Formally Proven and Certified Off-The-Shelf Software Components: the Critical Links for Securing the Internet of Things

Dominique Bolignano

Prove & Run
77, avenue Niel, 75017 Paris, FRANCE

dominique.bolignano@provenrun.com

Abstract. The Internet of Things is bringing new security challenges that need to be addressed before deployment reaches a larger scale. We believe this can be done using a few key security software components and will illustrate this using a few security use cases that are representative of various Internet of Things market segments.

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 [6,7], [10,11,12,13,14,15], smart grids, smart homes [16,17,18], 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 other areas of the IoT (home health care, medical equipment, on-demand manufacturing, *etc.*) until quickly becoming essential and critical to the successful deployment of the IoT.

In this paper we will first describe and justify the approach we propose in order to achieve a proper level of security. We will in particular 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. We will then illustrate the critical part of the approach on a few representative use cases.

© Prove & Run S.A.S. 2016

2 An Approach to IoT Security

A large number of cyber-attacks have been reported lately. Even if these attacks were mostly targeted to cars and to mobile phones we believe that most of them (i.e. [3,4,5,6,7], [10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26]) are quite general and could happen in virtually any area of the IoT. Whereas the potential for such attacks is still extremely important we believe that such remote attacks on IoT devices could be avoided by following a proper security methodology and using technologies readily available on the market. We will in the sequel present the proposed approach and illustrate it on known attacks.

2.1 Following a Proper Security Methodology

It is first important to follow a proper security methodology. This was obviously not the case for many of the systems affected by reported attacks, probably because the security issues were not taken seriously enough. Many acceptable security methodologies exist and we do not recommend here a particular one. Yet, a proper security methodology should at least involve a proper risk analysis including the identification of assets and a study of the risks inherent to the architecture and associated functionalities for the expected usage environment. Such a risk analysis should lead to the definition of primary and secondary assets with associated security properties (integrity, confidentiality, availability, *etc*.). It should also lead to the identification of attackers' profile, of threats, of assumptions on the usage environment, of organizational and technical security objectives for the intended usage, *etc*. It should lead to the definition of a targeted resistance level that is commensurate with the risks at stake and to the potential business models for attackers.

Such a risk analysis should typically be followed by the definition of (product) security requirements.

When such a proper security methodology is followed both the security architect and the development team have a clear framework with proper objectives and requirements to perform and guide their work as well as a way to assess the adequacy of their solution to the security context. In the case of IoT where (massive) remote attacks are the most critical ones following such a methodology almost always leads to distinguishing two phases in an attack.

Identification versus Exploitation Phase of the Attack. There are indeed in many cases 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 attackers in the first phase is much higher, much more sophisticated tools, than the investment that can be reasonably applied to each single device in the second phase.

Addressing the Identification Phase of the Attack. When addressing the identification phase of the attack, security architects and developers will have to consider potentially sophisticated physical attacks. But technologies and know-how to resist to such attacks are widely available and in the end the only issue is cost. So the main strategy will be to keep to the very minimum the number of assets that need to be protected against sophisticated and expensive hardware attacks, and/or to minimize their value for attackers. Secure elements, HSMs and cryptographic processors can be used for that purpose, provided that the list of assets has been brought down to a few (root) secrets and keys that only need to be stored, and used to perform cryptographic operations. Even more important is to build a security architecture that reduces the value of assets by ensuring in particular that if such assets are compromised during the identification phase on one device, they cannot become key in performing the exploitation attack on a large number of remote devices.

Addressing the Exploitation Phase of the Attack. The importance of logical attacks. Once the problem of resistance with respect to the identification phase has been properly handled, physical attacks don't have to be considered anymore, besides side-channel attacks (mainly pertaining to cryptographic operations). These latter can easily be addressed by using state-of-the-art know-how and technologies that are readily available and that can be kept quite local. Now at the end the protection against logical attacks becomes the main challenge, and this is quite new.

Resisting logical attacks has indeed 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.

