A person in a suit holding a pair of scissors and a pair of tweezers

Description automatically generated

Louis Pung

PROXMARK3 SETUP GUIDE

Contents

[1. Set-up for Proxmark3 on Windows/Linux 3](#_Toc163509069)

[PM3 Software 3](#_Toc163509070)

[Test Hardware 3](#_Toc163509071)

[Preparing ProxSpace (for Windows) 4](#_Toc163509072)

[Getting latest firmware 5](#_Toc163509073)

[Makefile.platform 6](#_Toc163509074)

[Flashing the PM3 8](#_Toc163509075)

[Running the client 9](#_Toc163509076)

[Check antennas 10](#_Toc163509077)

[Updating firmware in the future 11](#_Toc163509078)

[A simple trick to speed things up in ProxSpace 11](#_Toc163509079)

[2. Enabling Script Support on the Proxmark3 13](#_Toc163509080)

[Overview 13](#_Toc163509081)

[Installation 13](#_Toc163509082)

[SWIG Setup 15](#_Toc163509083)

[Getting Scripts to work 16](#_Toc163509084)

[3. Exploring with the Proxmark3 18](#_Toc163509085)

[Proof of Concept 18](#_Toc163509086)

[HID SEOS - Ensign Work-pass 20](#_Toc163509087)

[Findings 20](#_Toc163509088)

[NYP Card – High Frequency MIFARE Classic 1K 21](#_Toc163509089)

[GymBoxx - High Frequency MIFARE Classic 1K with Static Encrypted Nonce 24](#_Toc163509090)

[Anytime Fitness Keyfob – Low Frequency, T55XX 25](#_Toc163509091)

For any questions and answers related to the Proxmark3, you can visit their [Discord channel](https://discord.com/invite/iceman).

# 1. Set-up for Proxmark3 on Windows/Linux

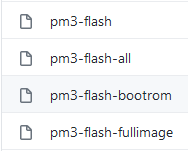
Adapted from Dangerous Things forum and [this video](https://www.youtube.com/watch?v=o6WOTM4D970)

## PM3 Software

[GitHub - RfidResearchGroup/proxmark3: Iceman Fork - Proxmark3](https://github.com/RfidResearchGroup/proxmark3)

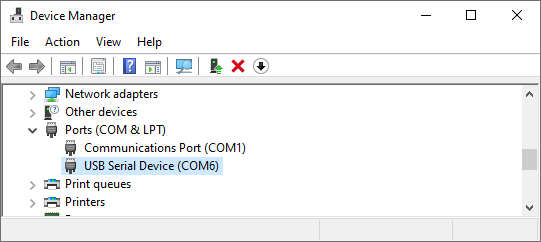
* Most well-rounded and somewhat user-friendly version of firmware is forked and maintained by Chris “Iceman” Hermann

There are two parts to the proxmark3 2.4k firmware - the bootloader and the application firmware. The application firmware is called the “full image” within the proxmark3 2.4k firmware development realm, but it does not contain the bootloader… so it is not a “full” representation of all the firmware elements that must be flashed (loaded) to the pm3 5.7k hardware…



## Test Hardware

USB hardware on the pm3 5.7k now acts like a virtual COM port using a very common HID (human interface device) driver interface, so no special drivers are necessary… you just plug it in and the hardware should be recognized immediately as a COM port.



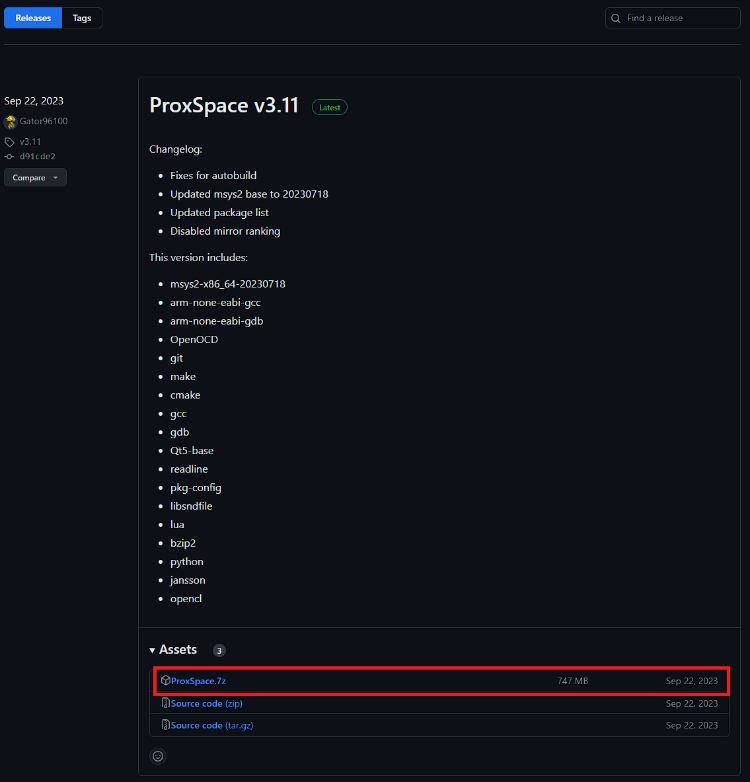
In Windows, you can open the Device Manager to check that your Proxmark3 2.4k is showing up as a virtual COM port.

