Artifact Evaluation (AE) abstracts

Joe Loughry
Netoir
joe@netoir.com

Kasper Rasmussen
University of Oxford
kasper.rasmussen@cs.ox.ac.uk

1 Git repository

This is a `git` repository of source code used to collect and analyze data from experiments in the paper.

1.1 Overview of experiments run

Two runs of experiments were done on LEDs and ESD protection diodes at 405, 532, 650, 780, 808, and 980 nm laser radiation on a variety of 5 mm visible LEDs and glass enclosed small signal diodes of the type used for ESD protection of I^{2}C bus connections on printed circuit boards.

C++ source code for the Arduino microcontroller that controls the stepper motors—raster scanning each laser in turn over the target area and measuring the SDA bus voltage (0–3.3 volt) with the Arduino's analog-to-digital converter (ADC) at each point in the raster scan, and writing it to a file—is in `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/LEDs/code/Arduino` for LEDs and `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/diodes/code/Arduino` for ESD protection devices.

1.2 Raw data

Transcripts of the output of these programs are in `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/LEDs/data/raw_data` and `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/diodes/data/raw_data`, respectively.

Gnuplot scripts to extract and plot the data are in `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/LEDs/data/` and `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/diodes/data/`, respectively.

R code for statistical analysis of the effect of elliptical beam axis rotation is in `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/diodes/data/beam_rotation.r`.

2 Installing and running the software

C++ source files are intended to be used with the Arduino IDE, available from `https://www.arduino.cc/en/software`.

2.1 Required libraries

To compile the Arduino code, once you have the Arduino IDE running, you need to install a couple of necessary libraries: the *Adafruit Motor Shield V2 Library*, and the *Adafruit PWM Servo Driver Library*, both of which are available from the Library Manager in the Arduino IDE, by searching for "Adafruit". This is the recommended way to install Arduino libraries now.

The *`Wire`* library should already be available by default. Also in the Arduino IDE, if asked to set a board type, just use "Arduino Uno", the simplest type of Arduino AVR board.

For further information on the Adafruit libraries, including step-by-step instructions for installing the library, see `https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software` and `https://learn.adafruit.com/16-channel-pwm-servo-driver/using-the-adafruit-library`.

2.2 The `basilisk` toolchain

The Arduino code generates the data processed by Gnuplot. Gnuplot is available from `http://www.gnuplot.info/`. Gnuplot scripts are provided with a `plot_all_data.sh` shell script in lieu of a `Makefile`.

Project IceStorm was successfully used on both macOS 14.4.1 and Ubuntu 20.04. To install the FPGA toolchain on Ubuntu, follow the instructions at `https://clifford.at/icestorm#Where_are_the_Tools_How_to_install`. See Appendix A for a list of dependencies.

2.3 Verilog and FPGA

The process for taking a Verilog description of a logic circuit and making it run on an FPGA is notionally:

yosys to take a Verilog source file and synthesize a logic circuit design.

nextpnr to takes the logic circuit from `yosys` and place & route logic gates to fit the FPGA chip's internal interconnection fabric.

icetime to simulate the design produced by `nextpnr` and check for timing hazards.

icepack to produce a binary image that will be loaded into the FPGA.

iceprog to upload the image into the chip.

After these steps are done, the FPGA chip acts like the specified logic circuit, until a different description is loaded.

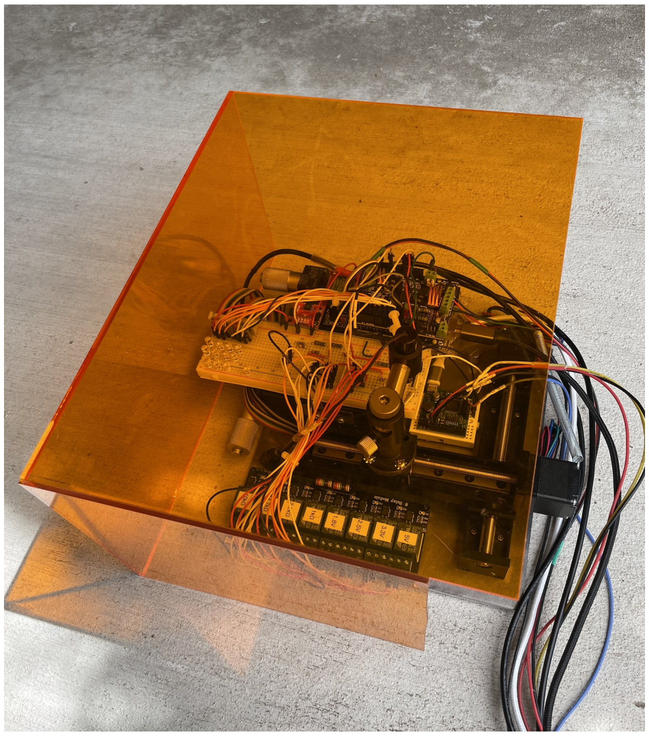
Figure 1: Raster scan apparatus used in experiments. This photo will replace the one in the final paper, as it shows the safety shield. A schematic of this device is Figure 5 of the paper.

