HARDWARE THAT IS LESS UNTRUSTED:
Open Source Down to the Silicon

RON MINNICH

HARDWARE THAT IS LESS UNTRUSTED: OPEN-SOURCE DOWN TO THE SILICON
RON MINNICH
JUNE 9, 2020

I. INTRODUCTION

In this essay, Ron Minnich argues "We can not build on a foundation that is compromised at all levels. There is no visibility into the system's behavior. The existing model assumes perfect software: 'Trust, but don't verify.' We need to start anew, from the gates, and work our way up."

Ron Minnich is a software engineer at Google.

A podcast with Ron Minnich, Philip Reiner and Alexa Wehsener can be found here

The views expressed in this report do not necessarily reflect the official policy or position of Technology for Global Security. Readers should note that Tech4GS seeks a diversity of views and opinions on significant topics in order to identify common ground.

Banner image is by Lauren Hostetter of Heyhoss Design

II. TECH4GS SPECIAL REPORT BY RON MINNICH
HARDWARE THAT IS LESS UNTRUSTED: OPEN-SOURCE DOWN TO THE SILICON
JUNE 9, 2020

The goal of CATALINK[1] is to build a more trustable system. What technical steps are required to build a system that users trust is doing what the system says it is doing? The system must be trusted all the way down to the transistors, and all of it must be open-source. This is inherently difficult, as the computing world is far less open today than it was 20 years ago. Adding to the difficulty, as we have learned in the last 10 years, those foundations have been completely compromised and are full of security holes—some accidental and some malicious. How did we get from the beginnings of computers to where we are today?

By far the computer type most commonly associated with computing is the so-called Personal Computer (PC). The Personal Computer defines a computing standard based around a physical

form. Though these started as personal, at least a billion computers following this standard are found in offices, data centers, automobiles, and personal electronics used around the world.

Shown below (Figure 1) is a typical PC of the 1980's—a motherboard. All the chips have an easily identified function. There is one central processing unit (CPU), memory chips, and Input/Output (IO) chips. And, in all this hardware, there is software. This software is contained in the larger chips with round white labels on the top, which blocks ultraviolet (UV) light from erasing the software. The software on those chips is called *firmware*, since it is software that is still there even when power is turned off. Firmware is less "hard" than hardware, since it is changeable software, yet "harder" than software, since it is persistent when power is removed and reapplied. When this board was created, anyone could write software to go in those chips, because all the information and resources to do so were openly available.
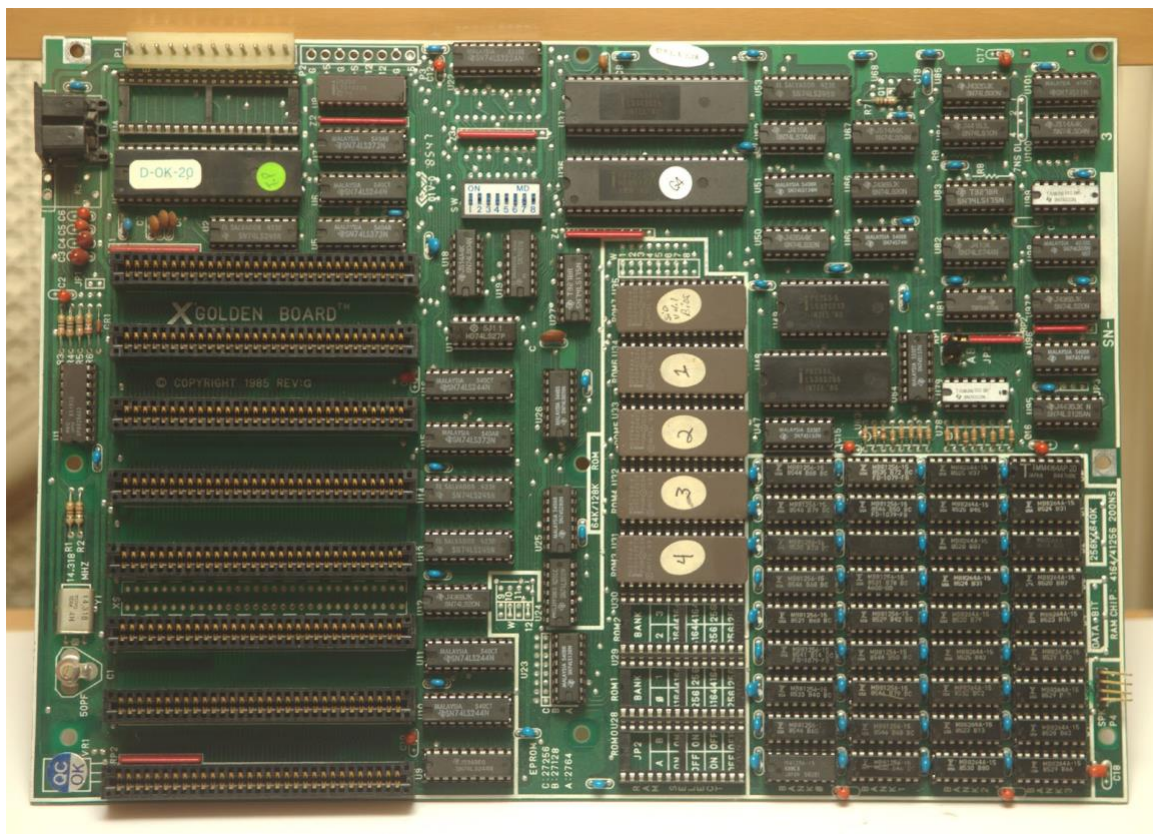