But this situation is radically changing, mainly due to the much larger attack surface exposed by IoT devices. An analysis of the attacks reported in conferences such as Black Hat 2013 to 2016 [3,4,5] indeed shows the ever-increasing importance of logical attacks. Hackers typically exploit errors (in the large sense) to break into systems: low-level implementation bugs, protocol or specification flaws, design, configuration or initialization errors, violations of organizational security policy, etc. See [1] for further detail on this topic.

Mobile Phone Security. The smartphone revolution triggered a change: mobile phones started to be used for more than just calling and texting. The new security needs 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, became too complex to secure.

In an attempt to cope with the new security needs coming from the mobile phone industry, the concept of the Trusted Execution Environment (TEE) was introduced in 2001 [1]. 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 [1]. However the TEE concept that is now widely deployed on smartphones was only the first step towards addressing new security needs. Even a small TCB such as the TEE is still too complex and error prone (e.g. [20,21]). The situation is even worse as the IoT brings many new challenges.

IoT: Logical TCB and Logical Attack Surface Becoming Too Large. With the IoT, the TCB of devices involved in connected architectures and their surface of attack become much wider. Devices to secure are indeed more diverse than "just" mobile phones: instead they handle a larger number of various peripherals to be secured. They also include large and complex 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. In many cases the risk for services and industries may become incommensurate.

It is quite obvious that it is essential to design architectures that rely on TCBs that are as small and simple as possible, and this was the idea behind the TEE concept. But while this is generally possible, even if the TCB becomes small it is always too complex and too error-prone to achieve the right level of security for the TCB if developed with standard development technologies, at least for the kernel part of them, and this even for a group of security experts.

The Challenge of Securing OSs and Kernels. Various public databases (such as [2]) indeed provide statistics on public bugs or vulnerabilities on all kinds of software. These databases clearly show that current OSs and kernels suffer from 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 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 rely more and more on complex and sophisticated hardware. OSs and kernels are by nature concurrent and hugely complex because of the need to support various kinds of peripherals (interruption handling becomes more and more difficult), the performance objectives (*e.g.*, complexity of cache management), the resource consumption issues (*e.g.*, the need for a sophisticated power management), and so on.

This complexity increases with time, increases with new IoT architectures and increases when it comes to real microprocessors (as opposed to microcontrollers).

Therefore, in the end, the main 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.* are free from errors, either in their design or implementation, that could be potentially exploited for logical attacks.

We believe that the challenge posed by logical attacks can only be addressed by applying deductive formal methods at least on the kernel parts of the TCB so as to get as close as possible to "zero-bug" for these complex software components.

2.2 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 to security properties.

Traditional software engineering techniques such as exhaustive testing or code inspections are clearly not sufficient anymore to bring the level of assurance that is needed to secure complex open kernels. This is due to the fact that there are too many different situations to consider for a kernel designer or tester and no real methods to review the quality of such kernel code in a systematic way, 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. Three categories of formal methods are usually distinguished:

1. Model-checking techniques, which can only be applied on models that are simple enough to be exhaustively checked. They are not applicable to 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 overflows, or the illegal use of pointers). They are certainly useful, but are far from being 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 (ProvenTools) that takes advantage of decades of research and advances in the scientific field of formal methods so as to make this formal verification process more efficient and also to bring even more confidence [8,9]. Using this environment, we can fully prove the most complex parts of any TCB, in particular OSs and kernels, so as 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: they are quite sequential and their complexity is amenable to traditional 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 validation of these "easy" parts is up to the security architect and/or up to the security evaluation and certification authorities. These parts will, depending on the situation, be formally verified or not, but in any cases they will have to be brought to the right level of confidence.

2.3 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 facilitate this task. In order to match the cost requirements of industrial deployment, it is critical to be able to re-use the same software components across industries and to mutualize costs.

We believe it is important to provide off-the-shelf reusable proven kernels, *i.e.* COTS, that will make securing IoT architectures simple and cheap, as well as fast to build and evaluate. In line with this vision, Prove & Run has already developed three such off-the-shelf kernels, formally proven and ready to use and certify:

1. ProvenCore, a fully proven and secure OS kernel, POSIX compliant and dedicated to microprocessors. ProvenCore has been optimized to run in the secure part of ARM® Cortex®-A TrustZone®-enabled microprocessors [8].
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 been also 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 functionalities as Xen[1], but with a much smaller TCB.

