January TATER Monthly Status Report
1-30-2018

Contents

1 Problem Statement

Our goal is to design a method to monitor and characterize the electromagnetic emissions of a microprocessor during boot to determine if foreign code has been injected. Limited by the types of instructions that have the most impact, the data acquisition system and analysis algorithm will be modeled accordingly. The finished product will consist of a system which can physically capture electromagnetic emissions with a custom antenna, collected using an amplifier paired with an ADC acquisition platform, and process collected data using a custom algorithm to create an EM profile of a processor's boot.

2 Hardware Summary

Hardware will be used to measure the EM signals produced by the execution of various instructions. In order to record consistent results and allow for accurate comparison between captures, the antenna position needs to be consistent. In addition, we need to ensure that no external noise from lights, other power sources, etc. interferes with data collection.

2.1 Achievements

- The target processor now operates at 16MHz. This produces a shorter boot time capture, better probe reception, and less need for algorithm preprocessing. Because instruction peaks are represented based on the amplitude of the signal, this also does not change the impact on the how the bootcode verification algorithm functions.
- A housing for the processor and antenna was constructed to prevent variations in signal collection while building the baseline. The metal case provides shielding from external sources. Attached peripheral switches were added to change the bootcode configuration during runtime, which is more efficient than loading from a file and achieves the same result. This is shown below in Figure 1.

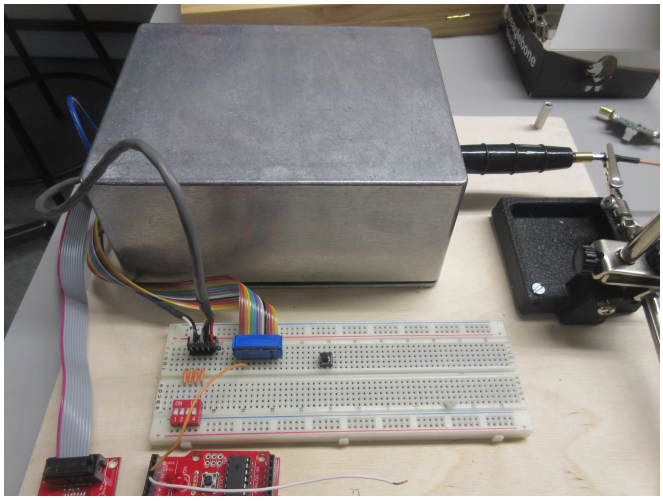

Figure 1: Housing and shield for target processor

- An external microprocessor (separate from the target) triggers a reset of the target processor when it receives a command from the host computer. The same microprocessor sends a trigger signal to the capture device in the same instant, in order to begin data capture. At the moment this is just the MSO700B oscilloscope, in the future data will be collected using the KC705 board in conjunction with the provided ADC. This setup is shown in Figure 2.

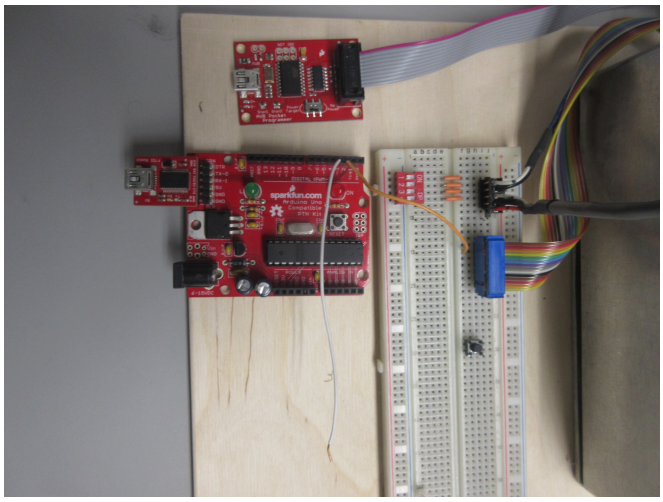

Figure 2: Microcontroller reset and trigger controlled from host computer

- Hardware captures were taken using the oscilloscope of various modifications to the bootcode. This includes: Set of multiple captures from the same segment of code. Set of multiple captures from various modifications of the same segment of code with heavy modification, moderate modification, and slight modification. This includes removal, insertion, and modification of instructions. This applies directly to the way configuration settings and foreign code detection will be analyzed. Comparisons of various instruction variations caused by similar runs are shown in Figure 3 on the next page.

2.2 Purpose / Next steps

Although the oscilloscope capture works for short time sequences < 10ms, we need to capture a full boot sequence which will be possible with the KC705 onboard memory. We will need to adapt the reference design to suit our needs, which will entail constructing the internal triggering and data streaming, in addition to the hardware driver running on the Linux machine. Once this is complete, the bootcode can be fully verified for foreign modifications.

