tripwire
CONFIDENCE: SECURED

How to Train Your RFID Hacking Tools

By Craig Young
Security Researcher
Tripwire VERT



Table of Contents

Introduction 3

RFID At A Glance 4
- **What is an RFID Tag** 4
- **Low Frequency Cloning (T55x7)** 5
- **High Frequency Cloning** 6

Introducing the Proxmark3 (pm3) 7
- **Proxmark3's FPGA** 7
- **Proxmark3's CPU** 8
- **Proxmark3's ADC** 9
- **Proxmark3's Connections** 9

The NXP PN533 NFC Transceiver Chipset 10

RFIDler 10
- **RFIDler Internals** 10

3d Printing 11
- **Printing Process** 12
  - Preparing the drawing 12
  - Preparing the Plate 12
  - Extrusion 12
  - Movement 12
  - Retrieving the Print 12
- **Printing Pitfalls** 13
- **Printing Coil (Antenna) Forms** 13
  - First Antenna Design 14
  - BADge Antenna 17
  - Clipwnd (Clipboard with a Kick) 18
- **Business Information Modeling (BIM), 3D Scanning, and RFID Hacking** 19

Using Proxmark3 like a Boss 19
- **Using 'lf search'** 19
- **Printing and using the demodulated buffer** 20
- **Developing New Features for The Proxmark3** 21
  - Adding real-time demodulation of AWID26 22
- **Simulation of AWID26 tags from facility-code and card number** 23
- **Cloning AWID26 from facility-code and card number** 24
- **Replacing LF standalone mode with NFC functions** 25

RFIDler: The New Kid on the Block 28
- **RFIDlerPi** 28

Concluding Remarks 28





Introduction

The Tripwire Vulnerabilities and Exposure Team (VERT) has been evaluating a variety of approaches for extending the functionality of our RFID/NFC analysis tools. This ongoing research documents accomplishments and ideas enabled through 3D printing, firmware enhancements, and device pairing. The tools used cross functional domains ranging from devices like the tiny proxmark3 with its excellent software-defined radio (SDR) based RFID Swiss-army knife to the more "dish washer" looking CubePro 3d printer. The collection also includes a DEF CON 22 acquired RFIDler v22-beta, NXP PN5333 USB stick, ChameleonMini, RaspberryPi, and USB Armory embedded USB stick.

In order to improve upon these tools it is critical to understand how they work at various levels so as to recognize their potential and become comfortable with using them in new ways. Many of these tools are extremely well documented with respect to forums and wiki articles explaining how they are operated. Few however go into much detail regarding how to get started working in the source code. Although in some cases developers gladly answer questions, it can be intimidating for a new comer who might not know where to begin. This research was performed partially to help consolidate some of the information about these devices making it easier for others to get involved. At the same time the contributions from this project are intended to be helpful for others using the technology.

Changes to the proxmark3 firmware (now available on GitHub) are documented in a tutorial-like demonstration for working with both high frequency and low frequency tags with and without an attached client. The feature enhancements introduce functionality specific for working with 125kHz AWID27 tags as well as for working with NFC in stand-alone mode. For AWID27, the changes allow the operator to act as a reader as well as to clone or emulate cards based solely on numbers found on a typical AWID card. The NFC stand-alone mode enhancement allows reading and emulating a UID from ISO14443a compatible tags. The new mode also allows writing a captured UID onto a special "magic" card.

Identified uses for 3D printing focus on the construction of antennas using 3D printed forms as well as concealment of hacking devices. The use of 3D printing for making coils is advantageous due to the possibility of lower cost and customized antennas. Device concealment is also explored with examples for outfitting a clipboard with a proxmark3, creation of fake badge readers and fake badges. Applications range between practical attack scenarios and research scenarios. As insecure low-frequency access cards still dominate the workplace and NFC technology is gaining steam, it is critical to get our tools ready now to identify emerging threats.

RFID At A Glance

RFID tags, badges, and cards come in two main categories: low frequency and high frequency. As of May 2013, legacy 125-kHz proximity technology was used in 70%-80% of all physical access control systems according to Stephanie Ardiley, product manager at HID Global. Implantable low frequency RFID chips have also seen widespread use for identifying lost pets. These are both fairly simplistic systems in which the reader energizes a coil to power up a chip and modulate back a stored number. The high-frequency RFID however includes applications like contactless credit cards, public transit fare cards, smart posters, and other functions utilizing Near Field Communication (NFC) built around ISO14443a.

What is an RFID Tag

The RFID tags described in this paper are passive devices. They are all essentially coils of wire tuned to a connected integrated circuit. When the coil is within the field produced by the carrier signal of the reader, it induces current to power the integrated circuit. The chip will then do something similar to opening and closing the circuit with the coil to control the damping factor. This has the effect of determining how much the coil will resonate from the reader. The reader detects this as a changing voltage and interprets the signal based on the modulation and/or encoding for that protocol. ISO14443a/NFC systems perform this system but with a far richer command set offering access to more data and the occasional encryption option.



Figure 1 RFID Tag Teardown (via Wikipedia)

RFID tags are used in everything from the common office access control systems to passports and mobile payment systems. Advertisers and game designers have been using NFC in toys and smart posters to provide a more meaningful interactive experience. We are also now seeing perhaps the first use of NFC as a protection mechanism in consumer products with the Keurig 2.0 coffee makers utilizing an NFC tag to differentiate unsanctioned grinds.

RFID tags have the potential to make our lives easier with contactless payment, automatic device configuration, and seemingly secure locks, but the reality is that much of this technology has been implemented insecurely because there is a general lack of knowledge on how to evaluate these systems.

Low Frequency Cloning (T55x7)

Cards using the Atmel T5557 and ATA5567 chipset are collectively referred to as T55x7 cards. With seven 32-bit optionally password protected EEPROMS, these cards can be configured to emulate a wide range of low-frequency tags. Modulation and encoding supported by the chip includes frequency-shift keying (FSK), phase-shift keying (PSK), Manchester encoding, Biphase encoding, and non-return to zero (NRZ) encoding. Each EEPROM is referred to as a block with block 0 being used to configure modulation and data rate options while block 1-7 may be used to supply up to 224 bits of raw data for transmission. T55x7 cards can simulate the most popular HID, Indala, EM410x, and AWID formats among others making them a very real threat to businesses relying on these cards alone for access control. With the help of a few simple proxmark3 commands, it is possible to easily convert any supported tag into a T55x7 clone.

Inside a T5557 card, it isn't much different from a "normal" RFID badge as shown in this illustration from http://www.proxclone.com:



Figure 2 T5557 Construction[1]

The block 0 configuration of a T55x7 controller varies slightly but the common T5557 configuration can be found in the Atmel datasheet[2]:

[1] http://proxclone.com/T55x7.html
[2] http://media.digikey.com/pdf/Data%20Sheets/Atmel%20PDFs/T5557.pdf

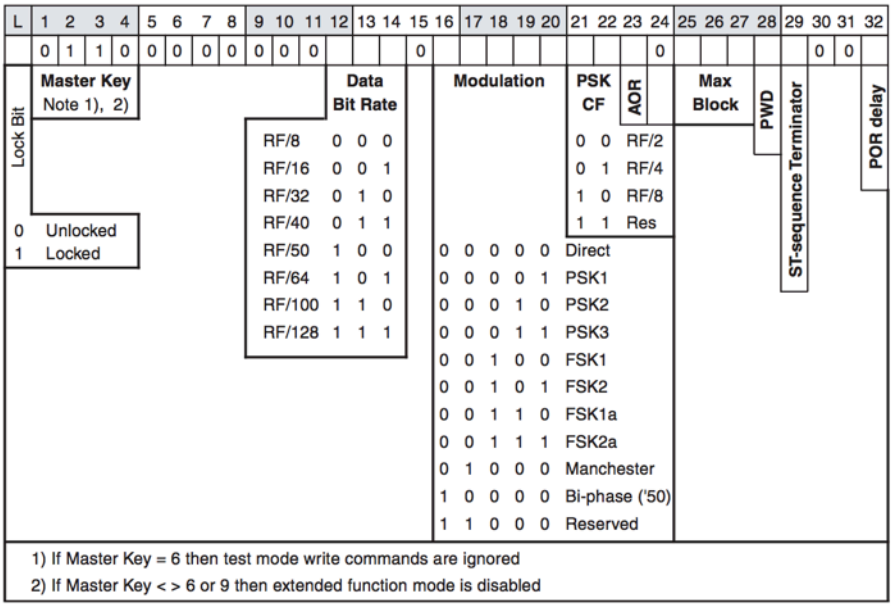


Figure 3 T5557 Block 0 Configuration from Datasheet

Low-frequency cards can of course also be emulated in-circuit by simply crafting the expected waveform and presenting it to the card reader. These techniques are covered in later sections.

