How to Build a Computer from Scratch

by Rotem Guttman and Gabe Somlo

Rotem Guttman: Hi, I'm Rotem Guttman.

Gabe Somlo: And I'm Gabe Somlo.

Rotem Guttman: And thank you for joining us. So Gabe, I understand that when I say I went and built my own computer, it doesn't mean the same thing as when you say you've decided to build your own computer.

Gabe Somlo: Boy, have I ever decided to build my own computer.

Rotem Guttman: I understand. So you're starting with a bucket of sand?

Gabe Somlo: Not yet. I actually plan on taking an ECE class on fabrication so that I could really build my own computer.

Rotem Guttman: I was joking, but.

Gabe Somlo: No, I mean--

Rotem Guttman: Okay, so let's get everybody up to speed. What was the impetus for this decision to build your own computer, and then what did you decide to do?

Gabe Somlo: So, back in-- no, let's start with the beginning, in the '60s, or maybe even before then. Microelectronics, the stuff various chipmakers fabricate, their customer base used to be overwhelmingly-- 90-something percent-- and this is not an exact number-- but some overwhelming percentage of their business used to be the United States government, and therefore the chips they fabricated and the standards they used and the locations they used were necessarily the United States and to the exacting specifications of the U.S. DoD. And so the DoD had their way with the chips for a virtual time, but then after a while the civilian market starting taking off, and now it is essentially the overwhelming part of what chipmakers sell and the government applications are a very small fragment of that. And supply chains and manufacturing and labor costs and optimizations to that effect sort of spread the whole thing around, and now you have chips being made in places where you don't even know what is all involved and who works there and who controls the fabrication.

And so we have a little bit of a problem for defense and sort of high-security applications because frankly no one really knows what went into the chips that we're using today, and most of the time if I'm updating my Facebook or ordering pizza online, I'm not important enough for anyone to mess with me, but the more paranoid you get and the more sort of important this is from a defense, security, sort of financial point of view, you have to start worrying about being a big, fat enough target for whoever did something to those chips. Typically--

Rotem Guttman: To be worth the effort for them to--

Gabe Somlo: Typically state actors, used you have the attention of important people, you have to start worrying about who cooked your hardware and whether there was a hardware backdoor inserted in it that's just sitting there waiting to be exploited when the target is important enough.

Rotem Guttman: Although to be fair, it's not just for a large-scale or have a security-critical application; supply chain attacks have happened for all sorts of levels. There have been supply chain attacks on cash registers that just went out into businesses so they could steal credit card information.

Gabe Somlo: Right. And then there's the whole idea of generally a hardware exploit is something that was inserted either during the design of the chip or the fabrication of the chip, or even by the tools that turned a clean design into a chip or a functioning computer. These are sort of like Easter eggs, and usually--

Rotem Guttman: Easter eggs can be good or bad.

Gabe Somlo: Well, they're there for whoever inserted them to use at some point in the future during a time of need. But the other problem is at some point somebody might just find one that they didn't put in there and then start using it. So everybody kind of has to worry about it. If somebody found one and you're important enough to then, then they'll use it on you.

Rotem Guttman: So what's the solution?

Gabe Somlo: So the idea is, "Okay, can we build a computer that is provably trustworthy?" And not in the sense of formal verification and lots of math. I actually haven't made it there yet. That's sort of a topic for the future. I'm actually probably headed toward taking a class about formal methods to sort of refresh my skill in that area. I haven't touched it in a long time. But just to prove that sources to the software and the hardware that went into your computer are--however much you can trust those sources, I would like to know that I can trust the thing I build from those sources exactly to the same amount.

Rotem Guttman: So essentially I can review my source, and I want to know that what I interviewed is what got made.

Gabe Somlo: Right. And so the typical rebuttal to that is, "Well, who has enough lifetime to read the sources to the Linux kernel and the C library and the CPU design that runs underneath it in case you actually get your hands on that source code?" There's not enough life in one person to actually read all of that, so if you can't do it anyway, why insist on the sort of open source equivalent to fielded system requirement? And that is sort of kind of a philosophical discussion about why I like open source so much, and it is not because I personally can't want to or have the energy or time or resources to look through all of it to ascertain that it's clean; it is possible for anyone to drill down into any corner of it in case they have any sort of suspicion. So it's sort of a democratization of security audits. It's proof that I have nothing to hide. If I give you the source code and someone eventually finds a problem with it, I am less likely to put it in there intentionally because I know somebody eventually might find it. Not that I expect everyone to go and sort of check the whole thing, but it's possible for one person to find one problem, and it's sort of cards on the table face up. It's sort of a guarantee from whoever wrote the source code that, "Hey, I have nothing to hide here. Look through it if you want."

Rotem Guttman: So it's not a strong guarantee, essentially-- it's not that it can't happen-- but at least we have a method of auditing it. We can go through and look at this stuff and see what's going on at least, rather than it being a black box.