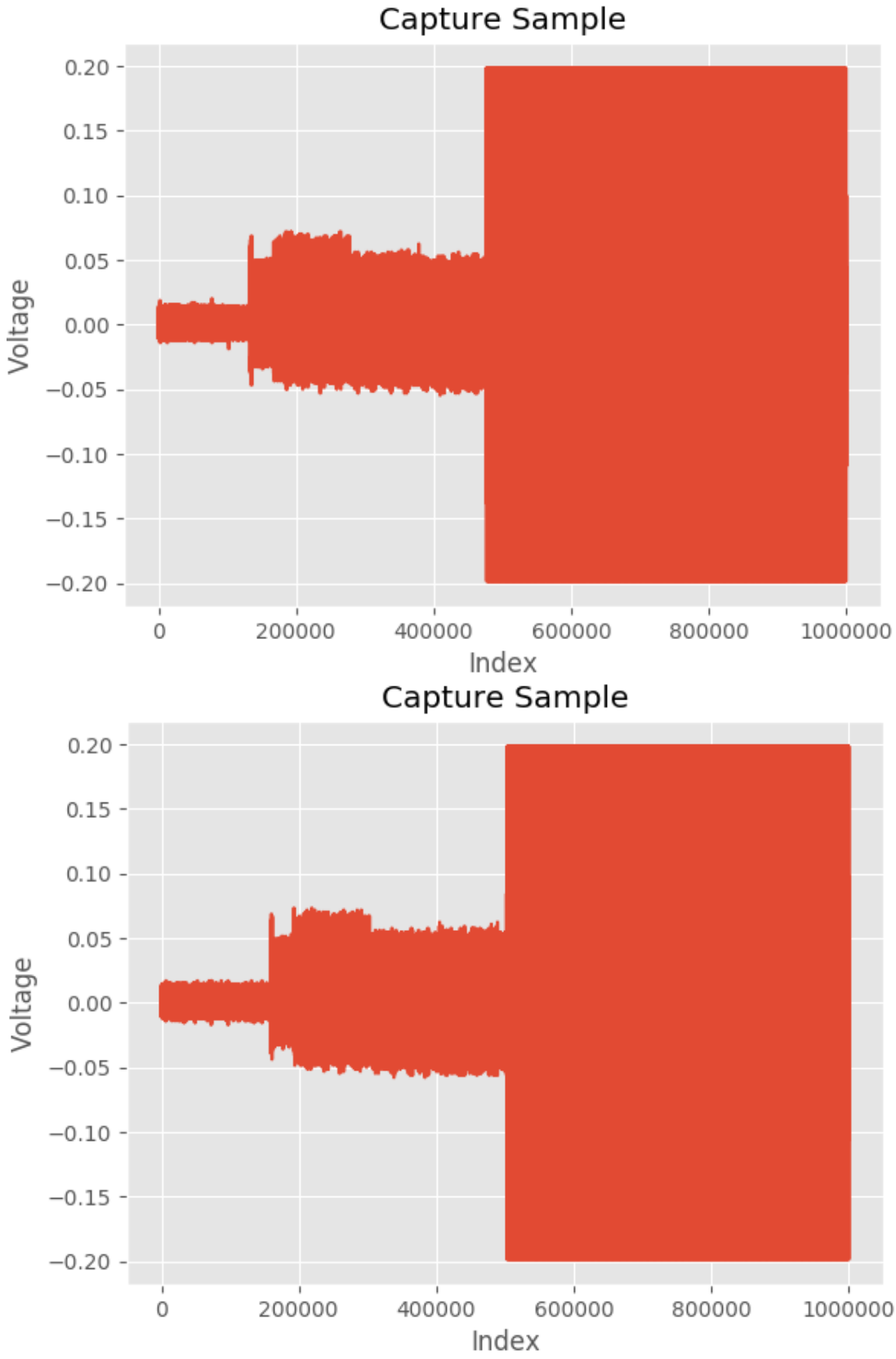

Figure 3:
These figures show the same code running on the target board. There are slight variations in the collected signal, which will need to be handled by the software detection. The large increase on the right is caused by peripheral interactions, which will be used to detect the end of a boot sequence. Captures were run at 2GSPS to gather as many points as possible

3 Antenna Summary

Many antennas were designed in the antenna design program FEKO this month, each of them having the quality in terms of dollars of roughly a $7 antenna that could be purchased from any RF site, unworthy of mention. After struggling to come up with a design that not only met our large bandwidth need of 1Hz-3.5GHz as we are capturing all potential emissions from the microprocessor, but also performed better than our probe in terms of gain, we started looking through professionally made antennas for design considerations. The following antennas are ones that stuck out as being perfectly qualified for our needs (if a little large in volume), while the one after that is a great example of the quality of antenna that is being custom designed in FEKO.

