Chapter 3
What Is an ICT System?

The full complexity of the information and communications technology (ICT) systems that we use every day is hard to fathom and it spans at least two dimensions. First, if I were to send an e-mail to my colleague in the office next door, the process easily involves more than a hundred devices over two continents. On its way from my computer in Norway to the mail server I use in the United States, it will traverse routers and switches in several countries. Each of these routers and switches will be dependent on several other components just to determine the next hop on the path towards the recipient of the e-mail.

The other dimension regards the complexity of every single one of these hundred devices. From the observable functionality of each device, there are multiple layers of technology all built on top of each other before we come down to the physical phenomena that allowed us to build the device in the first place. In this chapter, we describe the most important of these layers and conclude on how each layer can be exploited by a dishonest maker of ICT equipment.

3.1 Transistors and Integrated Circuits

In a discussion on the most important inventions of mankind, a strong argument can be made that the *transistor* [1] is on par with fire, the wheel, and agriculture. A few decades after its invention, it has changed the lives of billions of people and is now central to the way we work, the way we communicate and interact, and our consumption of cultural expressions.

However important, the transistor is conceptually a very simple thing. It can be explained as an electrically controlled light switch. The transistor is simply a device that lets the electric current through one conductor open and close a switch, and thereby start or stop electric current in another conductor. At a higher level of abstraction, we observe the transistor move information from one electric circuit to another and, from this observation, we can understand that the invention of the transistor was the seminal start of what came to be known as information technology. Circuits consisting of transistors and resistors can be constructed so that they implement all the logic functions of Boolean algebra. Circuits implementing a Boolean function are generally called *logic gates* and, through clever combinations of such gates, all the computing functions of a modern computer can be constructed.

Of course, electrically controlled switches in the form of relays and vacuum tubes had been designed before the transistor was invented and they had been used for building computers as well [3, 10]. What the transistor did bring to the table was two crucial properties. First, it could switch much faster and far more reliably than the old technologies and, second, it could be miniaturized. These features have given rise to modern *integrated circuits* with a number of transistors that today (2017) can run in the multiple billions and operate at a switching speed of multiple gigahertz.[1]

Already at the level of the transistor we find places where a manufacturer can build in malicious functionality. The physical phenomena that allowed us to build transistors are the link between the analog continuous world and the discrete world of the computer, but this link is by no means trivial to engineer. There are a number of ways a transistor can become unstable or stop working as intended. Therefore, already here we find places where a manufacturer can plan to do harm. A kill switch can be implemented by overloading selected transistors based on a given input signal, thus leaving the integrated circuit unusable. Completely investigating a product for malicious functionality therefore goes all the way down to studying the countless transistors on each chip.

3.2 Memory and Communication

The three central elements of a computer are data processing, the storage of information, and communication. All of these elements can be implemented by transistors and logic gates.

Communication in the sense of moving information from one place to another can, in its simplest form, be done by transmitting high or low power on a conductor. Thus communication can easily be implemented by means of transistors. A great deal of development has gone into improving the speed and distance of communication links, as well as reducing the fault rate, and the most successful developments are based on using light waves in optic fibres rather than electric signals in metallic conductors. Apart from the added complexity that accompanies these developments, they are not directly relevant to the discussions we cover in this book. We therefore refrain from giving further details.

There are a number of ways in which one can store information using logic gates. The general idea is that to store a bit, you can design a circuit with a feedback loop, so

[1]In 1965, Gordon Moore observed that the number of transistors on an integrated circuit seemed to double every two years. The truth of the observation has been remarkably persistent ever since and has therefore been referred to as Moore's law. The switching speed of transistors grew as fast as the increase in the number of transistors for a long time, but it started to level off around 2010.

that the input to a logic gate depends on the output of the same gate. Such a feedback loop can create a perpetually unstable circuit: for example, when an output value of 1 fed back into the circuit will generate an output value of 0 and vice versa. If made correctly, however, the circuit will have exactly two stable states, corresponding to a stored 0 and a stored 1, respectively. Storage built from logic gates in this way are known as volatile memory, meaning that the information is retained only as long as the system is powered on. This setup is used for registers in the CPU and for RAM. There are a number of technologies available for storing information in a more durable form, such as hard disks and flash memory.[2] These are dependent on physical phenomena other than those of the transistor but they still depend on transistor-based logic circuits for their control logic.

