Fodder, but these games are rare. Other games such as Pitfall or Alien vs. Predator run at a very low frame-rate (below 30 FPS), and, as I already said, many games don't work at all. For example, Rayman won't start on the libretro core.

The alternative is the standalone emulator. The Virtual Jaguar emulator is working well, but not well enough to run on all ODROIDs. The C1 will never be able to run a game fluently. I'm not sure about the C2 since I'm not sure if it even works without X11 drivers. Nor do I know the quality of the drivers yet, since at the time of the creation of this article there were no X11 drivers for C2 yet. The U3, although twice as fast as the C1, is probably too slow for many games as well. Some games, like Cannon Fodder, might work, but generally I wouldn't count on it. That means that the Atari Jaguar is most likely limited to the XU3/XU4 only. The emulator actually has a filter that allows you to use bilinear filtering for graphics, which can greatly increase picture quality:

Differences between no filter and with bilinear filter - the upper half uses the filter and the lower half does not

I took the time to test 56 different ROMs for the Atari Jaguar and will add a compatibility list for it. I also would like to highlight some of the games and tell you what I think about them.

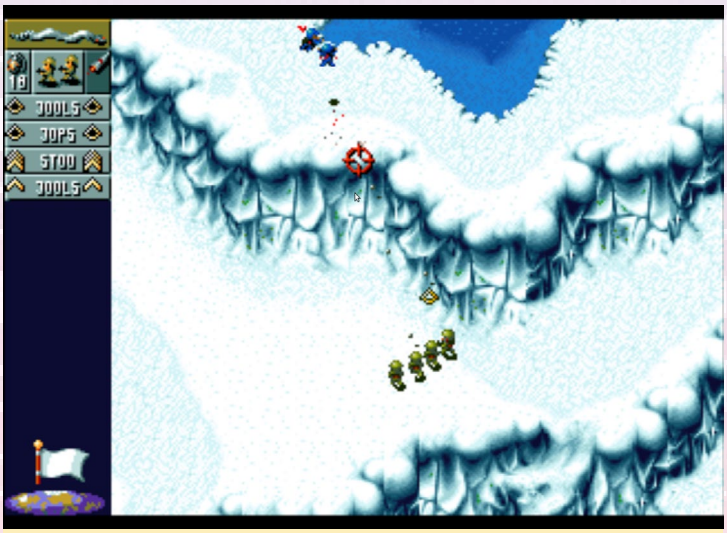
Cannon Fodder

Cannon Fodder is a strategic action game where you send a small group of soldiers into battle to fight other soldier and destroy buildings. The game is very fun to play, and although the original game was controlled via a mouse, such as on the

As the name implies, your soldiers are basically Cannon Fodder, and are very replaceable

Amiga, it works surprisingly well on the Atari Jaguar. The virtual mouse is very responsive using a controller, and since you only need two buttons to control the game it's easy to handle.

You have a lot of soldiers, called recruits, waiting for you to send into battle, so if one of your soldiers dies, he's simply replaced by a new recruit. The sheer number of recruits you get should show you how often you can die. It also shows that soldiers are very replaceable. You fight in different settings: Jungle, Arctic, and Desert.



A nice arctic level on Cannon Fodder where you fight with your soldier

I really like the game, but there are some alternatives to choose from. The game came out for many different consoles. I used to play it on the Amiga, where there's even an Cannon Fodder 2 and some other spin-offs of the series. Still, the Atari Jaguar version is nice and you should try it.

Flashback

Flashback – The Quest for Identity is another nice game that I know and love from the Amiga. This port to the Atari

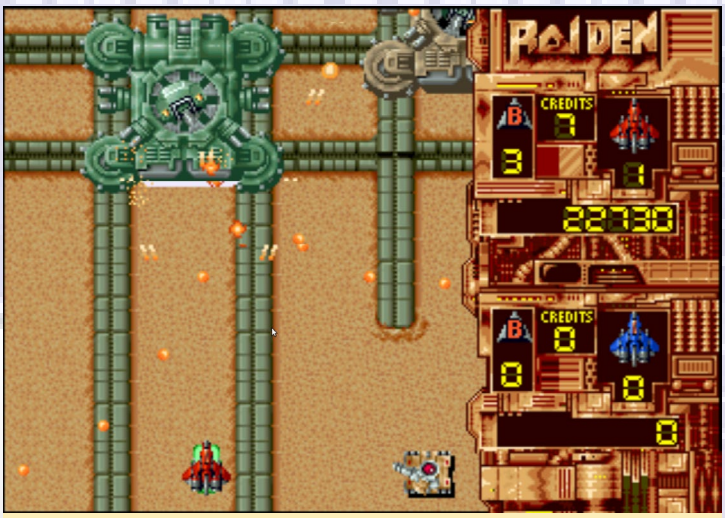
The Jungle is where you start your journey

Jaguar is also very good and it's a nice action platformer game to have for the Atari Jaguar.

Although I miss the better tunes of the Amiga, which had much better music quality in my opinion, it's also missing some of the cinematic scenes of other consoles. However, the game is fully there and as great to play as any other version of the game. I guess there's a reason why this game holds the Guinness World Record as the best-selling French game of all time, and from my childhood and playing this games for many many many hours on the Amiga, I can tell that it's well earned.

Fighting in futuristic settings, mines, and teleporting enemies are only a few obstacles you face

The game can be very hard at times, especially when finding out the right way how to deal with enemies. Sometimes it's best to distract them rather than directly confront them, and other times you just hope your shield will save you. I still



Raiden for the Atari Jaguar nice to play and should even support 2 player mode

prefer the Amiga version over the Atari version, but it's a good game no matter what.

There are also some very good arcade shooters for the Atari Jaguar, for example Raiden and Tempest 2000. Raiden actually has nice graphics and very good music which I can say I really enjoy. The gameplay, on the other hand, is somewhat hard, at least for me. There are some power ups spread throughout the level, but they are still rare, and if you die, you lose all power ups and have to start anew with collecting them, which doesn't mean you're going to get them again in the same level. Without power ups your craft will be extremely weak. Even "mid-boss" enemies will take a very long time to kill if you don't have any power ups for your weapons, which makes the game very hard in my opinion. You do so little damage, that without bombs you probably don't have the chance to kill an enemy at all. Still, the game is fun to play and I guess multiplayer is the best way to beat the game.

Tempest 2000 was an effort from Atari to catch up on with games that had 3D environments, and show how different they were from what was available on the PSI/Sega Saturn systems

Tempest 2000 is very interesting, although it doesn't look like much at first, it's actually quite fun to play. It has nice music, and the graphics are actually not that bad, even if they are very minimalistic. It comes with nice particle effects for explosions and items to collect. You can collect a couple of power ups and even get an NPC supporter that helps you shoot enemies. It's slightly laggy on the ODROID, but still very playable. It's actually quite addicting.

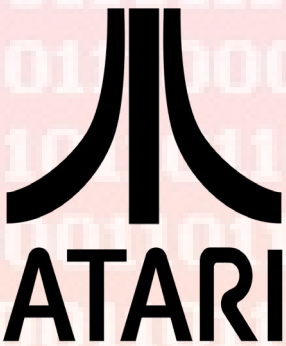
There are more arcade shooters which are notable, such as Defender 2000 andProtector. Atari Carts is a nice racing game similar to Mario Carts. Pinball Fantasies is a good Pinball game on the Atari. I also love the port of Worms which plays quite nice on the Atari Jaguar, although also originally controlled with mouse and keyboard.

General thoughts

Overall, the Atari Jaguar is a good console similar to a Sega Genesis or the SNES. Sadly. there are only a few games available and not even all games are working. Only the XU3/XU4 are currently able to play Atari Jaguar at an acceptable speed. The Libretro core of Virtual Jaguar is not working well, since only a few games work with it. The standalone emulator works much better.

Many of the games are actually ports from other consoles, and are either as good as on the Atari or might even be a little bit better, so it is worth trying other versions of the games to see what you like the best. Many games that I know from the Amiga are actually best played as the original version. Other games, like Pitfall, also exist on other consoles, which are less demanding, like Sega 32X, and can therefore run on more devices than just the XU3/XU4. Atari Jaguar CD games don't seem to work so these games can't even be played. Fortunately, you can now play the games on controllers like the XBox360 controller rather than the clunky Atari Jaguar controller. The setup of the controller is rather easy thanks to the Qt5 based GUI and the SDL2 drivers for joystick support.

It's an interesting console for the XU4, but it's not too much of a problem if your ODROID is not able to run the games, since there are probably ports for other consoles as well. If you want to try it Atari Jaguar on your ODROID XU3/XU4 you can download it from my repository. I hope you will have some fun with the piece of gaming history on your ODROID.



