Security Observance throughout the Life-Cycle of Embedded Systems

S. Hasan Mirjalili, Arjen K. Lenstra
IV, EPFL, Lausanne, Switzerland

Abstract - *Embedded systems are an established part of life. Their security requirements underline the importance of properly formulated, implemented, and enforced security policies throughout their life-cycle. Currently, security is just an afterthought, and most solutions are meant to thwart particular attacks. However, the increasing number of security breaches, the ensuing economical losses, and potential dangers all emphasize the importance of fundamental security solutions. This paper first surveys the current situation and then proposes a holistic approach where security is considered from the beginning of the design of embedded systems throughout their entire life-cycle. In our approach, the entire system life-cycle is analyzed and appropriate countermeasures are incorporated in the design. Obviously, prevention is not the complete solution. A 4-level defense strategy assures not only that a system has been properly designed in terms of security, but also that the liabilities of its designers are adequately covered.*

Keywords: Security, Embedded Systems, System life-cycle, Design Methodologies.

1 Introduction

The idea of pervasive computing is growing and computing devices will be available anywhere and anytime. Our lives and our businesses depend unavoidably on computing systems and, increasingly, on embedded systems in particular. In this paper, our focus is on embedded systems, which are ubiquitously used to sense, capture, store, process, transmit personal, private and vital data. When an embedded system performs any of these tasks, security observation is a necessity. Moreover, security of embedded systems provides new business opportunities and prevents losing many opportunities. For example, it prevents safety disasters and should help preserving user privacy. Time limited services or on-demand digital services are examples of new business opportunities benefiting from well established secure embedded devices. The increasing number of security breaches which have been detected in embedded systems in recent years also reveals the importance of fundamental security solutions. Current solutions are mostly the addition of features, such as specific cryptographic algorithms and security protocols to the system. This is mostly done at the end of the design phase or only as addition to a part of the system, such as the transceiver part of the system to encrypt and decrypt outgoing and incoming data. In fact, solutions of this sort are comparable to 'patching' a system. They cannot result in a complete solution and are often not integrated into the entire system. Sometimes these solutions violate the criteria that designers have taken into consideration from the beginning. These are subtle points that are not addressed by designers who tend to focus mainly on functionality and by companies that tend to focus on short term profits.

In this paper we set out to have a comprehensive view on the security of embedded systems and propose a design methodology that can help designers and developers to deliver more secure systems. To clarify what embedded systems are, we provide an informal definition of embedded systems. This also explains what distinguishes them that we must consider their security as a special case. A definition of security also helps to know when a system is considered secure and what should be done to make it secure. In the following paragraphs, several definitions are provided that are used in the rest of the paper.

1.1 Embedded systems

Embedded systems are specialized electronic systems that are part of a larger system. They are normally not directly visible to the user [1]. Examples of embedded systems are computing systems, which are inside, for instance, automobiles, planes, trains, space vehicles, consumer electronics, medical equipments, vending machines, network appliances, smart cards, cell phones, PDAs and other handhelds as well as robots and toys. They are designed and developed for a specific application and not as general purpose computing devices. They play a significant role in areas such as education, health care, ambient intelligence, consumer electronics, avionics, car industry, and controllers in industrial plants. The uses are endless, and their number is increasing in our new world of pervasive computing. Embedded systems have common characteristics such as: They should be efficient in terms of criteria such as power consumption, size, run-time requirements, weight, and cost. Also they are often embedded in portable systems with network capabilities.

1.2 Computing Systems Security

Computer security is the protection of computing systems against threats to confidentiality, integrity and availability [2]. In other words, a secure computing system provides three properties: confidentiality, integrity, and availability. All three are essential but depending on the application of system, one or two of them may receive more attention. Confidentiality means that information is disclosed only according to a security policy. Integrity means that information or system

structure can be changed according to a security policy and availability means that services of system are available according to a security policy. The security policy addresses constraints on functions, flow among functions and constraints on users' access. Constraints on function and access may be correlated with time, location and/or other parameters. Users are external systems or human users. All of the details about users and constraints is explained precisely in a security policy.