## Preparing ProxSpace (for Windows)

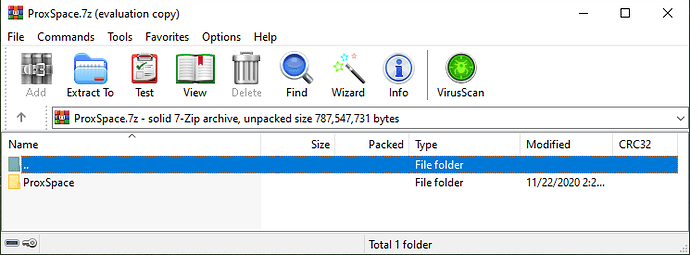
**(If you are doing this on a Linux environment, you can skip this step. ProxSpace is meant to provide a Linux environment solely meant for using the Proxmark3.)**

We will get Windows to provide an environment to compile, deploy firmware and run the client software via a software package called [ProxSpace](https://github.com/Gator96100/ProxSpace).

[Releases - Gator96100/ProxSpace](Releases%20-%20Gator96100/ProxSpace)



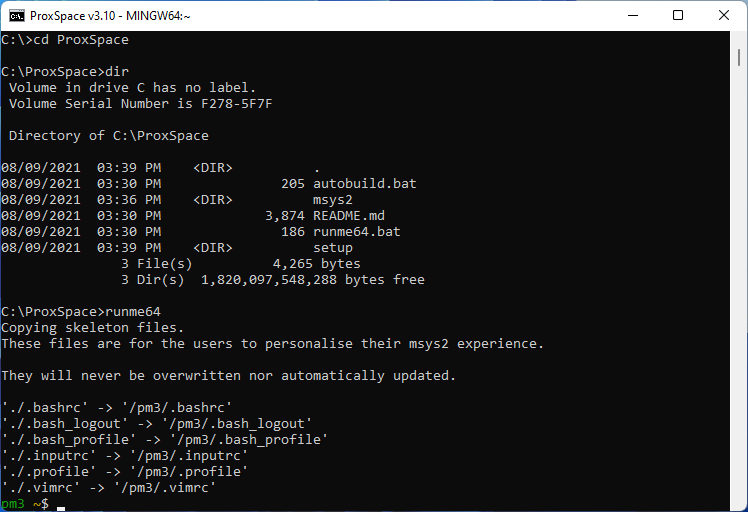
Download the **ProxSpace.7z** file as highlighted in the image.



Upon extraction of the ProxSpace to a folder of choice

* Suggest you put it into the root of your hard drive like C:\ProxSpace for example. It may be tempting to place it into a Program Files type location, but those folders have spaces in the path, special permissions, certain system protections, and all sorts of other nonsense can get in the way of a successfully working deployment

Once you’ve extracted the files, run the runme64.bat file and wait while ProxSpace configures itself.



Once finished, you should end up inside the ProxSpace environment with a green pm3 prompt.

## Getting latest firmware

We can now grab the Iceman firmware from GitHub. To do this, simply enter this command at the pm3 prompt:

**git clone https://github.com/RfidResearchGroup/proxmark3.git**

**cd proxmark3**

The proxmark3 folder is where the source files live for the firmware and client, and it will be where the compiled versions live after we run the compile commands.

## Makefile.platform

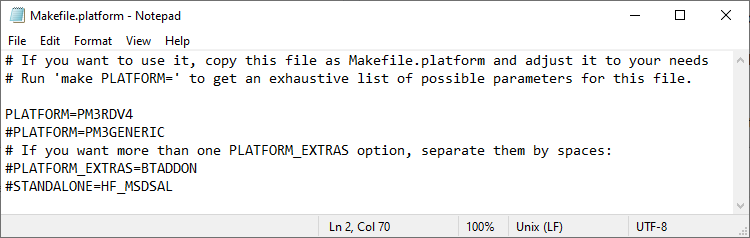
Before we compile anything, we need to be sure the firmware is going to compile for the correct hardware.