Both memory circuits and communication circuits can be exploited by an untrusted vendor. Above we argued that a circuit can be constructed so that it self-destructs on a given input signal. For memory and communication circuits, it is clear how such input signals can be received. For a communication circuit, the signal can arrive from the outside world, whereas a memory circuit can be designed to self-destruct when a particular sequence of bits is either written to or read from memory. Self-destruction is, however, a very simple form of sabotage that can be controlled by an untrusted vendor. More advanced operations, such as espionage and fraud, are primarily made possible through the logic that controls the memory and communication circuits. We discuss such control logic and processors next.

3.3 Processors and Instruction Sets

What drives the actions of the ICT system forward are the circuits of logic gates that perform actions on the memory and the communication channels. Such circuits read bits from memory and/or a communication channel, let the logic gates react to the read input to compute some new value, and write the computed bits into (possibly other parts of) memory or send them on to a (possibly different) communication channel. Long sequences of such actions can be constructed to perform arbitrarily complex operations and, in principle, everything electronic equipment can possibly do can be realized through such sequences.

The flexibility of a modern computer does, however, not come from fixed sequences of operations. Early on, it became clear that the exact operation of a computer could be coded into memory itself. A very complex circuit of logic gates could read an instruction coded into memory, perform one very specific action based on that instruction, and then move on to read the next instruction [10]. The semantics of each instruction would be as simple as *read the contents of register A and add them to the contents of register B*. Other instructions could apply to communications between the circuit and neighbouring circuits in the same system or they could be a

[2]Non-volatile memory is generally slower in operation; therefore all computers presently uses volatile memory for registers and other memory close to the core of the architecture.

message to the system that it should start reading instructions from another part of memory, so-called *jump* instructions.

An electronic circuit reading instructions from memory in this way is called a *processor* and the set of different instructions it is able to understand is called the *instruction set* of the processor. Each processor model has its own unique instruction set, which can be quite large; the number of transistors needed to implement these instructions is therefore also quite high. An example that illustrates some of the complexity is the 22-core Intel Xeon Broadwell E5 [6]. This is a CPU chip containing 22 processors and cache memory implemented by 7.2 billion transistors and operating at a clock switching speed of up to 3.8 GHz.

The instruction sets implemented by a processor form the bridge between the tangible hardware and the software that controls its behaviour. The divide between hardware and software is very distinct in ICT, in that its respective designs require different skill sets from the engineers. Also pertinent to our problem, it represents a divide in the world of security experts. Most security experts are concerned with weaknesses and attack vectors in software, often assuming that the hardware itself can always be trusted. If we do not trust the maker of the hardware, however, this assumption is completely broken. A vendor can include undocumented computer instructions known only to itself and let their execution be triggered at a predefined input signal of the vendor's choice. Documented instructions can have undocumented side effects that leak information from memory and send it on to a communication channel.

In principle, all the malicious functionality one can think of could be implemented in the hardware of a CPU. In particular, one should be aware how this influences the security provided by cryptographic software. Cryptography is the main building block of computer security. If cryptographic software runs on untrusted hardware, there is no reason to believe that the system is secure.

3.4 Firmware

Firmware is often described as something residing in between hardware and software. In essence, it is a piece of code that is loaded onto non-volatile memory and is read by the processing unit at start-up [7]. The firmware is what ultimately defines the instruction set and functionality of a processor.

The huge benefit of firmware is the added flexibility of a chip after its manufacture. Firmware can be used to correct mistakes in the chip design and to add new functionality after the hardware has been made. For instance, updates to firmware are used to add new codec formats to portable music players and other firmware updates have improved the power consumption of mobile phones.

