Chapter 12
Summary and Way Forward

In this book, we have asked the following question: What if one or more of the providers of the core components of an information and communication technology (ICT) system are dishonest? This question has been actualized by recent discussions and events, such as the Snowden revelations, the discussions that have taken place in many Western countries on the inclusion of equipment from Chinese providers into telecommunications infrastructures, and the case of Volkswagen cars having electronics recognizing that they were being tested for emissions.

Our problem is widely distinct from the traditional problem of the prevention and detection of malware in a system, first, because the malware is already present at the time of purchase, so preventing it from entering into the system is meaningless, and, second, because there is no clean malware-free sample for comparison, something that is the basis for most effective malware detection techniques. In this book, we have wandered through the landscape of ICT knowledge. Throughout the journey, we have tried to determine how one might verify if a producer of an artefact consisting of electronic hardware and software has included unwanted functionality in the product.

12.1 Summary of Findings

The notion of trust has been heavily studied in modern philosophy since the 1980s and onwards. In most aspects of life, our relationships to the people and institutions we interact with are influenced by the amount of trust we have in the people and institutions in question. Our need to verify their actions depends on our earlier history with them, their reputation among other people we trust, and the consequences for ourselves and the other party if they defect on us. In Chap. 2, we discuss the trust relationship between a buyer and a vendor of electronic equipment. Ultimately, we find that meaningful trust in this relationship can only be attained through an ability to verify the functionality of the equipment that is being sold. Most of this

O. Lysne, *The Huawei and Snowden Questions*, Simula SpringerBriefs on Computing 4, https://doi.org/10.1007/978-3-319-74950-1_12

book is therefore devoted to the technical verification of the properties of electronic equipment.

In Chap. 3, we embarked on the notion of a system, pointing out that it spans many dimensions. The most obvious dimension is that going from the user interface of a given system – through many layers of software, firmware, and hardware – down to the physical phenomena that allow us to build the machines in the first place. Another dimension emerges from the distributed nature of most complex systems. These are typically constituted by a series of subsystems, many of which physically reside on different machines. In addition, these subsystems may be owned and run by different organizations in different countries. Since the outsourcing of services and the usage of cloud facilities are becoming commonplace, the diversity of the ownership of subsystems is likely to increase.

A usual assumption when looking for weaknesses in computer systems is that the weakness, or, in our case, the security hole, will be visible from the source code. Several companies have therefore made efforts to let their customers study the source code of their systems. This is a strong act of goodwill but, unfortunately, we cannot assume that the absence of malicious actions in the source code guarantees freedom from malicious behaviour in the equipment itself. In Chap. 4, we use a result from the 1980s to show that malicious executable code and electronic circuits can be introduced by the tools used by developers. In these cases, not even the developers of the product need be aware that malicious code is being embedded into the product they are building. Therefore, to look for malicious code or malicious circuits, we have to study gate-level electronics and the executable code.

One reason reverse engineering and the search for malware are challenging is that there are theoretical limits to what can be achieved through code analysis. This is studied in Chap. 5. We showed proof that decision procedures for detecting malicious functionality are a mathematical impossibility.

Understanding and making sense of executable code and gate-level definitions of hardware in life-size systems is a massive task. There is, however, an active community with well-developed tools for the reverse engineering of systems. The expertise of this community is used in analysing the effects of known malware, as well as in reverse engineering programmer interfaces for software development. We provided an overview of the classes of tools used for the reverse engineering of hardware and software in Chap. 6. Although these tools have shown their worth in analysing limited problems, they are a far cry from bringing us close to analysing entire systems in the full complexity illustrated in Chap. 3.

The detection of intrusion is perhaps the most developed field in computer security. Such detection consists of scanning a system in the search for a computer virus, Trojan, or unwanted active processes placed there by third parties. Chapter 7 gave a high-level overview of the different known methods. The success of this field is one of the finest stories in the tale of computer security. Still, its methods fall short of solving our problem. The most efficient modus for intrusion detection is to scan a system for the signature of a piece of code that is not supposed to exist in a healthy system. When malware is inserted into a system at the development phase, no system

will be healthy. The distinction between code that is supposed to be there and code that is not disappears.

An alternative to analysing an electronic product before it is powered on is to start it up, let it run, and then study its behaviour. In Chap. 8, we explained that this can be done in one of two ways. One is to build a controlled environment around the piece of equipment, a so-called sandbox This has the advantage of containing the malicious acts the equipment could carry out so that no real harm is done. Another benefit of this approach is that the sandbox itself can conduct a systematic test of how the piece of equipment acts under different stimuli. The main weaknesses of sandboxes are, first, that they only run for a limited time and, second, that malicious pieces of code may try to check if it is running in a sandbox before it exposes its malicious behaviour. These weaknesses form separate battlefields where malware creators try to hide malicious functionality from the sandbox and the sandboxes creators work on tricking the malware into exposing itself.