* Iceman firmware was made specifically for the **Proxmark3 RDV4 hardware version**, so some tweaks have to be made if you are not using the RDV4 version

We do this by creating a Makefile.platform file and setting a value inside. Luckily there is a sample version of this file we will make a copy of, then edit. To do that, use the copy command:

**cp Makefile.platform.sample Makefile.platform**

Using an editor of your choice (nano, notepad, mousepad), open up the Makefile.platform

Note the lines:

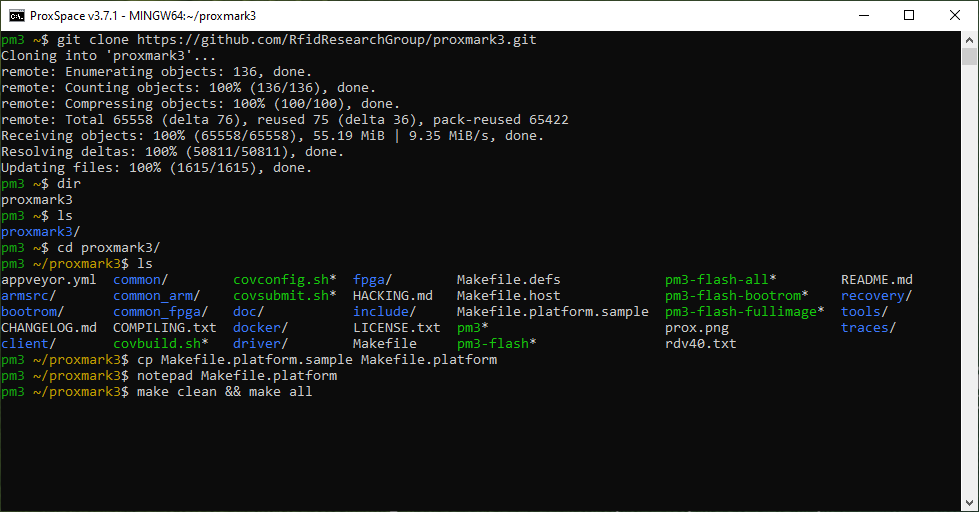
* PLATFORM=PM3RDV4
* #PLATFORM=PM3GENERIC

Comment out the line with a # if you are not using that specific version.

Save the file (CTRL-S) and close Notepad and go back to the pm3 prompt. The compile commands in Linux are bundled into a simple make command. Now we just tell ProxSpace to compile the firmware and client software.

**make clean && make -j8 all**

**Note that this process is likely to take a while.**



After the make build process, you will have new files and folders… a collection of pm3-flash files and a client folder as well.

## Flashing the PM3

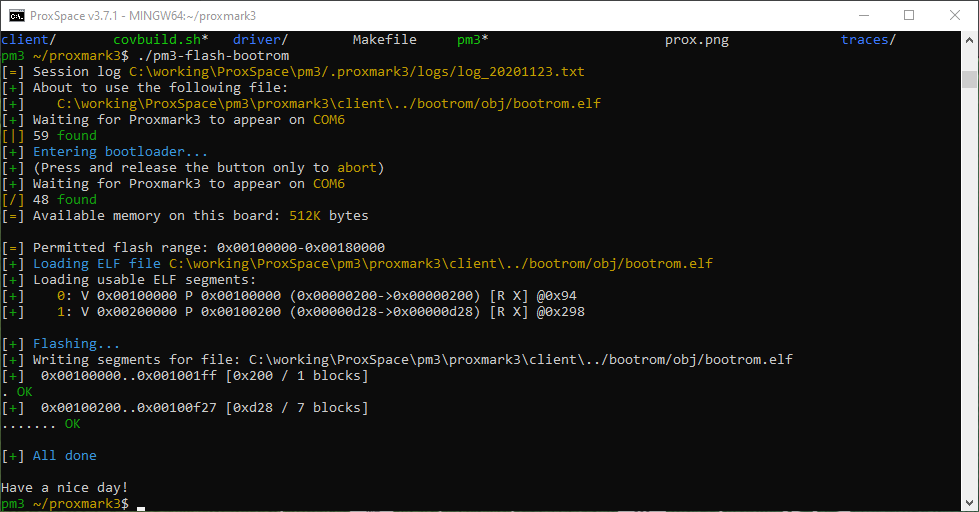
If your PM3 is currently plugged into your computer, take it out and plug it in again, making sure that you are holding onto the built-in button as you do so.