High Frequency Cloning

So called "magic" are well known in the industry because they allow the end user to reprogram the normally write-only block 0 values containing most importantly the card's unique identifier (UID). Along with well-documented cracks against the MiFare Classic encryption scheme, this allows an attacker to make a complete duplicate MiFare Classic card in potentially under a minute even when encryption is used. Also referred to as "changeable uid" or "Chinese", these cards are available for a variety of shapes sizes and formats with some companies such as Clone My Key[3] offering full duplication service.

Another approach comes from hardware devices designed to simulate the waveform and even responses from ISO14443a compliant readers. One of the most interesting techniques however has to be the use of a Chameleon circuit. Designed at Ruhr University Bochum, this circuit is designed to store and emulate multiple NFC tags with various data sets as well as snooping on NFC communication to potentially uncover UIDs or other sensitive data. Among other things, this device was used to highlight fundamental flaws in the Akademisches Förderungswerk (AKAFÖ) contactless payment system commonly used in German universities.[4]

[3] http://www.clonemykey.com/
[4] Timo Kapser, Milking The Digital Cash Cow [29c3] https://www.youtube.com/watch?v=Y1o2ST03O8I

Introducing the Proxmark3 (pm3)

Proxmark3 is the brainchild of Koning Gans who developed the basis of the system while analyzing the security of the local transit cards. The idea is to use a less expensive FPGA to perform high-quality software-defined radio (SDR) paired with an ARM processor capable of performing encoding operations as well as moderating instructions from the user. The project is a completely open source (hardware and software) project for affordable RFID research on both high frequency and low frequency target systems.

The proxmark3 can interact with or simulate a wide range of RFID tags with additional support being added through community maintained firmware[5]. An assembled board can currently be purchased for $229 without antennae or an enclosure.[6] Tuning commands within the ARM operating system assist in construction of high quality coils for as little as the cost of a cable.

Proxmark3's FPGA

At the core of the Proxmark3 is a Xilinx Spartan-II FPGA driven by code authored in Verilog and compiled with the ISE WebPACK. A modular design allows for different discrete functionality blocks available for switching by simply redirecting pin connections on the fly. The FPGA defines a serial peripheral interface (SPI) used for some of the most important functions such as driving the coil and reading from the analogue-digital converters. The FPGA also implements a synchronous serial port (SSP) used for communicating data from the FPGA to the ARM in low frequency mode.



Figure 4 Annotated Proxmark3 (Xiling Spartan-II FPGA)

[5] https://github.com/Proxmark/proxmark3
[6] http://store.ryscc.com/collections/proxmark-3/products/naked-proxmark-3 [$229 as of 7/17/2015]

Proxmark3's CPU

An ATMEL 32-bit RISC processor (AT91SAM7SXX series) handles all of the high level functions on the proxmark3 board. The ARM processor also loads configurations into the FPGA for different modes of operations such as switching between high frequency and low frequency modes. All USB communication from the client application is handled within the ARM processor before deciding what if any action must occur with the FPGA.

Source for the program running on the CPU is found in the *armsrc* path.[7] The main logic on the ARM processor is in *AppMain()* within the *appmain.*c. After initializing the device, the ARM loops checking for USB commands or button presses. Commands sent over the USB serial link are represented with opcodes defined across header files[8] and also maintained in a LUA script[9]. UsbPacketReceived() translates these commands into function calls on the ARM.

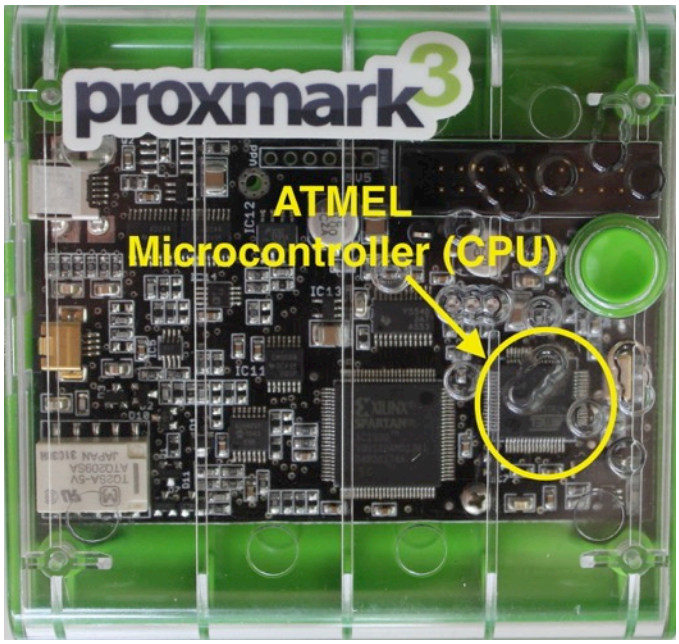


Figure 5 ATMEL 32-bit RISK Microcontroller

Samples from the FPGA are stored in a DMA buffer on the ARM referred to as *BigBuf.* This data may be samples from the ADC or data for the Mifare emulator. In sniffer mode, the ARM can attempt to simultaneously decode signals with both Manchester and Modified Miller encoding looking for a valid signal. Contents can also naturally be sent to the host for other types of analysis including visual inspection. (Refer to *BigBuf.c* for buffer related functions.)

[7] https://github.com/Proxmark/proxmark3/tree/master/armsrc
[8] https://github.com/Proxmark/proxmark3/search?q="define+CMD_ACK"&type=Code
[9] https://github.com/Proxmark/proxmark3/blob/master/client/lualibs/commands.lua

Proxmark3's ADC

The Spartan-II FPGA connects to a Texas Instruments TLC5540 analogue-to-digital converter (ADC). This 8-bit ADC sends readings from the coil to the FPGA across an 8-pin bus. The FPGA uses the serial peripheral interface (SPI) to control the ADC and SSP to share data to the ARM. The TLC5540 collects up to 40 million samples per second giving a theoretical maximum of 20MHz per the Nyquist rate.

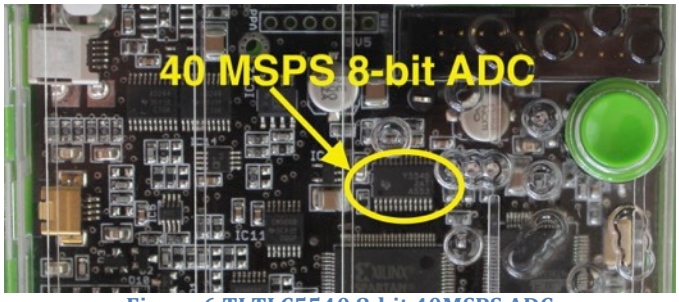


Figure 6 TI TLC5540 8-bit 40MSPS ADC

Proxmark3's Connections

The proxmark3 design uses a hirose 4-pin connector for the antenna along with mini-USB for both data and power. USB support emulates a USB serial connection allowing it to operate without any driver installation on most systems. Proxmark3 also offers general purpose and JTAG pins outside of the scope of this document. For human I/O, there is a single button and a whole lot of lights to provide feedback about the proxmark3 status while in operation.