Gabe Somlo: Right. So we could wander around and find the problems, the garbage dumps in the park. If it's sort of a fenced off area and you're not allowed to go in there, then very few people are ever going to be able to just walk by and find some toxic waste dump that's buried in there.

Rotem Guttman: And to take the example of open SSL, auditing that-- that's a massive code base and once flaws were discovered, the community came together, and so once it's important enough for any given community, now you can divide up the load and say, "Let's go all look at this, and then check each other's work."

Gabe Somlo: There was a toxic waste dump sitting there for a very long time, but somebody just happened to walk by it and call attention to it, right? If it had been fenced off and nobody was allowed to walk through that park, it would have stayed unfixed and the bad guys, who are more determined by definition than the good guys, will have found it eventually and used it without us knowing it's there. And it took us a while to find it, yes, but somebody did.

Rotem Guttman: But we found it.

Gabe Somlo: Yes.

Rotem Guttman: And so what's the solution? How do we do that for hardware?

Gabe Somlo: So there is a sort of burgeoning open source hardware community out there on the internet. We have open source CPU designs. Linux has been ported to that architecture. The only thing that's missing is the ability to sort of systematically prove that all your source code taken together is exactly as trustworthy as the system you built. And so the concept there is the concept of a self-hosting computing environment or computing ecosystem, and self-hosting means-- the best analogy is that whole ship of Theseus-- we're sailing it on the ocean while we're rebuilding it from scratch kind of thing. It has everything it needs within itself to rebuild itself without assistance from the outside. And if you want to think about it, Linux and then VSD in the software world are self-hosting systems. You have a kernel running applications on top of a library. One of those applications is the C compiler. The C compiler can compile both the location and the kernel. So the trio of kernel, C library, and C compiler is--

Rotem Guttman: A C compiler could compile a C compiler.

Gabe Somlo: Right. Yes, it is everything you need to build all these three components and all the other applications that run on top of it. So this is a self-hosting environment, essentially, because it can develop itself. It doesn't need external help to further develop or rebuild itself. And so that kind of breaks when you're trying to include the CPU that's running underneath all this stuff, because the CPU is a piece of silicon. It's been made by some manufacturer according to weird design methodologies an sort of proprietary processes that you have no visibility into. And so that's where the open-source hardware--

Rotem Guttman: So if a commercially available CPU is off the table, how do you do your calculations?

Gabe Somlo: So the next best thing to a commercially dedicated silicon, or application-specific integrated circuit, that's dedicated, hardcoded silicon, like most of the chips that you have in your computer, there's another alternative called field-programmable gate arrays, FPGAs, and what those are, they're essentially-- at the microscopic level, they're a regular grid of configurable logic blocks which can be turned into whatever hardware design you want, including a CPU. And so they're easier to inspect visually for bugs and the manufacturer doesn't know what you're going to use one for when you configure it. So you could put a CPU on top of it, configure it to act as a CPU, and that precludes a large class of hardware attacks. It's going to be a slower CPU. It's not going to be a high-performance computer, probably not going to do particle simulations or mining for cryptocurrency on this thing, but it can-- it's good enough--

Rotem Guttman: But for security-critical applications where you really need to trust--

Gabe Somlo: Those are more likely to be embedded systems, controlling some kind of industrial process or things that you don't want to have hardware holes in. So you could build a CPU on an FPGA, and the idea is can we use open source software tools to generate configuration for FPGAs. Those things tend to be proprietary today, and there's also an open source community that's sort of growing and getting closer and closer to being able to have open source software generate code for FPGAs that will turn them into CPUs and embedded computers, and if you can run an open source stack with Linux and the C library and the C compiler on top of the FPGA itself, then hopefully you can run the hardware compiler tools that turn-- so essentially you can have a computer that can rebuild its own-- and not just kernel, but also its own CPU and system on a chip. That's the dream.

Rotem Guttman: So you mentioned system on a chip. What are the components that we really-- what's the critical minimum set that we need in order to do this? So we have a CPU that we've created out of this FPGA. What else do we need?

Gabe Somlo: The CPU core essentially is just sort of the middle, center part of a hardware computer. There are so-called peripheral devices, things that'll let you use a keyboard, things that'll let you use a display or a terminal, things that will let you connect to a hard drive or just a little--

Rotem Guttman: Now to be clear, we're not talking about the actual keyboard or mouse or screen; we're talking about the interfaces to them from the--

Gabe Somlo: Yeah. So it's the thing you hook a keyboard into that's on the inside of the computer that is the keyboard's driver-- it's not a driver-- it's a device, basically. And the thing you plug a hard drive into, like SATA or SCSI, those are things that are called peripherals inside the computer, and essentially a system on a chip is a CPU plus a bunch of hardware that accepts connections to different devices that get jammed into the same logical chip, which is why it's a system on a chip rather than just a CPU.

Rotem Guttman: And you're going to get some sort of memory as well, right?