We all have a place in our old school gaming hearts for Atari, which could have been a better console, but was outclassed the likes of the Sega Dreamcast

Compatibility List

Air Cars	not working	
Alien vs Predator	working	a little laggy
Atari Karts	working	feels slow (not laggy but slow)/ Menu seems off screen
Attack of the Mutant Penguins	not working	
Battle Sphere Gold	not working	only works until menu (but you can listen to nice music)
Brutal Sports Football	not working	
Bubsy - Fractured Furry Tails	working	
Cannon Fodder	working	Main Theme sounds slighly too fast
Checkered Flag	working	
Club Drive	partly working	graphics seems off and it crashes
Cybermorph	not working	
Defender 2000	working	
Doom - Evil Unleashed	not working	
Double Dragon V	working	
Dragon - The Bruce Lee Story	partly working	Background and Menu are not working correctly
Evolution - Dino Dudes	working	Button Layout?
Fever Pitch Soccer	working	
Fight For Your Life	working	issues in Background
Flashback	working	
Flip Out	not working	
Hover Strike	not working	crashes when you start a game
Hyper Force	not working	stops when you start a game
International Sensible Soccer	not working	crashes when you start a game
Iron Soldier	not working	stops when you start a game
Iron Soldier 2	not working	terrible sound issues in menu, stops when you start a game
I-War	working	
Kasumi Ninja	working	may require multiple starts, slow menu, missing ground

Missile Command 3D	working	all very slow (frameskipping?)
Music Demo	not working	hangs emulator
Native Demo	working	no sound
NBA Jam Tournament Edition	working	
Pinball Fantasies	working	odd controls
Pitfall - The Mayan Adventure	working	
Power Drive Rally	not working	crashes when you start a game
Protector - Special Edition	working	
Raiden	working	(only starts with BIOS)
Rayman	working	very fun to play
Ruiner Pinball	not working	
Skyhammer	partly working	terrible sound issues, you can get in game, but runs unstable
Soccer Kid	working	
Space War 2000	working	confusing game could have been nice
Super Burnout	not working	Only menu works, stops when going into game
Super Cross 3D	not working	partly working without BIOS, but with no sounds and many glitches
Syndicate	working	not a good game for Consoles
Tempest 2000	working	takes a long time to load
Theme Park	partly working	graphical glitches, runs unstable
Total Carnage	not working	crashes soon after you're in the game
Trevor McFur in the Crescent Galaxy	working	hard game (no music)
Troy Aikman NFL Football	working	sometimes crashes
Ultra Vortek	not working	
Val D'Isere Skiing & Snowboarding	working	Graphical issues, too vague controlls
White Men Can't Jump	not working	freezes in various situation
Wolfenstein 3D	not working	
Worms	working	menu slightly off screen
Zero 5	working	hard to maneuver
Zool 2	working	

ANDROID DEVELOPMENT

ANDROID SUPPORT LIBRARY

by Nanik Tolaram

One common problem with Android app development is the number of different Android version that need to be supported, which can be daunting at times. In the early days of Android, most developers struggled to make sure that their app could be run on older version of Android. As time passed, Google came out with a library to make developing apps easier. For the last few years, Google has strongly encouraged developers to use the Android Support Libraries. The end goal of the library is to reduce the level of code needed for application to run on different Android versions. Google has been frequently releasing updates to this library. With the help of Android Support Library, developers just have to focus on writing their apps and leave the hard work of Android version portability to the library. In this article we're going to take a look at the Android Support Libraries and how to use them.

Setup

The first thing to do is to make sure you have the support library installed in Android Studio. Use the "android" application that resides inside <android_sdk_folder>/tools directory. Under the Extras section, select "Android Support Library" to download it into your SDK. Refer to Figure 2 to see how your SDK directory should look once the library is downloaded.

Library Versions

As seen in Table 1, there are a number of support libraries available, but you only need to use what is right for your project. The directory listed in the table contains the all the relevant files such as source code, resources, and config files of the library.

Looking at the table, you can see that majority of the support libraries are related to user interface, which is a large source of problems that developers have to deal with when designing Android

Figure 1 - Android SDK Downloader

Extras		
GPU Debugging tools	1.0.3	Installed
Android Support Repository	30	Installed
Android Support Library	23.2.1	Installed
Android Auto Desktop Head Unit emulator	1.1	Installed
Google Play services for Fit Preview	1	Installed
Google Play services for Froyo	12	Installed
Google Play services	29	Installed
Google Repository	25	Installed
Google Play APK Expansion Library	3	Installed
Google Play Billing Library	5	Installed
Google Play Licensing Library	2	Installed

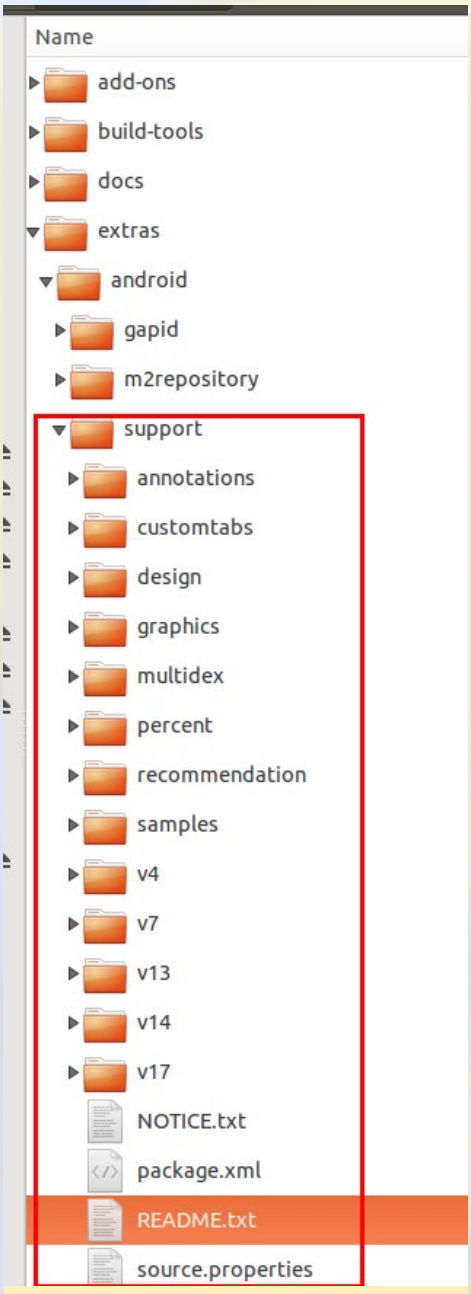


Figure 2 - Support library inside the SDK folder

Support Library	SDK Directory
v4 Support Library	<sdk>/extras/android/support/v4/
Multidex Support Library	<sdk>/extras/android/support/multidex/
v7 Support Libraries	<sdk>/extras/android/support/v7/appcompat/
v7 cardview library	<sdk>/extras/android/support/v7/cardview/
v7 gridlayout library	<sdk>/extras/android/support/v7/gridlayout/
v7 mediarouter library	<sdk>/extras/android/support/v7/mediarouter/
v7 palette library	<sdk>/extras/android/support/v7/palette/
v7 recyclerview library	<sdk>/extras/android/support/v7/recyclerview/
v7 Preference Support Library	<sdk>/extras/android/support/v7/preference
v13 Support Library	<sdk>/extras/android/support/v13/
v14 Preference Support Library	<sdk>/extras/android/support/v14/
v17 Preference Support Library for TV	<sdk>/extras/android/support/v17/
v17 Leanback Library	<sdk>/extras/android/support/v17/leanback
Annotations Support Library	<sdk>/extras/android/support/annotations
Design Support Library	<sdk>/extras/android/support/design
Custom Tabs Support Library	<sdk>/extras/android/support/customtabs
Percent Support Library	<sdk>/extras/android/support/percent
App Recommendation Support Library for TV	<sdk>/extras/android/support/recommendation

Table I - Android Support Libraries and SDK Location

application. Also notice that the library uses numbering such as, v7, v13, v14, etc. Table 2 outlines the different meaning for each of the version number

v4	Android 1.6, API level 4 and higher. The library includes a large set of APIs, compared to the other libraries, for: data handling, network connectivity, and programming utilities.
v7	Android 2.1, API level 7 and higher
v13	Android 3.2, API level 13 and higher.
v14	Adds support for prefered interface
v17	Targeted for Android TV devices.

Table 2 : Android Support Libraries Version

Let's take a look at the content of the Design Support Library that resides inside the "<sdk>/extras/android/support/design" directory. The content of the directory can be seen in Figure 3.

