Trust, but verify: why and how to establish trust in embedded devices

(v1.1: This paper is an updated version to the DATE 2016 invited paper)

Aurélien Francillon
EURECOM, France
Email: aurelien.francillon@eurecom.fr

***Abstract*—A lot of research efforts have been put into constructing secure systems. However, experience has shown that, while there are many products which have a good level of security, others are really insecure. Some are security devices: security is at the core of their purpose; while other are not. We nevertheless often rely on the their security in our daily life and their failure can have serious consequences. In this paper, we discuss why we are in this situation and what we can do to improve the situation. In particular, we defend the thesis that more transparency and more openness in embedded systems hardware and software will foster a more secure ecosystem.**

First, there is an economic problem. Besides being a difficult problem to solve correctly, security is most of the times expensive.

Second, trust is something that is not blindly granted but that is earned by verifying it. Currently, trusted computing mechanisms often rely on unconditional trust on the systems manufacturer. However, users have too few ways to verify that the systems are trustworthy other than blindly trust the manufacturer. We should design systems where the users, i.e., the devices owners, can decide whom and what to trust. We call this *Design For User Trust*, where users are in control of the system.

Finally, one can only trust a system fully if he can inspect it. Unfortunately, the first security measures that are implemented in embedded systems often prevent such an independent analysis (e.g., deactivation of a debug port, secure boot, encrypted file system, obfuscation). But such measures are more hiding the problems (making it difficult to discover software vulnerabilities) than solving it. They are often useful in securing a system (slowing down an attacker) but should not jeopardize our ability to analyze them. We call this *Design For Security Testing*.

We conclude that more research is needed to make it easier to build secure systems, in particular, in the areas of concrete architectures for *Design For User Trust* and *Design For Security Testing*.

I. Introduction

Embedded devices have been present since almost the beginning of computing, yet in the past years embedded devices became more complex, more connected and more pervasive. An average person might carry ten independent embedded devices (watch, phone, smart card, medical device, RFID tags...).

Moreover, such devices often operate without the user's knowledge (RFID tag), are always on and communicating (a smartphone). People will eventually be unable to know how many devices they are carrying, which ones are currently connected and what data they contain. Is the data personal or not? Who is able to access it? Who is able to perform software update without the user's knowledge?

Security (or lack thereof) in embedded devices is becoming a public concern, and is almost daily in the mainstream news. There are too many devices that are reported to have surprisingly large number of vulnerabilities, which are often consequences of obvious negligence [1].

In this paper we will take the position of a user and owner of an embedded system. A user here may be an individual, a company or even a state. While this is a subjective position, it is often overlooked. We mainly consider the commodity embedded products and systems that people interact with everyday such as off-the-shelf devices and SOHO equipment. We will also mostly discuss trust in embedded software rather than hardware (such as hardware Trojans...). Trust in hardware is as important as trust in software but will not be discussed here.

We analyze the reasons for which so many products are insecure and describe the factors that lead to this situation. We then look at what needs to be done to improve the situation and describe difficulties in implementing *Design For User Trust (DFUT)* and *Design For Security Testing (DFST)*.

II. An economic problem

Securing embedded devices, requires an holistic view, which includes the economy of embedded systems. Indeed, such devices are produced to be sold to customers which will use them. Such devices therefore only exist because there is a market for them.

Economists consider that a market is fair when buyers and sellers are equally informed. Akerlof [2] famously descried the used cars market as a market where there is an imbalance of information (*market for lemons*). In this market the seller knows whether the car is a good one or a bad one (*a lemon*). This is because either he used it for long time or because he is a professional car dealer. On the other hand the buyer cannot tell and is making an uninformed choice. As a consequence users are not willing to pay much more than for a *lemon* and the prices of all products are driven by the prices of *lemons*. This essentially remove the quality goods from the market, because selling them at a price close to that of a *lemon* would lead to a net loss.

Anderson [3] first discussed that security is often more an economic problem than a technical problem. Anderson discusses several effects that influence security decisions like the tragedy of the commons, customer lock in, the need to get

into markets quickly (which leads to shipping products which are incomplete and insecure) and the problem of asymmetric information.

