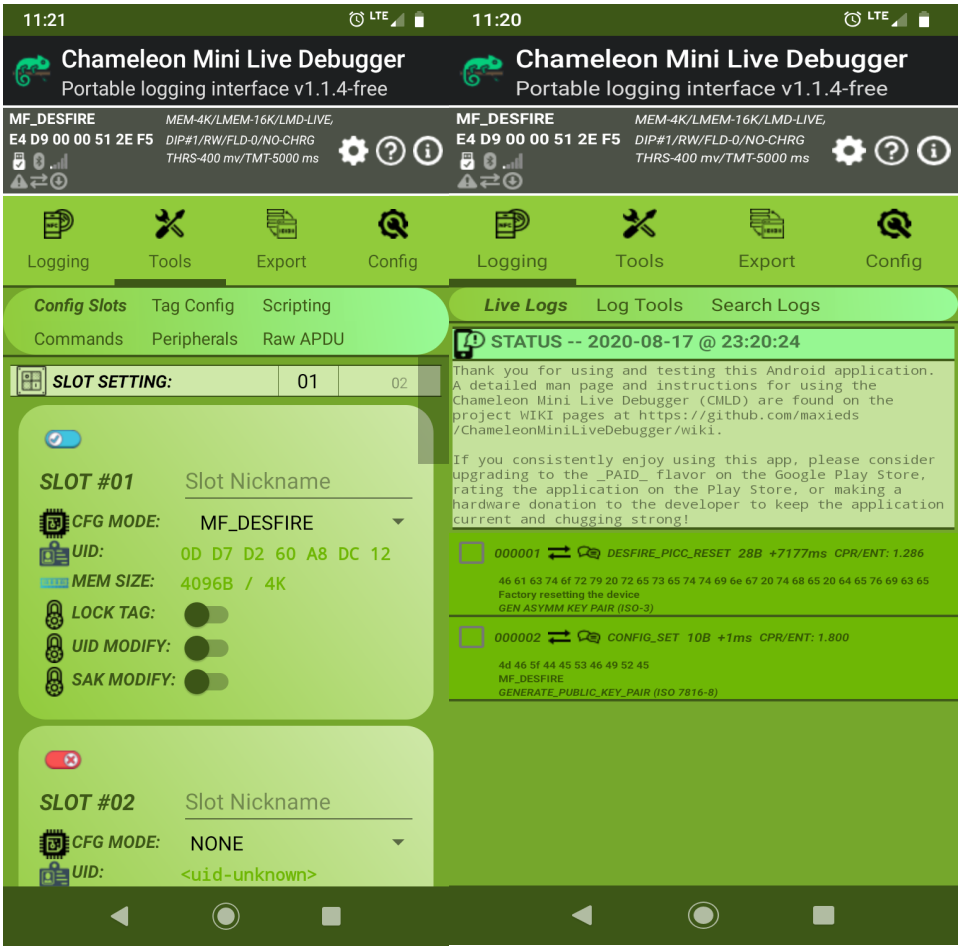
Proposal for funding of a functional DESFire NFC firmware stack for Chameleon Mini devices

I have gradually become interested in developing software to log and control the NFC pen-testing boards called Chameleon Mini devices since I arrived at GA Tech in 2017. My initial work with these boards involved writing an Android device based logger and controller for these boards hooked to a Droid phone over serial USB cable. As the technology has advanced, wireless Bluetooth connection support from phone to device is currently in the works (this week, thanks to a generous donor to my open source software application: https://github.com/maxieds/ChameleonMiniLiveDebugger – screenshots below):



The Chameleon Mini devices are fairly affordable NFC pen-testing and logging devices compared to other commercial hardware options. Additionally, they have well tested, advanced firmware written in C for the onboard AVR devices that can be compiled with standardized `gcc/g++` toolchains specialized to this platform. I have worked on some freelance software work outside of my studies at GA Tech in 2019 to incorporate more cryptographic routines and

encrypted XModem-type uploads to these Chameleon devices. As I have started working on a newer, more challenging project with the firmware sources recently, my ability to construct software for these AVR-based boards has only become more confident and well practiced.