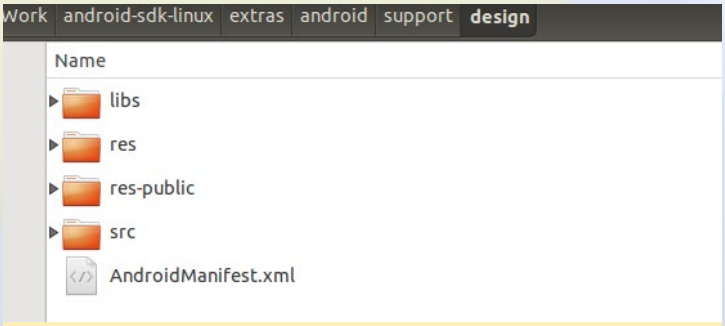


Figure 3 - Design Support Library directory

There is a /src directory, which is empty. The source code for the support library is available for download from the AOSP repository at http://bit.ly/1Ua2clG.

Demo

We're going to take a look at some demo code that you

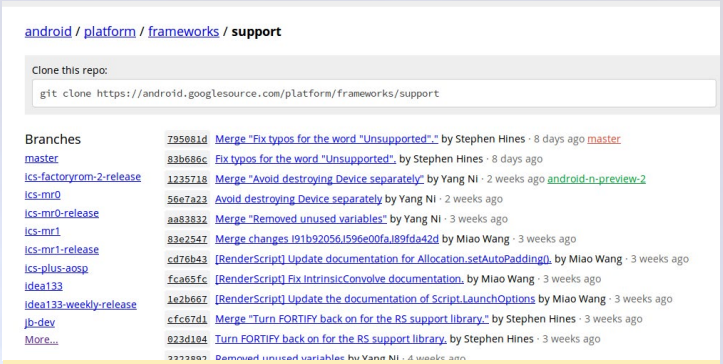


Figure 4 - Support Library Source Repository

can checkout from the GitHub repository at http://bit.ly/1QuHhmK. Figure 5 shows the app running on the three different Android devices: Nexus S - Android 4.1.2, Nexus 5 - Android 6.0, and Samsung Galaxy Express - Android 4.1.2. Notice how the app still looks and behave the same on all the devices.

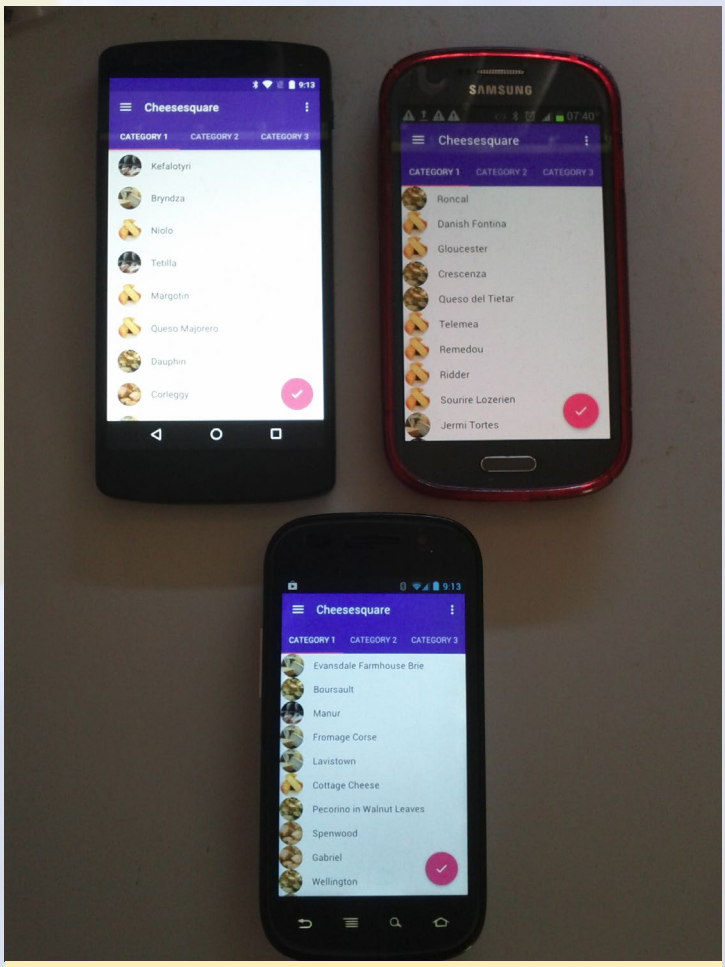


figure 5 - Three different devices with two different Android versions running on them

One thing that you need to decide when writing an Android app is the lowest version of Android you want your app to support. There is no rule, since it all depends on the market that you are targeting, resources that you have, and many other things. The demo app we are looking at supports devices from

Gingerbread, Android 2.3 API Level 9, and later. You can see the devices in Figures 6a and 6b.

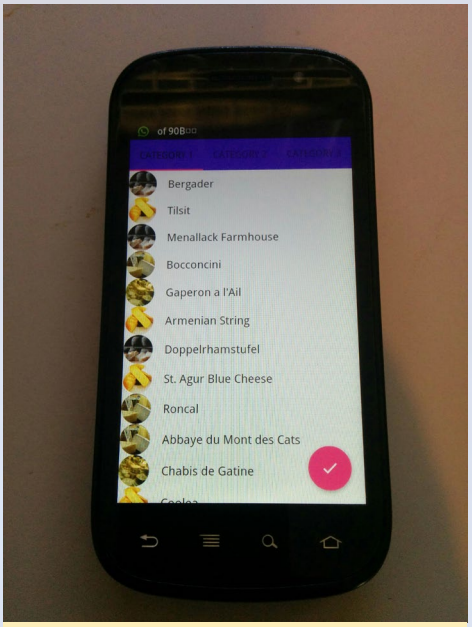


figure 6a – Nexus S Gingerbread

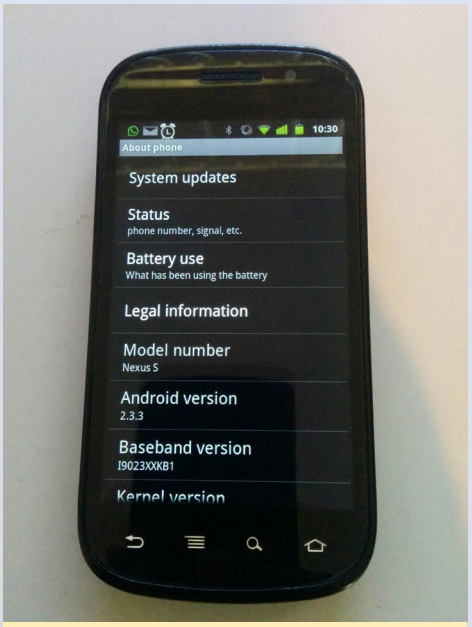


Figure 6b – Nexus S Device Configuration

The configuration that specifies the lowest Android version your app wants to run on is set in the build.gradle file. The code below shows the minSdkVersion attribute that controls this. Android API and version numbering information can be found at http://bit.ly/1Ua4mlk.

```
defaultConfig {
    applicationId "com.support.
android.designlibdemo"
    minSdkVersion 9
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}
```

The code below shows the declared dependencies inside the build.gradle file that indicates to the build system which support library to use. The demo app uses the Design Support and Cardview library, v7, both using version 23.1.1.

```
dependencies {
    compile 'com.android.
support:design:23.1.1'
    compile 'com.android.
support:cardview-v7:23.1.1'
    compile 'com.github.bumptech.
glide:glide:3.6.0'
    compile 'de.hdodenhof:circlei
mageview:1.3.0'
}
```

The custom view that the app is using is shown below, which can be found inside the res/layout/activity_main.xml file:

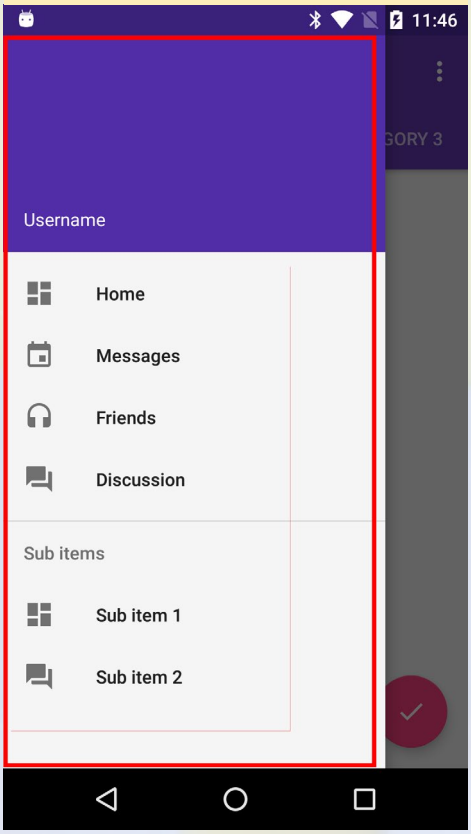
```
<android.support.v4.widget.Draw-
erLayout xmlns:android="http://
schemas.android.com/apk/res/an-
droid"
    xmlns:app="http://schemas.
android.com/apk/res-auto"
    android:id="@+id/drawer_lay-
out"
    android:layout_height="match_
parent"
    android:layout_width="match_
parent"
    android:fitsSystemWindows="t
rue">
    <include layout="@layout/in-
clude_list_viewpager"/>
    <android.support.design.wid-
get.NavigationView
        android:id="@+id/nav_
view"
        android:layout_
height="match_parent"
        android:layout_
width="wrap_content"
        android:layout_
gravity="start"
        android:fitsSystemWindows
="true"
        app:headerLayout="@lay-
out/nav_header"
        app:menu="@menu/drawer_
view"/>
</android.support.v4.widget.Draw-
erLayout>
```

Figure 7 shows which part of the app is using the new support library's custom view, called android.support.design.widget.NavigationView.