What firmware means for the problem we address is that it constitutes another possible point for the introduction of malicious functionality. Firmware that changes the semantics of a processor's instruction set can clearly be used by untrusted vendors to carry out espionage and fraud, as well as sabotage. In our journey up the technology stack, this is the first and lowest point where we can see that a vendor can introduce unwanted functionality after the product has been purchased and received. A firmware update has all the conceivable potential to render your product unsafe; therefore, if your vendor is not to be trusted, firmware updates should be regarded with considerable suspicion.

3.5 Operating Systems, Device Drivers, Hardware Adaptation Layers, and Hypervisors

Starting from the instruction sets of the underlying hardware, the *operating system* [8] is a piece of software that builds higher-level abstractions and concepts that are closer to what a user or programmer will need. From basic physics that allow you to store and read a bit of information, it builds the concept of *blocks* and *files*. From the instruction set of the processor, it builds the notion of *process*. Some equipment will allow for multiple users and therefore the concept of *user* must be formed. In systems that will have multiple simultaneous processes, provisions must be made for the time-sharing and virtualization of the computer's resources. In addition, security mechanisms must be built into the operating system such that each process is isolated from the other processes, ensuring that they cannot overwrite each other's portions of memory, and that processes and users do not have access to information on the system that should be hidden from them.

Ideally, once you have created an operating system, you would like it to be able to run on several different versions of hardware. To accommodate this, a thin layer of software often runs directly on the hardware, making the hardware appear as if it were of a predefined generic type. These thin layers of software are called *hardware adaptation layers* for the core part of a computer and *device drivers* for peripheral equipment. In particular, the notion of a driver should be well known to most: whenever you connect external equipment to your computer, a driver for that piece of equipment needs to be installed.

A *hypervisor* [2] is a more recent notion. It stems from the need to be able to virtualize a computer so that it appears as several computers, allowing the same piece of equipment to run multiple operating systems at the same time. This is particularly useful for cloud services that offer platforms as a service. Disconnecting the instances of operating systems from the hardware they run on allows them to offer a number of software platforms to its customers that is not fixed to the number of instances of hardware it has installed. A hypervisor can run directly on the hardware (native, or type 1 hypervisor) or it can itself run as a process on top of another operating system (hosted, or type 2 hypervisor).

An untrusted maker of operating systems, device drivers, hardware adaptation layers, and hypervisors has optimal working conditions. If any of these components does not work, the entire system would most likely stop; these components are therefore ideal places for introducing kill switches. Second, since most of a computer's security mechanisms are placed in these components, it is the ideal place to introduce code that steals and leaks information. The fact that the operating system is a very complex and large piece of software makes it extremely hard to examine all its parts fully. As an illustration, it is assumed that most versions of the Windows operating system have several tens of millions of code lines.

3.6 Bytecode Interpreters

In the same way that device drivers and hardware adaptation layers render operating systems less dependent on the specifics of the hardware they run on, the intention of bytecode interpreters is to make the application independent of the operating system and platform. In essence, bytecode is a program written in a low-level but generic instruction set. Once the bytecode has been created for an application, this application can run on any combination of hardware and operating system for which a bytecode interpreter exists that can run it.

A plethora of different bytecode definitions exist, but the one most of us have been directly in touch with is that realized by the Java virtual machine [4]. It allows the same executable Java bytecode to run on a vast set of different platforms, from huge servers down to handheld devices.

Not all applications come in the form of bytecode, since many applications run directly on the operating system/hardware platform itself. This is indeed the case for the bytecode interpreters themselves. The widespread use of bytecode still urges us to consider how a dishonest bytecode interpreter creator could harm us. Such ways are quite easy to find, since the bytecode interpreter is in full control of everything an application does. It can therefore implement kill switches based on given inputs and leak any information that is made available to the application itself.