1.3 Security model and definitions

Vulnerability, threat, attack and safeguard or countermeasures make up a framework that enables arguing about computer security. Vulnerability is some weakness or fault in the system that could allow security to be violated. A threat is a circumstance or event that could cause harm by violating security. An adversary exploits a vulnerability in the system to perform an attack. A safeguard is any technique, procedure, or other measure that reduces vulnerability. Safeguards make threats weaker or less likely. This framework ─vulnerability, threat, attack and safeguard─ is useful for analyzing and evaluating system security, also for deciding what safeguards to use [2].

The rest of this paper is structured as follows. Section 2 argues why the security of embedded devices is important and in section 3 challenges in designing secure embedded systems are presented. In section 4, it is discussed why security is treated as an afterthought. In a system, different stakeholders may have different security expectations. This is discussed with an example in section 5. In section 6, a holistic approach is proposed in which, by studying the system life-cycle, vulnerabilities are predicted and their countermeasures are applied during the system design. Since the complete behavior of the system cannot be predicted, all threats and attacks cannot be prevented. In section 7 a 4-level defense is described that assures a system has been properly designed in terms of security and may be expected not to pose a security hazard.

2 Why Embedded Security

Making embedded systems secure is not only to protect resources and assets; it also provides opportunities for new services and new businesses. In [3] it is argued why the security of embedded systems is important. More arguments are listed here that emphasize the significance of embedded systems security.

Pervasive security
Embedded systems are becoming pervasive as they are becoming cheaper. Their networking degree also is growing to let them have a better synergy by sharing resources and connecting users. They also contain assets of different stakeholders. Networking, sharing resources and holding assets exposes embedded systems to a growing range of threats.

New look
In the past decades a lot of research has been carried out in the area of information security and system security. Many mature and well studied solutions exist. Some of the solutions are applicable to embedded systems but some of them cannot be utilized. The common characteristics of embedded systems ─mobile and resource constrained systems─ enforce researchers to take a new look at current solutions. For example, security solutions that consider the life-cycle of software do not consider the disposal phase as we would do in embedded systems because software does not have a disposal phase.

Safety
Application of embedded systems in areas such as health care, avionics, or car industry where humans are involved raises the issue of safety. For example, the violation of integrity and availability of an artificial hearth, brake of a car and navigation system of an airplane may have disastrous consequences [4]. Attacks are turning from digital-data attacks to human attacks.

Financials
M-commerce is followed by e-commerce, where mobile devices are the main player in financial transactions. Smart cards with e-wallet function or micropayments are examples of embedded systems in finance. There is enough incentive to break into these systems and there is high benefit for financial institutes to protect their systems.

New business model
There will be many new applications or business models that strongly depend on the security techniques of embedded systems e.g., pay-TV, video on demand or time-limited services. Investors will invest in these businesses when they are sure their revenue is properly protected.

Privacy
Some embedded systems are able to sense and capture a huge amount of data about location or status of a user to provide them some services. For example GPS systems process a lot of data about whereabouts of a user. By this information the location and personal information of a user can be observed easily which may affect the user's privacy.

Legal issues
Some applications have legal concerns, e.g., e-voting or road-toll systems. They should meet applicable governmental standards to be acceptable for usage. They should not be manipulated easily. Producers should implement sound security techniques in their products to receive approval from authorities.

Secure identification of components
Third parties will contribute components and subsystems to a system. The secure identification of them is a major concern for a large number of applications. Counterfeiting products and parts (e.g. printer cartridges and ICs) are areas with urgent need for strong and secure device identification. Also, secure identification is important for access control.

Light-weight crypto
Since resources are limited in embedded systems, some of the current security solutions are not applicable. New security solutions with less computational requirements, smaller size and lower energy consumption are necessary.

3 Embedded Security Challenges

Designing secure embedded systems is not straightforward. There are many challenges that should be defied in order to secure them. Some of the challenges are explained below.