3 Hardware

The photograph shown above (Figure 1) will replace Figure 6 in the final paper, as it shows the laser safety shield in place.

The experimental apparatus consists of a pair of stepper-motor–driven linear actuators configured for *x*–*y* motion to raster scan a focused laser over the target area, approximately 1 mm^2–25 mm^2 depending on whether the target is an ESD protection diode or LED. A schematic is given in Figure 5 of the paper. To switch in different values of pull-up resistors and vary the voltage on the I^2C bus, a bank of relays is used. Bus voltage is varied from 1.8 V to 5 V to mimic LVCMOS, CMOS, and TTL logic families, and pull-up resistor values varied from 1 kΩ to 10 kΩ in logarithmic steps. Sixteen runs are done for every combination of LED wavelength and laser wavelength, unless testing (for example) the effect of elliptical beam axis rotation.

3.1 M5 CPU

M5, shown in Figure 2, is a minimalist CPU intended not so much to show the practicality of the attack against real hardware (Lattice Semiconductor iCE40-HX8K FPGA evaluation board) but rather to highlight certain unique difficulties of the attack, beyond the obvious ones like aiming and focusing.

This is a 4-bit computer with an accumulator that is visible on the front panel. (Visibility is key to establishing a phase lock on the internal state of the CPU.) It has a very simple instruction set to make feasible the reachability analysis in Figure 15 of the paper.

The particular FPGA chosen is not significant; the attack should work on any CMOS logic family, as explained in the paper. But most FPGA development tools are proprietary, closed source, and expensive; in contrast, a completely open source toolchain—Project IceStorm—exists for the Lattice iCE40. Every step in the process is visible, down to a plain text file containing an array of ones and zeros that is what is actually uploaded into the interconnection fabric of the FPGA.

Project IceStorm: `https://clifford.at/icestorm`.

Verilog source code for the CPU is available at `https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/M5/code/Verilog` and C++ source code for the attacker is available at `https://github.com/jloughry/basilisk_artifacts/tree/main/experiments/M5/code/Arduino`.

`Makefile`s are provided to synthesize, place & route, simulate (to check for timing hazards), and upload to the FPGA.

3.2 How the attack works

M5 runs the microcode for each instruction on a sixteen-step cycle. Figure 3 shows a typical instruction, opcode mnemonic STA, which stores the value currently in the accumulator register to a specified memory location.

The attacker needs to know a lot about the CPU and the program that is currently running to perform the attack suc-

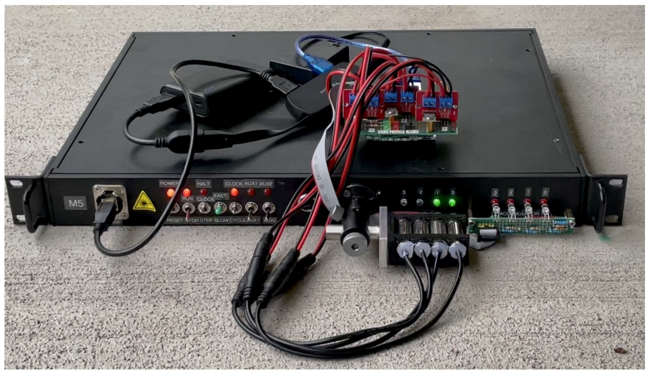
Figure 2: M5 CPU. A schematic of M5 will be included in the final paper, an oversight noted by the reviewers.

cessfully. Firstly, the attacker needs to establish a phase lock on the internal state of the CPU, and from it to accurately measure the cycle time, because the entire attack is predicated on cycle counting.

Here, phase locking is accomplished by watching the accumulator display for changes, because the display always changes at a known microcode cycle. From the direction and magnitude of the change, the attacker can deduce what instruction was running (for example, INC or DEC if the accumulator value changed by one, or STA if the value changed by more than one). From the time between changes, the attacker can calculate the cycle time by dividing by the number of instructions executed between changes and looking up the cycle time of each instruction, which may be different.

All this could be accomplished a different way simply by watching the bus LEDs. We do it by means of the accumulator simply to illustrate the general principle that the attacker is not necessarily attacking the same LED as the one being watched.