- This first antenna, a Pulse Larsen Outdoor Multi Band designed for utility boxes and smart metering could meet all our specifications, as it had a fantastic bandwidth that was large enough to capture an ideal amount of power emissions at a range of different frequencies, and an even better isolation than our probe (by about 20dB). The downside to this antenna is its cost of $50, though the quality and performance of its output really make it a design to consider. The antenna's spreadsheet can be found here:

  https://www.mouser.com/catalog/specsheets/pulse_W4165XXX.pdf

- The second antenna is a simple patch antenna, which has a worse isolation than our probe (around 25dB at the best of times) and has a bandwidth that would be completely unacceptable for our data acquisition. The upside of this antenna is its price, coming in at $7, but it has too poor of a performance to be a real contender with the first antenna. The antenna's spreadsheet can be found here:

  https://www.mouser.com/catalog/specsheets/SL_PA251615025SALF.PDF

3.1 Moving Forward

- We intend to have a working antenna mounted on our board by the 10th of March at the latest.
- Before making any decisions about buying a professionally made antenna, we would like to try and emulate this first antenna by building it from scratch, and see if we can get an isolation that is at least as good as our probes.
- Failing this, we will try and see if we can find a cheaper antenna to possibly purchase, that would still be efficient enough that our data is comparable to that of the probe data, while still attempting to implement a sufficient design of our own conception.

4 Software Summary

Implementation of various sections for analysis of the collected data is underway. Below are descriptions of software algorithms that will simulate, preprocess, and cross correlate the data to

meet the goal to determine whether the monitored code has been modified.

1. Preprocessing algorithm: This algorithm filters out noise that is generated in between execution of instructions by the processor
2. Establish a baseline: Collect several sets of data from repeated runs of the program that is being analyzed, and average these together to establish as accurate of a baseline as possible. Once the basic correlation algorithm is working to specification, allowances for optional configurations will be added to the algorithm and included in the baseline.
3. Cross Correlation algorithm: This algorithm applies cross correlation between the baseline data and the actual data to find any injected or missing data. We plan to achieve windowing (apply cross correlation over a set of points) over the preprocessed data, where each window will have its own correlation value.

4.1 Simulation Algorithms

We have also been using two additional algorithms to simulate data in place of data acquired from hardware captures, for the purpose of testing code for analysis and preprocessing.

1. Simcode algorithm: This algorithm is a simulation of the data. It creates a file of random data.
2. Simwave algorithm: This algorithm is a simulation of the data to wave data. It takes the output file from the simcode algorithm and creates peaks depending on the values.

Implementation Language Choices: Both Simcode and Simwave have been implemented in C, but the preprocessing algorithm is written in Python. Going forward, the cross correlation algorithm will also be written in Python as it offers libraries of easy-to-use, optimized correlation functions. In addition, Python is very quick to implement and modify and thus ideal for prototyping and proof-of-concept type work. Once a working prototype is complete, we may design a second correlation algorithm in C to improve the runtime.

4.2 Achievements

- We discussed the schedule and plan of attack for this semester. We plan to have a simulation of all algorithms, including the cross-correlation algorithm for the detailed design review (2/7). When we write the code for the algorithms, we plan to write unit tests to go along with the code.
- Preprocessing Algorithm: The preprocessing algorithm filters out noise collected from the processor in between instructions. Noise is distinguished from actual instructions by its very low amplitude and is far more prevalent than the significantly larger waveforms generated

when actual instructions are executed. The python program written to filter out noise is based on the assumptions that noise occurs significantly more often and is of lower amplitude than all relevant data points.

- Correlation Algorithm: So far, we have identified possible Python library functions to use for correlation. Among these are `numpy.correlate`, which looks like a promising first algorithm to test on the capture data that will be gathered soon, `numpy.corrcoef`, and `scipy.signal.correlate`.

4.3 Next Steps

We will use data obtained from hardware captures to develop and test a correlation algorithm. This process will include using captures from the same boot sequence to identify the likelihood of false positives. We will also compare code that has been changed to varying degrees to assess the correlation algorithm's accuracy in identifying modifications.

Once the basic correlation algorithm is working as desired with a high detection rate and a low false positive rate, we will begin working on identifying legitimate configuration changes.

5 Risks

Main risks, severity, impact, and how we are mitigating the risks:

5.1 Risk 0

Data acquisition, specifically materials required.

`Impact`: Gathering of profile for execution.

`Severity`: High

`Status`: We have received the KC705 and are awaiting the delivery of the ADC board. However, we will need to locate a suitable computer in order to program the FPGA.

5.2 Risk 1

Antenna specifications are highly dependent on processor speed and type.

`Impact`: This might change the antenna size and efficiency as the center frequency and the bandwidth of the signal will be modified.

`Severity`: Low

`Status`: Using a shielded antenna as close as possible to the processor in a way that it does not interfere with wired peripherals is ideal. This entails a custom housing surrounding the processor to mitigate external noise.

5.3 Risk 2

Unable to identify legitimate configurations in a single run.

`Impact`: May have to run comparison algorithm repeatedly for each allowable configuration. This would significantly increase the algorithm's runtime.

`Severity`: Medium

`Status`: We have collected many different samples of hardware captures in order to begin testing of the various methods of analysis. Possible routes include averaging of the collected signals, sensitivity controls, and similar.