Grigg [4] revisits asymmetric information by observing that the market for security products is a market where both the seller and the buyer have incomplete information. I.e., even the seller does not know if the product is secure, or will defend against threats. This seem relevant for security products (e.g., neither the anti-virus vendor nor the buyer will know if the anti-virus will stop the next generation of malware[1]) rather than for the security of products. In the latter case the vendor knows if he considered security from the beginning or not, and how much he did invest in building a secure system.

In [5] Schneier suggests that software vendors should be liable for the security of the software they provide. This is because without software liability the software vendors will not want to invest into securing the products:

> Liability enforcement is essential. [...] the costs of bad security are not borne by the software vendors that produce the bad security. In economics this is known as an externality: a cost of a decision that is borne by people other than those making the decision.

Ten years after this essay has been published, we don't really see liability enforcement happening. Will this happen in the world of embedded devices? Yes, probably in some critical markets, like airplanes, cars or medical devices. If an airplane is crashing because of a software bug (or a vulnerability has been exploited) the manufacturer will probably be found to have some responsibility. For example, Toyota has been sued for faults in its throttle control system and found responsible for the problem [6], [7].

Will similar liability appear for commodity devices? Like IP Cameras and home routers? It may not, in particular for the devices whose failure is not life threatening, i.e., most *Internet of Things* devices. The security of the devices may improve in the future. In [8] Shostack argues that mature markets need to differentiate and that security will be a differentiating factor. Attacks and vulnerabilities of embedded devices also attract a significant interest by the media and, even if the reality gets often distorted in news reports, this bad image also leads to a growing awareness of the problem.

A. Building secure devices may be an error in the short term

It is largely recognized that building secure systems is difficult. There are numerous methods [9] and techniques but applying them require planning, experience and skilled workforce. All this has a cost and often users prefer to acquire devices that have many features, are easy to use and are cheap. On the other hand, security often makes those systems more expensive, harder to use and it is recommended to limit the attack surface, i.e., the number of features to make secure devices. Schneier captures this in [5]:

> Any smart software vendor will talk big about security, but do as little as possible, because that's what makes the most economic sense.

In addition to this, those markets are dynamic and for a small company time to market may be important. A more secure product will often be more expensive, more complicated to use and take longer to develop. If a company brings such a product to the market it may be difficult to sell, as the market will already be captured by cheaper products with more features [8].

B. But building devices with insufficient security is an error in the long term

However, in the long term, the lack of security may lead to serious issues. Some will be bad image or user complaints, but in other cases a security problem can lead to a massive data compromise and a serious consequences. Such consequences may have a higher cost than an initial investment in security or even as the company value, leading to bankruptcy.[2]

Toyota's throttle system problems we mentioned previously were due to poor code and design quality. This lead to both high financial loss[3] and serious damages to Toyota's image. It's interesting to note that in this case the system failed by itself. I.e., this was a reliability problem, a bug that appears under some unusual circumstances. An attacker, however, will usually beat the odds. Writing a reliable exploit consists in abusing such very infrequent conditions to make them happen reliably. If similar problem would have been abused by an intelligent attacker (as opposed to a random fault) damages could be have been much larger, all vulnerable cars could have failed.

In the long term such problems will become even more important. Devices are often deployed for a very long life time and in those conditions a real device life-cycle needs to be considered. For example, for how long firmware updates need to be provided after production is over? How are those supposed to be installed? By whom? How to perform updates on device which are not supported anymore by the manufacturer? Should the device be replaced? When a device is compromised, how to perform forensics on it? In particular forensics is very difficult if access to the device software is not possible.

III. Motivations for trust and security

Manufacturer motivations for securing a platform are not the same as those of the end user or device owner. In particular, manufacturers will often be scared of device cloning, will want to retain control on an ecosystem or protect media content. When devices are under constant attack, and losses are significant, manufacturers make serious efforts to secure the systems. For example, smart cards used in pay-TV have

[1] In [3] Gene Spafford is cited as an example:

> You're proposing to build a box with a light on top of it. The light is supposed to go off when you carry the box into a room that has a Unicorn in it. How do you show that it works?