Heterogeneity
Most embedded systems are heterogeneous. They include software, hardware, mechanical components, optics, etc., and may consist of different components based on different technologies. Securing a heterogeneous system may be more challenging than a homogeneous system.

Complexity
Embedded systems have constraints which make the application of general security solutions difficult or impossible. Integrating security mechanisms with other functionality requirements is also not straightforward. Some embedded systems have real-time requirements, low power considerations and reliability requirements that should be considered besides security requirements. In fact, security is now a new metric that should be considered besides the other metrics. Meanwhile, security policies may violate other parameters [5]. These issues make security of embedded systems complicated.

Flexibility
Personalization of a system is a desirable feature for users. This implies provision of some flexibility and ability of customization in the system. On the other hand, this flexibility may impact the security of a system. It is challenging to find an equilibrium point of flexibility and security in a system. Also, it is desirable to have a flexible security policy in the system. Since security utilizes resources, in some environments we prefer to reduce the level of security to save resources.

Decentralized control
Not all embedded systems are controlled centrally; some of them are working independently. In some situations maintaining, repairing or restoration of them is done remotely. Some have adaptive behaviors in different environments. These systems will communicate and interact in ways that were unforeseen during their design. In these scenarios, there should be self-adaptive, self-configuring or self-restoring techniques to preserve security.

Alternative energy sources
Side channel attacks are strong attacks based on information gained from the physical implementation of a cryptosystem, e.g., power consumption, electromagnetic leaks, timing information, or even sound [6, 7]. These can provide an extra source of information which can be exploited to attack the system. These attacks and their countermeasures have been studied for a long time [8]. Introduction of alternative energy sources e.g., light, vibration, walking, etc. might introduce new types of side channel attacks.

Time-to-market
The first product that reaches the market is the winner. Time-to-market is a criterion that forces producers to prevent applying well studied security solutions. In this case, producers emphasize more on legal enforcements. Security solutions which will not cause a delay in time-to-market are essential and valuable for producers.

Security Cost
Security needs more management which leads to higher costs. Having cheap security solutions would make systems more secure, since manufacturers avoid utilizing costly solutions. They prefer to add more functionality than securing current functionalities. Affordable security mechanisms are demanding.

We explained the importance of the security of embedded devices, and existing challenges. Current solutions are mostly as an afterthought and the security is not considered from the beginning. In the next section, it is discussed why security is an afterthought.

4 Security: an Afterthought

The software industry and embedded device developers rarely think about security from the beginning. Security is usually an afterthought because the primary consideration of producers and consumers is not security. Companies pay more attention to: Sending the product to the market as soon as possible; Producing a user friendly product; A product with more features that competes better in the market than a more secure product; Massive production for more income and a cheaper device.

Since security affects all of the above considerations it is not an economic priority for companies. Meanwhile, at the moment, security is also not a primary consideration for all users. Most users pay more attention to: Saving money; Ease of use, features and functionalities. Hence companies know there is no immediate return by making their product secure. They have little incentive if the consumer does not consider it to be important. Moreover most consumers do not know the difference between a secure product and an insecure one before purchase. They are more interested in the technologies that solve their problem in the short term and if they want to

opt for more secure technologies, then companies discourage them by higher prices. However, producers and consumers will worry about security of their product when they lose their assets by a breach of their device. As more news of security breaches and hacks are reported, the awareness and importance of security is raised. The best comprehensive solution is considering the security from the beginning and throughout the life-cycle of the system. In next sections, this approach is explained in detail.

5 Security Expectations from a System

The first step in designing a secure embedded system is its security analysis. For security analysis of a system, all the resources that should be protected are specified. Also all stakeholders, both for and against the system should be identified. In fact, whoever has an asset in a system has a security requirement for it and whoever interacts with the system could be a potential adversary of part or the entire system. Different stakeholders have different expectations from the security of a system.

With an example, we explain how different stakeholders are concerned about the security of a system. We suppose a taxi agency that owns some cars and several drivers working for them. Also we suppose the cars in this agency are all modern cars equipped with the latest electronics.

