Proven Security for the Internet of Things

To be presented at the Embedded World Conference 2016

Dominique Bolignano
Prove & Run
Paris, France
dominique.bolignano@provenrun.com

***Abstract*— The large-scale deployment of the Internet of Things will not be possible without resolving current security issues and challenges. We believe this can be addressed using a few key security software components and will illustrate this using representative examples drawn mainly from the connected car use case.**

1 INTRODUCTION

The Internet of Things (IoT) is gaining traction and will probably accelerate its deployment in the future. Cybersecurity (referred to as "security" in this paper) issues have begun to appear in several areas of the IoT (connected cars, smart grids, smart homes, smart offices, smart cities, avionics, adaptive maintenance, Industry 4.0, etc.). Such security problems will tend to grow at an even higher rate than the IoT itself as hackers will also benefit from more profitable business models coming from higher volumes, higher value of individual objects involved, more IoT features, etc. The problems are also very likely to spread across to other areas of the IoT (home medical assistance, medical equipment, production on demand, etc.) until quickly and critically hampering the successful deployment of the IoT.

This paper will start with a review of the main components of the security chain. We will demonstrate that the main sources of security issues can be attributed to faulty software where errors in the software architecture, design, implementation or configuration of an IoT system create vulnerabilities that can be exploited to mount a successful attack. The challenge is therefore to produce software that is as close as possible to "zero-bug": this paper will explain how this challenge can be met in a cost-effective way. As an example, this paper will present how ProvenCore and ProvenVisor, two highly secure components of Prove & Run's range of security bricks have been designed to reach the highest level of security.

2 SECURITY: INS AND OUTS

Many believe that security boils down to (properly) using a few basic ingredients: cryptographic protocols (such as signature and encryption), secure elements, etc. However this is just a small part of the whole security story.

The strength of a security chain is known to be that of its weakest link. In our opinion, it can be further claimed that there are basically four main kinds of links:

1. Cryptographic algorithms,

2. Cryptographic protocols,
3. Technologies and know-how to resist physical attacks,
4. Technologies and know-how to resist logical attacks.

The first three kinds of links have reached a level of maturity to the extent that the cognizant and well-trained security architect will be able to bring them to the required level of resistance. However the last kind of link (resistance to logical attacks) has become the weak link in new IoT architectures: this paper will focus on showing how to strengthen this link.

2.1 Cryptographic Algorithms

Cryptographic algorithms can either be symmetric key-based (such as DES, Triple-DES, AES) or public key-based (such as Diffie-Hellman, RSA, or elliptic curves). These algorithms, which are at the heart of cryptography, have indeed benefited greatly from extensive and successful scientific research and expertise. As a result, relevant know-how and secure implementations are now widely available. The resistance of various cryptographic mechanisms has been quite well understood and documented. Security architects will first carefully choose the cryptographic algorithm to use. Then, depending on the threat model at stake, they will either rely on cryptographic hardware co-processors to implement the selected cryptographic algorithms they selected, or on available cryptographic expertise to implement the algorithm and their countermeasures in software. They may also use a combination of software and hardware measures.

2.2 Cryptographic Protocols

The second kind of link, i.e. cryptographic protocols, is also now very well understood. Cryptographic protocols (such as Kerberos, IPsec, TLS, FIDO, and many others) have benefited from extensive advances in both research and industrial applications. Many hacks reported at security conferences are due to improper use of available state-of-the-art validation techniques. In particular, formal methods now allow researchers to verify that the protocols at stake are free of flaws in a cost-effective manner. This is typically done using deductive or model-checking techniques ([4], [5], [6], [7], [8], [9]). More accounts on formal methods will be given in a dedicated section further in this paper.

Even if the main security challenge with cryptographic protocols is the abstract protocol itself, as described previously, some flaws have also been reported in widely deployed implementations. Such flaws in implementation could also have been avoided by carefully using low-level validation techniques that are now available. These low-level validation techniques involve validation and code inspection, as well as static analysis techniques, a basic form of a formal method. Such state-of-the-art validation techniques can indeed be used without increasing the cost or the duration of the development process, and the relevant know-how is widely available.

2.3 Technologies and Know-How to Resist Physical Attacks

Physical attacks, i.e. attacks that can be semi-invasive or invasive, exploit the physical properties of the targeted component, which is a usually a chip (for example [1], [2] and [3]). Semi-invasive attacks involve power analysis, UV EEPROM or Flash erasure, laser glitching and laser-assisted power analysis. Invasive analysis usually involves the removal of the chip package, and typically the use of heavy machinery such as focused ion beamed (FIB) workstations.

The technologies and know-how to resist such attacks have benefited from decades of advances in various security-related industries: the smartcard industry, as well as the content protection industry (e.g. pay TV), the military industry, the payment terminal industry and even the Hardware Security Module (HSM) industry. These technologies and know–how are widely available but have a per-unit cost which limits their use, at least with the current state of art. Securing a chip at the hardware level indeed adds many constraints that significantly raise the cost of the chip. For example, the communication buses between the processor, its memory and the other peripherals need to be protected either physically (a technique used for smartcard chips), or cryptographically (which involves more hardware but also costly management phases, such as personalization).

Thus when it comes to larger embedded devices that are deployed in millions of units, such technologies are either restricted to dedicated microcontrollers with few peripherals, or to very limited sections of larger microprocessors (e.g. the secure boot mechanism). For this reason these technologies can usually only be used for very limited sections of the secure part of modern IoT architectures: for example they will be used to secure the root of trust, the key store, cryptographic operations, or more generally for basic security transactions (as found on Trusted Platform Modules, i.e. TPM, personal HSM[1] , etc.).

2.4 Technologies and Know-How to Resist Logical Attacks – The New Challenge

Resisting logical attacks has been until recently the easy and marginal part of the security challenge. This is because the hardware components to secure were both very small and had very small attack surfaces. The smartcard is a very good and representative example of such situations: the hardware features are simple (very few peripherals and interruptions to handle, simple cache mechanisms, etc.) so as a result the corresponding OS is very simple, which in turn exposes a very small attack surface compared to modern operating systems. The latter fact is mainly due to the very basic communication means used by such devices and to the stricter (technical and organizational) constraints placed on the applications that are loaded on their OS.