There are 2 parameters inside the new widget: app:headerLayout and app:menu. These point to the file res/layout/nav_header.xml and res/layout/drawer_view.xml. The drawer_view contains the submenu text along with the icons to be displayed while the nav_header contains the top part of the view. In Figure 7, it's the text "Username".

The other custom view used is an

Figure 7 - NavigationView widget



droid.support.v4.widget.DrawerLayout which, is the main layout for the whole app encapsulating the android.support.design.widget.NavigationView. The DrawerLayout view knows which view to popup based on the code below:

```
switch (item.getItemId()) {
    case android.R.id.home:
        mDrawerLayout.openDrawer(GravityCompat.
START);
        return true;
}
```

The GravityCompat.START tells the library to look inside the DrawerLayout view to iterate through its child views to find which view has set the layout_gravity to "start", which is defined inside activity_main.xml in the NavigationView.

This should give you a headstart on using the Android Support Library. We will look more in depth at the support library in future articles.

Resources

Android Support Library Features
http://bit.ly/1ei6hQ5

Android Support Library
http://bit.ly/1qIpxiK

BATTLE FOR THE SOLAR SYSTEM: PANDORAN WAR

A FUN 2D MISSION-BASED SPACE SHOOTER

by Tobias Schaaf

The Pandoran War offers a very straightforward gaming experience for your ODROID. The game is a non-linear mission system, which means that you simply jump in by choosing any one of the dozens of missions that interest you across the entire game. It takes place between books 2 and 3 of the novel trilogy, so it might be worth reading some of the free chapter samples on the Pandoran War website at http://bit.ly/1YRi6Rj to learn more about the game's backstory. The game runs on SDL2 and OpenGL ES, so it will take advantage of the Mali GPU and powerful 4 or 8 core processors available on devices like the ODROID-XU4 or ODROID-C2.

Installation

To install the Pandoran War, you'll need to first add @meveric's repository to your distribution, if you haven't already:

```
$ su
# cd /etc/apt/sources.list.d/
# wget http://oph.mdrjr.net/meveric/\
sources.lists/meveric-all-main.list
# wget -O- http://oph.mdrjr.net/meveric/meveric.asc
| apt-key add -
```

Next, simply install it as you would any other game or application via apt-get:

```
$ apt-get update
$ apt-get install tbftss-odroid
```

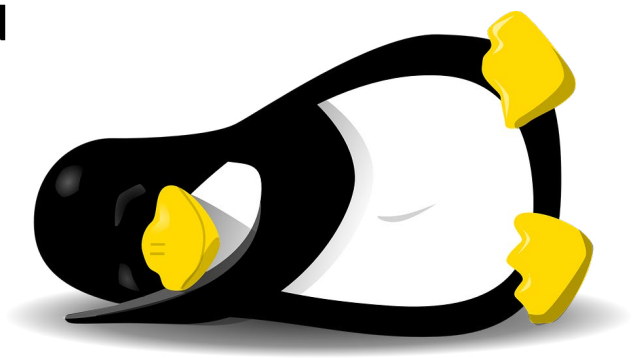
A fun 2D shooter that is a gem for Linux gamers on ODROID

For questions, comments and suggestions, please visit the original thread on the ODROID forums at http://bit.ly/1prxGY1.

BABY NAP (NIGHT ACTIVITY PROGRAM)

PART I - HARDWARE CONFIGURATION

by Marian Mihailescu



Being a parent for the first time is a big challenge. Having a crying baby and not knowing how to calm him/her down quickly can be extremely taxing, especially on working parents that take turns taking care of the baby. I hope the Baby Night Activity Program (Baby NAP) will help parents in this predicament.

Consider a baby crib with a pressure mat sensor that would go under the child. A Radio-frequency identification (RFID) reader attached to the baby crib, coupled with RFID bracelets or rings worn by the parents, can determine when the baby is picked up, for how long, and by which parent. A sound sensor can be used to record the amplitude of baby's voice and determine if the baby is crying, for how long, and how hard. A Pyroelectric (Passive) InfraRed sensor (PIR) sensor and a light sensor can be used to record the ambient motion and light. All these sensors can be connected a Single Board Computer (SBC) like the ODROID C1/C1+. This board can upload the sensor data to a cloud based service like the Amazon Web Services (AWS), for additional processing.

For those wanting to extend this setup, they can for example, use a Philips Hue bulb to control the light color in the baby room, and a microphone can filter out the baby noise and determine, if the parents are singing, how effective that is in putting the baby to sleep, and so on.

With an AWS lambda function and a dashboard with all the sensor readings, the parents can make intelligent conclusions about the baby's sleep preferences such as:

- does (s)he prefer a certain ambient light color?
- does (s)he like to be moved around when put to sleep?
- is (s)he getting asleep faster when singing?
- is (s)he waking up more when there is light in the room?

Parents can also check how much night time they spend with the baby, to settle more quickly the discussion on whose turn it is to put the baby to sleep. Although this project does not intend to replace a baby monitor, it could send notifications to the parents when the baby starts crying using Amazon Simple Notification Service (SNS).

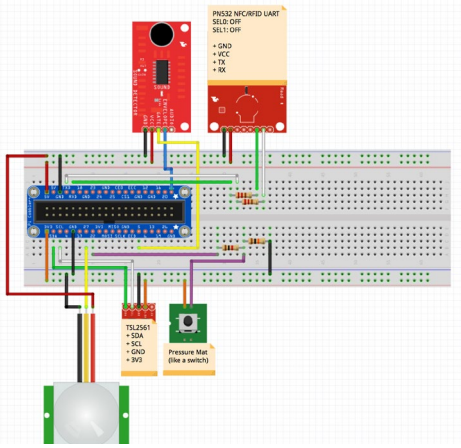
Components used

- ODROID C1/C1+
- Adafruit PN532 NFC/RFID controller breakout board
- http://bit.ly/1O0wK30
- Adafruit 13.56MHz RFID/NFC Bracelet
- http://bit.ly/1pP4Q3G
- Adafruit RFID/NFC Smart Ring
- http://bit.ly/1WXBrSi
- Adafruit TSL2561 Digital Luminosity/Lux/Light Sensor Breakout
- http://bit.ly/1rFj0WX
- PIR Motion Sensor (Ebay)
- SparkFun Sound Detector
- http://bit.ly/1HZN6Y1
- Pressure Mat PM1 (Ebay)

Hardware connections

The ODROID C1 has a 40-pin connector that is mostly compatible with the 40-pin connector of the Raspberry Pi (except the 2 analog inputs), so I will be using a Pi Cobbler+ in the accompanying diagrams. There are 2 power rails on the breadboard, one for 5V (red wires) and one for 3.3V (orange wires). There are also 2 rails for ground (black wires), on each side of the breadboard. I will be using some test programs for

Figure I - First step of wiring



each sensor, written in Python and using the WiringPi2 library, available for the ODROID at http://bit.ly/23Rwf7T. The WiringPi GPIO mapping for the C1 is available at http://bit.ly/1Ejubsm. Figure 1 illustrates the circuit used.

Note that one can also use relevant components from Hardkernel's C1 Tinkering Kit (http://bit.ly/1YNPN6k). If using this kit, one would have to pay attention to the pin assignments on the T-Breakout PCB and alter the information presented here to match the variations.

PIR Motion Sensor

The PIR motion sensor has 3 pins: VCC, going to the 5V rail, GND and DATA (yellow wire). We will be using header #13 for the DATA pin, exported by WiringPi as GPIO 2. The input from the PIR sensor is 1 when there is motion and 0 otherwise. The sensor has 2 potentiometers, one for sensitivity and one for time (how long after the motion was detected - the DATA output will be 1). Using trial and error, I set the potentiometer to output 1 for a 10-second detection duration. Figure 2 indicates the relevant section of the circuit.