Prove & Run's COTS come for example with Common Criteria (CC) 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 are proven down to the code itself and go in fact much beyond what is required by the highest levels of CC certification.

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

3 Securing the IoT with a Proven and Secure OS Kernel

In this section, we illustrate on a few use cases the most innovative and challenging part of the approach, *i.e.* building a very strong TCB by reusing formally proven COTS, and in particular here using ProvenCore.

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

The way we, at Prove & Run, propose to address this is the following. We use a separate proven secure OS kernel, *i.e.* in our case ProvenCore, to address peripherals that need to be secured and to run secure applications, in a way that allows us to:

- Keep the normal OS (Linux or any other OS on which the main functionalities of the product have been developed) and thus retain all its features and benefits,
- Use a proven OS to perform security functions so that any error in the normal OS cannot be used to compromise the TCB; this pushes the normal OS outside of the TCB,
- 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 processors, 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, by splitting 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 on Cortex-A) act as an extremely small and security oriented two-VM asymmetric security hypervisor that allows:

- The Normal World to run on its own, potentially oblivious of the existence of the Secure World and,
- The Secure World to have extra privileges such as the ability to have 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, 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/left in the Normal World. The critical peripherals that have been selected to be controlled by the secure OS are configured, typically during the secure boot, to be only accessible and visible to the Secure World.

So in the architecture we propose, only applications that implement security functions (*i.e.* firmware update, firewalling and more generally security policy enforcement applications, IDS, VPNs, authentication protocols agents, security trace loggers, *etc.*), and potentially the drivers of security-sensitive peripherals need to be adapted. The biggest part of the software stack, *i.e.* the Normal OS and its applications are left untouched.

3.1 Securing Firmware Update Over-The-Air

As a first representative example we present how to secure a Firmware Over-The-Air (FOTA) management system. The ability to update firmware in the field is now a requirement in most markets. Firmware updates are an essential security mechanism, with both a curative use to update the firmware when vulnerabilities have been identified, and a preventive use to block unauthorized firmware updates by attackers.

From a high-level point of view, FOTA systems:

- Improve the value of existing devices by enhancing their functionality and performance: Tesla Motors uses firmware updates to add new features to existing cars.
- Eliminate costly recalls/local maintenance or physical replacements because of functional or security bugs. Again Tesla Motors has recently used its FOTA system to fix a vulnerability in their cars before the vulnerability was disclosed publicly.
- Reduce testing and support costs by keeping all devices at the same version, so there is no need to support older versions of the software.

A firmware update is a highly sensitive operation, carrying a massive security risk, as an attacker can misuse it to break or disable the device, unlock restricted features, or load a modified version of the firmware with disabled security and/or safety features.

We show here how to secure a FOTA system (and more generally any firmware update mechanism). The approach can be used to integrate any given FOTA system (for example a third-party one), or to design a new one. As in most security applications, only a part of the application is security sensitive: the so-called TCB. The idea is to port or develop the TCB part of the application as a ProvenCore (*i.e.* secure) process, the other part potentially remaining unchanged and running on the normal OS. By running the secure part of the application on ProvenCore we achieve two objectives: (1) we can make sure that the secure part is actually doing what it is meant to do, because it is relying and executing on a sound foundation, *i.e.* the formally proven ProvenCore and (2) that the secure part cannot be tampered with as it is protected (and isolated) from other applications and from the external world by ProvenCore.

As an example, our Secure Firmware Update (SFU) solution uses a split architecture:

- The SFU Agent, which is the secure part of the FOTA remote application, runs in the Secure World, on top of ProvenCore, and performs the most sensitive operations, such as verifying the authenticity of the update, making the installation decision and installing the update.
- The SFU Client runs in the Normal World, and retrieves the firmware image, for instance from the Internet or from another peripheral, potentially rebuilding it if delta binaries are used for bandwidth considerations. The SFU client has no security responsibility, and other third-party software can replace it.

