IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727
PP 11-15
www.iosrjournals.org

Security in Embedded Systems: Vulnerabilities, Pigeonholing of Attacks and Countermeasures

Vijeet H. Meshram[1], Ashish B. Sasankar[2]
[1]vijeet.meshram@gmail.com ,*Department Of Computer Science, Dr.Ambedkar College, India*
[2]ashish_sasankar@yahoo.com, *G.H.Raisoni College of Information and Technology, India*

Abstract: *Embedded systems are growing by leaps and bounds in many domains such as automobiles, industrial control, mobile phones etc. As these devices have started integrating in our daily life we need to look into for some critical measures that is security of these embedded systems. In this paper we look at existing threats and vulnerabilities in embedded systems. We envision that the findings in this paper provide a valuable insight of the threat landscape facing embedded systems. The knowledge can be used for a better understanding and the identification of security risks in system analysis and design.*
Keywords –*Embedded Systems, threats, vulnerabilities, attacks*

I. INTRODUCTION

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. 98 percent of all microprocessors are manufactured as components of embedded systems.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers, but ordinary microprocessors are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.Embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

So the security is an important issue because of the roles of embedded systems in many mission and safety-critical systems. Attacks on cyber systems are proved to cause physical damages [1]. However, comparing to conventional IT systems, security of embedded systems is no better due to poor security design and implementation and the difficulty of continuous patching [2]. Although many approaches have been proposed in the past to secure embedded systems [3], [4], various facts such as deployment scale, resource limitations, the difficulty of physical protection, and cost consideration all make it very challenging to secure them [5], particularly for devices with remote control, maintenance and operation functions.

Having a comprehensive view and understanding of an attacker's capability, i.e. knowing the enemy, is prerequisite for security engineering of embedded systems. Security analysis, secure design and development must take into account the full spectrum of the threat landscape in order to identify security requirements, innovate and apply security controls within the boundary of constraints. As a result, in the scope of embedded systems security, the following questions arise:

- What are the main causes of those successful attacks?
- What are the main vulnerabilities?
- What are the commonalities of the attacks?
- How can we use the knowledge to improve the security of embedded systems?

In this paper, we conduct a systematic review of existing threats and vulnerabilities.

1. Characteristics and Vulnerabilities of Embedded Systems

Many of the inherent characteristics of embedded systems have direct impact on securityrelatedissues. We discuss some of their implications on vulnerabilities in embedded systems.

1.1 Characteristics

Embedded systems are used in special application domains where conventional workstationor server computers are not suitable due to functionality, cost, power requirements, size, or weight. The specialization of embedded system often comes with one or more drawbacks of the following type:

- Limited processing power implies that an embedded system typically cannot run applications that are used for defences against attacks in conventional computer systems (e.g., virus scanner).
- Limited available power is one of the key constraints in embedded systems. Many suchsystems operate on batteries and increased power consumption reduces system lifetime. Therefore embedded system can dedicate only limitedpower resources to providing system security.
- Physical exposure is typical of embedded systems that are deployed outside the immediatecontrol of the owner or operator (e.g., public location, customer premise). Thus,embedded systems are inherently vulnerable to attacks that exploit physical proximity ofthe attacker.
- Remoteness and unmanned operation is necessary for embedded system that are deployedin inaccessible locations (e.g., harsh environment). This limitationimplies that deploying updates and patches as done with conventional workstations isdifficult and has to be automated. Such automated mechanisms provide potential targetsfor attacks.
- Network connectivity via wireless or wired access is increasingly common for embeddedsystems. Such access is necessary for remote control, data collection, updates. In caseswhere the embedded system is connected to the Internet, vulnerabilities can be exploitedremotely from anywhere.

These characteristics lead to a unique set of vulnerabilities that need to be considered in embedded systems.