3.7 The Application on Top

All throughout transistors, logic gates, integrated circuits, printed circuit boards, hardware adaptation layers, and finally operating systems and drivers, the intention of the different fields of engineering is to build generic platforms. The intention of the complete electronic device itself is determined by the application running on top of this platform. The set of possible applications is exceptionally diverse and spans all conceivable uses of electronic equipment.

Many known cyberattacks go through an application to compromise a system. The attack can appear in the form of a webpage and go through the web browser or it can come through an email attachment. Still, when what we fear is wrongdoing on the part of the maker of the equipment, we do have some defences against the maker of applications. Since security mechanisms against third party attacks will usually be built into the operating system, applications must exploit weaknesses in the operating system to cause harm to the entire installation. If the operating system is secure, applications will be limited to leaking information that is accessible by the user running the application. This does not mean that untrusted applications are not dangerous; it only means that, as opposed to makers of operating systems, we have at least some level of defence against malicious developers of applications.

3.8 Infrastructures and Distributed Systems

The days of standalone electronic devices are long gone. Most devices are connected to the Internet and the applications they run are part of a distributed system. Two obvious examples known to everyone are the World Wide Web and e-mail. The truth is that the applications that are currently standalone and running on only one device are quite few. If nothing else, most devices are part of a distributed system that provides software updates over the net.

A distributed system is as complex from the operating system and up to the user interface as it is from the operating system down to the physical phenomena of a transistor. As an example, your e-mail client is built on top of at least three different layers of path-finding systems before your machine is able to talk to your mail server. Typically, these are the Ethernet protocol that finds the path between your PC and the gateway of your building [11], the interior gateway protocol that finds the path within the network of your Internet provider [9], and the border gateway protocol that finds the way between different Internet providers [5]. On top of this, a separate distributed system, called the domain name service, is required that translates e-mail addresses to the IP addresses of the mail server and a distributed system for authentication is needed that can secure the integrity of the mail accounts. Only when all of these components are in place can the mail system start working and the key insight is that all of these underlying components are themselves complex distributed systems.

In some cases, the same vendor will be responsible for making all the components of a distributed system. In other cases, the system is open, in the sense that new components can be added continuously and these new components can come from different vendors. Typical examples of the latter are e-mail systems and communication networks. In the former case, the harm that a malicious vendor could do is quite similar to what we described in the section above. All information trusted to the distributed system could leak and the system itself can be used to gain access to resources that should be secured by the operating system. In the latter case, we must assume that the product of the untrusted vendor is integrated into an already existing distributed system. This could be a new e-mail server or a new router that extends a country's critical network infrastructure. In the latter case, a new challenge arises, in that the new component could create chaos in the existing distributed system by behaving in unpredicted and malicious ways. A new router in a system could, for instance, sabotage the path-finding algorithm in the network and thus disconnect the entire network (Fig. 3.1).

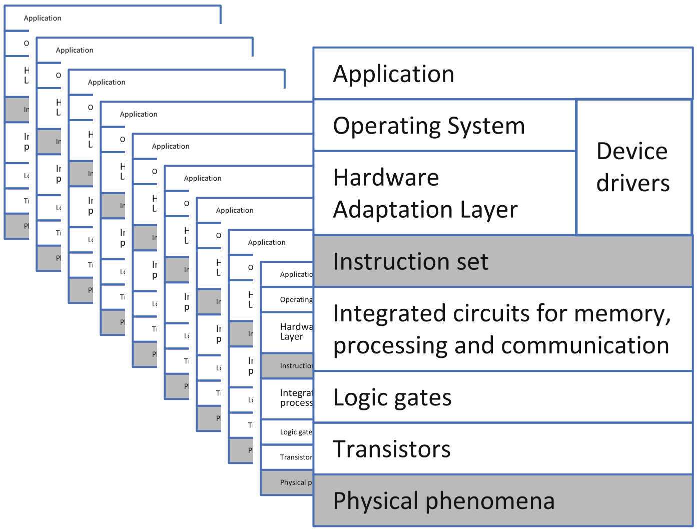

Fig. 3.1 Schematic overview of a distributed system. Each device builds instruction sets from physical phenomena through the hardware layers of transistors, logic gates, and integrated circuits. On top of the instruction sets, there are multiple layers of software before we reach the application. This technology stack can easily consist of billions of transistors and millions of lines of software code. Such a highly complex structure appears in all the – possibly hundreds of – devices that constitute the distributed system. Before a simple thing such as e-mail can work, several different distributed systems must be in place and work well. The domain name system, the interior gateway protocol, and the border gateway protocol are the three most obvious ones