The security comes here from the isolation of the sensitive update process in the SFU agent. The agent runs independently of the Normal World, and ProvenCore protects it from interference by other software components. In addition ProvenCore provides a solid ground for the execution of the SFU Agent that makes sure that the Agent is effectively doing what it is written and meant to be.

In practice, Prove & Run's SFU solution has been designed to strengthen existing firmware management systems by protecting the most sensitive parts of the firmware update operation: the integrity and authenticity checks, and the firmware's activation. Our SFU solution brings many advantages:

- The integrity and authenticity of the firmware is guaranteed by the isolation of the firmware update process, even if the operating system from which the firmware image comes has been compromised.
- Attackers cannot abuse the firmware update mechanism by bypassing integrity checks. If an interface exists with the secure boot solution, they also cannot modify the firmware directly.

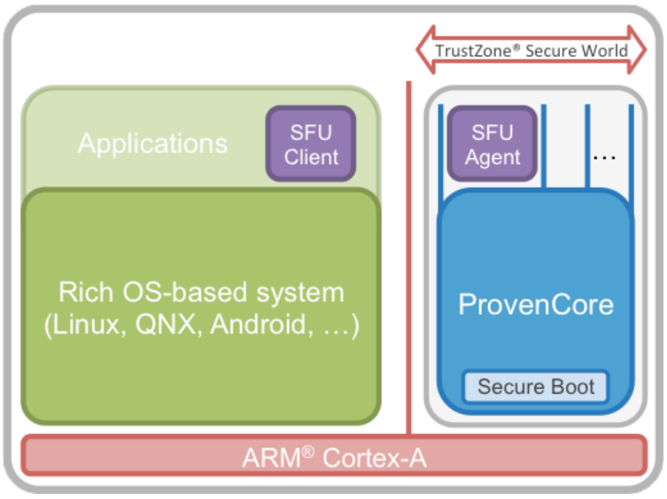

Figure 1: FOTA Architecture

The way we use ProvenCore to secure the FOTA process typically works for any other security function. In the end it comes down to the following:

- Identifying in a given security function the security sensitive part, *i.e.* the so-called TCB, then developing/porting this part as a ProvenCore process.
- Configuring ProvenCore to control sensitive resources and peripherals whenever needed, potentially rewriting some of the drivers for the secure peripherals into ProvenCore drivers. In the case of the SFU controlled resources were for example the secure memory (only accessible to the Secure World) and potentially the cryptographic functions or peripherals used to enable the secure boot.

3.2 Securing a Virtual Private Network

Applications running on connected embedded devices need to communicate securely with remote peers (other embedded devices, gateways or servers): they must be assured that their communications cannot be listened to or modified on the way. Even on “private” networks, who can be listening is never clear.

More precisely, applications and remote peers must be able to:

- Authenticate each other to make sure they know for sure who is on the other end of the line,
- Authenticate the messages they exchange to make sure that the messages they exchange are not modified on the way,
- Encrypt the messages they exchange to make sure that the content of the messages are not disclosed to any potential listener.

It is hard to provide this level of security, as it requires experience with cryptography and protocol design. For example, SSL:

- Requires the application to embed a large and complex library,
- Requires the application to be modified to make use of this feature,
- Requires the developer of the application to understand how to use this library securely.

Virtual Private Network (VPN) implementations offer a solution to these issues:

- Supported at the OS level, so applications don’t have to be modified,
- Simpler configuration, performed once for the whole device,
- Can apply to some or all of the communications going in and out of a device,

Limitations of Traditional VPN Implementations. Nevertheless, the confidentiality and authenticity of messages exchanged across a VPN can only be trusted if the VPN agents running on each device are protected from attacks:

- If an attacker can use a local application to inject a new public key certificate in the certificate store of the VPN agent, the attacker can perform a man-in-the-middle attack.
- If an attacker can use a local application to read the private certificate store of the VPN agent, the certificates can be leaked, enabling the attacker to impersonate this device or to perform a man-in-the-middle attack.
- More importantly if an attacker can perform a privilege escalation or more generally corrupt the hosting OS kernel they can easily perform any of the attacks above and much more.