[2] This, for example, happened to DigiNotar, a certificate authority that bankrupted after it was found that it was compromised [10].

[3] Loss are counted in billions of dollars [11].

been attacked constantly for decades and are made quite secure with many countermeasures in place.

Regulation and certification are also driving factors to improve security, this is the case for example for chips in passports which need a certain level of security and need to undergo an external evaluation to be certified. Critics of certification [3] mention that it provides a limited incentive, as the goal is to obtain certification rather than really securing the device. This leads some vendors to look for the lenient testers [3].

It is very common for secure products (e.g., chips) to have confidential documentation or data sheets. This is often claimed to be incompatible with the Kerckhoffs's principle. The well known Kerckhoffs principle states that the security of a system should only depend on a cryptographic key and not on the secrecy of the design of the algorithms. As [12] argues, this does not mean that the design needs to be published, indeed, hiding such information is slowing down attackers.

However, we argue that this also restricts the ability for developers to get educated and understand the available systems. But more importantly, this also prevents users and developers to compare various security solutions. This aggravates the information asymmetry, is this one of the cause of the market for lemons in security?

Secrecy also leads to suspicion, if the internals of this device are secret, maybe it is because there is something to hide? Is it actually trying to hide its poor level of security? If it were secure hiding its design would not be needed? Or maybe a shameful behavior is hidden?

For example, the *Baseband* software (modem) of a smartphone is generally a large piece of code that is protected by a secure boot and is opaque. Because this code is also very privileged, this easily leads to theories and suspicion [13], [14].

A. Security or lock out?

Does more security helps to trust the system? It's often quite the opposite. Richard Stallman famously [15] referred to *Trusted Computing* as *Treacherous Computing*. This position was mainly expressing the fear that such a security mechanism could be used against the user. E.g., preventing the user to install operating system of his choice or making it impossible to access to some documents. However, the problem here is not only technical, but depends on who has control over the system and who can inspect it.

In particular, there was a strong opposition against enforcement mechanisms that would prevent the computer owner from running the software of his choice. Those fears mostly appeared unfounded as a TPMs is not a secure boot mechanism. This however changed with Microsoft pushing for secure boot with UEFI [16]. This makes it harder for users to install alternative software [17]. This has been the case for long time on smartphones, tablets and many embedded devices. UEFI actually does not state who should be in control of the root signing keys but in practice Microsoft does.

Back to embedded devices, users which purchase a device own the objects, but were they designed to protect user's best interests (data security, privacy)? Or are the devices designed in the best interests of the issuer (company or governments)?

On the other hand, users want (or should) require to keep control on the devices they own. Of course it is not realistic that every user will inspect the software on his device or recompile its operating system. However, making it available to all makes it possible for a few advanced users to inspect the device's internals, and overall increases the trust in such devices.

Users are often locked out of the devices they own. In such a case, the only option for users is to exploit vulnerabilities to be able to take control of their hardware. Such vulnerabilities are not disclosed to the vendors, to prevent vendors to fix them. Therefore, users become the attackers of their own goods. However, such vulnerabilities can also be abused by attackers to compromise user's devices. This is a pathological situation where users have no incentive to report the vulnerabilities to vendors but will rather keep them for themselves so as to take control back on the devices they own.

When manufacturers lock the users out of the system, this makes it impossible for users to manage the security of the devices they own. Therefore, manufacturers should be fully liable for the consequences of a security issue with the device. Most manufacturers will not welcome such a liability, as this represents a huge risk. However, liability cannot be put on the device owner as he does not have any way to secure the system.