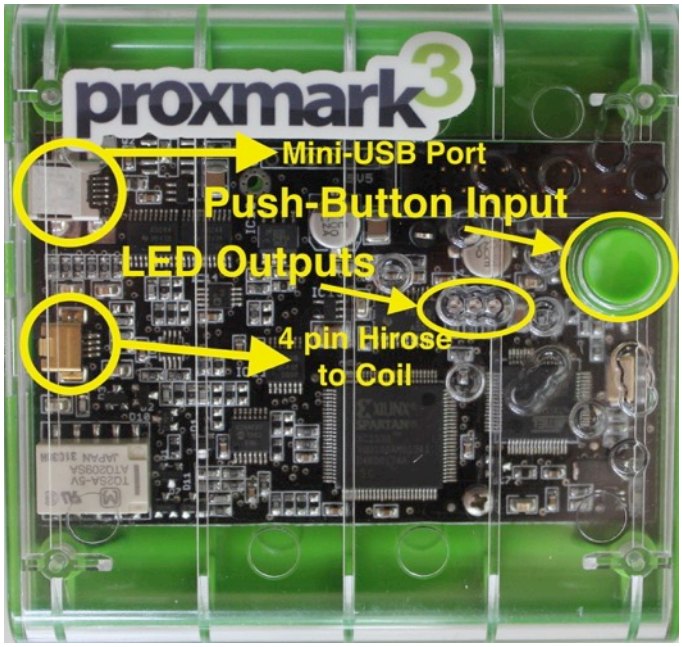


Figure 7 Proxmark3 I/O

The NXP PN533 NFC Transceiver Chipset

The PN5333 transceiver module is fully compatible with ISO14443a, ISO14443b, and FeliCa tags in reader and writer mode[10]. It is also interoperable with ISO18092, ECMA 340 peer-to-peer communication and comes in a convenient USB enclosure complete with libNFC support. This tool benefits from the NXP chipset ensuring better interoperability especially with changeable UID tags. This USB dongle also gives the opportunity to experiment a little more freely with Linux RFID tools such as the NFC tools included with Kali Linux. The PN533 USB package is available from a variety of sources generally ranging from $40-$60 including test cards.



Figure 8 Example PN533 USB Dongle

With support for most popular Linux distributions as well as OS X and Windows, Libnfc is a great tool for identifying tags and performing research on target systems.[11] The PN533 is just one of many available NFC USB peripherals available with support for libNFC.

RFIDler

A product of Aperture Labs, RFIDler, originally was funded via Kickstarter[12] and was available for sale as the 'v22-beta' revision board and coil at DEF CON 22. The goal of RFIDler is to create an extremely simply and low cost circuitry for interacting with low frequency RFID. The circuits use outputs from a PIC microcontroller along with basic analogue circuitry to form and read 125kHz modulated signals. A 3d case was designed by Aperture Labs and published as a free model on Thingiverse[13].

RFIDler Internals

A basic LC tank is used for the antenna driven by the PIC32 microcontroller and amplified with the analogue components. Digital potentiometers are used to control thresholds for determining logic levels while interpreting signals. Numerous LED outputs can be used to visually indicate the device status. Automatic command execution allows for a basic stand-alone operation.

[10] http://www.nxp.com/documents/short_data_sheet/PN533_SDS.pdf
[11] http://nfc-tools.org/index.php?title=Libnfc
[12] https://www.kickstarter.com/projects/1708444109/rfidler-a-software-defined-rfid-reader-writer-emul
[13] http://www.thingiverse.com/thing:427536

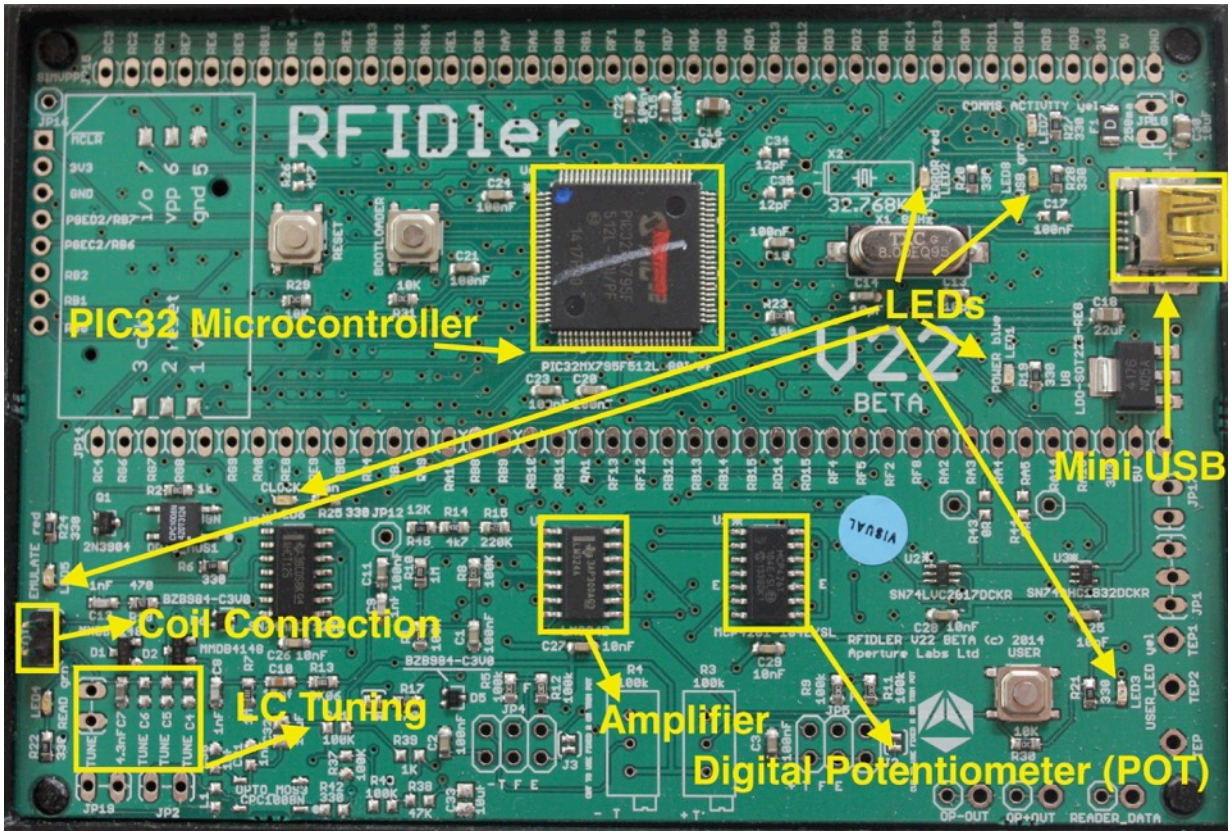


Figure 9 The Annotated RFIDler

Unfortunately the RFIDler acquired at DEF CON 22 for this research has problems reading most tag formats. With each tag it has been necessary to spend a good bit of time finding the sweet spot and adjusting POTSET values. A rectangular multi-layer coil antenna tuned for the RFIDler v22-beta board was made but it was still not possible to get consistent tag reads. Furthermore our RFIDler has a tendency to actually indicate a successful tag decode but report back an incorrect decoding making practical application difficult.

3d Printing

Additive manufacturing is the process by which a substance is gradually deposited layer by layer to build up a 3-dimensional model. VERT's CubePro from Cubify supports both PLA and ABS plastic with layer thickness options of 70, 200, or 300 microns and 3 patterns for internal structure. With a large build volume (11.2" x 10.6" x 9.06") it is on the large end of consumer printers[14].

[14] http://www.cubify.com/



Figure 10 CubePro Promotional Picture

Printing Process

Preparing the drawing

3D models are drawn in standard CAD software and then imported (STL/etc) into the CubePro (or other CAM software) to slice the model into 2D layers suitable for printing.

Preparing the Plate

The CubePro uses a print plate with a coating of water-soluble glue that is dissolved after the print allowing the model to be scraped off. The first step is to coat the printed surface with this glue so that the extruded filament adheres to the print plate rather than itself.

Extrusion

Drive gears on the extruder feed filament into a hot print nozzle. When the print starts, the extruder and the print area both are heated until the filament begins to extrude. Excess filament is pushed into a trash bin and then wiped off before a print commences.