The proposed architecture consists in placing the VPN agent in the Secure World and developing a proxy in the Normal World that provides the VPN interface to Normal World applications. It is also possible to reduce the scope of the agent by only keeping its Trusted Computing Base (TCB) on the secure side, and putting the non-TCB part of the agent into the proxy. We did this for example for OpenVPN. The Normal World (in our case Linux) uses OpenVPN (or more generally the secured VPN) as if it were a normal Linux-based OpenVPN implementation. But the secure channel is fully secured against remote attacks. The architecture can be summarized in the following figure:

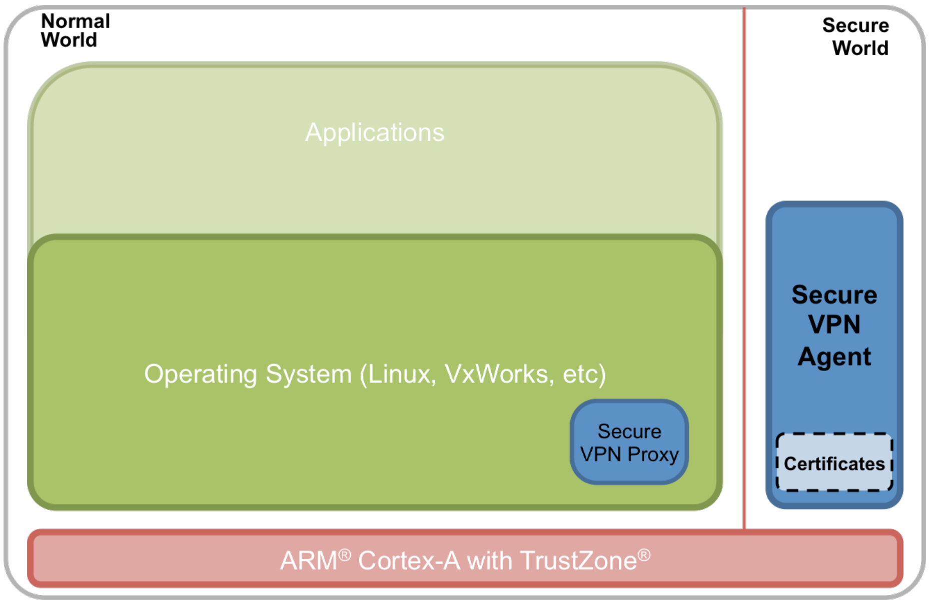

Figure 2: Architecture for the secured VPN implementation

Security Rationale. VPN agents cannot be fully trusted because they run in the same address space as the OS (Android, Linux, *etc*.), where they are vulnerable to the huge number of local and remote attacks that affect traditional operating systems. This also means that it may not be possible to recover control of compromised devices remotely.

The Secure VPN Solution is secure because:

- Thanks to TrustZone, the Secure VPN Agent executes in a secure area, protected from attacks from applications running on the OS.
- Thanks to TrustZone, the Secure VPN Agent relies on certificates whose authenticity and confidentiality are protected from any attacks coming from the OS.
- The kernel of the Secure VPN is formally proven for higher security.
- The authenticity of the Secure VPN is protected by a secure boot mechanism anchored in the hardware security features of the board.

In order to better explain the importance of using a formally proven secure kernel we consider a simpler use case, that of a secure filter.

3.3 Securing a Command and Data Filter

Let us consider the situation where commands are received from the outside (typically from an administration server) and data is regularly sent from the device to the cloud. This is typical of many IoT endpoints. Let’s then identify the complete list, type and constraints that should apply to both incoming commands and outgoing data. Once this is done, it is then possible to develop a security filtering application that ensures that only messages that comply with the identified types and constraints go in and out.

Such a security application would have been more than useful to prevent attacks such as [7].