[1] The HSM terminology is used here with different meanings in different fields: we use HSM in its traditional and original meaning (for servers) and use “personal HSM” for its usage in the automotive industry.

But this situation is radically changing. An analysis of the attacks reported in conferences such as Black Hat 2013 to 2015 ([11], [12], [13]) shows the importance of logical attacks. Hackers typically exploit errors to break into systems: low-level implementation bugs, protocol or specification flaws, design, configuration or initialization errors, violations of organizational security policy, etc.

2.4.1 Mobile Phone Security

The mobile phone revolution triggered the change when mobile phones started to be used for more than just calling and texting. The new security needs involved were mainly driven by secure transactions or secure processing requiring the involvement of more peripherals than secure elements and smartcards could possibly handle (at least at an acceptable price). For example, peripherals for the user interface needed to be trusted so as to be sure in particular that "what you sign is what you see". As a result, the part of the device to secure, more precisely the scope of the software and hardware that needed to be trusted, the so-called Trusted Computing Base (TCB), was extended to include the OS kernel, and because of that, was becoming too complex to secure.

In an attempt to cope with the new security needs coming from the mobile phone industry, the author introduced the concept of the Trusted Execution Environment (TEE) in 2001 and pushed for its standardization (through OMTP and then GlobalPlatform) and adoption through his previous company[2]. The TEE concept was mainly driven by the idea that taking a large OS such as Linux or Android out of the scope of the TCB and replacing it with a secure, small and carefully designed microkernel was a major step toward building an architecture that could resist logical attacks. However the TEE concept that is now widely deployed on smartphones was only the first step towards addressing new security needs.

2.4.2 IoT: Logical TCB and Logical Attack Surface Becoming Too Large

With the IoT, the TCB of devices involved in connected architectures becomes much wider. Devices to secure are indeed more diverse than 'just' mobile phones; they handle a larger number of various peripherals to be secured. They also include complex and large software stacks, with rich OSs and kernels, some of which are essential to security. The IoT is thus taking the need for security into a new era where sub-systems and peripherals that need to be secured are complex and have a very large surface of attack. In other words the scope of the TCB significantly expands and becomes the new weak link and the one we refer to here as the "resistance to logical attacks".

In addition, IoT use cases create new situations where assets that need to be protected are not just virtual, but also physical: goods, infrastructures, lives, etc. The effects of large-scale attacks are no longer limited to tampering with crucial data or creating improper transactions (issues which can usually be avoided or traced back with proper risk management processes), but could also potentially include irremediable physical destruction. The prospects and business model for attackers become much more attractive. The risk for services and industries in many cases may become incommensurate.

The car industry provides a number of relevant use cases, which we believe are very representative of the IoT security challenge, with the advantage of being very well documented. Most cars are now connected in one way or another.

- Let us first consider the hypothetical use case of a pay-as-you-go service (toll road, parking, etc.). For such a use case a secure element (e.g. a TPM, or a personal HSM) might be just enough to provide an adequate level of security. We are still in the conventional world where risk management, potentially reinforced by the use of secure elements, is typically considered good enough. Risk management and organizational security measures (such as immediately reporting a stolen secure element or a stolen car) are enough to limit fraud to an acceptable level.
- Now consider another use case of an active car tracking system designed to geo-locate a car remotely. This use case drags at least two additional peripherals into the scope of the TCB: the GPS module and an independent communication link with the outside (such as a cellular modem). Such peripherals cannot be cost-efficiently controlled by a secure element. The TCB hence needs to include a richer OS than those found on secure elements, which by construction will lead to a much broader attack surface.

In this case, we need to distinguish the physical attack surface from the logical attack surface. When designing modern open IoT architectures, trained security architects will almost always be able to keep the physical attack surface small but regardless of the level of their skills, they will not be able to prevent a large logical attack surface.

In order to go into more detail we also need to distinguish the different phases of an attack here. There are usually two phases to consider for an attack: (1) the identification phase where the attack is identified and prepared and (2) the exploitation phase which corresponds to the use of the analysis, data, technique and tools defined during the first phase. The investment that can be reasonably made by hackers in the first phase is much higher and the tools much more sophisticated than the one that can be reasonably applied to each single attack in the second phase.

In our car tracking system example, putting the tracking box in a place that is physically difficult to access might be an efficient countermeasure for the exploitation phase but certainly not so for the identification phase. Thus if physically tampering with one tracking box does not help to break into other boxes, direct exploitation attacks will be the main challenge for the hacker.

So in other words, the identification phase may involve very sophisticated hardware attacks, but the assets that need to be protected in this phase may be kept to a very limited list (typically some keys). For this phase we are still in the conventional security challenge. The new challenge is found in

[2] Trusted Logic S.A., that gave rise to the Trustonic joint venture, Selected announcements: http://www.arm.com/about/newsroom/5688.php, http://newscenter.ti.com/index.php?s=32851&item=127309

the exploitation phase where the scope of the logical TCB (referred to as TCB in the following sections) and the corresponding logical attack surface (logical attacks are referred to as "attacks" in the following sections) are becoming much larger. Our discussion here mainly applies to this new challenge. In this exploitation phase remote attacks that can be raised to a large scale are the main problem.

Coming back to our connected car use case, the security architect will typically try to keep the modem stack outside the TCB. This is possible as will be discussed at a later stage. But even if the security problems due to the software modem stacks are handled in that way, we are still in a situation where an OS kernel has to handle on one hand a sophisticated communication channel to the outside world, and on the other hand a GPS module. In that case, the OS kernel will be part of the TCB and would need to be secured.

A sophisticated car tracking system will typically need to have access to other peripherals as well, such as locks, door sensors, gyroscopes, etc. in order to make smarter and better decisions. For cost considerations, all these sensors will typically be shared with some other Electronic Control Units (ECUs) through the car's communication buses. But regardless of whether there is one ECU or multiple ECUs managing all these peripherals, ECUs have to use OSs and kernels whose complexity becomes the problem.