Movement

The extruder is on a gantry moving along the x and y-axis while the print bed moves down gradually to provide the z axis. As each layer is drawn, fans are switched on and off as needed to perform cooling.

Retrieving the Print

After the printer has completely generated a model, it will be stuck the to print plate thanks to the glue applied before printing. The object is removed by soaking in warm water for several minutes and then carefully prying the model away from the build plate.

Printing Pitfalls

3D printing is not yet a consumer friendly technology. CAD skills as well as some sense of industrial design and materials science are needed to excel at creating custom 3d components. Printers can also have tedious calibration processes wasting time and filament for the operator. Beyond this, operation of an extrusion based 3D printer will almost inevitably lead to debris jamming the drive gears or clogging the print nozzle. This is why it is important to know how to clean critical printing parts disassembling them as needed. For the VERT Makerbot replicator 5, this happened between basically every single print. On the CubePro the first clog/jam did not occur until after more than 700g of models had been successfully printed. At that point, the debris show below led to filament flow errors and disassembly of the extruder.

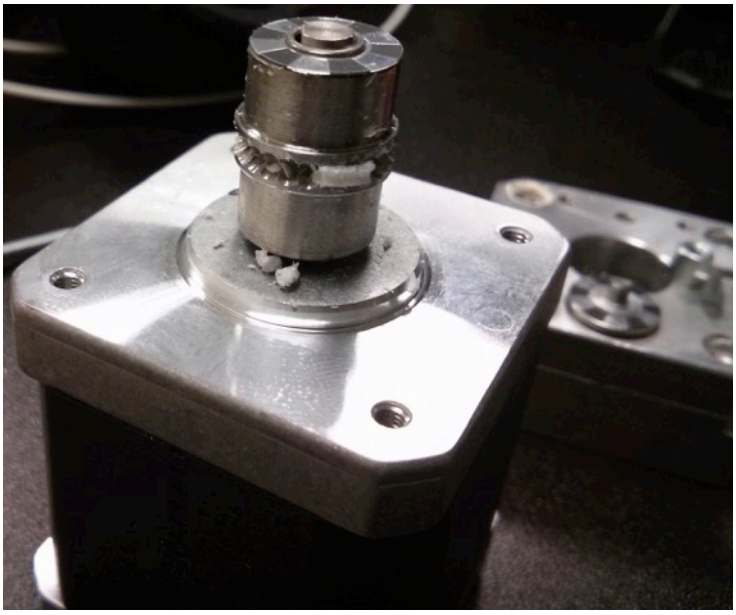


Figure 11 Filament debris in extruder drive gears

Printing Coil (Antenna) Forms

The LC tank used by RFID devices involves a coil (inductor) tuned to the capacitor for the desired resonance frequency. The induction of the coil is dependent on the shape, size, wire diameter and number of turns in the coil. Microchip has an excellent application note describing the calculations for various shapes and properties of RFID antennas[15]. The proxmark3 GitHub also contains details of antenna[16] designs specific to the proxmark3 but which can be modified for other applications with the appropriate calculations or measurements. 3d printing can easily produce the shapes needed to form wires into the desired shapes with precise measurements. Once an antenna form has been designed, it can be integrated with other models to make coils hidden within other objects such as clipboards, lanyards, phone cases, or even fake card readers.

[15] http://ww1.microchip.com/downloads/en/AppNotes/00710c.pdf
[16] https://github.com/Proxmark/proxmark3/wiki/Antennas

First Antenna Design

The first antenna design tested was inspired by D18c7db's LF antenna from the proxmark3 wiki[17] but using 3D printed components to create the shape. The first antenna constructed demonstrated higher voltages than the $60 commercial antenna purchased with the proxmark3.

```
proxmark3> hw tune

Measuring antenna characteristics, please wait...
# LF antenna: 31.21 V @   125.00 kHz
# LF antenna: 18.29 V @   134.00 kHz
# LF optimal: 31.21 V @   125.00 kHz
# HF antenna:  2.23 V @    13.56 MHz
# Your HF antenna is unusable.
Displaying LF tuning graph. Divisor 89 is 134khz, 95 is 125khz.
```

Figure 12 First Coil Tuning Output

Constructing this antenna required some supplies (note that CubePro filament costs above average):

40 AWG Magnet/Enamel Wire	$9.99 (enough for 1000+ coils)
Retractable 32" USB<->Hirose	$5.75 (any Hirose cable is fine)
Printed form	$4 (CubePro), $1 (generic), Shapeways ($6)
Silicon squeeze tube (optional)	$3 (enough for multiple coils)
Heat shrink tubing (optional)	

And some basic tools:

Soldering iron
Wire strippers

To start, tape down a length of wire on the front and pull it to the back with the wire's groove. Now count with each bend of the wire (i.e. count 4 per turn) until you reach 480 (120 turns). Be careful with each turn that the wire is not caught on the side of the form or anywhere other than directly in the path for the wire. At the end, pull a length of the wire to the front, tape it down, and solder a short wire to each end of the coil. This is more than enough turns but this is a case where extra is definitely better than not having enough. Removing turns from the loose end is easy but if you don't have enough turns after you've cut the wire and started testing, you are out of luck for adding new turns. (If you have a good meter, you can verify that the connections are stable before the next step.) It is also important to note that the enamel of the wire will prevent conductivity but the heat of the soldering iron allows a joint to be formed. The enamel can also be removed by gently scraping with a razor.

The hirose cable has a wire pair for low frequency tuned coils and high frequency as indicated in the referenced wiki. The wire within the hirose connector VERT tested had stranded wires great for

[17] https://github.com/Proxmark/proxmark3/wiki/Antennas#D18c7db's-LF-Antenna

connecting into a proto-board. The antenna can now be attached to the proxmark3 through a breadboard and tested on the proxmark3 with the 'hw tune' command. The initial tune execution should show an optimal frequency below 125kHz as in figure 13.

```
proxmark3> hw tune

Measuring antenna characteristics, please wait...
# LF antenna:  4.54 V @   125.00 kHz
# LF antenna:  3.85 V @   134.00 kHz
# LF optimal: 30.52 V @    90.91 kHz
# HF antenna:  2.23 V @    13.56 MHz
# Your HF antenna is unusable.
Displaying LF tuning graph. Divisor 89 is 134khz, 95 is 125khz.
```

Figure 13 Untuned LF antenna

By gradually removing coils, the optimal frequency increases little by little as shown in figure 14.

```
proxmark3> hw tune

Measuring antenna characteristics, please wait...
# LF antenna:  4.81 V @   125.00 kHz
# LF antenna:  3.99 V @   134.00 kHz
# LF optimal: 29.84 V @    93.02 kHz
# HF antenna:  2.27 V @    13.56 MHz
# Your HF antenna is unusable.
Displaying LF tuning graph. Divisor 89 is 134khz, 95 is 125khz.
proxmark3> hw tune

Measuring antenna characteristics, please wait...
# LF antenna:  5.09 V @   125.00 kHz
# LF antenna:  4.26 V @   134.00 kHz
# LF optimal: 29.56 V @    94.49 kHz
# HF antenna:  2.38 V @    13.56 MHz
# Your HF antenna is unusable.
Displaying LF tuning graph. Divisor 89 is 134khz, 95 is 125khz.
```

Figure 14 Unwinding the coil to tune it

The coil is unwound enough when the tune function indicates that your target frequency is the optimal frequency for the circuit. Since 125kHz is the carrier frequency for most badges used for access control,

this frequency was selected for the coil. For my proxmark3, this turned out to be around 90 turns of the coil rather than the calculated 107 turns. (This could be the result of component tolerances or uneven wire coiling but in the end shouldn't matter as long as decent voltage is observed.)

```
proxmark3> hw tune

Measuring antenna characteristics, please wait...
# LF antenna: 31.21 V @   125.00 kHz
# LF antenna: 18.29 V @   134.00 kHz
# LF optimal: 31.21 V @   125.00 kHz
# HF antenna:  2.23 V @    13.56 MHz
# Your HF antenna is unusable.
Displaying LF tuning graph. Divisor 89 is 134khz, 95 is 125khz.
```

Figure 15 Tuned coil for first antenna

The new antenna itself is not much different in size than a typical building access control card making it ideal for 'blending in' when used in card simulation mode. When building an antenna like this remember that 40 AWG wire is incredibly thin and easy to break so it helps to use adhesives to hold in the wires of the coil while also better securing the antenna as a whole to the cable. Otherwise it can be very frustrating to finish up an antenna only to tear too much of the lead off by accident.

