Reversing with Ghidra

Hardcoded password fail

Ali Raheem

10th of June 2020

Contents

Introduction

When I buy a piece of hardware I expect to own it and be able to do what I want with it. On the other hand manufacturers want me to buy their hardware and then do what they want with it. When I am feeling particularly cynical this is why I think router manufacturers 'encrypt' configuration files. Of course from their point of view the files are encrypted to protect the sensitive data from being stolen and preventing confusion by having those files open and editable on your desktop.

But staying cynical, the quotation marks around encrypt are on purpose, because of course if the hardware that does the encrypting and decrypting is in my hands then that encryption is no more than a challenge to take the hardware apart and find the keys.

Even if you don't have the device in question read on and you can grab all you need to follow this post-mortem for free.

The Quarry

One of my favourite things is looking around eBay for cheap junk, my favourite thing is to buy that junk, play with it for a day or two and then chuck it in the junk pile. When that piece of junk challenges me to hack it all the better. That's what happened when I bought a TP-Link TL-MR3020 V3 (I got it for a song at £18), ostensibly I wanted to use it tether my phone and get a reliable internet connection. But finding the configuration backup encrypted was just such a challenge to hack it that I can't resist.

Let's get crackin'

Lucky for me TP-Link provides firmware on their website for download making it easy to poke around in it's innards before the router even arrived.

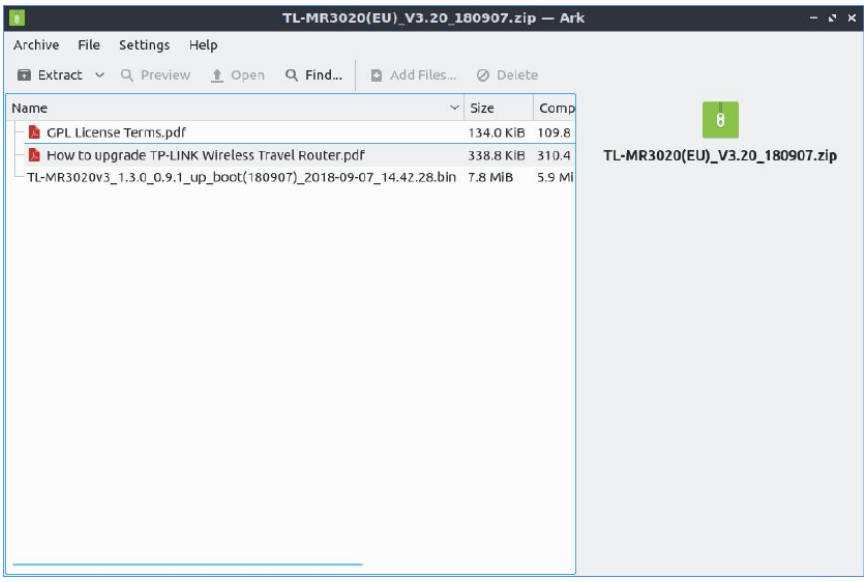

Figure 1: The manufacturers website helpfully provides the fireware for download

Downloading the file and unzipping we're greeted with a couple PDFs (I've never even read the titles) and what we're interested in a file ending with a .bin extension. But we should take note that they exist incase we get stuck, heck maybe one of them describes in perfect detail the format of the config files?

These .bin files are typically just smooshed together smaller files. The most popular tool to explore such a file is binwalk. Binwalk steps through the file and can work out the types and sizes of embedded files. If we use the `-e` option we can even have binwalk pull these chunks out for us. It'll automatically unzip or otherwise unpackage these subfiles. It worth noting the output of binwalk as if we want to repack the firmware (for flashing a modified version) matching the offsets (at least) and settings will maximise our chances of success.