Start with the bootloader by typing:

**./pm3-flash-bootrom**

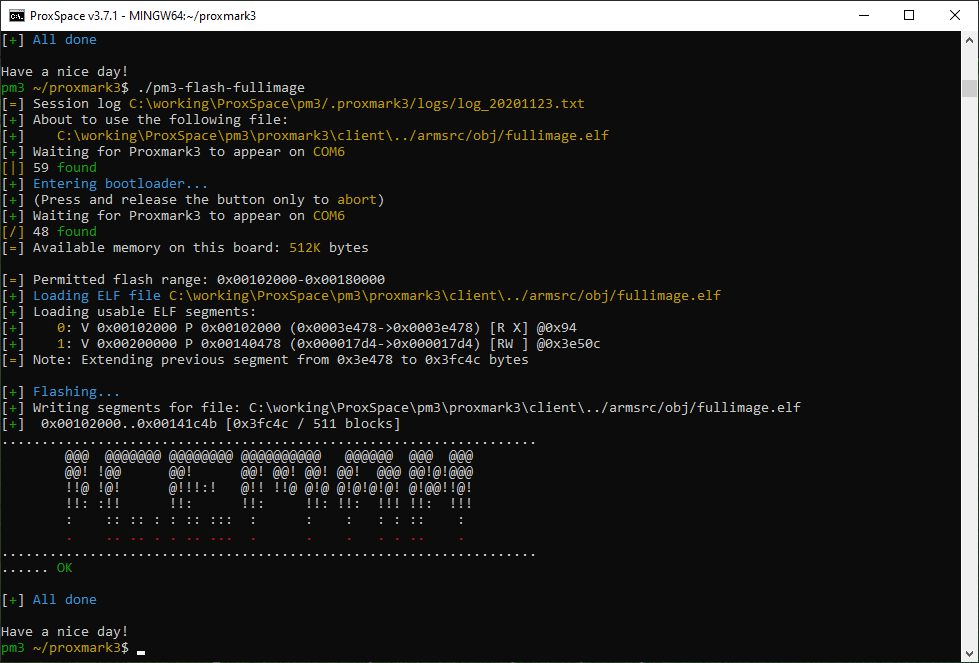
* The bootloader is a small chunk of code that gets the proxmark3 hardware up and running, and then it executes the application “full image” firmware. For the most part, you can update the bootloader and full image all in one go if you are simply updating your firmware to the next minor revision on the same fork… but if you are changing forks from say the factory firmware over to Iceman, then it’s important to attempt to update just the bootloader first, then the application firmware. Doing so can be tricky of course, which is why we have jumped forks over to Iceman.
* The bootloader also determines how your Proxmark3 communicates over USB.

When you press enter you will see a series of things happen and the bootloader will be updated…



Now we need to flash the full image by typing:

**./pm3-flash-fullimage**



## Running the client

With the firmware updated to the current version, now it’s time to run the client **(in the /proxmark3 directory)** via:

**pm3**

This command will automatically find the COM port the proxmark3 is operating on and launch the client.