2.4.3 The Challenge of Securing OSs and Kernels

Various public databases (such as [10]) provide statistics on public bugs or vulnerabilities on all kinds of software. These databases clearly show that existing OSs and kernels are prone to a great number of errors and weaknesses, no matter who writes them, and no matter how long they have been in the field. For example, new errors are still being reported in the thousands every year on "well-known" systems such as Linux.

This situation is basically due to the inherent complexity of such OSs and kernels, which increasingly rely on complex and sophisticated hardware. OSs and kernels are hugely complex because of the need to support various kinds of peripherals (interruption handling becomes increasingly difficult), performance objectives (e.g. complexity of cache management), resource consumption issues (e.g. need for sophisticated power management), etc. This complexity grows with time, with new IoT architectures and when it comes to real microprocessors (as opposed to microcontrollers).

Therefore, the issue boils down to being able to produce and demonstrate that the OSs and kernels that are part of the TCB are as close as possible to "zero-bug", i.e., free from errors, either in their design or implementation, that could be potentially exploited for logical attacks. As will be seen in the following sections, such errors may be considered errors in what we will call the "security rationale".

3 ADDRESSING THE NEW CHALLENGE

We believe that the challenge posed by logical attacks can be addressed using a focused combination of principles:

- Use a state-of-the-art security methodology that will in particular clearly set out what we call a complete "security rationale" for the targeted system.
- For the OSs and kernels that are included in the TCB, apply deductive formal methods to get as close as possible to "zero-bug" for these complex software components.
- Reuse proven Commercial Off-The-Self (COTS) software components to control the cost of developing a secure product.

3.1 Security Methodology

In this paper we address security technologies and know-how almost exclusively, deliberately omitting security methodologies. Security methodologies are of course essential and it is very important to use existing ones.

We assume implicitly in our discussions that state-of-the-art security methodologies as imposed by the Common Criteria framework are followed. In particular, we assume that for each architecture and use case, a proper risk analysis has been conducted and a threat model has been provided, that the security requirements, and more generally a proper security target, have been defined and are used as a guide both by the security architect and for the security evaluation. It is worth noting that such documents can be reused from one evaluation to another to further reduce cost. Prove & Run's COTS come for example with Common Criteria EAL7-ready documents which can be used by any integrator to rapidly go to any level of security certification they wish. Prove & Run's formally proven COTS have been proven down to the code itself and go in fact far beyond what is required by the highest levels of CC certification.

We believe that the IoT does not cause any global disruption that would render these security methodologies inapplicable or incomplete, except for the part that relates to the exact scope of what needs to be formally verified. We are addressing this in the rest of the section.

It is generally agreed that a clear security rationale explaining the hypotheses, conditions and reasons why the security architecture meets the desired security properties needs to exist, either implicitly or preferably explicitly. In the Common Criteria framework for example, this security rationale is split up into a few dedicated documents such as the security target, and various other correctness documents. For the highest levels (EAL7 in particular), these documents must be described using a formal method. But there are no requirements for the scope of the formal proof to go all the way down to the source code.

We believe that neglecting the need to go down to the source code is a weakness when targeting new IoT architectures, as some errors may exist (and will exist) if the complex part of the security rationale is informal and cannot be checked by a tool (i.e. a formal prover). In our opinion, an additional security level, i.e., typically an extended EAL7+, should be considered and should involve "full proof down to the code" for the most complex part of the security rationale,

and this is what we are targeting for the Prove & Run secure COTS. We will call this recommended security level that we propose the "enhanced formal proof-based security level".

One way of evaluating a given security methodology such as the one we propose is to evaluate a posteriori whether for each reported attack, the said methodology would have allowed preventing the said attack. We conducted this exercise for the proposed approach for each attack reported at the Black Hat conferences from 2013 to 2015, and can confirm that the proposed methodology would have prevented all such attacks, except for a few that corresponded to the exploitation of previously unknown hardware behaviors, such as the Rowhammer attack presented in [17]. Each of these new kinds of attacks would nevertheless have been very rapidly taken care of, and integrated into the revised security rationales.

3.2 Using Deductive Formal Methods for OS Kernels Included in the TCB

To the extent that the OS and kernel are included in the TCB, they need to resist hackers who will try to exploit bugs and weaknesses, i.e. errors in the security rationale. These software parts need in particular to be as close as possible to "zero-bug" with proven and auditable compliance with security properties.

Traditional software engineering techniques such as exhaustive testing or code inspections are clearly no longer sufficient for providing the level of assurance needed to secure complex open kernels. This is due to the fact that there are too many different situations for a kernel designer or tester to consider and no real methods to review the quality of such kernel code systematically, beside the use of proof techniques.

Instead, we believe the only valid response to such complexity is a special class of formal methods, which are known as "deductive techniques" or "proof techniques". There are indeed various kinds of formal methods, all using rigorous techniques (typically mathematical and/or logical techniques) to prove corrections or compliance with security properties. There are usually three distinct categories of formal methods:

1. Model-checking techniques, which can only be applied on simple models that can be exhaustively checked. They are not applicable on real kernels, or only on oversimplified abstractions of them.
2. Static techniques, which can be applied on real code, but can only check some limited low-level properties (such as the absence of buffer overflow, or the illegal use of pointers). They are certainly useful, but are far from sufficient, as they cannot address the high-level properties at stake (integrity, confidentiality, isolation, etc.).
3. Deductive (or proof) techniques, which are the most expensive to apply, as formal proof cannot be done completely automatically. But they are the only ones which allow proving virtually any security or safety property.

Prove & Run has developed a formal language and a dedicated environment (i.e. ProvenTools) that takes advantage of decades of research and advances in the scientific field of formal methods in order to make this formal verification process more efficient, and also to increase the level of trust ([18] and [19]). Using this environment, we can fully prove the most complex parts of any TCB, in particular OSs and kernels, in order to leave hackers with no vulnerabilities to exploit, even in large, complex and open systems.