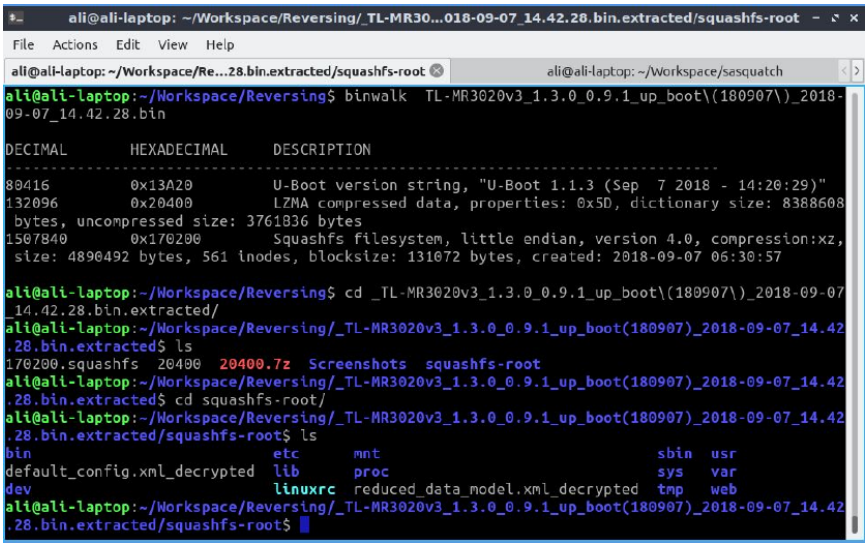

Figure 2: Binwalk is the swiss army knife of binary deconstruction, here we used it to detect and extract filesystems from the manufacturers provided firmware

Unzip for me

In our case the most interesting bit is `Squashfs filesystem`. Squashfs is a popular choice for engineers to package a filesystem into such a firmware file. Using squashfs you can take a directory full of files and folders and produce a single binary. Binwalk has helpfully unpacked it into a folder for us. Moving into the filesystem directory we find what looks like a bogstandard linux directory. With the right tool we could even run these some of these binaries, we could `sudo chroot . bin/busybox sh`, or go for something more elaborate like firmadyne which would let us emulate the entire system. We wont need any of that here, well so far.

Personally I like to look at `etc/` directory first when I get into a victim filesystem. This is where most system configuration files are stored. John the ripper, makes short work of the password file (`etc/passwd.bak` found here, in this case the username is admin, password is 1234). Another point of interest are the boot scripts in `etc/init.d/`. Looking at the boot scripts we see that `etc/passwd.bak` is copied over `etc/passwd` the standard password file location.

It's also often the case that the system will start out in a far less secure state than it runs in. It's common to see `telnetd` start only to be killed in a

later script. Giving a brief window where one could login and make changes.

Things get interesting

But poking around the filesystem and something piqued my interest, `etc/default_config.xml`. It's also not recognised as xml, but "data" by the `file` command.

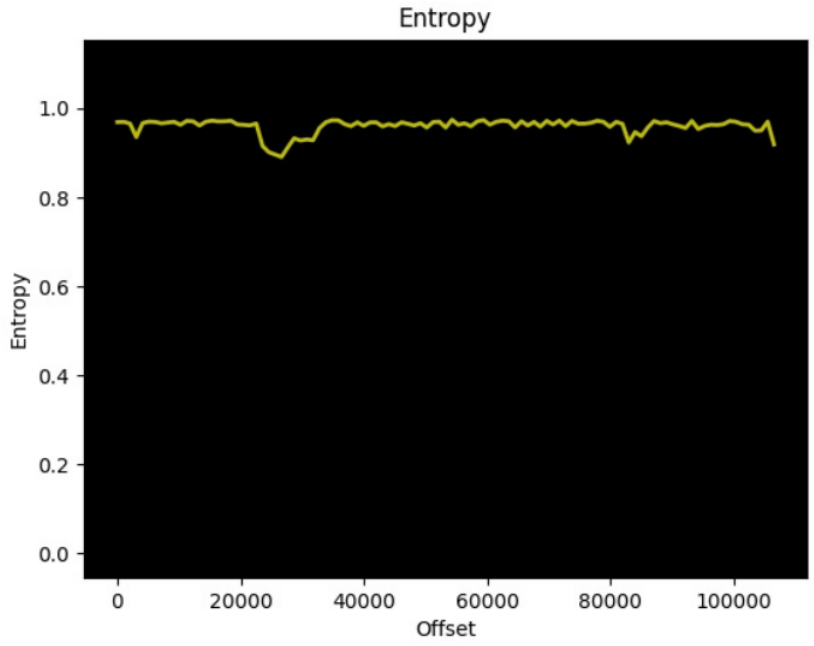

Figure 3: Binkwalk will let us calculate the entropy (a kind of measure of information) per byte, encrypted or compressed data has high entropy. Most files will have varying entropy. Here we see a few dips but otherwise constant high entropy.