3.9 Discussion

The takeaway from this chapter is that verifying an ICT system to ensure that it is not doing harm is a monumental task. In principle, it entails a careful study of all the software components of several distributed systems running on top of each other. Each of these systems can easily consist of tens of thousands to hundreds of thousands of lines of program code. For each of the possibly hundreds of devices running the code that is involved in these distributed systems, we would have to study the operating system, device drivers, and hardware adaptation layers that run on them. The operating system itself can consist of tens of millions of lines of code. If it runs on a virtualized platform, we would also need to include the hypervisor in the study.

All of the above would only touch the software side of the problem. Each device would consist of a number of chips integrated on a number of printed circuit boards. The CPU of the system alone will consist of tens of millions of logic gates and billions of transistors and, even if the CPU is probably be the most complex of the chips in the device, there will usually be more than a handful of other chips that need to be studied.

Most buyers of electronic equipment are only exposed to parts of this problem. The decision to buy electronic devices will rarely encompass all the infrastructure necessary to make the device work. If you do not trust the maker of a mobile phone, you will only need to investigate that device, because the network it connects to is not under your control. If, on the other hand, you are the network provider, you have control over what equipment you buy and use in your network but not over the equipment that your customers might connect to it or the equipment of the network provider you peer with to provide connectivity. Even under these limitations, the task of securing against dishonest makers of electronic equipment spans the physical phenomena underpinning the transistors of the system all the way up to the software running the user interface of the application. In large parts of this book, we will be concerned with ways of approaching this problem to understand if it is at all doable. Before that, in the next chapter, we will discuss the design process of ICT equipment to understand how a dishonest vendor could introduce malicious functionality into a product.

References

1. Bardeen, J., Brattain, W.H.: The transistor, a semi-conductor triode. Phys. Rev. **74**(2), 230 (1948)
2. Bressoud, T.C., Schneider, F.B.: Hypervisor-based fault tolerance. ACM **29** (1995)
3. Copeland, B.J.: Colossus: The Secrets of Bletchley Park's Code-Breaking Computers. Oxford University Press, Oxford (2010)
4. Lindholm, T., Yellin, F., Bracha, G., Buckley, A.: The Java Virtual Machine Specification: Java SE 8 Edition. Pearson Education, London (2014)
5. Lougheed, K., Rekhter, Y.: Border gateway protocol 3 (bgp-3). Technical report (1991)
6. Nalamalpu, A., Kurd, N., Deval, A., Mozak, C., Douglas, J., Khanna, A., Paillet, F., Schrom, G., Phelps, B.: Broadwell: a family of ia 14 nm processors. In: 2015 Symposium on VLSI Circuits (VLSI Circuits), pp. C314–C315. IEEE (2015)
7. Opler, A.: 4th generation software. Datamation **13**(1), 22–24 (1967)
8. Peterson, J.L., Silberschatz, A.: Operating System Concepts, vol. 2. Addison-Wesley, Reading (1985)
9. Rekhter, J.: NSFNET backbone SPF based interior gateway protocol. Database **4**, 5 (1988)
10. Rojas, R.: Konrad Zuse's legacy: the architecture of the Z1 and Z3. IEEE Ann. Hist. Comput. **19**(2), 5–16 (1997)
11. Seifert, R.: Gigabit Ethernet: Technology and Applications for High-Speed LANs. Addison-Wesley Longman Publishing Co. Inc., Reading (1998)