After the attacker has established a phase lock and measured the cycle time, the attack proceeds by counting cycles into a predicted part of the fetch–execute cycle and firing the lasers at the instant when the desired value is known to be on the bus. Typically, the laser fires more than once during a particular instruction; for example, once to change the opcode, again four and a half cycles later to change the memory address, and five cycles after that to change the data being written to memory.

We did not collect any data on the probability of the attack being successful, because we found it to be completely reliable under the conditions we set up. The lasers are bolted in position, aimed at the bus LEDs from a range of 2 cm, because it removes an independent variable (aiming error) from the experiment.

3.3 Lasers

The lasers used in this experiment were 405 nm near-UV diode laser modules of unknown power rating that were

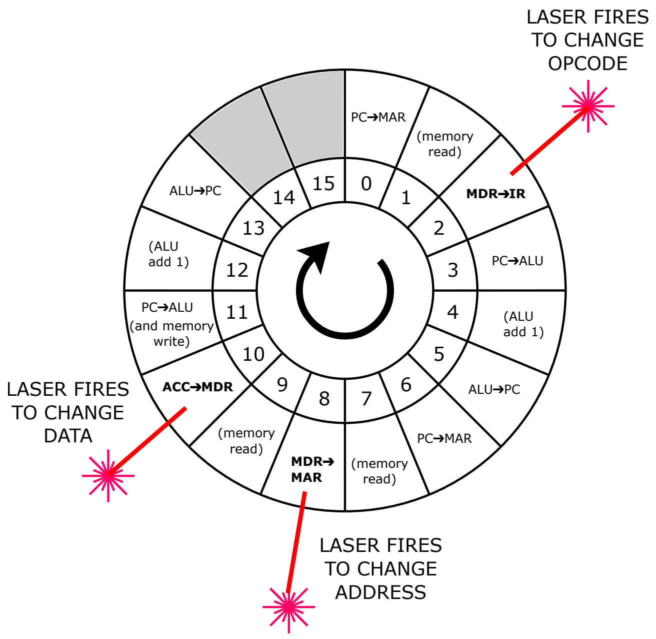

Figure 3: Timing diagram showing where the laser fires during fetch and execution of a single instruction (here, STA for "store accumulator").

extracted from "cat toys" sold on Amazon.com at https://www.amazon.com/gp/product/B09Y4D7NFB/. In operation, they draw approximately 150 mA each from a 3.3 V supply, so their optical power must be $<$ 500 mW and is probably considerably less, as they get warm in continuous operation. *These are absolutely not eye-safe and should never have been sold as cat toys.*

For safety when using 405 nm lasers, we recommend an enclosure made from #2422 transparent orange polycarbonate sheet 3 mm thick, as shown above when that apparatus was running at 405 nm.

The lasers are modulated by switching their power supply on and off with a MOSFET. Two important considerations apply to these lasers; firstly, they need 3.3 V and will burn out quickly at 5 V, but the MOSFETs won't switch a load less than their gate (control) voltage. So to make the MOSFETs work and avoid burning out the lasers, always switch 5 V through the MOSFET, and drop it down to 3.3 V with an LM317 voltage regulator between the MOSFET and the laser. The MOSFET modules used are available from Amazon.com: https://www.amazon.com/dp/B07F5JPXYS and the voltage regulators at https://www.amazon.com/gp/product/B08CDMZMDN/.

These lasers were chosen for use because they exhibit quick response when modulated in this way, typically $<$ 100 µs turn-on and turn-off latency. Many other laser modules from other sources, when measured, had a turn-on latency of more than 4000 µs, limiting modulation to $<$ 0.25 kHz.

See the external file
M5_remote_code_execution_480p.mov
for video.

Figure 4: Video of the remote code execution attack. It begins with the computer running a scanning pattern on the accumulator register. When the button on top is pressed, the lasers begin firing at the green bus LEDs in the middle. After a few seconds, the pattern on the accumulator LEDs has changed, indicating that the computer is now running a completely different program.