Figure 1. A 1980's motherboard.

Today's situation is drastically different. Shown below is a more recent motherboard (Figure 2). Some of the smallest chips on that motherboard—1 centimeter (cm) by 1 cm—are more complex than the entire motherboard shown above. Some of these chips contain 1,000 times as much firmware as the motherboard shown above, and like in the motherboard above, all are vulnerable to supply chain attacks. This firmware is difficult, if not impossible, to change once it is installed. The source is only available under the strictest Non-Disclosure Agreements. Moreover, there is no way to verify that the firmware on this board is doing what it should do, or that it is not doing what it should not do. The model of this proprietary firmware is, "trust, but don't verify." This model has proven too ineffective for operational security.

It has been demonstrated that it is possible to attach wires to the board, trace activity, and reverse engineer the software from those activity traces[2]. From there, it is not a significant leap to inserting malicious software. It would be even easier to modify the software somewhere in the supply chain; for instance, when the software is written, the board is built, or the board is en route to the final customer. Modifying software in this manner is called a *supply chain attack*.

Figure 2. A modern motherboard.

While there are many CPUs on this board, the most well known is the main CPU—the x86. The x86 is the most visible to the user, because it runs Windows or Linux, which in turn run all the applications users use. The x86 has firmware, called UEFI, which is as complex in many ways as Linux or Windows. UEFI is generally only released in binary form, and hence unverifiable: "Trust, but don't verify." Many security researchers have decoded UEFI for fun, however, and criminals have likewise done so for profit. A search for "UEFI exploits" reveals an astounding 204,000 results. As a result, in any given week there are many active exploits, some of the most recent being this year.[3]

We can not build on a foundation that is compromised at all levels. There is no visibility into the system's behavior. The existing model assumes perfect software: "Trust, but don't verify." We

need to start anew, from the gates, and work our way up. But what does this new model look like?