- *User:* entities that use services of a system. In our example, a driver is a user of system. For safety reasons, the availability of computing parts of the car and the location privacy of the driver are some of driver's security concerns.
- *Owner:* entities that have ownership of a system. Owner can be the user of system or they may be separate entities. In our example, the taxi agency is the owner. The agency may record some information on the car, the confidentiality and integrity of which is important for them. Except agency nobody else even the driver of car should have access to that information.
- *Manufacturer:* entities that produce or manufacture the system. Car manufacturers design some components for the car and it is their intellectual property. The confidentiality and integrity of their design, their code and non-forgeability of components are their concern.
- *Repairer or component provider:* entities that maintain or repair the system. This entity can be the manufacturer itself or they may be two separate entities. If the car should be repaired or transferred for updating of some parts or for safety control, a repairer should not have access to the information of the driver or owner. Meanwhile, maintainer may add some components or codes that are their intellectual property.
- *Platform provider:* entities that provide the infrastructure or specialized services to the system such as network facilities, communication links, power sources, etc. Future cars can communicate with each other and forward safety messages. They can also receive information from road side equipment. Such wireless platforms help cars to operate safely, so the availability of such platforms is important. Integrity and confidentiality of messages communicated by cars or road side equipment over these platforms are also a concern.
- *Service provider:* entities that provide some services to the end user or to the owner, such as infotainment or games, etc. Cars can download music, movies or games on their media player and DRM is an issue that infotainment providers worry about.
- *Dealer:* entities that act between manufacturer and end users or between two users. Sometimes a dealer is just a middleman who delivers a system to the user. A dealer may store some information on a car that should be kept confidential. Some information is not confidential to manufacturer or user but a dealer should not have access to them. Integrity and confidentiality of information stored on the device by the manufacturer to be used by the user or owner should be secure from dealer attack. Another example is delivery of a smart card to a user or a dealer may install malicious software on a mobile phone and sell it to a user.
- *Legal bodies (e.g., police):* entities that enforce the legislations of government into a product, e.g., police and standards bodies. Suppose that a car has a digital identification or digital license plate. Its integrity is the police's concern or the digital speedometer of trucks, for example, should not be manipulated by driver.
- *Beneficiary (e.g. bank):* in some systems, in addition to user and manufacturer, there might be another entity that benefits from the usage of the system by its user. Like a bank that is the beneficiary of ATM machines. In our example, the taxi agency is the beneficiary of the cars.
- *Other systems:* entities are not always human beings; they may be other systems which are a user or provider of services to the system. In our example, cars transmit safety messages to other cars. Integrity of these messages is a security concern.

As one can see from this example, many entities are involved in the security requirements of an embedded system. Accomplishing all the security requirements of the various stakeholders is challenging. A strong security modeling is necessary to have a secure design leading to a secure product.

6 Considering security from the beginning

In the literature, the need for considering security from the early stages of design has been emphasized [9, 10, 11]. Some initial efforts towards design methodologies to support security are described in [12, 13, 14, 15] but they don't present a holistic approach as we intend to do in this paper. A sound approach to a secure design is to consider the security from the beginning. There should be an analysis of the life-cycle of the system to detect all the conceivable vulnerabilities and to have

appropriate solutions for preventing those vulnerabilities or threats from happening.

Below the life-cycle of an embedded system is analyzed and possible sources of vulnerabilities in each step are explained. We call this strategy *insecurity prevention*.

6.1 System Life-cycle

The life-cycle of an embedded system consists of three phases: development, use and disposal.

The *development phase* includes all activities from the requirement specification of a product to the decision that the system has passed all acceptance tests and is ready to be delivered. The development phase also consists of some sub-phases such as: Requirement specification, Design, Production, Product shipment and Support/maintenance.

The *use phase* of a system's life-cycle begins when the system is accepted for use and starts the delivery of its services to users. Use phase consists of alternating periods of correct service delivery, service outage, service shutdown and maintenance.