When such a security application runs on a normal OS such as Linux, we get the architecture shown in the following figure:

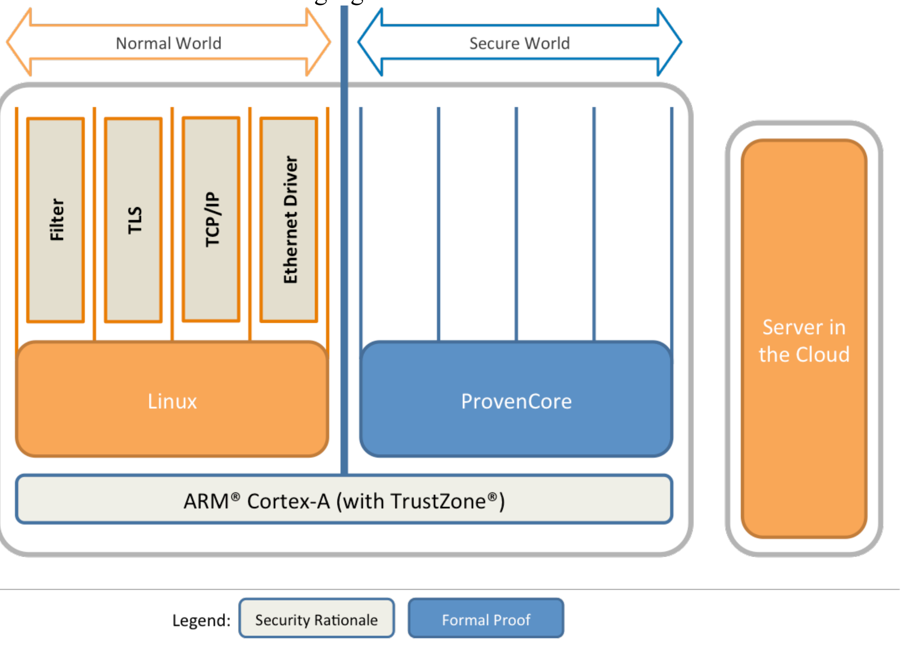

Figure 3: Traditional architecture for a command and data filter

A hacker will typically try to send corrupted messages (*e.g.*, by injecting messages on the communication link between the trusted server and the device). Depending on the message, the tampering will ideally be detected and blocked either by the communication driver, the TCP/IP stack, the TLS layer (we assume here there exists such a security layer) or by the filtering application, each one checking its part of the message. The problem is that any problem in any of these layers will potentially be exploited to perform a privilege escalation attack, to corrupt or tamper with the underlying OS kernel and to bypass the complete filtering chain, as shown in the following figure:

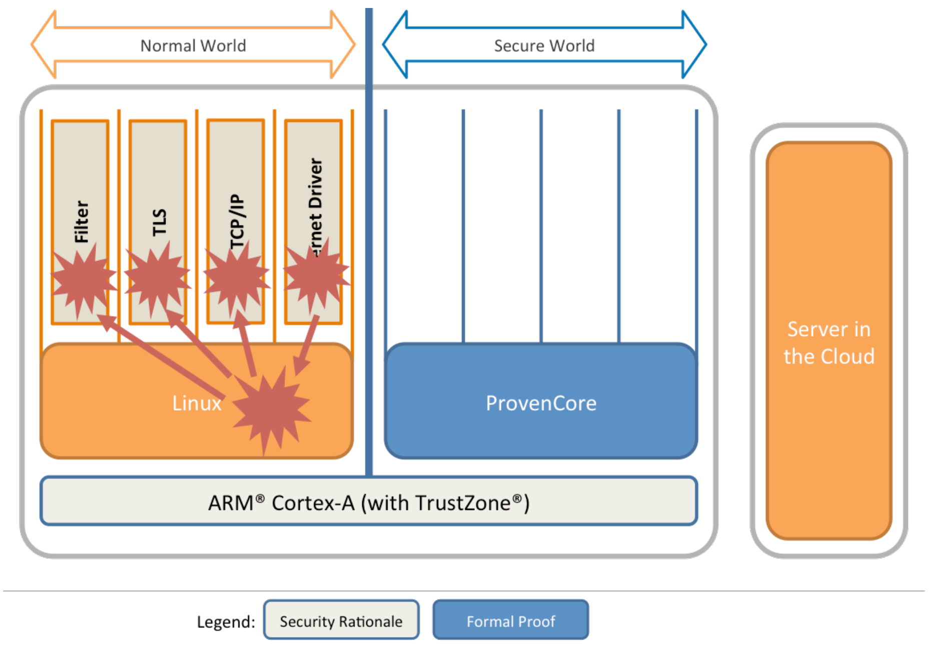

Figure 4: Vulnerabilities of the traditional architecture

Now in the case of Linux (or of most normal OSs), the privilege escalation is even not required and the attack is even easier because software components such as the drivers or (part of) the TCP/IP stack are executing in privileged kernel mode.