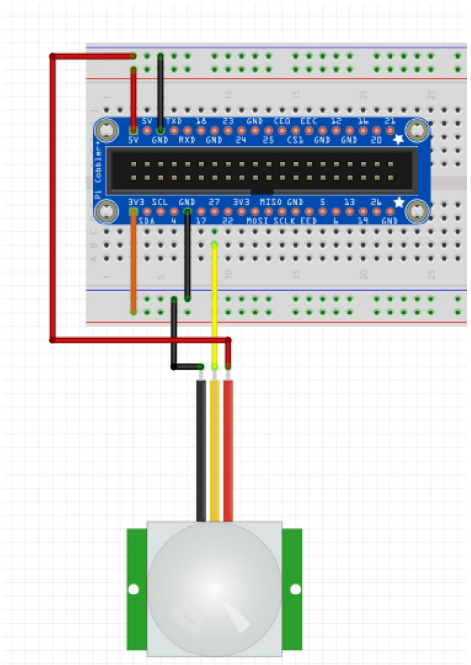


Figure 2 - Second step of wiring

The test program used is a very simple Python script that polls the GPIO input every second:

```
import wiringpi2 as wpi
import time
wpi.wiringPiSetup()
# GPIO pin setup
wpi.pinMode(2, wpi.GPIO.INPUT)
while True:
      i=wpi.digitalRead(2)
      if i==0:
      print "no motion ", i
      elif i==1:
      print "motion detected ", i
      time.sleep(1)
```

Running the program and moving around the sensor should result in an output similar to this:

```
# python pir.py
no motion  0
no motion  0
no motion  0
motion detected  1
motion detected  1
motion detected  1
motion detected  1
motion detected  1
motion detected  1
motion detected  1
motion detected  1
motion detected  1
no motion  0
no motion  0
```

Pressure Mat

The pressure mat acts as a switch: when the mat is pressed, the switch is turned on. There are 2 wires to be connected - one to 3.3V (with orange wire) and one to the GPIO input pin #15 (WiringPi GPIO 3, with purple wire). We will be using a pulldown 10k resistor to connect the data wire to the ground (the black wire), which means that when the switch is open there is a path to ground and the GPIO will read 0. When the mat is pressed, because of the pin connected to 3.3V there will be a lower resistance path to high and the GPIO will read 1. We will be using also a 1k current limiting resistor on the data wire, to make sure the board will handle the current drawn when the switch is ON. Figure 3 reflects this setup.

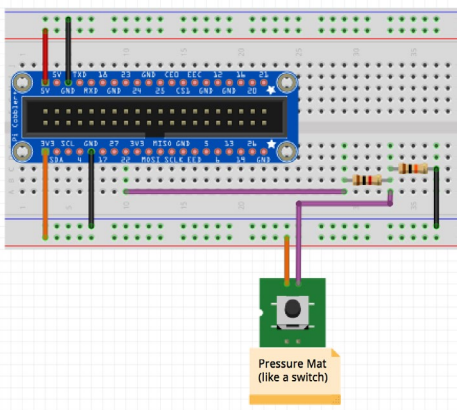
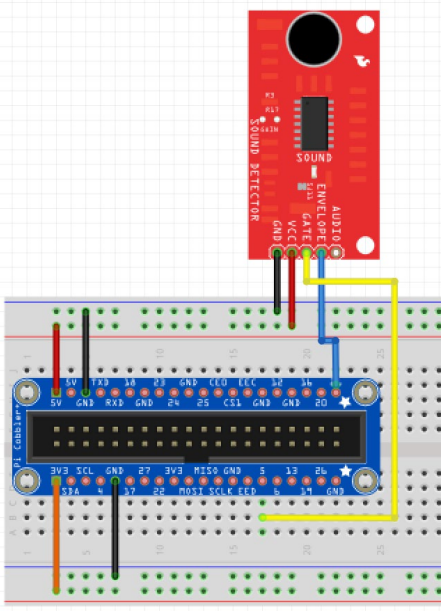


Figure 3 - Third step of wiring

The test program used here is also very simple:

```
import wiringpi2 as wpi
import time
wpi.wiringPiSetup()
# GPIO pin setup
wpi.pinMode(3, 0)
while True:
      i=wpi.digitalRead(3)
      if i==0:
      print "not pressed ", i
      elif i==1:
      print "pressed ", i
      time.sleep(1)
```

Figure 4 - Fourth step of wiring



Sound Sensor

The Sparkfun sound detector that we will be using has 5 pins: VCC (5V), GND, GATE, ENVELOPE and AUDIO. We will be using the GATE digital output, which is 1 when sound is detected and 0 otherwise, and the ENVELOPE analog output, which represents the amplitude of the sound. We will be connecting the GATE pin to header #29 (WiringPi GPIO 21, using yellow wire) and the ENVELOPE pin to ADC.AIN0 on header #40 (blue wire). Figure 4 illustrates this part of the overall setup.

The following script can be used to test sound detection:

```
import wiringpi2 as wpi
import time
wpi.wiringPiSetup()
# GPIO pin setup
wpi.pinMode(21, 0)
while True:
      i=wpi.digitalRead(21)
      if i==0:
      print "no sound ", i
      elif i==1:
      print "sound detected ", i
      time.sleep(1)
```

The sound volume can be determined by trial and error. In my case, I reduced the output to the 0-127 value range and used the 10-30 range as moderate (conversational) sound volume. An output below 10 means the room is quiet and a value above 30 implies there is a high volume sound, with a likelihood that the baby is crying.

Below is another test application, which should be saved to a file caled sound-env.py:

```
import wiringpi2 as wpi
import time
wpi.wiringPiSetup()
while True:
      i=wpi.analogRead(0)
      ampl = i*255/2047 # not
sure this is correct
      if ampl <= 10:
      print "quiet: ", ampl, ",
", i
      elif ampl <= 30:
      print "moderate: ", ampl,
", ", i
      else:
      print "loud: ", ampl, ", ",
i

      time.sleep(0.5)
```

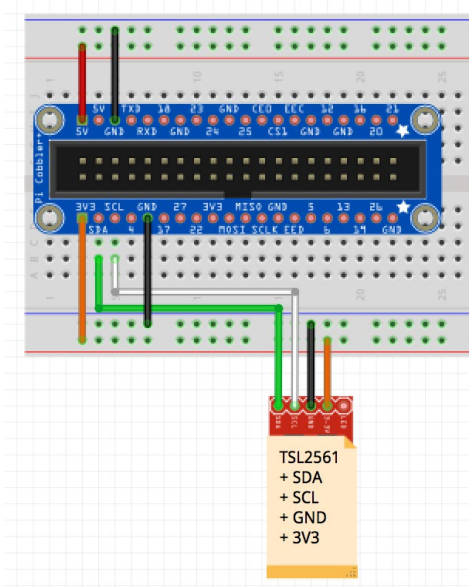
When run, it produces output similar to that below:

```
# python sound-env.py
quiet:  10 ,  88
quiet:  3 ,  31
moderate:  27 ,  220
moderate:  24 ,  199
moderate:  28 ,  229
moderate:  16 ,  130
loud:  52 ,  419
moderate:  16 ,  129
moderate:  30 ,  244
quiet:  4 ,  33
quiet:  6 ,  49
quiet:  3 ,  31
quiet:  5 ,  44
```

Light sensor

The Adafruit TSL2561 can be connected to the ODROID board via I2C. The connections are straightforward: 3V3 to 3.3V rail (orange wire), GND to ground (black wire), SDA to SDA (green wire) and SCL to SCL (white wire). Figure 5 demonstrates this part of the setup.

Figure 5 - Fifth step of wiring



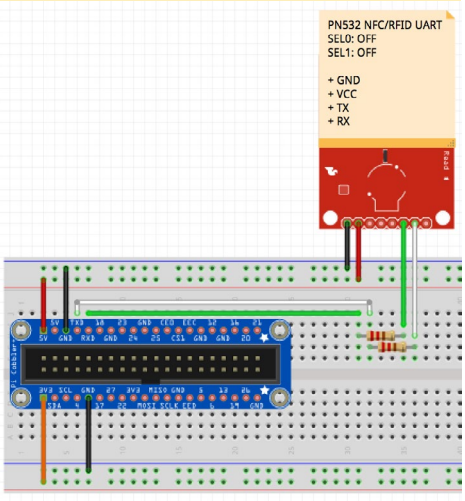
To enable I2C on the ODROID C1, you need to load the aml_i2c kernel module. It can be done by appending aml_i2c to the /etc/modules file. We will also be using the tentacle_pi python module, available at http://bit.ly/1AiDZfk. It allows for communication with TSL2561 from a Python application outputting the light intensity in lux units:

```
from tentacle_pi.TSL2561 import
TSL2561
import time
tsl = TSL2561(0x39,"/dev/i2c-1")
tsl.enable_autogain()
tsl.set_time(0x00)
while True:
      print "lux %s" % tsl.lux()
      time.sleep(1)
```

RFID Reader

The RFID card reader can be connected to the ODROID C1 using the SPI, I2C or UART interface. We will be using the nfc Linux library, which is the easiest option is to use the UART serial connection. For this mode, both SEL0 and SEL1 on the PN532 RFID breakout board need to be set to OFF state. As with any serial connection, we will

Figure 6 - Sixth step of wiring



be using 5 wires: VCC (5V, red wire), ground (black wire), RX and TX (green and white) - connected to TX and RX of UART1 (/dev/ttyS2) on the board on headers #8 and #10. Figure 6 indicates this part of the setup.