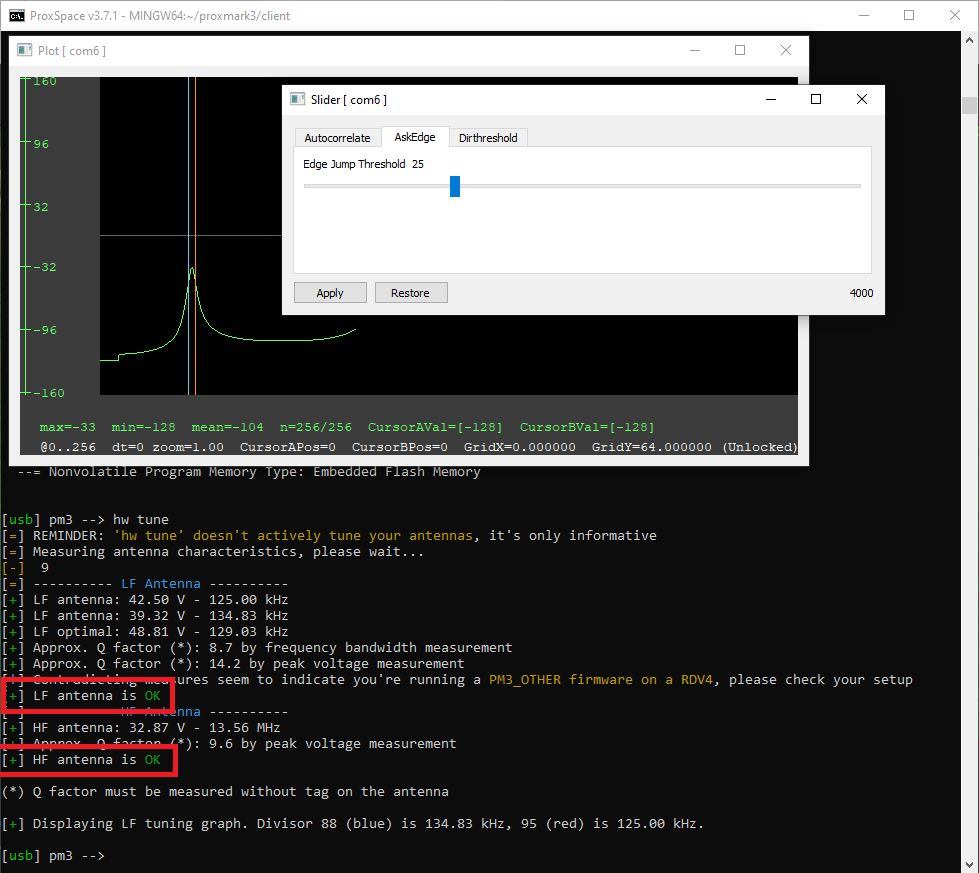


**At this point you are in the Proxmark3 client, inside the ProxSpace environment. To get back here in the future, you have to use the runme64.bat file to get into ProxSpace, then cd proxmark3 /client, then ./proxmark3 to launch the pm3 client software.**

## Check antennas

Now the first thing you want to do with any new proxmark3 hardware is check the tuning of your antennas. To do this make sure the pm3 is sitting on a wood table (not metal) with no RFID or NFC tags around it and type the command:

**hw tune**



Ignore the graph and slider windows for now. You are only concerned about the lines highlighted.

## Updating firmware in the future

If you should ever want to update firmware in the future, the simplest way at this point is to launch ProxSpace with runme64.bat then rename the existing proxmark3 folder with the Linux move command;

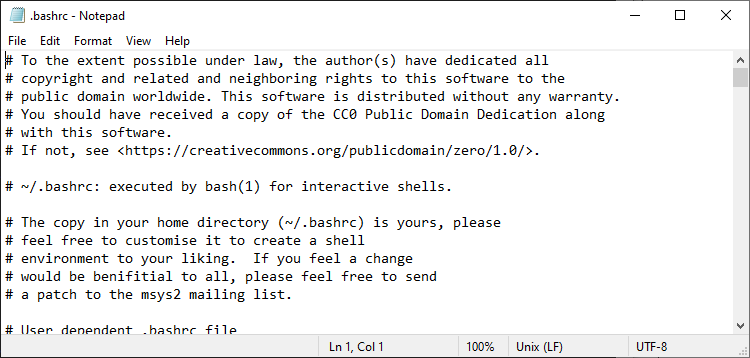
**mv proxmark3 proxmark3.old**

Then use the GitHub command git to pull down the current Iceman repo and follow those steps again.

* Navigate into the proxmark3 folder
* Pull down the latest source code by issuing git pull
* Set the Makefile.platform file
* Change the lines to activate PM3OTHER
* Run make clean && make all
* Flash your pm3 bootloader and app image
* Run the pm3 client in the proxmark3 folder

## A simple trick to speed things up in ProxSpace

Since ProxSpace is just providing a useful Linux environment for doing one thing and one thing only - using your proxmark3, then there is no reason why it should not start the client immediately for you once you enter the ProxSpace environment. To get ProxSpace to do this, you have to edit the **.bashrc** file by issuing the command notepad **.bashrc** at the prompt as soon as you enter ProxSpace. Then notepad will open, and you’ll see something like this:



Add this line at the very end of the file:

**proxmark3/pm3**

Save the file, and the next time you enter ProxSpace the proxmark3 client will immediately launch.

* If you ever want out of the client to issue commands in ProxSpace for things like updating the proxmark3 source code and re-compiling, then you can just use the **quit** command to get out of the proxmark3 client and go back to ProxSpace.