Some parts of the TCB such as the secure boot, secure configuration software or security applications do not necessarily have to be proved. This is due to the fact that they are quite sequential and that their complexity is amenable to conventional validation techniques. They would also be, and this is probably not a surprise, the easiest to prove, if needed. So the decision on which techniques to use for the validation of these "easy" parts is up to the security architect and/or the security evaluation and certification authorities. These parts, depending on the situation, may or may not be formally verified, but in any case they will have to be brought to the right level of trust.

3.3 Reusing Proven Building Bricks Across Industries

Producing a security-proven OS kernel using deductive formal methods is not a straightforward and simple task, even if our formal language and dedicated environment facilitates this. In order to match the cost requirements of industrial deployment, the possibility of reusing the same software components across industries and pooling development costs is critical.

We believe it is important to provide off-the-shelf reusable proven kernels, i.e. COTS, that will simplify and lower the cost of securing IoT architectures, making them fast to build and evaluate. In line with this vision, Prove & Run has already developed three such off-the-shelf kernels, that have been formally proven and are ready to use and certify:

1. ProvenCore, a fully proven and secure OS kernel, POSIX-Conformant and dedicated to microprocessors. ProvenCore has been optimized to run in the secure part of ARM Cortex-A TrustZone-enabled microprocessors.
2. ProvenCore-M, a fully proven and secure OS kernel, providing a large subset of the same POSIX APIs and dedicated to microcontrollers. It has also been optimized to run in the newly announced secure part of ARM Cortex-M TrustZone-enabled microcontrollers.
3. ProvenVisor, a modern fully proven and secure hypervisor, providing similar features as Xen[3], but with a much smaller TCB.

We believe that by combining these basic security bricks, we can secure virtually any IoT architecture at a very high level of security with lower cost and in less time. We will illustrate this in the following part of this paper.

[3] http://www.xenproject.org

4 SECURING THE IOT WITH A PROVEN AND SECURE OS KERNEL

In this section, we will take as a supporting and illustrative example one of the most complex IoT use cases, i.e. a modern connected car, and show how to apply the steps presented above to resolve the security challenge using a proven and secure OS kernel.

This (connected car) use case features most of the complexity of current IoT architectures and has the advantage of being well documented in various research papers. We will in particular use as an illustration two recent and well-documented attacks that were reported and described in 2015. These attacks will be respectively called the "BMW attack" presented in [20] and the "Jeep attack" presented in [21] in the following section. These attacks are also very representative of the ones that could happen in other domains of the IoT (smart grids, smart homes, smart cities, Industry 4.0, etc.) once hackers start taking an interest in them.

4.1 Identifying the Most Problematic Security Issues

4.1.1 Remote Attacks exploit Entry Points

Modern cars include many embedded systems in the form of ECUs. The number of ECUs is now close to a hundred in high-end cars. They are linked to each other through a bus, or bus architecture, which can be more or less complex depending on car models. Traditionally the bus type is CAN but newer bus architectures involve other technologies such as Ethernet for example.

Modern cars are connected to the outside world through the remote connectivity features of some of these ECUs. ECUs that have direct access to the outside world typically include the Telematic Control Unit (TCU), the car infotainment system, and potentially other ECUs, as shown in the Jeep attack. Such connections indirectly expose the other ECUs to remote attacks.

In the rest of this paper, we will call such devices "entry points". These entry points can communicate with the outside world using potentially various technologies such as Wi-Fi, cellular communications and Bluetooth, or use communication links that are part of the Radio Data System, the Tire Pressure Monitoring System, the Remote Keyless Entry System or the Passive Anti-Theft System.

For the IoT in general, remote attacks will typically exploit the communication links of the entry points as well as a combination of the device's weaknesses (more typically its software stack's weaknesses), so as to gain some control over the device (typically to perform some form of privilege escalation attack). The hacker will then use this compromised device as a new base from which they will try to compromise another device connected to it, repeatedly until they get to the target device(s) they want to compromise. In many cases (and in particular in the connected car use case) there is typically more than one entry point, but also more than one communication link that can be used and also more than one set of weaknesses that can be exploited. Both the Jeep and BMW attacks are representative illustrations of such remote attacks through entry points.

4.1.2 Attacks Exploit Errors in the Security Rationale

All the attacks reported so far for connected cars and more generally for the IoT can be seen as exploitations of some kinds of errors either in the architecture, hypotheses, protocols, design, implementation, initialization, or configuration, etc. A more technical way of characterizing such errors is to recognize that they can all be seen as errors in the "security rationale" as defined previously. This also means that all such errors could have been avoided (or at least would have been reduced to a marginal number) if the security rationale had targeted our proposed "enhanced formal proof-based security level" as defined in section 3.1.

In their description of the BMW attack, the authors present how they can remotely open the doors of virtually any recent BMW car through their ConnectedDrive system (a component that, for its compromised part, would be typically part of the TCU on more recent car architectures). They identify six security vulnerabilities. Most of these vulnerabilities correspond to high-level errors, i.e. errors relating to parts of the security rationale that handle the high-level descriptions of the architectures (i.e. specification, high-level design, configuration, initialization, organizational security policies, etc.).

One of these errors is the fact that all cars use the same symmetric keys, so an attack performed on a given car (in the identification phase) can easily be used to perform attacks on other cars (i.e. can easily be used in the exploitation phase). A basic security evaluation would have easily pointed this out as a problem. The weakness would have been obvious at least at the time the security rationale was written, provided of course that such rationale has been clearly expressed and with enough detail.

4.1.3 Errors in Complex Sub-systems such as the Kernel are the Real Challenge

As previously discussed, OSs and kernels, when developed with conventional development methodologies, are prone to a large number of errors, which are the real challenge when such OSs and kernels are included in the TCB. Anyone who has tried to convince himself or herself that a monolithic kernel for a modern processor is free of bugs or of security weaknesses can easily confirm that.

This is first illustrated with the BMW attack. Even if all the identified vulnerabilities had been removed, and the authors implicitly suggest how this could be done, a much harder security challenge would remain since the TCB would still include complex sub-systems, including at least a kernel and a modem software stack (probably implemented as a kernel as well, and in any case with the same level of complexity as a kernel stack).