The *disposal phase* is the disposal of an embedded system including media, software, components and data stored on the device. It starts when the system is no longer used or it no longer delivers services. It can also be transferring of the system from one person to another. The disposal phase is essential to prevent inadvertent release of data, information, or software. Security consideration in this phase is beneficial to protect sensitive information from disclosure and adhere to copyright, statutory, and regulatory requirements. In this phase, if user is not going to use the system anymore and he/she discards it, the availability of the system is not of his/her concern, although if the system is transferred to another user, its availability is the new user's concern. In disposal phase, the confidentiality and integrity of system and the data stored on it, is not less important to the user than in use phase.

6.2 Sources of vulnerabilities in the Life-cycle

After analyzing the life-cycle of a system, we should think about the sources of vulnerability in each phase. All the entities involved in the life-cycle phases of an embedded system can be the reason of vulnerabilities and threats.

Sources of the development vulnerabilities

In the development phase, the *physical world* with its natural phenomena, *developers* who are lacking competence or having malicious purposes, *development tools*, *production and test facilities/tools* which are software and hardware used by the developers to assist them in the development process can be the source of vulnerabilities.

Sources of use vulnerabilities

In the use phase, the *physical world*, *Administrators* (authorized people), *Service Users* (limited authorized people), *Service providers*, *Service infrastructure*, *other systems* and *adversaries* are the sources of the vulnerabilities. Involved entities may lack competence or they may have malicious purposes.

Sources of disposal vulnerabilities

The disposal phase can be considered as a special case of use phase with the difference that in disposal phase, if the system is discarded, its availability is not a concern. Therefore availability vulnerabilities and threats are ignored in the security analysis.

The next step in life-cycle security analysis of embedded systems is to find the vulnerabilities originating from these sources and to apply solutions in the design methodology. In this paper, it is not discussed how safeguards are applied in the design methodology.

7 4-Level defense strategy

To design a highly secure system, a 4-Level defense design strategy is proposed. This strategy consists of prevention, tolerance, removal and forecasting. Prevention and tolerance are basic security strategies while removal and forecasting are strategies for security assurance. Many applications need more than security, they need assurance, e.g. military equipments.

7.1 Prevention

The first and the best strategy to create a secure system is insecurity prevention. Prevention means preventing the occurrence or introduction of vulnerabilities. Mostly this is done by solutions and techniques during the development of the system. Improvement of design and development methods can result in good strategies for preventing security vulnerabilities. We mentioned in section 6 a development technique for prevention in which security is considered from the beginning.

Although designers and developers do their best to prevent insecurity in the system, it is not guaranteed that the system will be absolutely secure during its use phase because it cannot be predicted fully where and how the system is going to be used. Therefore, we need to consider solutions for occasions when vulnerabilities were not detected or could not be detected during the development phase and possible attacks can happen.

7.2 Tolerance

Tolerance means providing service in spite of some vulnerabilities or faults in the system. Different techniques can

be applied towards tolerance. We can categorize them into three categories: Vulnerability detection, Recovery and Self-adaptive techniques.

Vulnerability detection

In *vulnerability detection*, a mechanism is provisioned to detect possible vulnerability in a system. This detection can be done in two manners; *concurrent detection* and *preemptive detection*. Concurrent detection takes place during normal service delivery of system. For example, every file is checked for virus infection before execution. Preemptive detection takes place while the normal service delivery is suspended and the system is checked for faults and vulnerabilities e.g., during its idle time, the system is checked for existence of viruses.

However, detecting vulnerabilities is not the whole process of tolerance, after detecting we should handle these vulnerabilities. In fact we consider that there might be some vulnerability. We might detect them before becoming a threat to the system or we might not detect them and an attack could happen. Therefore, we should have strategies for both cases; handling detected vulnerabilities, handling threats or actual attacks. Vulnerability handling prevents vulnerabilities from being activated again. There can be different methods such as: *Diagnosis*, *Isolation*, *Reconfiguration* and *Re-initialization*.