# 2. Enabling Script Support on the Proxmark3

Adapted from [this video by Iceman Channel](https://www.youtube.com/watch?v=05v1A4pzZWQ&t=440s).

## Overview

By now, you should be able to open the Proxmark3 client. Within the client, running the command:

**script list**

A screen shot of a computer

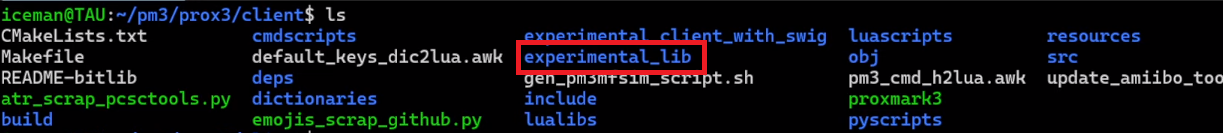
Description automatically generated

These are the scripts that come with the Iceman fork. They are considered as external scripts that can be run inside the Proxmark3 client. It supports both Lua, Python and native Proxmark3 commands (if typed out in a text file).

However, sometimes we do not want to be in the client to run these scripts. Instead, we want to operate from a Python script.

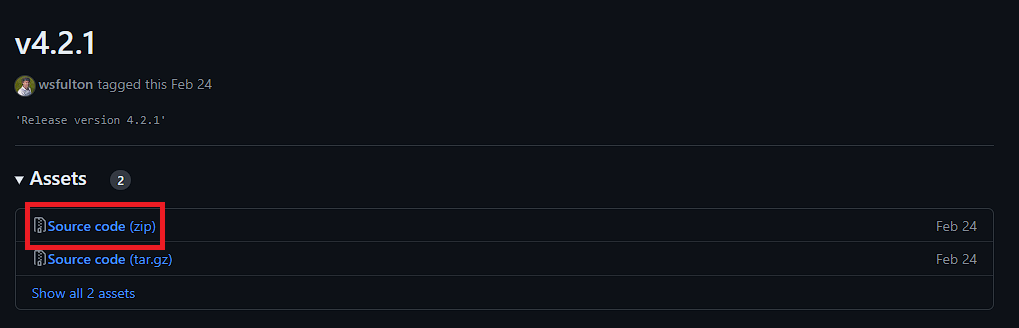
## Installation

If you **cd client** (quit the Proxmark3 client first), you will see the following within the /client directory:



This folder makes use of [SWIG](https://github.com/swig/swig) to generate wrappers around the C code, into different higher level languages like Python. We would need to install SWIG to make this work.

To install SWIG, you need to head over to the [releases on the SWIG GitHub](https://github.com/swig/swig/tags).



This guide will not list out the steps to install SWIG, as this widely varies by the operating system.

* Please read the Doc/Manual/Preface.html#Preface\_installation for full installation instructions for Windows, Unix and Mac OS X using the release tarball/zip file. The INSTALL file has generic build and installation instructions for Unix users. Users wishing to build and install code from Github should visit <https://swig.org/svn.html> to obtain the more detailed instructions required for building code obtained from Github – extra steps are required compared to building from the release tarball.

For the installation of Python script support, as stated in the [Github repository for Proxmark3](https://github.com/RfidResearchGroup/proxmark3) (within the Windows and Linux installation guide), we require the installation of **libpython3-dev**.

A black background with white text

Description automatically generated

Hence, for your respective operating system, run the installation for the requirements as recommended by the guides on the repository.

Linux:

**sudo apt-get install --no-install-recommends git ca-certificates build-essential pkg-config libreadline-dev gcc-arm-none-eabi libnewlib-dev qtbase5-dev libbz2-dev liblz4-dev libbluetooth-dev libpython3-dev libssl-dev libgd-dev**

Windows:

**sudo apt-get install --no-install-recommends git ca-certificates build-essential pkg-config libreadline-dev gcc-arm-none-eabi libnewlib-dev libbz2-dev liblz4-dev libpython3-dev qtbase5-dev libssl-dev libgd-dev**

## SWIG Setup

Within the **/client/experimental\_lib** directory:

A screen shot of a computer

Description automatically generated

Upon further inspection of **00make\_swig.sh**:

A black screen with blue text

Description automatically generated

* There are special commands within the Makefiles to enable Python and Lua script support

**./00make\_swig.sh**

A computer screen shot of a black screen

Description automatically generated

Now sequentially, run the **01make\_lib.sh** and **01make\_lib\_continue.sh**

* You can check out the details of the scripts on your own

When we are compiling these, we are creating a shared library for the Proxmark3 client.