Shown below (Figure 3) is a new board called the Hi-Five, from SiFIve. SiFive is a new startup which is designing and selling RISCV (pronounced "risk five") CPUs. RISCV is an open, unlicensed architecture, which means that companies wishing to build RISCV chips do not need to pay a license fee or ask permission. RISCV is a specification managed by the RISCV foundation, originally based in the United States and now based in Switzerland. The RISCV specification is a definition of how the CPU has to work and not a CPU itself. Other companies such as Nvidia, Western Digital, and SiFive build the CPU itself. Today, there are also open-source implementations of RISCV available at places like GitHub.
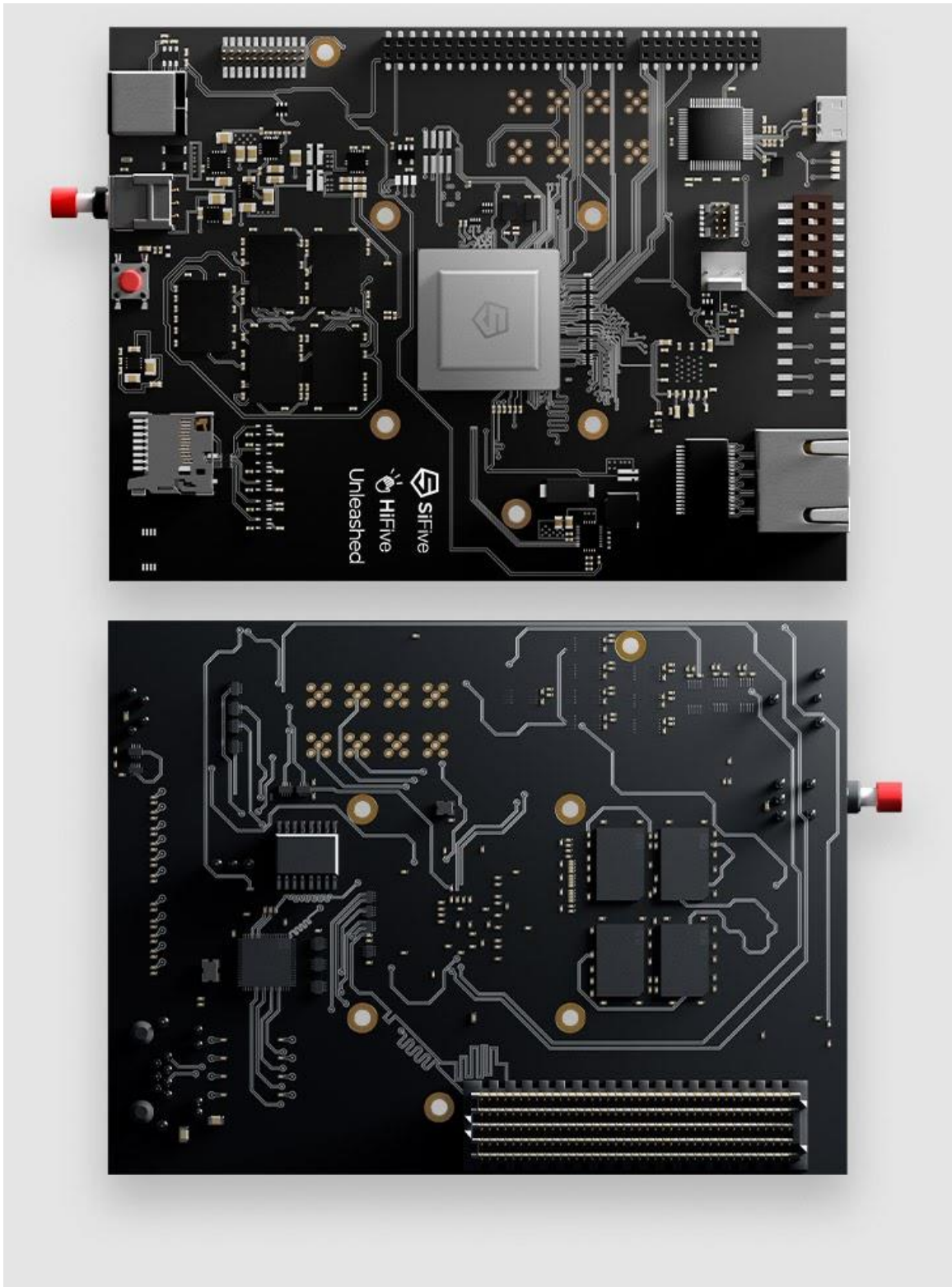
Figure 3: Hi-Five motherboard.

Open-source CPUs do not benefit from the 50 years of continuous performance gains achieved in the proprietary world. The HiFive board costs $1,000, compared to a nearly ten times more

powerful x86 board that costs a mere $30. When compared to proprietary systems, the open-source performance-to-price ratio is 300 times less.