Gabe Somlo: Right. So that's to talk to the outside. So to talk to memory, you have a large or small or whatever, in-between, RAM, memory, and so the CPU being able to talk to the RAM and write and read from it, it needs another piece of hardware that is called a DRAM controller. So that is also part of the SOC, system-on-a-chip, assembly.

Rotem Guttman: So it seems you need source for all of these different pieces--

Gabe Somlo: Correct.

Rotem Guttman: --an FPGA that is large enough for you to create all these pieces with, and then some sort of system for translating from the source to actually having the FPGA, some sort of bit stream that's going to--

Gabe Somlo: That is called a hardware compiler toolchain. It's kind of like the C compiler for software. You take hardware sources and you run them through your hardware compiler toolchain to get things like bit stream for an FPGA. Or in a different sort of branch of hardware development, you could get things that are like a giant graph of transistors that can turn into a mask to make dedicate silicon, but that's not what we're talking about here.

Rotem Guttman: So where's the problem right now? Can I just do all that right now?

Gabe Somlo: No. Right now when you buy an FPGA, typically the only way to program that FPGA is with the vendor-supplied compiler toolchain that you're very unlikely to ever see the source code for.

Rotem Guttman: So another black box.

Gabe Somlo: It's a black box. So in that black box, theoretically there could be all sorts of exploits and problems. The only way to rule those out is to compile it yourself from source and then you have that property back that you could look through the sources or you could have someone look through the sources and find problems and call attention to them.

Rotem Guttman: So is that what you're working on right now?

Gabe Somlo: There is an open-source compiler toolchain being developed on the internet by a collaborative effort. I am participating in that effort. I'm trying to actually get a usefully large FPGA targeted by the open-source toolchain so we can make it run a system on a chip, so we can run Linux on it.

Rotem Guttman: So let's now talk about what's the use case for all this. So let's say you succeed, this is done. We have an open-source toolchain, we have sources, we have a million of these FPGAs sitting in a shipping container that we can do whatever we want with. What do we use them for now?

Gabe Somlo: So we have a little embedded computer that's capable of running Linux and proving that it is as secure as the source code. What should we be able to do with these things? Well, things like industrial control, SCADA systems, the government will want to know that various different embedded systems for military applications are trustworthy and not compromised.

Rotem Guttman: Well, it seems like it wouldn't just be the government, but also for commercial industry. There would be an incentive for any case where they're worried about a sponsorship attack; this could be useful for that.

Gabe Somlo: Right. Anyone who was worried about the chips having Easter eggs in them, and then being a big enough target to bother with by whoever inserted the Easter eggs or found them, could benefit from this. Again, this is not going to be a super powerful computer, but it's going to be a fully trustworthy computer for sort of lower-product, lower-intensity applications-- embedded systems, typically.

Rotem Guttman: And that's right now. We're at an early stage. You're always going to be slower than the cutting-edge, dedicated hardware, but--

Gabe Somlo: Right. So once we have a prototype, I am hopeful that we're going to keep optimizing it and bringing it up to speed to make it as fast as possible. So maybe if we're lucky in a few years we could have a fully trustworthy, open, transparent computer that is up to the standards of maybe the early 2000s, late '90s, which I could be happy using a desktop computer of that level of power for writing on my email and my finances and things like that. I'm still not going to run particle simulations on it.

Rotem Guttman: No, absolutely. This is not going to be for high-powered computing, but we can get it to the point where it's useful for security-critical applications, banking, for healthcare.

Gabe Somlo: Right. So if I want to know for 100 percent of a fact that this computer works for me and no one else-- there was nobody who left it in a state where it would somehow maybe betray me to the people who were involved in making it-- if I want it to be mine, work for me 100 percent of the time-- as in for the paranoid type of applications-- that's the kind of thing you could be using this for.

Rotem Guttman: Here's hoping that you're wildly successful, because it sounds like we could use it.

Gabe Somlo: Fingers crossed. Thank you.

Rotem Guttman: Thank you for joining us. For more information on the work that we do, see the website below. This has been Rotem Guttman and Gabe Somlo. Thank you.

Gabe Somlo: Thank you.

Related Resources

Berkeley & SiFive implementation of 64-bit capable open-source RISC-V CPU:
https://github.com/freechipsproject/rocket-chip

Free/Open System-on-Chip (SoC) project (currently 32-bit only, acquired capability to boot Linux as of a few weeks ago):
https://github.com/enjoy-digital/litex

Work-in-progress (by me) to get 64-bit RocketChip to work with the LiteX SoC and its peripherals (serial console, ethernet, microSD):
https://github.com/gsomlo/litex/tree/gls-rocket

Free/Open Verilog (Hardware Description Language – HDL) toolchain (equivalent to e.g. Xilinx Vivado), works for programming Lattice FPGAs:
https://github.com/YosysHQ/yosys
https://github.com/SymbiFlow/prjtrellis
https://github.com/YosysHQ/nextpnr