## Getting Scripts to work

Notice within the **experimental\_lib** directory that there is a folder called **example.py** with the contents:

A screenshot of a computer screen

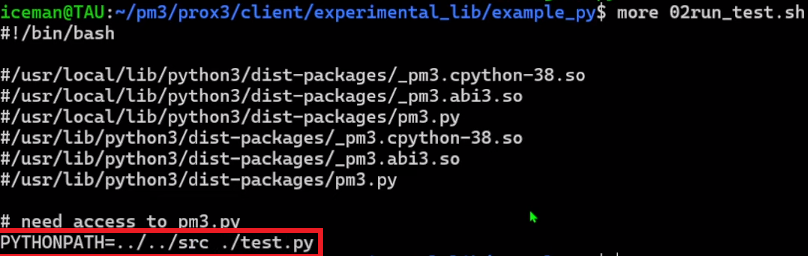
Description automatically generated

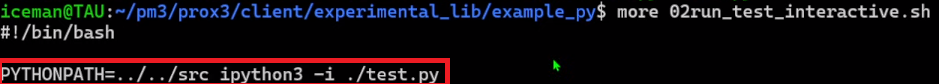
The contents of the **01link\_lib.sh**, as suggested by the name, creates a symbolic link to the shared library folder created in the previous step.

A screen shot of a computer

Description automatically generated

In order to run a python script via a bash script, we need to include the following, as shown in the examples for **02run\_test.sh** and **02run\_test\_interactive.sh**:





* The details of the python scripts within these test files will not be delved into within this guide. Check it out for yourself to figure out the functionality and pm3 API.
* **ipython3 -i** causes the output to allow you to have access to the pm3 console immediately after the run of the script

**Within this guide, it specifically delves into Python scripts. However, this should also work for Lua and C scripts within the other example folders.**

# 3. Exploring with the Proxmark3

This section contains some of the experimentation I had with the Proxmark3 with my limited access to certain passes, and my various findings.

[Here are all the commands for the Proxmark3](https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/commands.md)

## Proof of Concept

**hf search  
hf 14a info**

* The first command will search for known High Frequency tags
  + Detects 14a
* Second command will get the tag information

We have 2 magic cards (MIFARE Classic 1K)

A screen shot of a computer

Description automatically generated A screen shot of a computer program

Description automatically generated

* **UID**: We are concerned with how many bytes of UID. The examples below has 4 bytes
* **ATQA**: Answer to request
* **SAK:**  Select Acknowledge
* Both do not have a tag signature

**hf mf fchk**

* **fchk** will check keys fast, and targets all keys on the card
  + You can also state a dictionary to use

A screenshot of a computer program

Description automatically generated

* The FFFFFFFFFFF normally is an indication that it is not in use.

## HID SEOS - Ensign Work-pass

A computer screen with text and numbers

Description automatically generated

**hf seos list**

* Queries for the SEOS history

A screenshot of a computer

Description automatically generated

### Findings

There isn’t much else you can do, SEOS is fairly secure. They are using a diversified key to encrypt the credential data and it’s diversified to the real UID of the card which is normally hidden, if you use an HID omnikey you may be able to read the raw PACS data (if it’s not an elite keyed card) but it can’t be cloned. If the system/reader was poorly configured a [downgrade attack](https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/hid_downgrade.md) might work, but it’s unlikely.

SEOS and DESFire both pass data encrypted and do not leak the keys so you are very limited in how to exploit them and with SEOS even if you get the key for a card it only applies to that card because the key is diversified to that specific card only.

## NYP Card – High Frequency MIFARE Classic 1K

**hf mf info**

A screen shot of a computer

Description automatically generated

* **hf mf mad** returns nothing substantial
  + **MAD** stands for MIFARE App Directory

**hf mf autopwn**

* Automatic key recovery tool for MIFARE Classic

A screenshot of a computer

Description automatically generated

Note the binary files outputted. This would be important for the following step.

Taking a magic card to be used as a clone, we conduct the following:

* Set the UID, ATQA and SAK of the magic card to that of the card we are cloning

**hf mf csetuid -u <UID> -a <ATQA> -s <SAK>**

Once you have set these successfully, running the following command would take the key binary previously outputted by **autopwn** and match it to the UID assigned to the clone card, taking it as a card it has to restore.

**hf mf restore**

A screen shot of a computer screen

Description automatically generated

* Key A and Key B in Block 0 “fails” to be restored as it is suspected that they are not-writeable, and contain the values of the UID, ATQA and SAK.