Background for the DESFire firmware stack

When I started logging bits on the campus door readers around the winter of 2017 with the Chameleon devices, my primary aim was to write an Android phone based application capable of replacing the campus BuzzCard (with its integrated DESFire chip). This task is, as it turns out complicated for a number of reasons, not the least of which is that the Android NFC stack is not low-level enough to directly communicate with the readers directly. The logging information I obtained with these experiments was kept on a private GitHub repository until I unearthed it later this spring in 2020.

The repetitive data it contained, including multiple swipes and door authentications with the BuzzCard to on campus door readers indicated to me a clear vulnerability. In short, the door authentication scheme used to secure many on campus buildings and facilities was bypassing the sophisticated 3DES/AES cryptographic mechanisms on the DESFire tag housed by the BuzzCard itself. This is a ***HUGE*** vulnerability as it simplifies the task of "spoofing", or illicitly cloning any user's BuzzCard credentials by a would be attacker substantially. I reported my data and suspicions (without working exploit code to demonstrate the vulnerability at that time) to OIT and campus authorities. I was willing to cooperate with them easily by vowing not to make my sources for the actively developed DESFire stack for my Chameleon devices public until after the end of the Summer of 2020 – after which point, I was assured the problem would be resolved by the in-house GA Tech security specialists. I have kept my word, but still intend to make good on my promise to the open source community of Chameleon Mini users to deliver a functional DESFire stack for these devices – and sooner, rather than later.

Challenges of the project

The development of the DESFire firmware stack for the Chameleon Mini devices is complicated for a number of reasons. First, the software, memory, and enhanced embedded cryptographic functionality one needs to develop to even start on the spec for support for DESFire tags is taxing, and hard to effectively optimize on the memory limited AVR chips available on the Chameleon devices. Secondly, the specification for the variants of the DESFire tags is mostly closely guarded by large corporations like Philips or NXP that require NDA signatures and substantial offerings of monetary payment to even gain access to some of the most rudimentary data about the implementations, e.g., APDU command codes for essential core

operations with these tags, storage, and exact procedures for how to exchange data (under which encryption scheme, and with what set of working permissions). Nonetheless, some documentation has become freely available about these specs, and the wide availability of mostly Java-based Android HCE implementations for these DESFire tags has made it approachable to forge an implementation for the Chameleon devices. These DESFire tags offer many commands, including support for a subset of ISO-standard compatible command codes. The spec also offers support for several file types, nested permissions on files, multiple crypto keys (3DES/AES++), and other more nuanced features that set the task of a successful implementation of this DESFire stack apart from more simplistic emulation schemes, such as for MIFARE Classic and MIFARE Ultralight tags (among others, already readily support in the Chameleon firmware sources).

In other words, this is a very challenging coding and applied crypto project that will to the best of its suitors (e.g., me, and cohorts) win bragging rights, security conference talk appearances, and possibly even papers depending on the sophistication of parts of the implementation. This feat has been attempted before, though not recently, but has since been abandoned by the original fork maintainer. It also happens to be the single most requested user feature for these devices since I started tinkering with the Chameleon mini hardware years ago. I (as it happens), opened my mouth earlier this year and asserted that I will make this happen – it's about time after all, and without saying this publicly, I know how to write this type of code forwards, backwards and upside down, as it were. I intend on delivering a working DESFire firmware stack to users as I promised, and as I typically expect people to take me at my word. This is a matter of pride and bragging rights --- not just for me, but for fellow funders that want to see this type of applied crypto project succeed from an academic research perspective.

Funding needed (and why)

I am close enough with my current code base, but at a critical gap in the testing and verification process of my code – namely, I require funding for better testing hardware to facilitate development to the point where the resulting firmware is stable enough for user (non-developer only) testing phases. My active problem is that the lower end USB stick readers (priced at around $30-50 USD on Amazon) are unreliable, and will only once in a blue moon react well to input from the Chameleon devices I am using to test my software. I believe that with a much more advanced, refined, and importantly high quality NFC device called the Proxmark I will easily be able to run through the rudimentary testing phase and begin to do more interesting things with the implementation – both on the software end dealing with authentication and file storage, and also with the end-user applications like torture testing physical authentication systems on campus – only, of course within the confines of my ethically

minded “white hat” hacker role here. A Proxmark device runs at approximately ~$320, which is way out of my personal budget and price range, but well within reasonable funding bounds for applied crypto and (loosely) algorithms research at a large university.