Turning now to the Jeep attack, which is a bit more complex and representative as it involves a richer class of problems, the entry point is the infotainment head-unit. The head-unit includes two chips, an OMAP DM3730 chip (an

application processor) that is the one communicating with the outside world, and a Renesas V850 chip (a microcontroller) that is linked to the OMAP chip on one side and to the CAN bus on the other side. So the real entry processor here is the OMAP chip, which runs QNX, an operating system commonly used in cars.

- The first step of the reported attack consists of compromising the software stack of the entry OMAP chip first. This is easily done as the QNX-enabled software stack contains an internal software bus, i.e. a D-Bus, which is accessible by any device on the same cellular network (i.e. Sprint in the US) through port 6667. This allows remote devices to issue commands to be executed by the QNX-powered OMAP chip. This problem is also a high-level problem, comparable to the one found through the BMW attack. But it is only the first step in the attack, as the OMAP chip will then be used to compromise the second chip of the head-unit, the Renesas V850 subsystem.
- The second step of the attack is more sophisticated as it involves finding and exploiting a nontrivial design bug in the V850 chip's reflashing/booting system, but it is still a high-level attack, which can be found by inspecting/reviewing the design of the reboot/reflash mechanism itself. Hackers ordinarily have no access to the security rationale (when it exists) and might have to tediously reverse engineer the design in order to identify such errors, but it is still possible. Once they have compromised the second chip by adding new commands to send arbitrary messages on the internal CAN bus, hackers can then attack any of the target devices, for example the one in charge of turning the wheels, or the one in charge of accelerating or braking. The hackers will do this by sending fake messages that simulate commands from the parking assistance unit or by the cruise control unit.
- The third and last step of the attack is still of a high level and even much easier to identify as the CAN protocol does not implement any kind of authentication. Solutions exist to cope with such authentication problems (such as [22]), but they are high-level and leave the implementer with the tough challenge and responsibility of securing the software stacks of such ECUs and in particular of their kernel parts. Even security requirements such as one of the German Secure Hardware Extensions (defined in [23]) do not cover this tricky area.

As shown in the Jeep attack, the authors had a lot of similar weaknesses to exploit, and they seemed spoiled for choice. But even if all of these high-level weaknesses were addressed using state-of-the-art security methodologies and techniques (which we believe is a must), significant issues would still remain.

Let us consider as an example the case of the OMAP chip "entry point" used in the Jeep attack (a similar analysis could be conducted with the other chips compromised in this attack). We shall also assume that all high-level problems have been addressed, and in particular in the case of the OMAP chip, let us assume that the problematic D-bus has been protected from the outside world: if remote commands are not needed, by making the D-bus inaccessible, or if remote commands are needed, by filtering commands and parameters received from the D-bus. The fact would still remain that QNX (or any other large OS such as Linux or Android in other IoT cases) would still be in the TCB and would therefore be extremely difficult to secure due to its inherent complexity.

4.2 Using a Proven and Secure OS Kernel for Security

In the case of the BMW and Jeep attacks, the OSs and kernels included in the TCB would need to be secured. Now, formally verifying such OSs and kernels is theoretically possible but would be extremely expensive. On systems such as Linux or Android, where new features and drivers are added on an ongoing basis to cater to new requirements, there would also be an additional time-to-market problem, as even if cost was not an issue, it still takes time to formally prove such complex systems.

At Prove & Run, the way we propose to address this is as follows. Using a separate proven secure OS kernel, i.e. in our case ProvenCore, to handle peripherals that need to be secured and to run secure applications in a way that allows us to:

- Keep the normal OS (for example QNX in the Jeep example, Linux or Android on other head-units or other devices, or just any other proprietary OS or RTOS) and take advantage of all its features and benefits,
- Push the normal OS outside the TCB, so that any error in the normal OS cannot be used to compromise the TCB; this is done by using a proven OS to perform security functions
- Design the proven OS in a way that makes it generic and secure enough to be used as COTS in virtually any IoT architecture.

We describe how this can be done on ARM architectures that account for the vast majority of the market, but the same can be transposed to other architectures. On ARM architectures and in particular on Cortex-A and Cortex-M, that is on ARM microprocessors and microcontrollers, there is a security mechanism called TrustZone that provides a low-cost alternative to adding a dedicated security core or co-processor, which splits the existing processor into two virtual processors backed by hardware-based access control. This lets the processor switch between two states, i.e. two worlds, typically the "Normal World" on one side and the "Secure World" on the other side. TrustZone access control mechanisms together with its software stack (called the Monitor) act as an extremely small and security-oriented dual-VM asymmetric security hypervisor that allows:

- The so-called Normal World to run on its own, potentially oblivious to the existence of the Secure World and,
- The Secure World to obtain extra privileges such as the ability to occupy some part of the memory, as well as some hardware peripherals, exclusively visible and accessible to itself.

In the proposed architecture, the proven secure OS kernel, i.e. ProvenCore in our case, is used as the kernel for the Secure World, and the rich but error-prone OS (Linux, Android, etc.) is placed in the Normal World. The peripherals that are to be secured are configured, typically during the secure boot, to be accessible and visible only to the Secure World. In the Jeep case we thus would:

- Place the QNX OS and its software stack in the Normal World,
- Place our proven secure kernel in the Secure World,
- Make all peripherals that allow external and internal communication accessible only from the Secure World. Depending on the peripheral or communication links, this can involve reserving some chunk of memory space (i.e. buffers) to the Secure World only and/or reserving the peripherals themselves.

At this point the Normal World software stack has not yet been modified[4] and cannot work as such as it is now in a world where the peripherals that have been secured are not visible and cannot be accessed anymore. We thus need to replace the relevant drivers in the Normal World with proxies offering the same features. The modifications to the original Normal World stack are very small: they merely involve rewriting some drivers. These proxies will forward the requests to the Secure World where at least two secure applications will be added per peripheral:

- The actual driver that has now been moved to the Secure World and can be executed in user mode in the Secure World (i.e. as an application of the proven secure OS kernel),
- An application in charge of making decisions, e.g. performing low-level or high-level filtering.