After detecting vulnerabilities, we can collect and record information about the cause of the vulnerability for a future prevention or diagnosis. However, to prevent the utilization of a vulnerability and turning it into a threat for the system, we can isolate physically or logically the faulty component from the process of service delivery; in fact we make the vulnerability dormant. However, if we isolate a component of the system, we may need to switch to either a new redundant component or a spare one. This change in the system implies reconfiguring the system to have new consistent configuration. Also we may need to re-initialize the system and apply some changes in the tables and databases or configuration registry of the system.

Recovery

Still there is a probability that vulnerabilities were not detected and adversary used it and performed an attack. What we can do in case of an attack? In these cases we should have *attack recovery*. Depending on the attack, different strategies can be followed. Attacks on confidentiality may not be recoverable although they may be prevented in future designs. For attacks on data integrity, we can for example replace the data from the backup. The usual mechanism for attacks on availability is redundancy. For integrity and availability, techniques such as *Rollback*, *Rollforward* and *Compensation* can be applied.

Rollback is returning the system back to a saved state that existed prior to vulnerability detection; that saved state is a checkpoint. *Compensation* is replacing the redundant component to enable vulnerability elimination. *Rollforward* is setting the state without detected vulnerability as a new state.

Self-adaptive techniques

Some embedded systems are used in places where they should adapt themselves to the condition of their working surroundings. Also if they were attacked they should recover themselves automatically. In these situations, self-adaptive security techniques are very valuable. Having the ability to recover automatically requires provisioning re-configurability or re-programmability in the system. However existence of these facilities in the system might be the source of threats to the system. Self-adaptive techniques are challenging for security of embedded systems.

Vulnerability prevention and tolerance aim to make a system secure so that the system can deliver a trustworthy service. However, being secure is not enough and in high secure systems, we need assurance. Vulnerability removal and forecasting are for assurance. They help to have confidence in the system by justifying that the functional and the dependability and security specifications are adequate and that the system is likely to meet them.

7.3 Vulnerability removal

Vulnerability removal is performed both during the development phase and during the operational life of a system, i.e., the use phase. Vulnerability removal during the development phase of a system life-cycle consists of three steps: *verification*, *diagnosis*, *correction*. Verification is the process of checking whether the system adheres to given properties. If it does not, the other two steps follow: diagnosing the vulnerabilities that prevented the verification conditions from being fulfilled, and then performing the necessary corrections. After correction, the verification process should be repeated in order to check that vulnerability removal had no undesired consequences.

Verification techniques can be applied statically or dynamically. Verifying a system without actual execution is static verification, via static analysis (e.g., inspections or walk-through), model-checking, and theorem proving. Verifying a system through exercising it constitutes dynamic verification. Verifying that the system cannot do more than what is specified, is especially important with respect to safety and security.

Vulnerability removal during the use phase is corrective or preventive maintenance. Corrective maintenance is aimed at removing vulnerabilities that have produced one or more threats and have been reported, while preventive maintenance is aimed to uncover and remove vulnerabilities before they might cause errors during normal operation. The latter vulnerabilities include a) physical faults that have occurred since the last preventive maintenance actions, and b) design

vulnerabilities that have led to threats in other similar systems. Corrective maintenance for design vulnerabilities is usually performed in stages: the vulnerability may be first isolated (e.g., by a workaround or a patch) before the actual removal is completed.

7.4 Vulnerability Forecasting

Vulnerability forecasting is conducted by performing an evaluation of the system behavior with respect to attack occurrence. Evaluation has two aspects:

- Qualitative, or ordinal, evaluation, which aims to identify, classify, rank attacks, or the event combinations that would lead to system attack.
- Quantitative, or probabilistic, evaluation, which aims to evaluate likelihood that a fault will exist or measuring the difficulty of an attack.

Quantitative evaluations are better understood than the qualitative evaluations. For example, we can evaluate the amount of effort involved in breaking a cryptosystem.
In security forecast we try to have an answer for questions such as:

- How to predict security flaws and human misuse?
- How to predict nature of attacks based on system assets and mission?
- Which parts of system are likely to become under attack?
- How to determine the nature of an attack in its early stages?
- Is it an attack or not? What are its goals? How severe is the attack?