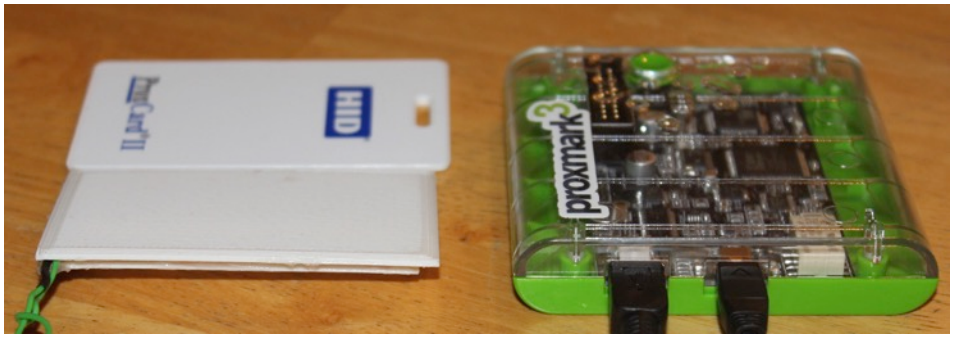


Figure 16 First antenna size perspective

The use of silicon sealant around the coils helps keep the coil in place and tightly wound, adding strength to the very fragile magnet wire. The use of heat shrink tubing and hot glue also helps reduce the tension



Figure 17 First antenna (back side)

BADge Antenna

With the antenna already being very similar in size to a typical RFID card, it is easy to extend the drawing by adding a lanyard hook to make for an antenna that can be worn with a lanyard and look very similar to a 'real' badge. Adding the loop was a trivial step in CAD once general dimensions were known.

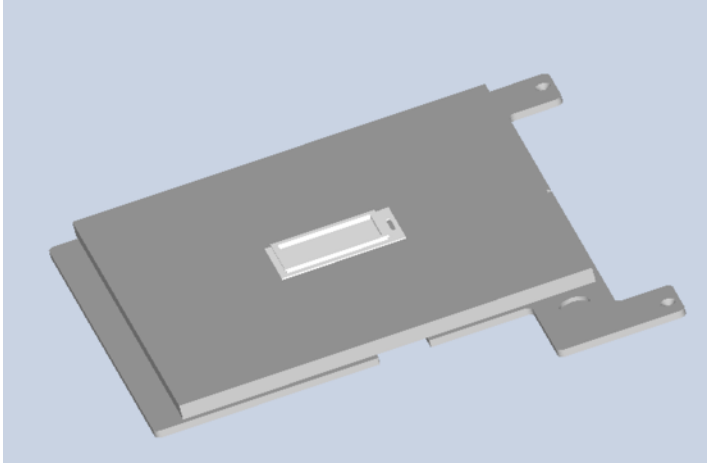


Figure 18 Antenna with Lanyard Loop in CubePro

The antenna was tuned similar to the process described above and then a black wire was run through a newly created hole in a black lanyard. The cable was able to come out at the top end with leads for the antenna while the hirose connector for the proxmark3 remained in my pocket out of site. The cost of this model on Shapeways was $12.68 including shipping and had excellent quality as shown in figure 19.



Figure 19 Antenna hidden in fake badge

Clipwnd (Clipboard with a Kick)

This storage clipboard comes packing a punch with 3D printed guide bars holding the proxmark3 and its accessories (including a USB battery pack) in position to start stealing badge data. The addition of embedded computers can open the doors for more advanced functionality with support via smartphone or by a third party overseeing your RFID hacking operation.



Figure 20 Storage clipboards can hide RFID tools in plain sight

Business Information Modeling (BIM), 3D Scanning, and RFID Hacking

3D printing opens up many doors for rapid prototyping of enclosures to make RFID hacking tools look like something other than what they are. The problem however is accurately capturing a likeness of the intended object without detailed measurements or excellent artistic skills. Fortunately there are some resources available for reducing the complexity of this problem. Building Information Models (BIMs) are a great example of this with more and more manufacturers contributing 3D models of their products for the purpose of simplifying the construction process. Catalogues of these models are available for architects to pull into AutoCAD files with the Autodesk Seek BIM catalogue.[18] A simple Google query for 'DWG file repository' also reveals several other options for obtaining useful model data.

Sometimes a model simply is not available for whatever reason and other steps must be taken to generate a model. Just a few years back this would generally mean purchasing a $1000+ desktop scanner or perhaps a digitizing arm to take precise measurements. This is no longer the case however as the rapid evolution of moderately priced 3D printing has created a market for hobbyist friendly 3D scanning solutions. Some software solutions like David 3D[19] process images of an object while a laser beam moves across it. This makes it possible to invest more or less depending on the required scan fidelity as the software can achieve quality scans with relatively low quality hardware. The technology to map out a 3D space has also penetrated into consumer devices with the most notable instance being the Xbox Kinnect gaming controller. With the release of APIs for working with the Kinnect, software packages such as Skannect and ReconstructMe now allow it to be used a general-purpose 3D scanner with impressive results. More recently VERT has experimented with 3D scanning using the free 123D Catch app from Autodesk[20] that processes a series of smartphone photos to reconstruct an object.

Using Proxmark3 like a Boss

The proxmark3 is only as powerful as the firmware it uses and the commands known to the operator. In the first half of 2015 there have been several new features implemented to make the system more powerful. This section of the white paper starts by giving an example of analyzing an unknown card and cloning it to a T55x7 using commands from pm3 developer Marshmellow42. Sometimes however a job may call for digging into the proxmark3 firmware and adding new features for specific tasks. This section provides an overview of how to use some of the newer advanced features explains by example how to design and implement new commands using the AWID26 format as an example.

Using 'lf search'

This section white paper focuses on a few of the changes in the low frequency (lf) context of the device. New features introduced in March 2015 allow automatic identification of low frequency tags and easy

[18] http://seek.autodesk.com/
[19] http://www.david-3d.com/
[20] http://123d.autodesk.com/catch/

recovery of the demodulated tag data. This is particularly useful when the goal is to simulate or clone a tag after interrogating it. The first step in this process is 'lf search,' a command to sample data from the coil and test each implemented demodulation looking for a match.

```
proxmark3> lf search
Reading 30000 bytes from device memory

Data fetched
Samples @ 8 bits/smpl, decimation 1:1
NOTE: some demods output possible binary
  if it finds something that looks like a tag
False Positives ARE possible


Checking for known tags:

HID Prox TAG ID: 0deadbeef (57207) - Format Len: 37bit - FC: 3562 - Card: 450423

Valid HID Prox ID Found!
```

Figure 21 Identifying HID37 tag via 'lf search'

With the samples in the proxmark3's graph buffer, it is possible to use other commands to inspect the waveform. Simulation of the tag from this point is as simple as running 'lf simfsk' with no other arguments. Clock and logic settings are used from the identified waveform as well as the demodulated bit stream.