Note that RISCV is not behind in every way; far from it. RISCV processors, being new, can be designed to avoid the kind of security problems that have plagued older CPUs in recent years. They have extremely low power consumption, and are arguably more power efficient than even ARM[4], which makes them practical for CATALink; a modern x86 processor is too power-hungry and hot. Because there is not a per-core license, and the cores are so compact, RISCV processors are not artificially constrained to small numbers of cores -- a recent system has 1049 cores, far more than any x86 processor offers today. Finally, there are some very cheap low-end RISCV 32-bit systems available today; in one case, one can buy a board for five dollars ($5)[5]! All that said, however, the highest-throughput RISCV CPU is at least 10x slower than a medium-throughput x86.

To effectively and efficiently run on RISCV, software needs to accommodate itself to constraints not seen since the early 2000's. CATALINK software needs to be efficient and thrifty. Therefore, we will need new software to accommodate these old constraints.

The requirement to write new software, rather than just taking what exists today, is an advantage. This process will give us a chance to write software designed for security as a top priority. Were we to simply take the RISCV hardware and drop existing software onto it, our system might still be unreliable and insecure. The problem is that much software is written in C, which is well known to be a problematic language when security problems are concerned. Further, most software is written to optimize features and speed instead of security.

In addition to the hardware issues previously discussed, we need to start anew on the software stack and consider modern programming languages with integrated safety attributes. For example, many projects are using a new language called Rust, which has many features to ensure safe programming, as well as minimizing the resource usage of the code. Other languages, such as Spark[6], are designed for *formal verification*, in which automated programs verify that the software does what it should.

The programming language will not on its own ensure a verifiably secure system. The rules for writing safe software are well understood, yet rarely followed. CATALINK will need rigorous controls to ensure software "follows the rules." These controls must be automated. One example of automated checking demonstrates how the "Go report card," generated by an automated system, grades the code quality continuously[7]. Another automation example is the Coverity code,[8] which scans 6,700 projects for code quality.

CATALINK can build on the foundation of RISCV, which is entirely open. Starting with that foundation, we can build new software in modern languages, such as Rust, that let us provide the assurance that the software does what it should, and does not do what it should not. Further, for all the components of CATALINK, we can adhere to a "Trust, but Verify" model, which is far superior to the alternatives we have today.

III. TECH4GS INVITES YOUR RESPONSE
Technology for Global Security invites your responses to this report. Please send responses to: info@tech4gs.org. Responses will be considered for redistribution to the network only if they include the author's name, affiliation, and explicit consent

IV. ENDNOTES

[1] Nautilus Institute, Stanley Center For Peace and Security, and Technology for Global Security, "Last Chance: Communicating at the Nuclear Brink, Scenarios and Solutions Workshop, Synthesis Report", NAPSNet Special Reports, May 23, 2020, https://www.tech4gs.org/uploads/1/1/1/5/111521085/last_chance_final_report-1__1_.pdf

[2] Andrew Huang, Hacking the Xbox: An Introduction to Reverse Engineering, No Starch Press, July 1 2003, updated March 2013, at: https://nostarch.com/xboxfree

[3] "Through the SMM-Class and a Vulnerability Found There.," SYNACKTIV Digital Security, January 14, 2020, https://www.synacktiv.com/posts/exploit/through-the-smm-class-and-a-vulnerability-found-there.html.

[4] https://www.datacenterknowledge.com/hardware/open-source-risc-v-ready-take-intel-amd-and-arm-data-center

[5] https://www.datacenterknowledge.com/hardware/open-source-risc-v-ready-take-intel-amd-and-arm-data-center

[6] The German Spark, not the Berkeley Spark. See [https://en.wikipedia.org/wiki/SPARK_(programming_language)].

[7] U-Root/u-Root, Go (2015; repr., u-root, 2020), https://github.com/u-root/u-root.

[8] "Coverity Scan - Static Analysis," Synopsys, accessed May 18, 2020, https://scan.coverity.com/