Using the entropy function for binwalk we can see that it's relatively high and relatively stable `binwalk -E etc/default_config.xml`. This is a good sign that the file is encrypted, or heavily compressed. Lets trust that binwalk would have identified if it was just compressed, and that it's unlikely something so small would have such an effective compression scheme.

All that said, the dip in the entropy suggests some structure in the file at offset 20000 which has made it through the encryption. If I had to guess this was a long run of zeros in the data. This could be a sign of a less than

stellar encryption scheme.

Target Acquired

So here's a candidate for us if we want to be able to edit and upload custom configurations. Clearly it's still encrypted and thats a problem, but we **know** that it must be encrypted and decrypted on our hardware and it's certain the software to do it is in this filesystem we've now got our hands on.

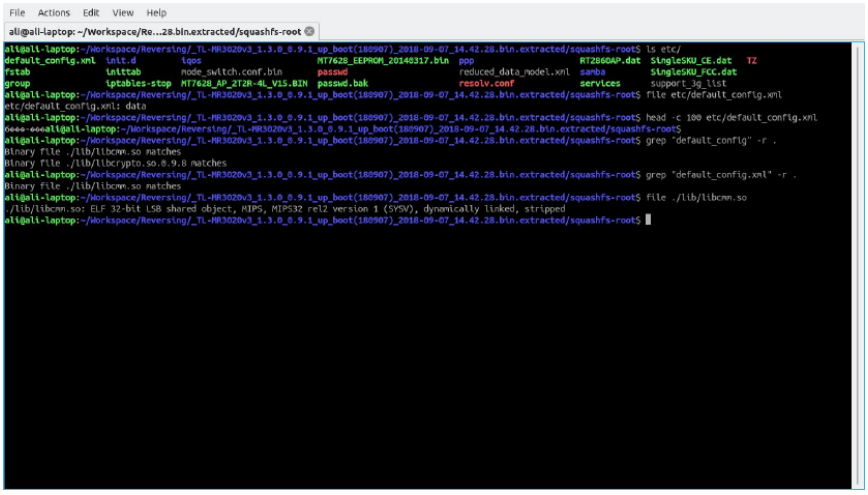

Figure 4: `default_config.xml` doesn't appear to be an XML file, and it's referenced in a binary.

Searching the filesystem for mentions of `default_config.xml` I was expecting to see a script somewhere but instead I got a match in a library `lib/libcmm.so`. Things just got interesting! I've recently discovered Ghidra an NSA backed dissembler/decompiler - I know I know, yes I am suggesting you download and run NSA software.

Ghidra is now an opensource project hosted on Github, it's powerful and I've found it to be incredibly user friendly compared to other options such as radare2, Hopper, rizen, binary.ninja and finally the famous IDA Pro. I am sure there are many, many more options out there and you can mix and match for example some tools now use Ghidra's decompiler as a plugin.

Let's give Ghidra a twirl on `lib/libcmm.so`. Grep already told us that it contained the string `default_config.xml`, but where? First we start a new project `ctrl + N` and import the file with `I`. Double clicking this imported