Printing and using the demodulated buffer

The hexadecimal representation of the demodulated buffer can be printed with the data command 'printdemodbuffer x' as shown in the following figure.

```
proxmark3> data printdemodbuffer x
DemodBuffer: 1D555555A6A999A69AA9A9AA
```

Figure 22 demodbuffer from HID37 waveform

With a little help from a forum post on the proxmark developers community forum[21] indicates the T55x7 block 0 configuration for different popular tag types. HID tags use FSK2a with an RF/50 data rate where a logical 0 consists of six logic 0 FSK cycles and a logical 1 is signaled by five logic 1 FSK cycles. This

[21] http://www.proxmark.org/forum/viewtopic.php?id=1767

corresponds to block 0 settings 0x00106070. The demodbuffer bytes can then be split up into blocks and programmed onto the T55x7 card.

```
proxmark3> lf t55xx write 0 0x00107060
Writing to block: 0  data  : 0x00107060
proxmark3> lf t55xx write 1 0x1D555555
Writing to block: 1  data  : 0x1D555555
proxmark3> lf t55xx write 2 0xA6A999A6
Writing to block: 2  data  : 0xA6A999A6
proxmark3> lf t55xx write 3 0x9AA9A9AA
Writing to block: 3  data  : 0x9AA9A9AA
```

Figure 23 Writing demodbuffer to T55xx

Developing New Features for The Proxmark3

Sometimes a task requires going beyond existing features to get the job done. The open source philosophy of tools like the proxmark3 make this possible but it is not always intuitive where to begin. Designs for the hardware and software of proxmark3 are maintained in a GitHub repository along with a helpful wiki for getting started.[22] This section is intended as a reference guide for getting started with proxmark3 firmware development.

Before jumping into the code it is helpful to introduce (or refresh) some key concepts related to how the pm3 operates. Proxmark3 is a USB powered device designed to either run independently with push-button input (stand-alone mode) or receiving commands over a USB serial data link. In contrast to tools like RFIDler or the recently released ChameleonMini, serial commands are sent to the pm3 using a custom client rather than having a native text driven command line interface. The proxmark client instead issues commands to the hardware by sending an *UsbCommand* data structure containing a command number and arguments to be consumed and processed by the ARM CPU. This device/client model reduces communication on the serial bus and allows developers to offload some operations from the resource constrained ARM. (Constraints must be considered in terms of both storage space and processing capability.)

The FPGA is a lot of what makes proxmark3 standout from the crowd but fortunately knowledge of Verilog is not required for most development tasks. In fact, it should be possible to add support for any low-frequency tag format entirely within C code. In low-frequency mode, the FPGA reads raw A/D samples via SPI and passes them onto the ARM for processing using SSP. In high-frequency mode the FPGA abstracts ISO compliant modulations so that the ARM can work with those types of tags as well. As a C developer looking to enhance the proxmark3, the three main directories of interest are *armsrc*, *client*, and *common*. These intuitively named directories contain code for the ARM, the proxmark client, and of course shared functionality accessible by either component.

[22] https://github.com/Proxmark/proxmark3

In order to illustrate the process of enhancing the proxmark3 firmware, this paper explores in depth the process of introducing client-driven support for a previously unsupported tag format and stand-alone support for ISO14443a (NFC) tags. Although proxmark3 has an extensive list of supported tag formats, there was no support for AWID tags when this research began and with access to both a card and reader it was a natural choice for contributing back to the community.[23]

Adding real-time demodulation of AWID26

While demodulation of an AWID FSK waveform from data samples was added in January 2015 with commit 1e090a61a149a58a57e9d9acbf5e5532387867a4, there was no corresponding functionality to make the pm3 an AWID reader in the way '*lf hid fskdemod*' allows. Having an AWID26 reader is useful for practical attacks in which the penetration tester is trying to skim badge numbers. The tester can leave real-time demodulation running and logging while casually attempting to get close enough to people with legitimate access to the facility under test.

The first step in supporting a new tag format is to identify the modulation scheme and parameters in use. In general when working with a completely unknown tag the starting point would be to energize the card and collect samples to plot. Once familiar with the general envelope of typical modulation schemes, it is often possible to recognize the modulation with a simple visual assessment of the waveform. Some characteristics such as the data rate can be calculated from the waveform while others may require more guesswork or trial and error. Fortunately the AWID cards are clearly marked and as with most cards of any popularity, the information is out there if you know where to look. In this case, I found all the information I needed from the previously referenced proxmark.org thread on t55x7 card programming. Decoding the block 0 value 0x00107060 tells us that AWID uses FSK2a with RF/50 data rate and the attached PDF illustrates the high and low logic sequences. This is coincidentally the same FSK modulation as used in the popular HID tag format but the similarity starts and ends there with AWID tags having a different preamble pattern and encoding technique.

Moving the AWID demodulation from a client side operation on the local dataset into the ARM is a reasonably straightforward process once the logistics are understood. To start, I defined a new command opcode with the pneumonic *CMD_AWID_DEMOD_FSK* by adding it to the appropriate header files referenced in the proxmark3 ARM section of this document. Logic to handle this command was added in the USB packet handling routine from *armsrc/appmain.c* following the pattern used for the '*lf hid fskdemod*' command. A new function *CmdAWIDdemodFSK()* was added to *armsrc/lfops.c* including the logic from '*data fskawiddemod*'. Migrating the logic from the client to the device side requires changing *PrintAndLog()* calls to the appropriate device side logging function (*Dbprintf/DbpString*), code to abort the function loop in response to a button press, and most critically that the demodulation is fed by live data acquisition rather than the graph buffer. On the device side we simply obtain a pointer to *BigBuf* with *BigBuf_get_addr()* and use *DoAcquisition_default()* in place of *getFromGraphBuf()*.

With the device end of the functionality in place, the only thing left to do is update the client to include the new context and function. For maintainability it is best to keep tag specific functionality in separate

[23] Client demodulation added Jan. 18 2015

files and so *client/cmdlfawid.c* was created in the likeness of *client/cmdlfhid.c* along with their associated header files. Within this file is a data structure *CommandTable[]* which exposes available commands within a context. *CommandTable* is a *command_s* struct containing a command name, function pointer, an offline flag, and description. Since the AWID functionality falls under the LF context, the AWID commands must be referenced from *cmdlf.c* with a pointer to a helper function to print command usage for the AWID context.

Example usage and output from '*lf awid fskdemod*' is contained in figure 24 below:

```
proxmark3> lf awid fskdemod
#db# AWID Found - BitLength: 26, FC: 1, Card: 31337 - Wiegand: 2f4d3, Raw: 011d8114eb2b281111111111
```

Figure 24 Sample output from 'lf awid fskdemod'

Simulation of AWID26 tags from facility-code and card number

As described earlier, it is possible to simulate most low-frequency tags with built-in generic functions allowing the user to specify a modulated bit stream or reproduce an acquired waveform. For example, the tag scanned in figure 24 could be automatically simulated using LF functions *search* and *simfsk*. (In this scenario, if the tag is supported, *lf search*, will populate the modulation options and specify the demodulated buffer.) It is also possible to use *lf simfsk* on its own by manually specifying clock speed, high and low logic sequences, and tag data. Without the tag present to scan however it becomes a bit more tedious since the operator would need to manually compute the bits to transmit taking into consideration the preamble and parity bits.

All of the AWID tags I have seen to date have a printed facility code and card number making it possible to duplicate the card even from a photograph. This section covers addition of the '*lf awid sim*' command to automate the process of simulating a specific AWID tag based on the numbers printed on the card. The first step in this process is of course to write a function for encoding a facility code and card number into a stream of bits to be transmitted. The logic was coded into *getAWIDBits()* derived entirely from the previously referenced AWID PDF referenced in the T55x7 thread. This function does the bit-wise operations needed to construct an array of bytes representing the card data. (Parity calculations are performed with the existing *parityTest()* function.) While this code could reside in either the client or the device layer, it seems most appropriate to keep this in the client since pm3 will not need to perform this calculation except when receiving commands from the client and so it was added to *client/cmdlfawid.c*.