The second approach to dynamic analysis consists of putting the device into ordinary production outside of a sandbox and then observing its behaviour. By doing so, we avoid sandboxes' drawbacks of time limitations and test detection. On the other hand, malicious acts performed by the device will have real-life effects. The severity of these effects will differ, depending on what the effects were. If the malicious effect is to render a piece of equipment unusable, then it will be too late by the time it is detected. If it is to leak confidential information, it may not be too late but, depending on the criticality of the function of the equipment and alternative ways of providing this function, it may be hard to stop.

The key to understanding if dynamic testing – either in a sandbox or in production – can help solve the problem lies in the question of whether the malicious acts can actually be detected. As stated, acts that render the equipment useless are easily detected when they occur. When they occur in real production, however, it will be too late and, since they can be triggered by arbitrarily complex external stimuli, one cannot expect to be able to trigger them in a sandbox. As for the leakage of information, there is more bad news. It has been shown through information theory that undetectable leakage is possible, although with limited bandwidth.

Although we have presented a series of undecidability results that seem to end our analysis, several theoretical developments have promise. In the field of formal methods, which we discuss in Chap. 9, we have a plethora of logic systems for programs and deduction systems that are *sound and complete*. This means that, for every property of a program that can be expressed in the logic system, there exists a proof for this property. Although the undecidability results mean there is no systematic way the absence of malicious behaviour can be proven, it is clear that an honest developer will have a clear mental picture of this during the development phase. We can therefore envision that the programmer provides proof of absence of malicious behaviour as part of the development process. Checking the correctness of such proofs is a trivial task that can be done by a machine.

However, even formal methods have limitations. In this case, the limitations are related to the fact that the developer mainly works on source code. This source code is automatically transformed into binary code before it is executed on a machine or

into gate-level structures before it is implemented in hardware. For the proofs to be watertight, they will therefore have to refer to code on another level of abstraction than what is in the developer's mind. Another difficulty is that an ICT system is generally exceptionally complex, as discussed in Chap. 3. This complexity overwhelms the current state of the art in formal methods. Finally and most importantly, the notion of sound and complete is deceptive. It leaves the impression that every property of a program can be proven. What it really means is that every property *expressible in the logic system* can be proven. A logic system that is sound and complete therefore has the limitation that properties that are undecidable in the Turing sense cannot be expressed in the system. As mentioned above, this will be the case for most of the malicious properties we are interested in finding.

We discussed in Chap. 10 the existence of various systematic approaches to the assurance of software quality. The most common one of these for computer security is called the Common Criteria and these are formed around seven quality assurance levels. Although the gist of these assurance levels is that the developer of the equipment collaborates with the customer to secure the system against third parties, they can be made relevant for our discussion as well. They do, however, encounter the same limitations as those discussed in the various chapters of this book, with regards to both the static analysis of code and dynamic testing.

If our system has only a few components that we do not trust, we could try to execute them in a controlled manner. They can be put in a sandbox environment where they cannot send information to the outside and where the information going into the components is controlled and even encrypted to ensure that all communications from the outside are scrambled before reaching the component in question. We study these mechanisms in Chap. 11. Unfortunately, they are usable only for a limited set of cases.

12.2 The Way Forward

The conclusion on the current state of the art with respect to our problem is rather bleak. If we do not trust the makers or vendors of our electronic equipment, there seems to be no framework of knowledge that adequately addresses the full complexity of the question. However, some approaches appear to be more promising than others and they should receive the additional attention that the seriousness of the problem seems to justify. We will mention some of them in the following sections.

12.2.1 Encryption

Distrust in the equipment you communicate through and collaborate with is one of basic assumptions in encryption algorithms and protocols. This is therefore an example – and possibly the only example – of a field of ICT security and robustness

where the premise of the field need not be changed as a result of distrusting the equipment vendor instead of accidents or a third party. The research field has therefore resulted in numerous methods that provide a solid basis for trust in our case as well, even when communication can be leaked and even in the presence of attempted man-in-the-middle attacks.

The development, harnessing, and proliferation of strong encryption techniques should therefore be encouraged and will be an important component in the solution to the problem of untrusted equipment. Still, encryption does not solve the problem altogether, for two reasons: First, the equipment used for encryption and decryption will also consist of electronic devices. The question of who built the encryption equipment, from the transistors up to the application, will be important. If one cannot place complete trust in the people who built the encryption equipment, in all the complexity we have described in Chap. 4, then the encryption methods offer nothing. Second, while encryption promises to build defences against threats such as eavesdropping and man-in-the-middle attacks, it does not touch the problem of remotely controlled kill switches. For that problem, we must look for solutions elsewhere.

12.2.2 Formal Methods

We have argued above that formal methods currently fall short of solving the problem of fully verifying electronic equipment. In spite of this, we choose to list it as one of the research fields that should be pursued with greater intensity in the search for solutions to our problem. Our reason for this is that, of all the existing approaches to quality and the scrutiny of code – such as software quality management, reverse engineering, code review, and static and dynamic analysis – the field of formal methods is the only one that holds the promise of being able to issue *guarantees* of the absence of unwanted functionality.