The libnfc library and related components can be installed easily on Ubuntu with the following command:

```
$ sudo apt-get install \
  libnfc-bin libnfc-examples \
  libnfc-pn53x-examples
```

The python wrapper can be downloaded and installed from http://nfcpy.org. In order to use libnfc, you need to configure the connection by creating a file called /etc/nfc/devices.d/pn532_uart.conf, which contains the following code:

```
## Typical configuration file for
PN532 device connected using UART
name = "PN532 board via UART"
connstring = pn532_uart:/dev/
ttyS2
allow_intrusive_scan = true
```

You can then test the connection with the pn53x-diagnose program:

```
$ sudo pn53x-diagnose
pn53x-diagnose uses libnfc 1.7.0
NFC device [pn532_uart:/dev/
ttyS2] opened.
 Communication line test: OK
 ROM test: OK
 RAM test: OK
```

The nfcpy application is well documented and easy to use. To read a tag, you only need a few python lines. Note the tty:S2:pn532 connection string that needs to be used.

```
>>> def connected(tag):
print(tag); return False
...
>>> clf = nfc.ContactlessFrontend
('tty:S2:pn532')
```

```
>>> clf.connect(rdwr={'on-con-
nect': connected}) # now touch a
tag
Type3Tag IDm=01010501b00ac30b
PMm=03014b024f4993ff SYS=12fc
<nfc.tag.tt3.Type3Tag object at
0x7f9e8302bfd0>
```

Since the connect function is blocking, the final code will be written to read the NFC tag in a separate thread.

Part 2 of this series, available in next month's issue, will detail the software components required.

References

- ODROID-C Tinkering Kit
- http://bit.ly/1YNPN6k
- Hardkernel's WiringPi2 GitHub repository
- http://bit.ly/23Rwf7T
- ODROID-C2
- http://bit.ly/1oTJBya
- Userspace Python drivers for TWI/I2C sensors
- http://bit.ly/1AiDZfk

MAKE YOUR OWN SMART CAR WITH THE ODROID-XU4

BRING NEW FUNCTIONALITY TO YOUR VEHICLE

by Jon Westgate

I'm an electronics engineer who also works in computer systems engineering, and am certified in the United Kingdom's City and Guilds Electronic Servicing to level three. I have been inspired with electronics ever since my father built a color TV from scratch in his spare time. This project brings the power of an ODROID-XU4 into your car for added functionality and features while on the road. This is a look at my journey from getting started with the project to getting everything in working order.

I've always wanted to build a PC for use in my car. It's something I've seen with a lot of potential ever since MP3s were made digital and could be played from a PC, bringing the ability to listen to music and other capabilities to my car. I've experimented in the past, but most devices just didn't meet my needs. The x86-based machines I used were far too slow, and the Android computer-on-a-stick I tried in the past couldn't easily use a touchscreen or GPS for the features I wanted. I also once tried this a Nook HD, which showed promise, but its battery created a lot of problems during testing.

Trial and error

All of these previous issues led me to trying this project out with an ODROID. My first ODROID was the U3, and I bought it without doing a lot of homework necessary to make this all work. The Chinese, no-name 7-inch touchscreen I found had a 1280x800 resolution, and I found he out that the ODROID-U3 can only handle a 1920x1080 or 1280x720 resolution, creating a distorted image on the screen I tested it with. There were also some touchscreen issues as I tried to get positioning to work accurately on the resistive touch controller. It was an even greater challenge to do so on Android, where it expects a capacitive screen in the first place.

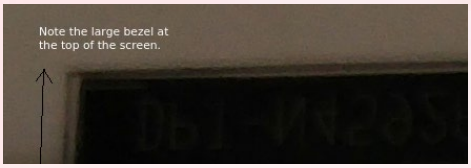
I needed something to just work with this touchscreen, so it looked into an alternative, including the XU3. The XU3 seemed like a better fit with its improved resolution compatibility and other features, but it was just too expensive for my project. But when the ODROID-XU4 came out and offered a far better price tag, I was able to finally afford a device that could meet my needs.

Finding a display device

After receiving my ODROID-XU4 two weeks later, I went ahead and got started with an Android image on my eMMC. Thankfully Android booted and the touchscreen worked immediately. However, I still wasn't happy with the display's quality and touchscreen, so I decided to see what else was out there. In the meantime, I found out about the 7-inch touchscreen offered by Chalkboard Electronics. It offered the capacitive multi-touch experience that I needed for Android, as well as better quality display overall than my Chinese no-name screen.

When the touchscreen arrived, I noticed that the bezel was quite large on the Chalkboard 7-inch touchscreen. It's interesting to also note that it's upside down, with the connections for the interfaces on top, rather than on the bottom of the screen as you may expect with other displays. This is something to bear in mind depending on how you plan out your own Car PC project, as you'll need to allow for about 7mm of space at the top when mounting your display.

I realized that the firmware that came with this touch screen emulated a mouse pointer, rather than the touch experience



you'd expect from a tablet or similar Android device. That way the device will register a swipe when you move your finger, rather than move a mouse cursor on the screen. There's a different firmware you need to flash for this touchscreen in order to get that working, according to the documentation that I read.

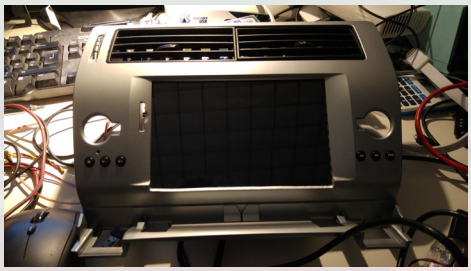
Flashing the touchscreen firmware

For this project with my ODROID-XU4, I used @voodik's CyanogenMod 12, featuring Android 5.1.1 (http://bit.ly/1VyXFKW). It's best to start with an out-of-the-box experience and add features as you experiment and get the basic hardware working. After finding the right driver to update the firmware, you'll need to use a Windows PC in order to update the firmware for the touchscreen to work with this project properly. The full details of these instructions are on the Chalkboard Electronics Website at http://bit.ly/1MULs0y. Once you finish the firmware update, you should be able to use pinch-to-zoom and all the other intuitive features that make this project easy to use from your car.

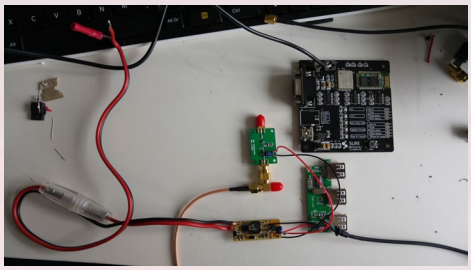
Getting GPS working

For this project to succeed, I wanted to also get a GPS working for satellite navigation. This was something not already available on my Citreon C4 Coupe. I tried and failed to get satellite navigation through other third party methods, and the manufacturer's GPS technology was far too expensive, and not worth installing on my car. Before beginning this project, I had, among other project experiments, gotten a WinCE device working with a 7-inch GPS unit along with a Bluetooth entertainment system tied into it all on a dashboard. But after having some crashes, bugs, and general poor performance, it was finally time to upgrade to my new XU4 with its powerful 8-core processor and Android operating system.

When I switched to the XU4, I was able to trash the old Chinese display built into my WinCE device, and instead use the bezel and dashboard to fit in my new, higher quality Chalkboard touchscreen display instead. I was also using a Denison Gateway Pro in order to program my car's buttons and features with the custom display and XU4 I was installing. This gave me a seamless Bluetooth connection for hands-free calling and music through the car's speakers.