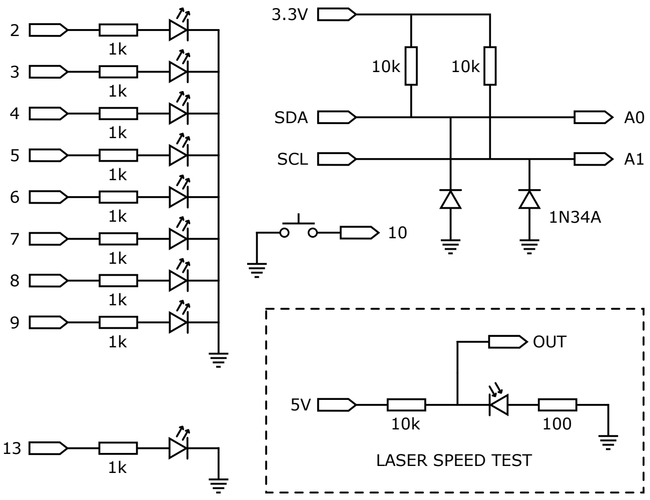

Figure 5: Schematic of the mezzanine board used as target in the I^{2}C attack. This is the green circuit board visible in the center of the apparatus.

4 I^{2}C experimental apparatus

The apparatus shown in Figures 6 and 4 below is an improved version of the one shown in Figure 16 of the paper, and will replace that photo in the final paper. It was designed for attacking a live I^{2}C bus and comprises several devices on the bus—addressable alphanumeric displays and a nonvolatile memory chip—together with a microcontroller and ESD protection for the bus.

The target of the attack is a pair of 1N34A glass-enclosed small signal diodes commonly used for electrostatic discharge (ESD) protection for an I^{2}C bus.[1]

These are mounted—for accessibility—on a mezzanine board plugged into the top of an Arduino Uno microcontroller (actually, a SparkFun clone with USB-C) in the center of the baseplate. The same mezzanine board holds the 10 kΩ pull-up resistors for the I^{2}C bus, and an isolated photodiode amplifier circuit used separately for speed tests on the lasers (Figure 5).

The Arduino here is really only used for two minor purposes: it sends out commands on the I^{2}C bus every few minutes to display the words “NORMAL OPERATION”, and it also measures the voltage on the bus with its internal ADC, drawing a bargraph on some off-board LEDs (connected to GPIO pins) for help with aiming. There is also a tiny pushbutton on the bargraph board (at the end of the rainbow ribbon cable) to reset the target so it refreshes the display.

Other devices on the I^{2}C bus, available for attacking, include an array of quad alphanumeric displays at address 0x70–0x73, and a nonvolatile memory chip at address 0x50.

The baseplate is 6 mm aluminum plate, drilled and tapped for mounting holes. The attacker is another Arduino Uno in the lower left corner of the baseplate; atop the attacker is another mezzanine board holding the MOSFETs used to modulate the lasers, and a red/green LED to indicate when the attacker is ready, and whenever the lasers are firing.

[1]Actually, these are modern Schottky equivalents of the original germanium component, but they work the same.

Below the attacker there is a pushbutton to start the attack, and a pair of BNC jacks for convenient access to connect and oscilloscope for measuring the I^{2}C bus, although these are not connected to the attacker in any way, except mechanically.

When the button on the attacker is pressed, it fires the lasers as fast as it can to impress a signal on the I^{2}C bus, attempting to write the words “PROOF OF CONCEPT” on the display. The lasers, in this case, are 780 nm near-infrared, because that is a good wavelength for ESD protection diodes. It has to do this blindly, as it has no feedback from the I^{2}C bus, so for example it can’t detect a collision, nor can it read data back from the memory chip. But the time needed to write a complete message to the alphanumeric display is around 10 ms, so there is a very small time window for collisions to occur.

Snapshots of the C++ source code for the attacker and the target are available at https://github.com/jloughry/basilisk_artifacts/blob/main/experiments/I2C/.

4.1 Data from experiments

See Figure 18 in the paper for an oscilloscope trace showing both (1) successful communication via laser with another device on the I^{2}C bus, and (2) the difference between a laser impressing a signal and conventional electrical communication. This is an illustration of the general principle that the attacker is often technically violating the specification but nevertheless communicating successfully.