In any case, if the same software components are executed and protected by ProvenCore such remote attacks are not possible anymore, as ProvenCore has been formally proven to resist to such logical attacks in all possible cases. This is explained in the following figure:

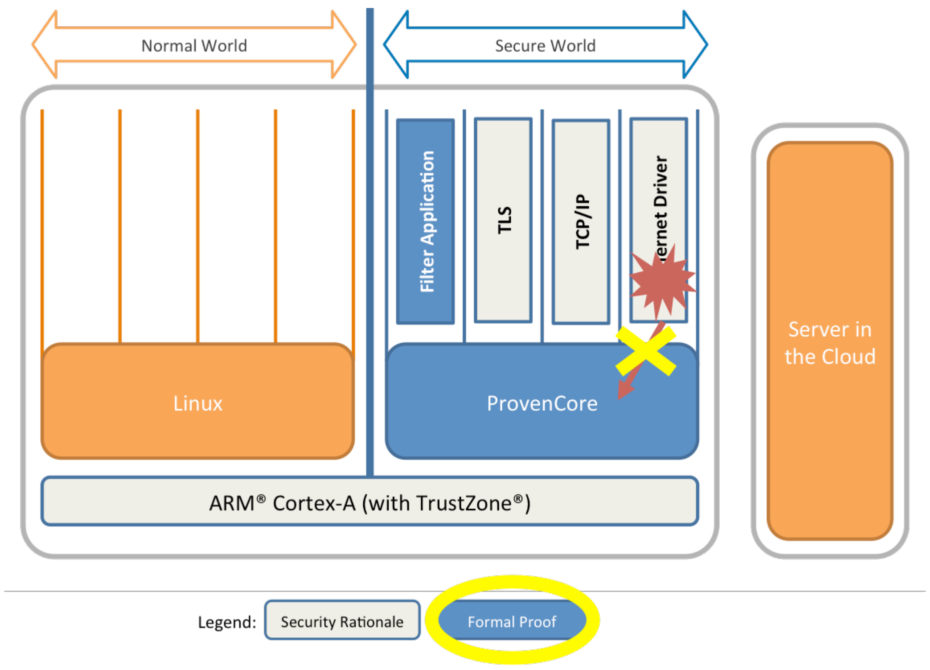

Figure 5: Secure architecture for a command and data filter

In addition, the Ethernet peripheral is in the Secure World, so, even if the Linux OS in the Normal World is compromised, it cannot get any access to the peripheral without going through the driver, protocol stack and filtering application running in the Secure World

In this setup it is not strictly necessary to prove the Filtering Application, as it is typically a very simple sequential application for which state-of-the-art development and quality-management methodologies alone allow to reach a very high level of quality and trust. Nevertheless proving such applications can be done at a relatively low cost and bring an even higher level of security.