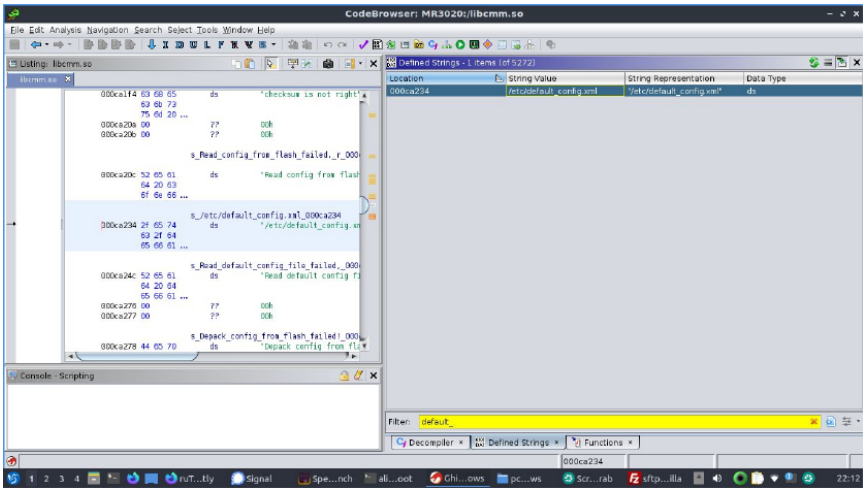

Figure 5: Searching binaries for strings can be helpful if they are file names, error messages in particular. They can narrow down the search in big files.

file we are transported to the main ghidra window once we accept its offer to analyze the binary.

Searching for defined strings in the binary we get a single hit. We can double click it in the listing view, we can then right click the label to search for references to that address in code. Another double click and we are taken to the code making use of it.

One of the most features I find so helpful in Ghidra is it's decompiler which will produce a pseudocode (usually valid C) representation of the binary. Often this will even compile directly if we want to see the code in motion. You might still have to look at the listing to understand precisely what's going on though.

The process of compiling is necessarily lossy and the main victims are variable names and types, even if debug data is present and we have function names. Lucky for us there has been no attempt to obstrufcate any of the code in this library and we can guess at types and what function the serve. Here we find a function called (helpfully) `dm_loadCfg`, I've relabelled (`L` or `right click > relabel`) a few variables and made guesses at their types (`ctrl + L` and again in the right click menu also).

I do this almost automatically as I try to understand the code, ghidra will update the decompilation as you do so, theres a positive feedback here: As ghidra uses your types to update its understanding of the function, you can add more types and labels. You can also experiment, types benefit most

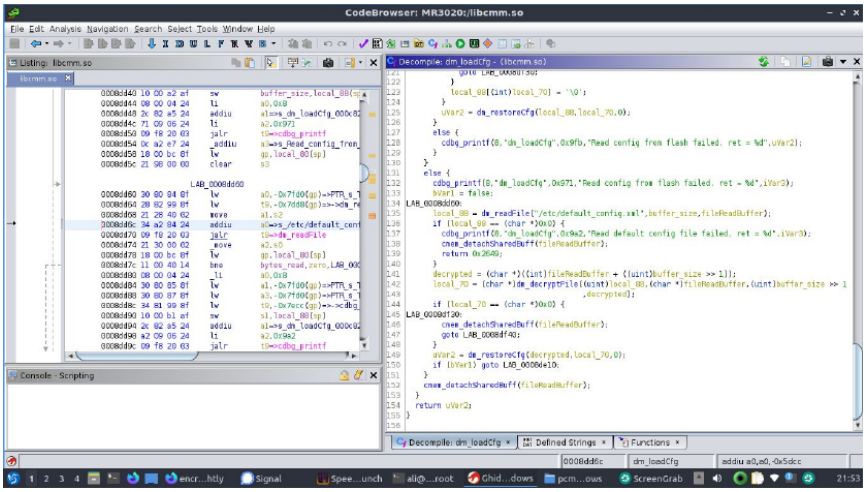

Figure 6: Ghidra's interface might be intimidating at first but its really clear and uncluttered I find. Here's a typical view of some decompiled code.