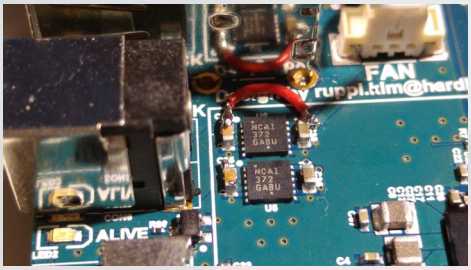
Rather than use an ODROID GPS Module, I ended up using a Chinese unbranded GPS module, which connected to a 4-in-1 antenna to save space and make things easier. This was all then connected to my XU4 using a serial to USB adapter. The goal was to use a single antenna that could handle not only my GPS, but also my AM, FM, and DAB radio connectivity as well through this 4-in-1 antenna connector. I ended up placing the antenna near the rear view mirror after experimenting with several antenna systems, including a powered GPS antenna that didn't work out in my current setup.



The GPS Software I used to use on WinCE was called iGO Primo. But now that I'm using Android, everything can be handled through an app available on the Play Store, which includes numerous offline GPS apps, as well as Google Maps. However, one last challenge with the GPS was ensuring that the real time clock could synchronize with the GPS satellites for time accuracy, which required an RTC battery so that the time wasn't lost each time the device lost power. I had a spare battery on hand and simply replicated something very similar to what Hardkernel offers on its website. I also figured out a way to gracefully turn off the XU4 through the PIO connector on the board.

A minor mishap

While working on getting the PIO connector working to gracefully shut down my XU4, I ended up accidentally shorting the 5V line, killing my XU4 and leaving it with a flashing LED. In case anyone is wondering, I was able to get it fixed by shorting out the protection circuits (ICs), which can be seen in Figure 5. If you ever make this mistake, you can try this in order to figure out how to bring your XU4 back from the dead.



By making this mistake, I ended up ruining my RTC circuit, making the battery useless. After searching around the Plays Store, I was able to find an app called "Smart Time Sync" (http://bit.ly/1rw76yA), which uses the GPS to set the time and adjust your Android clock after each boot. You might find this useful if you don't want to deal with a battery, or can't deal with it due to a mistake like mine. It's also just a really useful app for those interested in getting their XU4 to work with a GPS on Android.

Getting a radio working

What's a car without a radio? Since the goal is to create a single experience

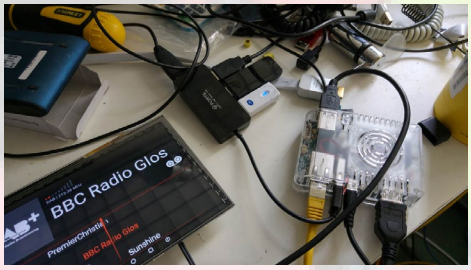
from my XU4, I also wanted to get a radio working with the help of Software Defined Radio (SDR). The RTL-SDR is a $25 gadget that lets you pick up a wide range of radio signals, including FM radio stations with DAB for station information for a rich experience.

However, the DAB app I was using did require an Internet connection to check its license, even though it needed no Internet connection to work on its own. This required some creativity and yet another gadget to get things working: a cellular data connection. On the bright side, it meant Google Maps would work too, but it required more work in order to get everything working. I eventually settled on a Huawei 3G/4G dongle that was compatible with my local British carrier, Everything Everywhere (EE).

The Dongle I chose was one that connected via WiFi to my XU4 for simplicity's sake. It could be designed to connect natively via Android, but this meant one less thing to wire, as it could be done right from the car's 12V socket plug. I ended up upgrading the antenna for improved connectivity.

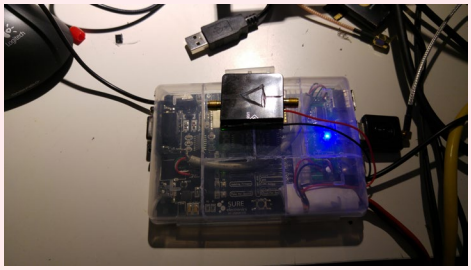
Putting everything together

As I continued testing, everything started to slowly come together. There was a working XU4 with Android, a working touchscreen, GPS connectivity, FM radio with DAB, and Internet connectivity. With all these different dongles connected, I needed to install a powered hub in order to maximize the power available to the XU4 without affecting all of the peripherals' own power needs.



Space is limited in a car's dash, so I wanted to minimize the space taken up by all of the different dongles and radios. It took some careful soldering and cutting, but I was able to break down the entire power module, XU4, and attached peripherals in a single, compact device, taking up far less room than when everything was connected for testing purposes.

Since the RTL-SDR I purchased includes a direct sampling modification, it's also able to support AM radio and other frequencies too. In total, the device can sample signals between 100 KHz and 1.8 GHz, a wide range that covers AM and FM radio, CB radio, and even private radio signals for police and fire departments too. Having such a broad scanner is very useful in my car if I ever want to track down a nearby signal.



Getting everything into the car

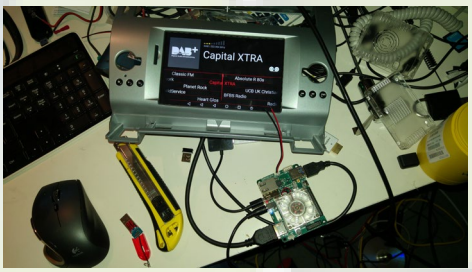
Now that everything was working and compacted, it was time to get everything configured and ready for installation into my car. This required installing the 4-in-1 antenna, as well as getting the right wiring between my new dashboard system and my car's own power systems. I cannot emphasize enough the importance of using fuses to prevent an electrical malfunction that can cause damage to your electronics, or worse harm to a friend or family member. It's also important to properly earth or ground your electronics, as you'll be stepping down a 12V DC connection to the 5V connectivity your XU4 and peripherals need. Even at these low wattages, it can be very dangerous to get a shock from these electronics. You really want to give everything the power it needs.

You can do everything through USB, but in my case it was easy to run power from the 12V power to a CPT component that stepped the 12V connection into a 5V 10A connection. Then everything is passed on through direct soldering to the USB hub, and finally the ODROID-XU4. This saves the redundant step of using an inverter or other AC adapter when everything will end up in DC power, and makes it easy to power

the antenna for a better signal. The CPT components are very helpful too, as they have protections built in to prevent an electrical short or other damage.

Right now, everything is switched on when the car is turned on with the ignition switch. This is how everything is currently configured, though I hope to change this with a timed supply in case I quickly turn off the car to make a stop or get fuel. USB will be the key here, though it will take some time in the future to figure it all out and effectively control how the device receives power from the battery even when the car is not on.

In the meantime, I carefully installed all the components into the bezel and dashboard of my car, taking care to ensure all the media buttons and other components would work once reinstalled.



Getting audio working

Getting audio working on this new Car PC configuration with my XU4 was a lot harder than first expected. There was a lot of bad noise due to the grounded connectivity using normal default outputs. The solution was to use an HDMI audio extractor to prevent all the noise you'd otherwise hear when playing music or listening to the radio.

I've tried a number of different options, including a USB digital audio controller rather than an HDMI audio extractor, but the goal was to make sure this new Car PC could support any audio device, such as an iPod that I plug in, a hands-free call on my smartphone, or the GPS navigation system. The entire process continues to be a work in progress with plans to purchase a better DAC with an amplifier. There's also an impedance mismatch between the audio jack in my Denison Gateway Pro, causing a tinny, thin, and quiet sound. This will continue to be something to look into and figure out as I continue tinkering with my Car PC.

Other odds and ends

There are a number of small modifications I've made as I fine tuned this project. This includes using a dremel to drill the Chalkboard touchscreen so that it fits into the bezel and dashboard perfectly. I also added a new connector to the XU4 so that a power button could be added through new external controls on a circuit board in the future. The goal is to also support future USB joysticks and other connectivity without having to reopen everything.

For this project, I ended up using a full size SD card for cheaper, larger storage capacities on the device, and this required an extension cable due to the limited space near the bezel and an HDMI connector blocking a spot where my full size SD card would have went.

Another innovative idea that I had was to use a reversing camera. The Chinese-made GPS system I used supported a reversing camera, although my Chalkboard touchscreen had no way to support this. The reversing camera works by detecting a composite video signal and forwarding that along to the screen, which can be converted for the Chalkboard's HDMI input. The only problem is that the composite signal looks really bad and has a poor aspect ratio on the screen.

The solution is two-fold. First, you need to install an upconverter to bring the composite signal up to a proper HDMI signal without aspect ratio issues. However, this will add yet another peripheral to install in your car in order to get everything working. You'll also need to find a camera that is good enough to provide an HD-quality signal for improved visibility. I recommend using an SDI-HD camera, which will convert the composite HD signal over coaxial cable to your upconverter, which then connects flawlessly to your Chalkboard touchscreen. This enables you to get up to 4K UHD resolutions from a single coaxial cable with BNC connectors, along with a 12V DC power connector, and no aspect ratio issues. This is still a work in progress though, as I'm currently developing a circuit that will detect the car's reverse mode, and switch the Chalkboard display from the XU4 and instead show the backup camera for a seamless experience.