4 Conclusion

In this paper we have shown that IoT architectures introduce new security challenges. We have shown that it is possible to bring a very high level of security to those IoT architectures, in a way that is both compatible with industrial requirements and with time-to-market and cost constraints. This involves reducing the scope of the TCB, which can be achieved using a combination of formally proven, secure and certification ready COTS. 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. D. Bolignano, "Proven Security for the Internet of Things,", Embedded World 2016, Nuremberg, Germany, February 23-25, 2016.
2. National Vulnerability Database. NIST, https://web.nvd.nist.gov/view/vuln/search
3. Briefings, 2013. Black Hat Conference, https://www.blackhat.com/us-13/archives.html
4. Briefings, 2014. Black Hat Conference, https://www.blackhat.com/us-14/archives.html
5. Briefings, 2015. Black Hat Conference, https://www.blackhat.com/us-15/briefings.html
6. C. Miller and C. Valasek, "A survey of remote automotive attack surfaces". http://illmatics.com/remote%20attack%20surfaces.pdf
7. C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle". IO-Active, Seattle, WA, Tech. Rep., 2015. http://www.ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf.
8. S. Lescuyer, "ProvenCore: Towards a verified isolation micro-kernel," EuroMILS Workshop, 10th HiPEAC Conference, Amsterdam, Netherlands, January 19-21, 2015.
9. P. Bolignano, T. Jensen and V. Siles, "Modeling and abstraction of memory management in a hypervisor," 19th International Conference, FASE, 2016.
10. Beemer, Open Thyself! – Security vulnerabilities in BMW's ConnectedDrive. http://m.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html
11. Hack it like you stole it. https://www.pentestpartners.com/blog/hacking-the-mitsubishi-outlander-phev-hybrid-suv/
12. Car Hacking Research: Remote Attack Tesla Motors. http://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/
13. C. Yan, W. Xu and J. Liu, "Can You Trust Autonomous Vehicles: Contactless Attacks against Sensors of Self-driving Vehicle," Defcon 24, August 4-7, 2016.
14. Y. Burakova, B. Hass, L. Millar and A. Weimerskirch, "Truck Hacking: An Experimental Analysis of the SAE J1939 Standard," 10th USENIX Workshop on Offensive Technologies (WOOT 16), Austin, TX, August 8-9, 2016.
15. C. Thuen, "Time to Get Progressive With ICS / IoT Cyber Security," http://www.digitalbond.com/blog/2015/02/02/time-to-get-progressive-with-ics-iot-cyber-security/
16. Neighbor Unlocks Front Door Without Permission With The Help Of Apple's Siri, http://www.forbes.com/sites/aarontilley/2016/09/17/neighbor-unlocks-front-door-without-permission-with-the-help-of-apples-siri/
17. R7-2016-10: Multiple OSRAM SYLVANIA Osram Lightify Vulnerabilities (CVE-2016-5051 through 5059), https://community.rapid7.com/community/infosec/blog/2016/07/26/r7-2016-10-multiple-osram-sylvania-osram-lightify-vulnerabilities-cve-2016-5051-through-5059
18. How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen, http://www.forbes.com/sites/thomasbrewster/2016/09/25/brian-krebs-overwatch-ovh-smashed-by-largest-ddos-attacks-ever/

19. C.Wang, X. Guo, Y. Wang, Y. Chen, B. Liu, "Friend or Foe?: Your Wearable Devices Reveal Your Personal PIN", Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, Xi'an, China, May 30 - June 3, 2016.
20. KNOXout (CVE-2016-6584) - Bypassing Samsung KNOX, `http://www.vsecgroup.com/single-post/2016/09/16/KNOXout---Bypassing-Samsung-KNOX`
21. Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption, `http://bits-please.blogspot.fr/2016/06/extracting-qualcomms-keymaster-keys.html`
22. Objets connectés : polémique sur la sécurité du réseau français Sigfox, `http://www.01net.com/actualites/objets-connectes-le-reseau-francais-sigfox-une-passoire-en-matiere-de-securite-957875.html`
23. R7-2016-07: Multiple Vulnerabilities in Animas OneTouch Ping Insulin Pump, `https://community.rapid7.com/community/infosec/blog/2016/10/04/r7-2016-07-multiple-vulnerabilities-in-animas-onetouch-ping-insulin-pump`
24. S. Skorobogatov, "The bumpy road towards iPhone 5c NAND mirroring".
25. K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida and H. Bos,"Flip Feng Shui: Hammering a Needle in the Software Stack", Proceedings of the 25th USENIX Security Symposium, Austin, TX, August 10–12, 2016.
26. V. Sivaraman, D. Chan, D. Earl and R. Boreli, "Smart-Phones Attacking Smart-Homes", Security & Privacy Week 2016, Darmstadt, Germany, July 18-22, 2016.