One particularly promising approach lies in the combination of specification-based techniques in defining proper benign system behaviour, discussed in Sect. 7.8, and proof-carrying code, discussed in Sect. 9.8. Specification-based techniques have the advantage over other static detection methods of not requiring a clean, uninfected sample of the system. Proof-carrying code has the advantage that the computational challenges of finding the proof of a system's correctness are transferred from the system's customer to its provider. The customer of the system will therefore only have to carry out proof-checking, usually a fairly simple computational task.

Several problems, however, need to be addressed through research. First, much research in the field has focused on providing specifications in which a piece of code performs some specified predefined actions. Our problem is somewhat different, in that we need to be able to prove that a system is free of unwanted side effects, such as the leakage of information or the halting of operations. The latter problem is particularly challenging, since it is well known that the halting problem is generally undecidable (see Chap. 5). A second challenge is that many of the existing results of formal methods relate to source code. As made clear in Chap. 4, it is the machine

code that needs to be verified. We will therefore need, not only scalable tools to help programmers construct correctness proofs of the source code, but also compilers that create machine code and translate proofs from source code to machine code. Finally, the production of proof-carrying code is highly resource intensive. The approach is therefore most likely only applicable to a relatively small trusted computing base (see Sect. 2.6).

The verification of hardware chips is, however, challenging in a different way. Whereas the development chain from an algorithmic description of the hardware down to gate-level descriptions can be framed in by formal methods, the actual production of the chip cannot. Furthermore, as described in Sect. 6.7, determining what logic was actually placed on a chip is no easy task. Whereas formal methods seem to be an interesting prospect in solving the software side of our problem, it is hard to see how it can help us prove that malicious functionality is not being included into hardware in the production phase.

12.2.3 Heterogeneity and Containment

The most potent approach to handling untrusted electronic equipment is through containment and heterogeneity. This basically means that systems must be constructed to the extent possible so that no code and no piece of hardware is universally trusted. Unfortunately, this field has received limited attention, as discussed in Chap. 11.

The strength of the approach is that it holds the promise to handle all sides of the problem. Disjoint pieces of equipment can handle disjoint pieces of information, so that damage through information leakage is limited. Whenever a kill switch is triggered, rendering a piece of equipment out of service, other pieces of equipment can take over. Once it has become clear that parts of an infrastructure can no longer be trusted, software can be replaced and hardware isolated for later replacement.

Another strong aspect of this approach is that it can be applied to the entire technology stack. Intellectual property integrated onto a chip can be controlled and contained by other pieces of intellectual property, pieces of software can be controlled and contained by other pieces of software, and entire devices can be controlled and contained by other devices.

Still, there are problematic aspects to the state of the art in this area. The first is that most methods currently applicable were developed for the purpose of fault tolerance. This means that much of the work assumes that faults strike in an arbitrary fashion and that, at a given time, there will be few faulty components to handle. These assumptions will not necessarily hold when there is an intelligent adversary and this adversary controls a large number of components in your system. Another problematic aspect is that of cost. Duplication of functionality is costly at all levels, and heterogeneity itself will significantly increase the cost of maintenance. Finally, the containment of malicious actions is not a trivial feature to build into a system. It will require a great deal of research to understand how to do so and most likely its implementation will come at the cost of loss of performance.

Even if many problems are involved in handling untrusted vendors through heterogeneity and containment, we still see these as the most promising way forward. Unlike other conceivable approaches, there are no important parts of the problem that we can identify upfront as impossible to handle. In particular, when we are challenged by possible kill switches implemented in hardware, it seems that heterogeneity and containment are the only viable path that can lead to a solution at all.

12.3 Concluding Remarks

Industrialized nation states are currently facing an almost impossible dilemma. On one hand, the critical functions of their societies, such as the water supply, the power supply, transportation, healthcare, and phone and messaging services, are built on top of a huge distributed digital infrastructure. On the other hand, equipment for the same infrastructure is made of components constructed in countries or by companies that are inherently not trusted. In this book, we have demonstrated that verifying the functionality of these components is not feasible given the current state of the art.

The security implications of this are enormous. The critical functions of society mentioned above are so instrumental to our well-being that threats to their integrity also threaten the integrity of entire nations. The procurement of electronic equipment for national infrastructures therefore represents serious exposure to risk and decisions on whom to buy equipment from should be treated accordingly. The problem also has an industrial dimension, in that companies fearing industrial espionage or sabotage should be cautious in choosing from whom to buy electronic components and equipment.

Honest providers of equipment and components see this problem from another angle. Large international companies have been shut out of entire markets because of allegations that their equipment cannot be trusted. For them, the problem is stated differently: How can they prove that the equipment they sell does not have hidden malicious functionality? We have seen throughout the chapters of this book that we are currently far from being able to solve the problem from that angle as well. This observation implies that our problem is not only a question of security but also a question of impediments to free trade.

Although difficult, the question of how to build verifiable trust in electronic equipment remains important and its importance shows every sign of growing. The problem should therefore receive considerably more attention from the research community as well as from decision makers than is currently the case. The question has implications for national security as well as for trade and is therefore simply too important to ignore.