The next step is to implement code in the client to initiate the tag simulation. While it would be trivial to add a specific command definition to the ARM as was done for the *fskdemod* routine, it is also completely unnecessary and since the ARM has a finite storage capacity it seems best to reuse existing code where possible. Since we know that the *simfsk* operation is already implemented, the easiest option is to have the client compose and send an appropriate *UsbCommand*. An example of sending this command is available in *armsrc/appmain.c* revealing that *CMD_FSK_SIM_TAG* is the appropriate command type leading to the ARM function *CmdFSKsimTAG()*. Reviewing the definition of this function (in *armsrc/lfops.c*) illustrates how the command arguments are used. In this function, *arg[0]* indicates the high and low logic sequence, *arg[1]* combines the clock rate and inversion setting, and *arg[2]* indicates the size in bits of the buffer. The buffer itself is passed in *d.asBytes* with each byte representing a bit in

the intended stream meaning that the 12-byte code will need to be represented as 96-bytes of true or false. The design of *getAWIDBits()* however is to return bytes in an easily printable format so a for loop is used to iterate over each bit setting *d.asBytes* one byte at a time by iterating over the bits masking out all but the relevant bit and setting the byte accordingly.

As illustrated in figure 25, the '*lf awid sim*' command also displays the equivalent '*lf simfsk*' parameters to simulate the specific tag.

```
proxmark3> lf awid sim 23 1337
Emulating AWID26 -- FC: 23; CN: 1337

Press pm3-button to abort simulation or run another command
Running 'lf simfsk c 50 H 10 L 8 d 011d81bd148e281111111111'
#db# Simulating with fcHigh: 10, fcLow: 8, clk: 50, invert: 0, n: 4798
```

Figure 25 Sample output from 'lf awid sim ...'

Cloning AWID26 from facility-code and card number

As previously described, low-frequency tags can be impersonated by cards with the T55x7 family chipset. Programming the card requires nothing more than calculating values for each T55x7 configuration block and writing them to a card. (Figures 22 and 23 illustrate this relationship.) As with the simulation code, the first step is to translate the badge details into a well-formed AWID26 demodulated buffer and then split this across the blocks of the T55x7. Once again this functionality could be performed by a dedicated function on the ARM side but since T55x7 programming functionality already exists it is an easy task to have the client automate the task of calculating and programming the tag blocks. Writing to the tag uses the *CMD_T55XX_WRITE_BLOCK* command with *arg[0]* for the block data (4-byte int), *arg[1]* indicates the block number, and *arg[2]* could be used for working with a password protected card. Similar to the '*lf awid sim*' command, the *clone* operation also prints the equivalent individual commands as shown below.

```
proxmark3> lf awid clone 123 45678
Preparing to clone AWID26 to T55x7 with FC: 123, CN: 45678 (Raw: 011db7de722bd81111111111)
Block 0: 0x00107060
Block 1: 0x011db7de
Block 2: 0x722bd811
Block 3: 0x11111111
proxmark3> lf search
Reading 30000 bytes from device memory

Data fetched
Samples @ 8 bits/smpl, decimation 1:1
NOTE: some demods output possible binary
  if it finds something that looks like a tag
False Positives ARE possible


Checking for known tags:

AWID Found - BitLength: 26, FC: 123, Card: 45678 - Wiegand: 2f764dd, Raw: 011db7de722bd81111111111

Valid AWID ID Found!
```

Figure 26 Sample output of 'lf awid clone ...'

Ironically this relatively simple functionality actually uncovered a quirky behavior in the communication between the client and the device. After submitting the pull request with this function, I received feedback from another developer that the clone method was failing after block 0. After some back and forth we were able to establish that the new code was exposing differences in the serial communication as performed by the Windows client versus Linux/OS X clients. On Windows, data was being lost rather than buffered when commands are sent from the client in rapid succession. The ideal solution for this is to have the client wait for a response from the ARM between each command. This not only eliminates the timing issue but also allows the operator to validate that commands were executed. As a short-term solution however the logic was updated to include a short delay between each command.

Replacing LF standalone mode with NFC functions

Low-frequency and high-frequency RFID both rely on a carrier wave to wirelessly power a chip through induction but in practice the two technologies are very different. Near field communication (NFC) is based on high-frequency RFID technology with communication as specified by ISO14443 standards. Whereas with AWID we saw that the tag would simply create the same waveform repeatedly and the reader would simply demodulate that signal, NFC tags allow for a variety of operating modes including the general sharing of data. While some tags support security features such as cryptography, some popular applications are designed to only verify basic unprotected data such as the tag's unique identifier. For example, Timo Kasper's PhD thesis from Ruhr-University Bochum describes how the Siemens SIPORT enterprise solution for access control was easily subverted because the system relied on the UID (transmitted in plaintext) and the first sector of a Mifare Classic card using default keys.[24] Certain other access control systems including the Android NFC 'Smart Unlock' feature and Samsung locks NFC

[24] Section 5.6.3 https://www.emsec.rub.de/media/attachments/files/2012/11/timo_phd_thesis.pdf

compatible locks (like the SHS-3321) only check the unprotected UID. As discussed earlier in this document, the UID block of a tag is intended to be a factory programmed unchangeable value to uniquely identify the tag. This section outlines how the proxmark3 firmware was enhanced to support stand-alone operation to capture, clone, and emulate UIDs from ISO1443a compliant tags. (Cloning capability in this initial design is limited to changeable Mifare Classic cards.)

Stand-alone mode runs entirely on the ARM with source code contained within the *armsrc/appmain.c* source file. Each loop of the main application checks if a command arrived over USB as well as checking whether the push-button was held down for a second. The traditional low-frequency operation invokes *SamyRun()* which provides replay and simulation logic for HID tags. Interacting with the high-frequency tags requires a different FPGA configuration and of course ISO14443 specific function calls. FPGA configurations are stored in the flash memory of the ARM and can be loaded on demand with *FpgaDownloadAndGo()*. The *FPGA_BITSTREAM_HF* configuration arranges the pin-out so that the appropriate coil is connected and the FPGA is ready to expect HF commands.

The general structure of stand alone operation can be modeled around the existing *SamyRun()* but it requires replacing HID logic with ISO1443 logic. For HID reading mode, *SamyRun()* uses *CmdHIDdemodFSK()* but the new logic will need to first command the FPGA to act as an ISO14443a reader with *iso14443a_setup(FPGA_HF_ISO1443A_READER_MOD)*. Reading the UID of a tag is covered with the *iso14443a_select_card()* call. This function will return false until a card was successfully selected and its UID saved to memory. The UID is printed and then rearranged to the expected byte order for the clone and replay functions.

Again for cloning, *SamyRun()* uses a single function *CopyHIDtoT55x7()* while the HF equivalent of writing to a changeable UID card requires a little more effort. First the UID value must be transformed to create a well-formed block 0. The relevant functions for working with changeable UID Mifare cards are *MifareCSetBlock()* and *MifareCGetBlock().* Example usage of these functions to write a new UID to a card is presented in *mfCSetUID()* which first reads block 0 from the target tag and then replaces the UID bytes while maintaining the reserved bytes 5-7 and setting an appropriate block check character (BCC).

The final stand-alone feature is tag simulation performed for HID tags with *CmdHIDsimTAG()*. For ISO14443, the equivalent function to use is *SimulateIso14443aTag()* which takes in high and low order bytes for the UID and configures the device to act as a tag with that UID. Simulation of an ISO14443a tag however may not work as well as cloning to a card. Using a PCB antenna from RyscCorp[25] works very well for simulating the tag when interrogated by some devices but hardly works on others. For example when the coil is brought in proximity to certain Samsung locks the simulation function reports unexpected commands while a cloned card works perfectly. (Specifically the SHS-3321 and SHS-3420 locks were evaluated.) This may be a consequence of timing discrepancies or perhaps related to the antenna shape but experiments with other hardware based tag simulation (PN533 w/ libNFC and ChameleonMini) had similar results. In general these devices are not recognized as tags by the lock. Further testing is needed to definitively debug this behavior and determine whether this may even be an advanced security feature of the lock. Over the course of many tests, the simulation mode worked

[25] http://store.ryscc.com/collections/proxmark-3/products/high-frequency-pcb-antenna

reliably with a Nexus 4 and Nexus 7 making it possible (as Google clearly warns) for a tag used by the Android Smart Lock[26] feature to be replicated in this manner. With the tested Samsung locks however only opened (one time only) after dozens of attempts with different antenna placement. None of the tested NFC devices had problems reading from clone cards.