Software

Here's a list of all the software I used:

- Voodik's CyanogenMod 12.1 running Android 5.1.1
- Google Apps including the Google Play Store
- Smart Time Sync Pro
- Google Chrome, as Firefox doesn't work well on the XU4 and needs to be woken up with the power button
- iGo GPS, but there are plenty of offline GPS apps available
- Autorun (FIT009C Soft Tools)
- K@mail, but note that you'll get an email certificate error you should ignore, due to Smart Time Sync Pro
- Google Maps and Google Streetview

For those who may need it, here are the changes I made to my build.prop file:

```
config.disable_bluetooth=false
wlan.modname=8192cu
ro.kernel.android.gps=ttyUSB0
ro.kernel.android.gps.speed=9600
```

Thanks for joining me on my journey to build a Car PC, and happy modding!

MEET AN ODROIDIAN

MILTIADIS MELISSAS

edited by Rob Roy

Miltiadis, known as Miltos, has been using programming and using computers since the early days of the Commodore 64

Please tell us a little about yourself.

My personal history begins back in 1965 in northern Greece and in the beautiful town of Xanthi. At the age of 18, I moved to Athens for my academic studies, and 5 years later I settled to the enchanting island of Corfu, where I work and evolve.

I got my Bachelor's degree as an Electronics Engineer together with the Pedagogical adequacy back in 1988. Shortly afterward, I worked for nearly 5 years with Barclays as an operator of the bank's Unix online system, and later continued with 2.5 years with Scotiabank holding the same position. In 1997, I resigned from the banks' working field and became a State Employee in a High School teaching Technology, a position that I still hold today. In 2006, I got my Master's of Science in Information Technology, and in 2012, I was promoted to a Sub-manager in the High School where I teach Technology and Electronics. My Master's of Education is pending, and my academic interests lie in the ontology field of organizing the information.

My career evolved alongside my personal life. I married my lovely wife Dori in 1995, and it's her continuing support and encouragement that made all things possible to our lives. My son Apostolos was accepted last year at Glyndwr University in Wales, studying Game Development: "Like father, like son". His dream is to work with the big names in the industry. The last member of our family, but not least is my daughter Eri. Eri is studying nursery in high school. She is only 16 years old but has a strong and firm career orientation.

How did you get started with computers?

In 1985, I purchased my first "home computer": the epic Commodore 64. With 64K of memory and a price less than $600, Commodore 64 was really a breakthrough at that time. That 8-bit machine was capable of drawing fantastic graphics using sprites, and outclassed every other computer in the market with its 3-channel sound card circuit. I can recall endless times of joy playing games like "Head over Heels" and the all times classic "Pacman". But the real power of Commodore 64 was its ease of programming. Mastering BASIC was a challenge back in the 1980s. Some of my first programming attempts were an address book, a phonebook and a database of recipes. Commodore 64 was actually a school in and of itself. I got all my knowledge of structured programming from this machine, and I went even deeper with some basic knowledge of assembly language afterwards. In subsequent years, I followed the evolution line of Intel's computers with Windows as the main OS, and it was in 2006 when I switched gears to Linux. The main reason for the switch was the sense of freedom that the open source programming had to offer, along with the strong supporting community behind Linux. Additionally, Linux is a challenge for every Electronics Engineer, since it can be found in many apparatus other than computers, such as set-top boxes, appliances, modems, phones, gaming machines, and embedded systems. It was Linux that brought me to Hardkernel's excellent line of products.

What attracted you to the ODROID platform?

Always keen to learn, I've came across an online manual from the creators of makeuseof.com (http://bit.ly/SInIlr) dealing with the Raspberry Pi. At the end of the book, the author proposed some Raspberry Pi alternatives, and the ODROID-U3 from Hardkernel was of course at the top of the list. It had better hardware specifications, more software OS implementations, and ran many flavors of both desktop and server editions of Linux, including Android, OpenVault, and Media Center OS. For me, it was "love at the first sight". I ordered the U3 almost immediately, and after six months of use I purchased another one. I was impressed from the very beginning as an Electronics Engineer by the quality of the board and its durability not to mention its small size and the low power consumption.

That was the time when the Rob Roy's excellent community images came into place. I enjoyed using "Fully Loaded: Ubuntu 12.11", "Quiet Giant: Ubuntu Server 13.05", and "Pocket Rocket: Android Jelly Bean 4.1.2", just to name a few. I discovered the strong supporting community (the ODROIDians) when I started to build my own images and compile some specific hardware drivers. Without their help I couldn't have gone far! The Odroid-C1 followed as a purchase and then 2 ODROID-XU4 units, which are my favorite.

How do you use your ODROIDs?

I've got OpenElec running perfectly on my ODROID-U3, allowing me to listen to music and stream videos, as well as watch YouTube channels, movies, live TV channels and programs from all over the world. The motto "cut that cord" became a reality in my life, and I've never looked back. On the other hand, my ODROID-C1 is sitting on my desk running "Pocket Rocket II: Android KitKat 4.4.4" with all the bells and whistles this OS has to offer. The XU4 serves as my main desktop PC, and frankly it's been months since I opened my high power consumption Windows PC. The ODROID-XU4 satisfies most, if not all, of my needs as a workstation and I even can play games with the excellent GameStation Turbo image provided by @meveric, another ODROIDian. The PPSSPP emulator works very well on this image, and most of the PSP games run at 60 fps flawlessly.

Lastly, I've created my own desktop environment based on the ArchLinux distribution, and installed the Mate desktop on top of it as a GUI. Undoubtedly there is a lot of work that has been done behind this excellent desktop environment. Surprisingly, this image doesn't have the black screen bug and the missing icons bug from the LibreOffice suite, and everything seems to work flawlessly. Hopefully, I will soon post a guide to the forum for everyone who wishes to have the same type of environment. The ODROID-XU4 is a real gem, and if those uses aren't enough, my second XU4 is used as a server. The possibilities are endless.

What innovations would you like to see in future Hardkernel products?

It couldn't be difficult for Hardkernel to include a wifi and bluetooth circuit on their next development board. On the other hand, I personally believe that there is a lot of room left for the software development of the boards. Surely, Android OS is quite mature for the C line of products (i.e C1, C1+, C2), as well as the XU4, but Linux OS can be evolved to its full potentiality in upcoming years. Furthermore, recent development operating systems like "Remix OS" look promising, even though it is not implemented yet on the Hardkernel's line of products, and I don't know if it can never be. Finally, some extra memory (RAM) could be added to all boards with unavoidably some extra cost of money of course. Personally I don't believe that the extra amount of 1 or 2 GB of RAM will surpass the cost of 100 dollars for even the most expensive ODROID.

What hobbies and interests do you have apart from computers?

"A healthy mind in a healthy body" it's a very famous ancient Greek saying. Besides computers, I love swimming and walking. I swim a lot, especially after some serious problems with my waist a year ago. The fact that I live in an island gives me the opportunity to swim almost the entire year. I cover a distance of about 35km per week on foot and another 8km by swimming. I am also motivated by my smart wristband and ODROID-C1, which run an Android app to help me to keep track of my activities, collect usage data and statistics, and view information about my health and fitness. In addition, I love reading educational and philosophical books and watching black and white movies from the old American cinema. I also take unique photos during my daily walks in the city. I've rediscovered my own city after 25 years of living through those walks and activities. I have also been thinking seriously of buying a bicycle.

What advice do you have for learning more about programming?

Programming is like learning a new language. It always takes time. Moreover it's more like a personal curve of development, since everyone learns and comprehends knowledge differently. There were periods where I was learning by leaps and bounds, and others where I was advancing very slowly. I started with BASIC a long time ago on the Commodore 64. BASIC helped me to learn the fundamentals of structured programming. I carried on with Visual BASIC back in 2000 by grasping the knowledge of optical programming, a computer language that I still use for educational purposes. Ten years ago, I took the "big leap" to object oriented programming using Python. I love Python because it's a general purpose, versatile and educational computer language with a big coding library and a vast supporting community behind it.

I advise starting with structured programming, and steadily and slowly go on to object oriented programming. Python (`http://bit.ly/1oDM6iq`) is a programming language that lets you work quickly and integrate systems more effectively. It provides a clean syntax and indentation structure that is easy to learn. I encourage my high school students to learn Python and have it as a basic set of skills. The Python programming environment can be installed very easily on every ODROID Linux computer, which greatly minimizes the cost of implementation. In addition, there are many free online courses on the web offered by various developers. Other online resources include Coursera (`http://bit.ly/18HdJkD`), Edx (`http://bit.ly/1bxqyhn`), and Udacity (`http://bit.ly/1dy71Lz`). As I teach my students, with the ODROID, the world wide web and Python, the sky's the limit, and the possibilities are endless.

Miltos and his wife, Dori, live on an island in Greece