4.3 Illustration: Securing a TCP/IP Entry Point

Let us consider as an illustration the TCP/IP entry point that was used for the Jeep attack. A first obvious security measure to apply would be to filter communications with a low-level packet firewall. At Prove & Run, we have developed a firewall that can run on our ProvenCore OS kernel, and is compatible with the NetFilter firewall available on Linux. The physical part of the entry point is configured to be accessible only from the Secure World. The Normal World can no longer communicate without the help and permission of the Secure World. We thus replace the Normal World driver with a Proxy Driver, which forwards (and receives) all requests to (and from) the secure side. The actual driver and the actual firewall are placed on the secure OS, as illustrated in Fig. 1.

[4] We are considering here for the sake of clarity that we are hypothetically revamping the flawed Jeep architecture by reusing as much of the flawed software stack as possible with minimal modifications. The proposed resulting architecture should of course be better used as an initial one when designing a new security architecture.

We could for example use this secure firewall to secure forbidden ports, for example, port 6667 used for the Jeep attack. This would be a much more secure solution compared to the standard use of a firewall which is either part of the normal OS kernel (such as NetFilter) or a pure application siting on top of the normal OS. In fact, even if one uses a proper firewall outside the Secure World and configures it correctly, the normal OS will always be subject to attacks targeting bugs and weaknesses that exist in such large OS kernels. Of course, some of these weaknesses could be blocked by the firewall itself. However, examining a representative list of past privilege escalation attacks (such as [24]) will easily show that regardless of the firewall configuration, it will still be possible to choose some attacks from the list that can be applied and allow compromising the OS, and then easily disconnect the firewall.

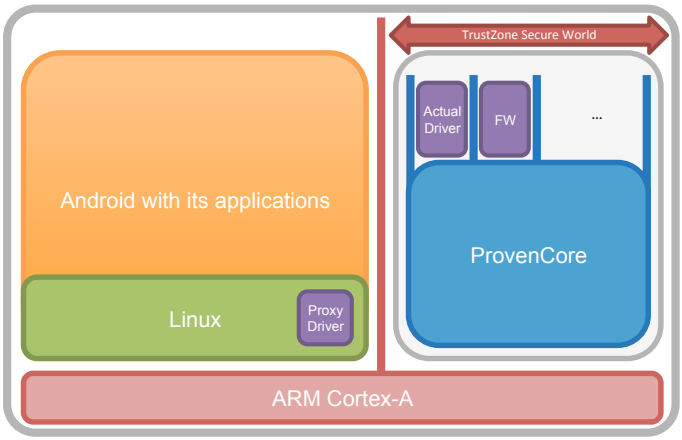

Fig. 1. ProvenCore-based architecture used to secure a TCP/IP entry point

The use of a secure low-level firewall on top of a proven OS kernel such as ProvenCore is necessary but still not enough, if certain ports have to be left open. Indeed, certain ports (6667 or others) or communication links will have to be left open by the firewall configuration, typically to perform certain functions that require open ports (e.g. Firmware update Over The Air, i.e. FOTA), or because some applications will require access to external data (e.g. a GPS system requiring traffic information).

Let us assume for example here that port 6667 has to be left open, that the external cellular network is not secure and that we need to use the internal software bus that the attacked Jeep was featuring to allow certain remote commands. In this case, we also need to filter higher-level commands (and not only packets) to authorize only a restricted list of commands with potentially some constraints on parameters. This can typically be done as an additional high-level filtering application running on top of our proven and secure kernel. We will typically also want to use a virtual private network (VPN) and authentication for such secure commands.

The complete TCP/IP software stack could of course be considered a secure application and placed on the secure side, but this is neither necessary nor a good thing. Only the part that needs to be trusted must be included in the TCB, in order to keep the TCB (and thus the security rationale) as small as

possible. For example, the sensitive software components of a VPN represent only a very small part of the software stack and TLS will handle authentication, initial session key negotiation, encryption and decryption. Many other parts of the TLS layer and of course all other lower communication layers can be left out of the TCB.

We have developed such secure applications that can be added on top of our ProvenCore OS kernel in order to secure IoT architectures quickly and at a lower cost, potentially a posteriori. Such secure applications include the TCB part of TLS, FIDO, OpenVPN, as well as various other firewalling and filtering applications, various FOTA systems, application management, etc. Certain specific or proprietary applications can be further developed and added for a particular architecture. The filtering of the virtual bus commands of the Jeep would be an example of such proprietary secure filtering applications that would need to be developed specifically for this architecture and that would execute on ProvenCore.

There are other ways to secure the same architecture with the same secure COTS, the important thing being for it to be done in a way that results in a highly secure architecture, based on a very small TCB, which is most often achieved by reusing existing COTS. In addition, it does not impose strong constraints on other, non-TCB parts of the architecture so that it can also be used to revamp existing architectures. The TCB also has the advantage of being formally proven for its complex part, and can be secured and certified at the highest level of certification in order to provide the highest level of trust.

On a car architecture, this security approach should be applied at least on all entry processors, typically the infotainment head-unit and the TCU on modern cars, and potentially the car gateway which, if it exists, may include the OBD-II maintenance connector. This provides a very strong first line of defense. It is of course possible to secure other processors using the same principles and technologies, to provide defense in depth.

5 The Need for a Proven and Secure Hypervisor

We have been analyzing and securing a wide variety of IoT architectures and have always managed to secure the proposed architectures at a very high level of security and very efficiently, using only a proven secure OS kernel such as ProvenCore for the TCB. In some rare cases though, the need arose for a proven and secure hypervisor to be used in conjunction with a proven and secure OS kernel. For this reason, we have developed and formally proved for security within Prove & Run a ready-to-be-certified hypervisor called ProvenVisor.

5.1 Hypervisors are not Magic Bullets but can be Nice Complements

Hypervisors often give false expectations in terms of security. Let us try to understand why.