Once the main functionality was working, the next challenge was to reconsider the usage workflow. In the traditional LF stand-alone mode it is necessary to first use the button to enter stand-alone mode and then again to enter record mode. It is also possible to go directly into play mode but since the ID is not initialized or preserved across sessions, this seemed extraneous. Instead the NFC code has been designed to enter directly into reader mode and then jump into replay mode when a tag is read. This makes it much easier to operate the device in a concealed container without exposing that the proxmark3 is hidden. With this change, the replay mode waits for instructions to clone or switch banks as displayed in the following flow-chart.

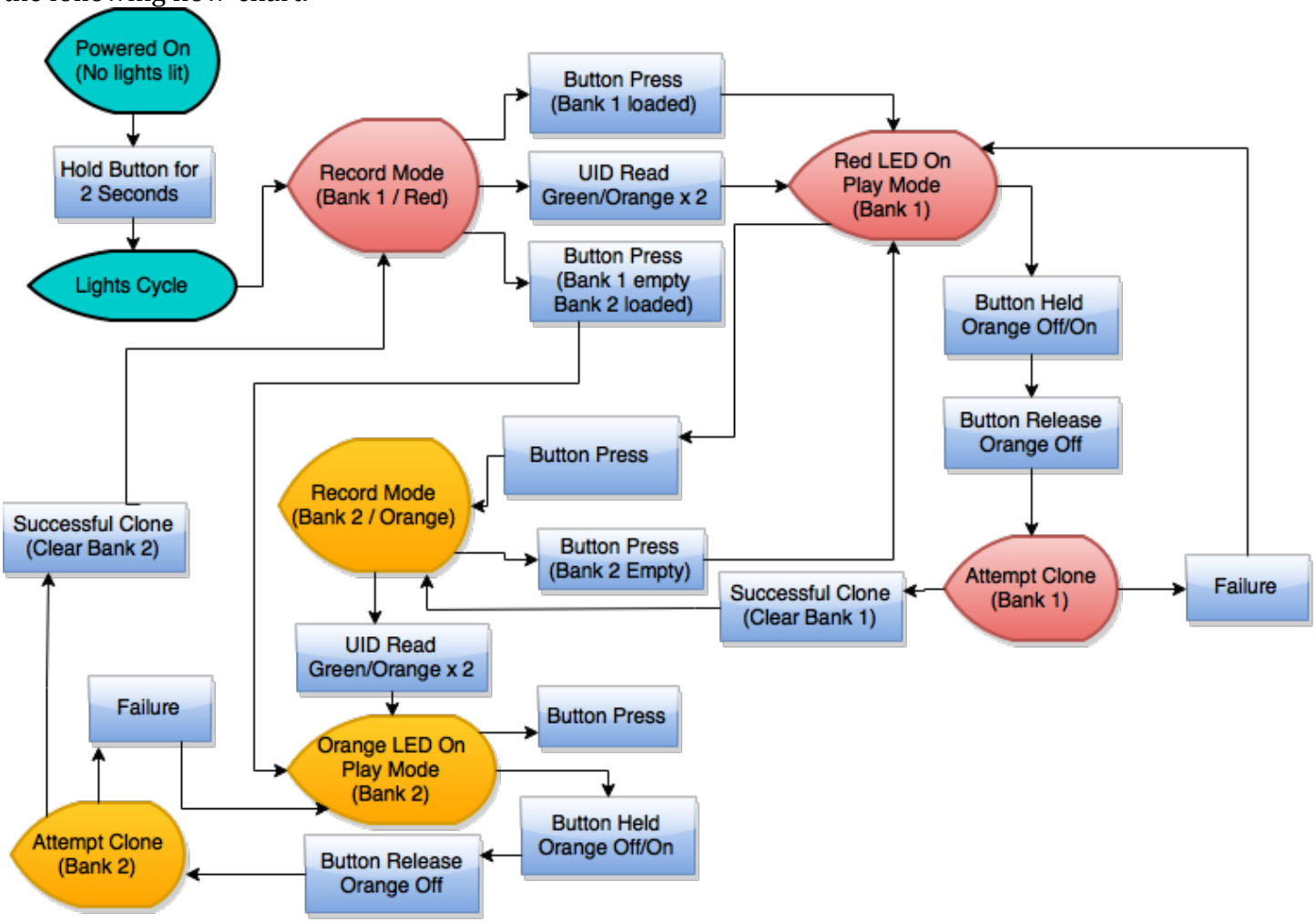


Figure 27 NFC Stand-Alone Mode Workflow

[26] "Setup Your Device for Automatic Unlock", https://support.google.com/nexus/answer/6093922?hl=en

RFIDler: The New Kid on the Block

If the proxmark3 is a nice 21-year-old bottle of single malt, RFIDler is more like a cheap bottle of vodka. While it might get the job done, it may not be the most enjoyable experience. While a 3D printed case definitely improves the read quality, it may still be necessary to spend a bit of time fiddling with coil placement or adjusting thresholds with the digital potentiometers. This is particularly the case when working with tags other than the prevalent HID tag format. The simple circuit design and inexpensive components make RFIDler potentially desirable for use as a more expendable device able to be left at a target site during a physical penetration test. Unfortunately the design does not include a method for storing interrogated tag IDs. Although someone with a background in schematic layouts and PIC controllers should be able to add an SD card slot with minimal difficulty, the overhead involved with designing and producing new boards is probably not the most cost effective unless a large quantity of devices are required. (Creating a new board to log the USB serial output could face similar challenges.)

RFIDlerPi

Pairing the RFIDler with an existing low-cost small computing device would be ideal. This is where the idea of RFIDlerPi began. The $35 Raspberry Pi board compliments the RFIDler quite well providing immediate data logging by setting the RFIDler to auto-start in the desired reader mode and the Pi configured with a basic script or program for dumping data off the serial line and onto an SD card or USB storage device. It is also a small leap from there to add a Wi-Fi dongle to the Pi allowing the operator to retrieve data and control the device over a network connection. Naturally these techniques can also be applied to other tools but the RFIDler is particularly easy to work with for this task because it requires minimal configuration particularly compared to use of a proxmark3 that would generally require compiling a new client for the target architecture and/or making various firmware changes. Alternatively if high-frequency RFID support is needed, a $50 USB based NFC reader will likely do the trick.

3D modeling and CAD software then makes it possible to produce a custom enclosure to conceal the gadgets. Building information modeling (BIM) libraries even make it possible to obtain decent likenesses of many objects commonly found in commercial buildings. Often times RFID vendors will even make available CAD drawings to customers making it possible for someone to make a very convincing fake reader. This is especially true with the growing assortment of manufacturing-as-a-service businesses allowing anyone to acquire prints from high-end commercial printers that may be prohibitively expensive to maintain and operate in house.

Concluding Remarks

Of the hacking tools used in this research, each has its strengths and weaknesses but it is clear that the proxmark3 is at this time the best option for overall RFID research of low and high frequency implementations. It is worth noting however that if only high frequency (NFC) capabilities are desired that more affordable options exist in the form of libNFC compatible USB devices. This approach is

primarily limited in that the low-level modulation functionality is typically not upgradeable as is possible with hardware implementations like the proxmark3. The ChameleonMini provides a good compliment to both of these tools enabling in circuit NFC tag emulation in a form factor similar to a typical contactless smartcard.

The RFIDler is also a project to keep an eye on if only interested in low frequency RFID but in its current state (v22-beta) it did not perform as well as proxmark3. Based on the collection of tags tested it would appear that HID reading is the most reliable. Other formats required a little more patience and reconfiguration of the POT settings and finding a sweet spot for how to place the card in relation to the antenna coil.

The intent of this project was to create a suitable introduction for newcomers and veteran hackers alike looking to harness the power of their RFID hacking tools. The firmware modifications documented in this paper have been merged into the official proxmark repository so that they may come in handy for users as well as being a partial guide for getting started with new development. Antenna form models are also being made available as STL files suitable for printing on most 3D printers or via an online printing service. The concealment designs discussed are intended as exercises for the reader as the needs of each user are expected to vary greatly based on the application.