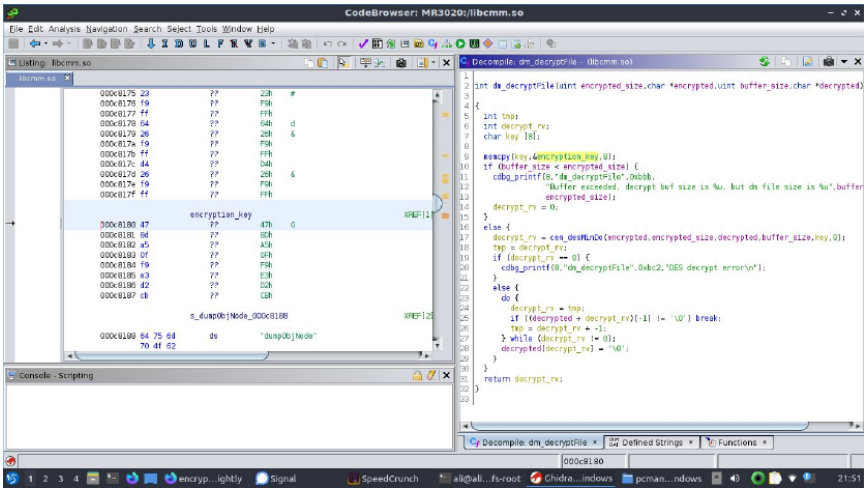

Figure 7: Ghidra's decompilation is its super power even assembler experts will struggle to take in the full picture because there are so many instructions per programmer idea - this is known as "expressiveness". C is much more expressive than assembler.

from guessing and seeing if ghidra can suddenly make sense of the movement of data.

The correct types in particular can dramitcally clean up the code. Ghidra can end up having to use obtuse types to carefully make all the data line up nor does it neccesarily know if a integer is signed, unsigned two 2 bytes, 4 bytes... If you're not sure what a particular type is circle back around hopefully once you've added the types you are sure of ghidra can help you with the rest.

It's not long before we happen on `dm_decryptFile`.

End game

Of course with a function named `decryptFile` we couldn't possibly not double click it and take a look. And I've again cross referenced `dm_decryptFile` with `dm_loadCfg` we can easily relabel and retype some of the variables to make it simpler to understand whats going on.

Quickly we find something else very intriguing `cen_desMinDo` and an error string. Helpfully, this error string warns us that there has been an error with `des` after checking the return value. Error strings can feel like comments for decompiled code - like comments they range from distracting and harmful to the cracker to illuminating. Well, `cen_desMinDo` is just a thunk but the parameters are very enlightening. We see that `cen_desMinDo` takes a few arguments but we find one argument unaccounted for. It was copied into a buffer and then passed into this decryption functon. This is exciting as it is just what we'd expect from a decryption key! If `cen_desMinDo` handles the decryption it either must be passed the key (most likely) or load it from somewhere which is very **unlikely** given it's an external function.

We can follow it to see where it came from it's derived from to see that it's 8 bytes loaded from a global variable. Double clicking it and ghidra takes us to the dissembly. `478DA50FF9E3D2CB` jackpot.

This just has to be a key, and although `cen_desMinDo` is external (likely libcrypto we saw above), the config file is encrypted it's almost certainly using a block cipher, if it fails it complains about DES, I'd bet you peanuts to pounds it uses a DES block cipher.

Openssl is to crypto what binwalk is to firmware (well that is somewhat under selling openssl!). Openssl is a beast of a software and I've always found the documentation lacking - often you need to look at code to work out how to use it. Luckily, soon after trying to decrypt with openss we find that the first 8 bytes pop out `<?xml ve` this is tantalyzingly close! Spend

some time playing with the openssl settings, quite quickily I identified the correct mode `des-ecb`. The final command I used was is `openssl enc -d -des-ecb -K 478DA50FF9E3D2CB -in default_config.xml` out pops valid XML!

Conclusion

Finally, we have the file dumped in it's XML glory. Googling the key above we find that not only did TP-Link use it as a hardcoded key for all routers of this model (likely so that they would be able to provide support with custom config files). But, inexplicably, they used it in other models too! That is the same exact key, not just the same scheme!

Helpfully someone has produced a script to automate encrypting/decrypting these configuration files. They also documented a code injection vulnerability using modified configuration files.

The obvious question is, why was this so easy? Well, for the main part it's probably because the engineers designing the firmware took no evasive/obfuscating actions. That and the above is presented without the many many dead ends I went down, but those deadends are just as valuable educationally as getting the right answer. They often also improve your understanding of the firmware you're working on so don't be disuaded by them!

This is just an introduction to some of the tools of a reverse engineer. But also a real life case, no contrived "crackmes" which have their place as educational tools. Hopefully after this you will feel confident in using `binwalk`, `ghidra` and even `john` the ripper.

The real question is if I'd tried this before buying the router, would I have been more or less likely to get it?

Formats

- Web
- Text
- PDF