In the Jeep example, on one of its products the head-unit provider, Harman Kardon, has now replaced the two chips that used to make up the head-unit with a single chip coupled with a hypervisor in order to host both software stacks on the same chip. While some believe this kind of architecture is a secure answer, this is clearly not the case. Using a hypervisor to reduce the number of chips is often advantageous in terms of cost, but cannot be an answer to security in itself, as virtual machines (VMs) will still communicate as the physical chips did, and therein lies the problem. These false expectations probably stem from the fact that a hypervisor provides separation and separation is usually good for security. But having separate chips also provides separation, the security of which is even easier to assess than that of a hypervisor. *In the very best case, a highly secure hypervisor used to reduce the number of physical processors will manage to bring multiple VMs onto the same chip without compromising security.* But for the same reasons as those explained for the secure OS kernel, such a highly secure hypervisor would become part of the TCB and would have to be formally proven for security.

Another reason for these false expectations probably comes from the fact that adding more processors may be a way of raising the level of security, so that using a hypervisor to virtually raise the number of processors raises the level of security by reducing the attack surface between the software stacks that are split among VMs. But this increased security would be genuine only if both of these conditions are met:

1. There is no communication at all between the various software stacks, i.e. the various VMs, or if the communications are secured using a secure OS and secure applications as described previously,
2. The hypervisor is really secure (and thus proven for security).

As an example, if the two processors of the Jeep head-unit had no means (direct or indirect) of communicating at all with each other, it would have been impossible to launch the described remote attack: the hacker would have been in a position to compromise the OMAP chip, but would have had no means of communicating with and compromising the second Renesas V850 chip. Replacing the two chips with one chip with a secure and bulletproof hypervisor would provide the same level of security as separate chips with no communication. But none of the above scenarios are feasible as communication links are in fact required between both chips(or both VMs).

More generally and in practice, many of the chips (or VMs) need to communicate with each other, and as soon as one opens a communication link between two such VMs, there is a risk that the corruption or compromise of one VM will be used to compromise the other one. The only way to secure this kind of architecture is to deploy some filtering or security applications. Such applications cannot be written on the bare metal or directly in a VM as otherwise they would incorporate some of the complexity of kernels and will be vulnerable unless formally proven. They cannot be written either on a normal OS as otherwise their foundations, i.e. the normal OS, would be weak and used to attack them. They will need to be executed by a proven secure OS kernel such as ProvenCore. Thus, when a hypervisor is necessary to reduce the number of physical processors, or to increase the number of virtual processors, a

high level of security can be achieved provided two security requirements are met:

- Such hypervisors, since they become part of the TCB, need to treated with the same level of security as any kernel that is part of an open TCB, i.e. they need to achieve the previously defined "enhanced formal proof-based security level,"
- They also need to be combined with a proven and secure OS kernel and its security applications that meet the same level.

This can be summarized in Fig. 2.

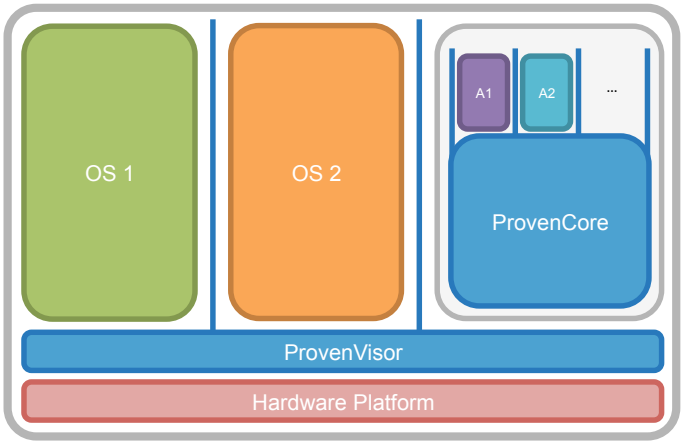

Fig. 2. ProvenVisor-based architecture

5.2 *Illustration: Securing a Radio Software Stack*

Radio software stacks are usually of a similar complexity to that of a kernel. They either need to be left out of the TCB or brought to this "enhanced formal proof-based security level". In most cases the first option is possible and is to be preferred, for the sake of cost and simplicity.

We can illustrate this with the well-known case of the cellular modem stack that can either be implemented on a separate baseband processor, or on a virtualized one. The fact that cellular modem stacks are error prone and easy to hack is quite well known. Many attacks that consist of exploiting errors in a modem stack for example have been publicized in [25]. Typically a fake base station is used to connect to the modem, and then to send invalid (out of range) messages so as to compromise the modem stack using known software errors and then to attack the OS that uses or cohabits with the modem stack.

The cellular modem stack is quite representative (both in terms of problem and solution) of the situation with the various kinds of radio software stacks, for various kind of technologies, i.e. cellular, Bluetooth, Wi-Fi, etc.

As previously mentioned, it is strongly recommended and usually possible to leave the radio software stack outside the TCB. To do so, we will use the same architecture again. If TrustZone is available, ProvenCore will be placed in the Secure World. Alternatively, ProvenVisor can be used to provide a secure VM. If the modem runs on a separate baseband processor, the achieved architecture is probably the one we want. If this is not the case, which means that the radio software stack runs in a VM on the application processor, it is recommended to replace the insecure hypervisor with a secure one, such as ProvenVisor. In any case, both ProvenVisor and ProvenCore can be further configured to enforce additional security policies (including but not limited to access control policies). As an example, ProvenVisor and/or ProvenCore may enforce a control policy to prevent the modem from accessing some of its hardware peripherals (e.g., a microphone) even when the modem becomes compromised.

5.3 *Securing More Complex IoT Architectures*

Complex IoT architectures involve sophisticated heterogeneous and multicore processors. In the previous discussions, relatively simple configurations were considered, which are still representative of a very wide range of commonly used IoT architectures.

Many other configurations are of course possible, both in terms of hardware configuration as well as for the TCB's precise role . As an example, ProvenCore can be placed in a separate physical core while still securing other cores. We have for example successfully tested this solution on NXP chips (the i.MX and QorIQ families offer security or multi-purpose cores that can be used to load a secure OS that will secure the other potentially heterogeneous cores).