IV. Verifying trust

We argued that independent evaluation and openness is needed for establishing trust in embedded systems. However, how can this be preformed while preserving security? I.e., how to make sure that the device is controlled by its owner and not an attacker (or manufacturer). How can we give sufficient access to the device to perform independent security tests while preventing abuse of such features? Devices needs to be designed specifically for this purpose. Building blocks, and methodologies, needs to be provided to facilitate such designs. Such design approaches are common in the microelectronics, for example, there is Design For Test (DFT) or Design For Manufacturing (DFM) which ensure that the device can be tested or manufactured efficiently. This is generalized in Design for X, i.e., considering many factors in device design [18]. We argue that we need to consider Design For User Trust (DFUT) and Design For Security Testing (DFST). In this section we only state what would be the requirement for those and mention a few difficulties in creating such systems.

A. Design for Security Analysis

Security analysis is difficult [19]. Unfortunately, the first security measures that are implemented in a system usually hinder testing. For example, restricting access to hardware documentation, encrypting firmware images, secure boot, deactivating debug access, stripping binaries all those makes security testing harder.

Traditional functional testing is always performed by the manufacturer before shipping a devices, why would security testing let open to third parties?

There are several reasons for this. First, the third party (customer and owner of the device) may not trust the manufacturer and require to inspect the device himself. Second, traditional testing is a positive testing, a feature works or not. On the other hand security testing is testing for the absence of a bad behavior. Security testing is often difficult for the developer or manufacturer, this is why an external view is important. Pen-testing is very successful because the tester is in the position of an attacker, a pen tester will focus on the weak links which may have been ignored by the developers.

Finally, such testing needs to be made available in the field. When a device is found to be compromised we need to be able to inspect it and maybe to analyze malicious code that was injected in the device. Forensics require similar access to the device internals as performing security testing. How could this be done if the access to the device internals are closed?

In summary, we need better security testing in embedded devices, this will only be possible if some support is available for security testing which was planned from the beginning, in the design phase.

B. Design for User Trust

Trusted computing should not be used to lock out users from the systems they own. We propose to consider the question of Designing systems for *User Trust* (DFUT). *User Trust* here means that the users are eventually in control (or can be if they want). However, providing such a design is not trivial.

This could consist in installing user keys and code to take ownership and control of the device's secure boot. This is actually difficult to perform without reducing the security of the devices themselves. For example, this cannot be done by software only, otherwise malware could install its own keys and defeat the purpose of the secure boot which is to prevent compromised systems to boot [20]. There should also be a mechanism to authenticate that the owner itself is loading new keys. Recovery mechanisms should also be designed. How to design a recovery mechanism without making it possible to use it without user consent?

Some examples of designs that are close to DFUT, in some systems. For example, Google Nexus phones can be turned into developers phones by unlocking the bootloader (which voids the warranty). On such developer phones it is possible to install custom images. Nokia Maemo phones were also providing a similar developer mode, once started with a non signed image the phone was still booting, but some (DRM) features are not available anymore [21], [22].

In summary, we need to empower users in the long term, for this we need to make it possible to users to be in control of the software that runs on their devices. This also requires that third parties (e.g., manufacturers or evil maids [23]) cannot abuse the system. Finally, such systems needs to be easy to use.

V. Conclusion

Security and trust are more than just technical problems. In this paper we argue that users, as owners of devices, but also customers, need to be involved in the security of the devices they use. First, this will reduce the liability of the device manufacturers. If a user has no control at all on a device, he cannot be held liable for security problems with the device. Therefore the responsibility has to be on the manufacturer. Second, we argue that making security relevant information available to users will also make users more informed and therefore reduce the information asymmetry. This in turn will generate a healthier market and make it easier to sell more expensive and secure devices. Finally, we argue that to make this possible, devices needs to be Designed For User Trust (DFUT), having trusted computing to be at the service of users and not against them. Users also need to be able to test the devices, we need to design devices so that they can be tested by users. Without introducing further vulnerabilities in the systems. This is Design for Security Testing (DFST). In such an Utopian world everyone would benefit from more security. We summary, we need DFST and DFUT building blocks that are simple and easy to integrate in real world designs from scratch.

References

[1] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A Large Scale Analysis of the Security of Embedded Firmwares," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*, August 2014.