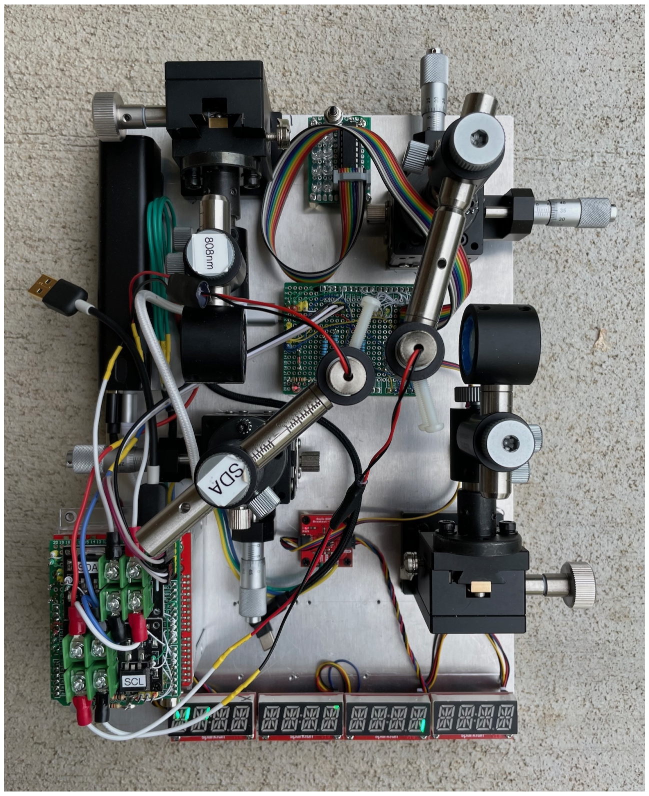

Figure 6: Experimental apparatus used to demonstrate I^{2}C bus attack.

4.2 Mechanical

Vernier adjustments are provided for aiming the lasers. Each vernier comprises a 40 × 40 mm *x*–*y* linear actuator (ThorLabs LX-20 or equivalent) bolted to the baseplate, onto which is bolted a Quarton model QLM-1125 polar laser mount. The polar axis of the mount provides easy coarse positioning—in polar coordinates—for aiming, with the linear actuator providing—in Cartesian coordinates—fine positioning capability. The polar and Cartesian coordinate systems are superimposed.

For precise focus, a second Quarton model QLM-1125 laser mount is paired with pieces of a Quarton model QLM-1225 to construct a Cartesian coordinate system from two polar coordinate systems, with θ_1 fixed at zero, and θ_2 fixed at $\frac{\pi}{2}$, letting us treat r_1 as x and r_2 as y. This is bolted atop a z-axis linear actuator (rack-and-pinion). It holds a 20 mm doublet converging lens with a focal length of around 25 mm. This arrangement allows the laser modules' internal lenses to be fixed at infinity.

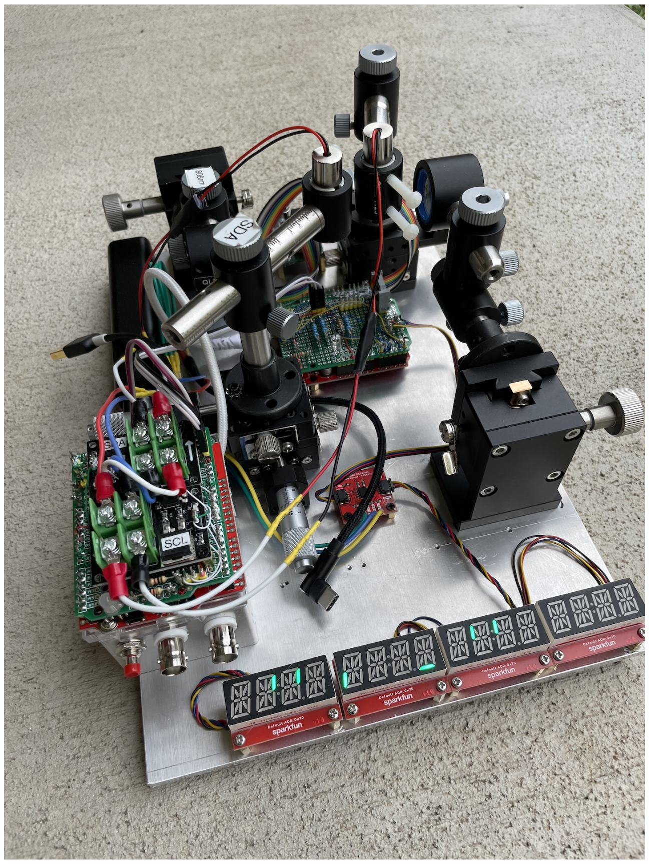

Figure 7: Oblique view.

Appendix

A List of dependencies for Ubuntu

```
sudo apt-get install build-essential clang \
                     bison flex \
                     libreadline-dev gawk \
                     tcl-dev libffi-dev git \
                     mercurial graphviz \
                     xdot pkg-config python \
                     python3 libftdi-dev \
                     qt5-default python3-dev \
                     libboost-all-dev cmake \
                     libeigen3-dev
```

B Portability

M5 and its attacker fit in a 2U rackmount enclosure that can sit on a table. The attacker is a separate 1U height rackmount box. Both devices are USB powered.

The I^{2}C apparatus is 250 × 350 mm and about 3 kg. It runs on internal USB batteries.