This cloned card should be able to successfully perform the same access rights as the original.

* This has been proven with other similarly configured MIFARE Classic 1K cards

## GymBoxx - High Frequency MIFARE Classic 1K with Static Encrypted Nonce

This follows the same steps as the section before, however now we discover that trying to decode the bytes would not work for some blocks, due to a static encrypted nonce.



It seems that the only solution (given that the static encrypted nonce is not a known key from the default key list) would be via sniffing.

A screenshot of a computer

Description automatically generated

In theory we should be able to look at the trace gotten from sniffing the trace between the reader and the card, and determine a pattern in order to get the static encrypted nonce.

* Requires on-site testing, which was not able to do in this scenario.

## Anytime Fitness Keyfob – Low Frequency, T55XX

We use this example to address a Low Frequency case, as this is the easiest use case out there.

**lf search**

We will find out that it is a known tag, T55XX.

**lf t55xx info**

A screen shot of a computer

Description automatically generated

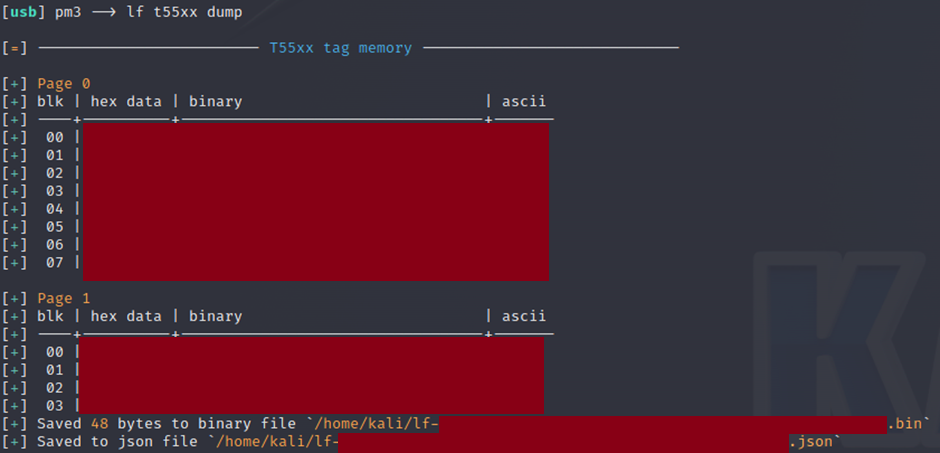
**lf t55xx detect**

A screen shot of a computer

Description automatically generated

Since there is no form of security set (password, One Time Pad), we can just dump the information.

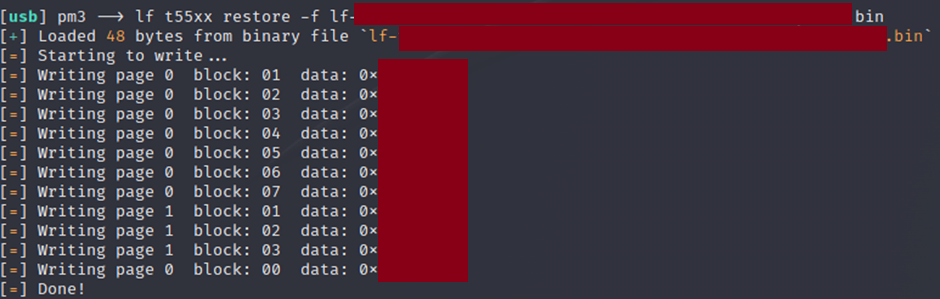
**lf t55xx dump**



We can see that for the T55XX tag, it contains 2 pages of hexadecimal data.

Taking into account the binary file dumped to, we put the new card to clone onto the PM3 and we run the following command:

**lf t55xx restore -f <BINARY FILE NAME.bin>**



# 4. Running Louis’ Scripts

* **cd client/clone\_scripts**

In this script, you should be able to see the following scripts:

A screenshot of a computer

Description automatically generated

The names are per intended. It is meant to automate the cloning processes for T55XX and MiFare Classic 1K cards. Names are per its function.

**Key things to note:**

* Only run the shell scripts (.sh) from command line.
  + You may need to enable permission to execute via **chmod +x** for both the python and shell scripts.
* Follow the steps as per instructions. However, if there is unintended behaviour, or if the script stops executing halfway, refer to the debug.txt file.
  + These could be due to many different reasons (antennae not detecting the card, card orientation is wrong, etc)