These are the questions whose answers can help forecast the security of a system and the measure of assurance that we can have for it. Security forecasting is an area that needs further study.

8 Conclusion

Security of embedded systems is very important. Strong security mechanisms prevent damages and economical losses while also offering new business opportunities. However, sound security solutions are not attained easily. There are many challenges that should be defied. Although security consideration as an afterthought seems to have short-term incomes and less development difficulties, one simple security breach in a product could result in deletion from the market. A sound solution considers the security from the beginning and analyzes the life-cycle of the system to detect the vulnerabilities from the birth to the death of system. After discovering the sources and the reasons of vulnerabilities, safeguards should be embedded in the design methodology. Although designers and developers try hard to prevent all conceivable attacks, since the use environments and behavior of users cannot be predicted, it is not fully guaranteed that the system is secure. In addition to prevention techniques, tolerance techniques applied in the system help to provide service in presence of failure or attack. Removal and forecasting techniques help to have assurance in the security of the system.

9 References

[1] Peter Marwedel, "Embedded System Design", 1st edition, Kluwer Academic Publishers:Hardbound, pp.1-8, 2003.

[2] Rita C. Summers, "Secure Computing: Threats and Safeguards", pp. 3-11, McGraw-Hill, 1997.

[3] Paar, C., Weimerskirch, A. "Embedded security in a pervasive world", Inf. Secur. Tech. Rep. 12,3, pp.155-161. Jan. 2007.

[4] Carey Goldberg, "Heart devices vulnerable to hack attack", The Boston Globe, March 12, 2008. available online: http://www.boston.com/news/local/articles/2008/03/12/heart_devices_vulnerable_to_hack_attack/

[5] Srivaths Ravi , Paul Kocher , Ruby Lee , Gary McGraw , Anand Raghunathan, "Security as a new dimension in embedded system design", In Proc. ACM/IEEE Design Automation Conf., pp. 753-760, June 2004.

[6] Weingart S., "Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses", Workshop on Cryptographic Hardware and Embedded Systems, 2000.

[7] J. J. Quisquater, D. Samide, "Side channel cryptanalysis", In proceedings of the SECI 2002, pp. 179-184.

[8] Ravi, S., Raghunathan, A., and Chakradhar, S. ,Tamper Resistance Mechanisms for Secure Embedded Systems", In Proceedings of the International Conference of VLSI Design. pp 605-611, 2004.

[9] Srivaths Ravi, Anand Raghunathan, Paul Kocher, Sunil Hattangady, "Security in embedded systems: Design challenges", ACM Transactions on Embedded Computing Systems (TECS) ,Volume 3 , Issue 3, Pages: 461 - 491, 2004.

[10] David Hwang, Patrick Schaumont, Ingrid Verbauwhede, Shenglin Yang, "Multilevel Design Validation in a Secure Embedded System", IEEE Transactions on Computers archive, Pages: 1380 - 1390, 2006.

[11] Joe Grand, "Practical Secure Hardware Design for Embedded Systems", Proceedings of the 2004 Embedded Systems Conference, San Francisco, California.

[12] Eric Uner, "A Framework for Considering Security in Embedded Systems", Embedded.com, Sept. 2005.

[13] Wayne Jansen , Serban Gavrila, Vlad Korolev, Thomas Heute, Clément Séveillac, "A Unified Framework for Mobile Device Security", Proceedings of the International Conference on Security and Management (SAM'04), pp. 9-14, June 2004.

[14] Ingrid Verbauwhede1 and Patrick Schaumont, "Design methods for Security and Trust", Design, Automation & Test in Europe Conference & Exhibition, pp. 1-6, DATE '07, 2007.

[15] Divya Arora, Srivaths Ravi, Anand Raghunathan and Niraj K. Jha, "Architectural Enhancements for Secure Embedded Processing", NATO Workshop on Security and Embedded Systems, VOL 2, pp. 18-25, August 2005.