It is also possible to change the role of the TCB and thus of the other components. For example, in our "revamped," secure version of the insecure Jeep architecture, we made the decision to not let any sensitive assets, such as confidential information, be managed in plaintext by the OS running in the Normal World. We could alternatively make confidential information available (for example, the position of the car and other types of information managed by the GPS system) to the Normal World but protect it from the outside world using the TCB itself. The normal OS would then execute in a sandbox managed by the TCB. This is just a matter of adjusting the configuration of the TCB.

6 CONCLUSION

In this paper, we have shown that IoT architectures create new security challenges that cannot be addressed with current security technology and methodology alone. However we have also shown that it is possible to take those IoT architectures to a very high level of security , in a way that is both compatible with industrial requirements and acceptable in terms of time-to-market and cost. This involves reducing the scope of the TCB, and using a combination of formally proven, secure and certification-ready COTS such as ProvenCore (an OS kernel) and ProvenVisor (a hypervisor). By combining these basic security bricks we can secure virtually any IoT architecture at a very high level of security with marginal and acceptable costs.

REFERENCES

[1] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," The Second USENIX Workshop on Electronic Commerce, Oakland, California, November 18-21, 1996

[2] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in M.Lomas et al. (ed.), Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997

[3] O. Kommerling and M. Kuhn, "Design principles for tamper-resistant smartcard processors," USENIX Workshop on Smartcard Technology, Chicago, Illinois, May 10-11, 1999

[4] D. Bolignano, "An approach to the formal verification of cryptographic protocols," CCS '96, New Delhi, India, March 14-15, 1996.

[5] D. Bolignano, "Formal verification of cryptographic protocols," CAV'98, Vancouver, Canada, June 28-July 2, 1998.

[6] T. Gibson-Robinson, A. Kamil and G. Lowe, "Verifying layered security protocols," Journal of Computer Security, vol. 23, no.3, pp. 259-307, 2015.

[7] V. Cortier, "Formal verification of E-voting: solutions and challenges," 3rd SigLog Newsletter, ACM Special Interest Group on Logic and Computation, vol. 2, no. 1, pp. 25–34, January 2015.

[8] C. Meadows, "Formal verification of cryptographic protocols: a survey," ASIACRYPT '94 Proceedings of the 4th International Conference on the Theory and Applications of Cryptology: Advances in Cryptology, Wollongong, Australia, November 28 - December 1, 1994, vol. 917, pp. 135-150.

[9] M. Abadi and R. Needham, "Prudent engineering practice forcryptographic protocols," IEEE Transactions on Software Engineering, vol. 22, no. 1, pp. 6–15, 1996.

[10] National Vulnerability Database. NIST. [Online]. Available: https://web.nvd.nist.gov/view/vuln/search [Accessed 15 Jan. 2016].

[11] Briefings, 2013. Black Hat Conference. [Online]. Available: https://www.blackhat.com/us-13/archives.html [Accessed 15 Jan. 2016].

[12] Briefings, 2014. Black Hat Conference. [Online]. Available: https://www.blackhat.com/us-14/archives.html [Accessed 15 Jan. 2016].

[13] Briefings, 2015. Black Hat Conference. [Online]. Available: https://www.blackhat.com/us-15/briefings.html [Accessed 15 Jan. 2016].

[14] D. Spaar. (2015, Feb. 2). Beemer, Open Thyself! – Security vulnerabilities in BMW's ConnectedDrive. c't. [Online]. Available: http://m.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html. [Accessed 15 Jan. 2016].

[15] C. Miller and C. Valasek. "A survey of remote automotive attack surfaces". [Online] Available: http://illmatics.com/remote%20attack%20surfaces.pdf [Accessed 15 Jan. 2016].

[16] Tracking & Hacking: Security & Privacy Gaps Put American Drivers at Risk. Office of U.S. Senator Ed Markey of Massachusetts, 2015. [Online]. Available: http://www.markey.senate.gov/imo/media/doc/2015-02-06_MarkeyReport-Tracking_Hacking_CarSecurity%202.pdf. [Accessed 15 Jan. 2016].

[17] M. Seaborn and T. Dulien, "Project Zero: Exploiting the DRAM rowhammer bug to gain kernel privileges," 2015. [Online]. Available: http://googleprojectzero.blogspot.fr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html. [Accessed 15 Jan. 2016].

[18] S. Lescuyer, "ProvenCore: Towards a verified isolation micro-kernel," EuroMILS Workshop, 10th HiPEAC Conference, Amsterdam, January 19-21, 2015.

[19] P. Bolignano, T. Jensen and V. Siles, "Modeling and abstraction of memory management in a hypervisor," To appear in: 19th International Conference, FASE, 2016.

[20] "ADAC deckt Sicherheitslücke auf - BMW-Fahrzeuge mit 'ConnectedDrive' können über Mobilfunk illegal von außen geöffnet werden," 2015. [Online]. Available: https://www.adac.de/infotestrat/adac-im-einsatz/motorwelt/bmw-luecke.aspx?ComponentId=227555&SourcePageId=6729. [Accessed 15 Jan. 2016].

[21] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle". IOActive, Seattle, WA, Tech. Rep., 2015. [Online]. Available: http://www.ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf. [Accessed 15 Jan. 2016].

[22] A. Van Herrewege, and I. Verbauwhede, "CANAuth - A simple, backward compatible broadcast authentication protocol for CAN bus," In ECRYPT Workshop on Lightweight Cryptography 2011, pp. 229-235, 2011.

[23] "HIS - Hersteller Initiative Software - Security". [Online]. Available: http://portal.automotive-his.de/index.php?option=com_content&task=view&id=31&Itemid=41&lang=english. [Accessed 15 Jan. 2016].

[24] Exploit-db.com. [Online]. Available: https://www.exploit-db.com/. [Accessed 15 Jan. 2016].

[25] R.-P. Weinmann, "Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks," 2012. In Proceedings of the 6th USENIX conference on Offensive Technologies (WOOT'12). USENIX Association, Berkeley, CA, USA, 2-2.