[2] G. A. Akerlof, "The market for" lemons": Quality uncertainty and the market mechanism," *The quarterly journal of economics*, pp. 488–500, 1970.

[3] R. Anderson, "Why information security is hard - an economic perspective," in *ACSAC*, 2001.

[4] I. Grigg, "The market for silver bullets," 2008.

[5] B. Schneier, "Liability changes everything," Heise Security, November 2003, https://www.schneier.com/essays/archives/2003/11/liability_changes_ev.html.

[6] M. Dunn, "Toyota's killer firmware: Bad design and its consequences," 2013, edn.com, http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences.

[7] C. Philip Koopman, "A case study of toyota unintended acceleration and software safety," 2013, http://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf.

[8] A. Shostack, "Avoiding liability: An alternative route to more secure product," in *Workshop on Economics and Information Security (WEIS)*, 2005, http://infosecon.net/workshop/pdf/44.pdf.

[9] H. Michael and S. Lipner, "The security development lifecycle sdl: A process for developing demonstrably more secure software," *Publishing House of Electronics Industry*, 2008.

[10] K. Zetter, "Diginotar files for bankruptcy in wake of devastating hack," Wired magazine, September 2011, http://www.wired.com/2011/09/diginotar-bankruptcy/.

[11] Wikipedia, "200911 toyota vehicle recalls," https://en.wikipedia.org/wiki/2009%E2%80%9311_Toyota_vehicle_recalls#Economic_impact.

[12] N. T. Courtois, "The dark side of security by obscurity and cloning mifare classic rail and building passes anywhere, anytime," Cryptology ePrint Archive, Report 2009/137, 2009, http://eprint.iacr.org/.

[13] The Free Software Fundation, "Replicant developers find and close samsung galaxy backdoor," March 2014, https://www.fsf.org/blogs/community/replicant-developers-find-and-close-samsung-galaxy-backdoor.

[14] D. Goodin, "virtually no evidence for claim of remote backdoor in samsung phones," arstechnica.com article, March 2014, http://arstechnica.com/security/2014/03/virtually-no-evidence-for-claim-of-remote-backdoor-in-samsung-galaxy-phones/.

[15] R. M. Stallman and J. Gay, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Free Software Foundation, June 2002, https://www.gnu.org/philosophy/fsfs/rms-essays.pdf.

[16] J. Brodkin, "The right to dual-boot: Linux groups plead case prior to windows 8 launch," October 2008, http://arstechnica.com/business/2011/10/the-right-to-dual-boot-linux-groups-plead-case-prior-to-windows-8-launch/.

[17] P. Bright, "Windows 10 to make the secure boot alt-os lock out a reality," March 2015, http://arstechnica.com/information-technology/2015/03/windows-10-to-make-the-secure-boot-alt-os-lock-out-a-reality/.

[18] H. Meerkamm and M. Koch, "Design for x," in *Design process improvement*, J. Clarkson and C. Eckert, Eds. Springer London, 2005, pp. 306–323. [Online]. Available: http://dx.doi.org/10.1007/978-1-84628-061-0_13

[19] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares," in *Network and Distributed System Security (NDSS) Symposium*, ser. NDSS 14, February 2014.

[20] J. Kerr, M. Garrett, and J. Bottomley, "Uefi secure boot impact on linux," October 2011, http://ozlabs.org/docs/uefi-secure-boot-impact-on-linux.pdf.

[21] E. Reshetova, "Maemo 6 security framework, making happy drm business and freedom lovers with the same device," february 2010, video of the talk available at https://www.youtube.com/watch?v=r08dFQQ2uZI LWN News http://lwn.net/Articles/373780/.

[22] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan, "Old, new, borrowed, blue –: A perspective on the evolution of mobile platform security architectures," in *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '11. New York, NY, USA: ACM, 2011, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/1943513.1943517

[23] J. Rutkowska, "Evil maid goes after truecrypt!" October 2009, http://theinvisiblethings.blogspot.fr/2009/10/evil-maid-goes-after-truecrypt.html.