1.2 Vulnerabilities

Embedded system are vulnerable to a range of abuses that can aim at stealing private information, draining the power supply, destroying the system, or hijacking the system for other than its intended purpose. Examples of vulnerabilities in embedded systems are:

- Energy drainage (exhaustion attack): Limited battery power in embedded systems makesthem vulnerable to attacks that drain this resource. Energy drainage can be achieved byincreasing the computational load, reducing sleep cycles, or increasing the use of sensorsor other peripherals.
- Physical intrusion (tampering): The proximity of embedded systems to a potential attackercreate vulnerabilities to attacks where physical access to the system is necessary.Examples are power analysis attacks or snooping attacks on the system bus.
- Network intrusion (malware attack): Networked embedded systems are vulnerable to thesame type of remote exploits that are common for workstations and servers. An exampleis a buffer overflow attacks.
- Information theft (privacy): Data stored on an embedded system is vulnerable to unauthorizedaccess since the embedded system may be deployed in a hostile environment.Example of data that should be protected are cryptographic keys or electronic currencyon smart cards.
- Introduction of forged information (authenticity): Embedded systems are vulnerable tomalicious introduction of incorrect data (either via the system's sensors or by direct writeto memory). Examples are wrong video feeds in security cameras or overwriting of measurementdata in an electricity meter.
- Confusing/damaging of sensor or other peripherals: Similar to the introduction of maliciousdata, embedded systems are vulnerable to attacks that cause incorrect operation ofsensors or peripherals. An examples is tampering with the calibration of a sensor.
- Thermal event (thermal virus or cooling system failure): Embedded systems need to operatewithin reasonable environmental conditions. Due to the highly exposed operatingenvironment of embedded systems, there is a potential vulnerability to attacks that overheatthe system (or cause other environmental damage).

- Reprogramming of systems for other purposes (stealing): While many embedded systemsare general-purpose processing systems, they are often intended to be used for a particularuse. These systems are vulnerable to unauthorized reprogramming for other uses. Anexample is the reprogramming of gaming consoles to run Linux.

In order to defend embedded systems from these attacks, it is necessary to consider different types of attacks and countermeasures in more detail.

II. PIGEONHOLING OF ATTACKS

Security threats to embedded systems can be classified by the objectives of the attacks or the means to launch the attack [6, 7]. As illustrated above, objectives of the attack can be to prevent privacy, overcome integrity or reduce availability. The means used to launch an attack can be either physical, logical or side channel based. Typical privacy attacks strike at authenticity, access control and confidentiality. Logical attacks on the other hand can be either software based or cryptographic. Examples of physical attacks include micro probing, reverse engineering and eavesdropping. The resources available for reverse engineering increase significantly if someone with manufacturing knowledge attempts to maliciously compromise the system. Integrated circuits may be vulnerable to micro probing or analysis under an electron microscope, once acid or chemical means have been used to expose the bare silicon circuitry. Eavesdropping is the intercepting of conversations by unintended recipients which are permed when sensitive information is passed via electronic media, such as e-mail or instant messaging. Fault injection attacks, power analysis attacks (both Simple Power Analysis (SPA) and Differential Power Analysis (DPA)), timing analysis attacks and electromagnetic analysis attacks are examples of side channel attacks. Side-channel attacks are performed based on observing properties of the system while it performs cryptographic operations.

2.1 Software attacks

Code injection attacks are examples of software attacks which today comprise the majority of all software attacks. The malicious code can be introduced remotely via the network. Cryptographic attacks exploit the weakness in the cryptographic protocol information to perform security attacks, such as breaking into a system by guessing the password. Solutions proposed in the literature to counter cryptographic attacks include run-time monitors that detect security policy violations [8] and the use of safe proof-carrying code [9].

Most of the recent security attacks result in demolishing code integrity of an application program. They include dynamically changing instructions with the intention of gaining control over a program execution flow. Attacks that are involved in violating software integrity are called code injection attacks. Code injection attacks often exploit common implementation mistakes in application programs and are often called security vulnerabilities. The number of malicious attacks always increases with the amount of software code [10,11]. Some of the attacks include stack-based buffer overflows, heap-based buffer overflows, exploitation of double-free vulnerability, integer errors, and the exploitation of format string vulnerabilities.

2.2 Side channel attacks

Side channel attacks are known for the ease with which they can be implemented, and for their effectiveness in stealing secret information from the device without leaving a trace. Adversaries observe side channels such as power usage, processing time and electromagnetic (EM) emissions while the chip is processing secure transactions.

The adversary feeds different input values into the system, while recording the side channels during the execution of a cryptographic algorithm (e.g., encryption using a secret key). These recorded external manifestations are then correlated with the internal computations. Side channel attacks can be performed successfully at either the sender or the receiver to identify the secret keys used for encryption and/or decryption. Power dissipation/consumption of a chip is the most exploited property to determine secret keys using side channel attacks. Kocher et al. [12] first introduced poweranalysis attacks in 1999, where secret keys used in an encryption program were successfully discovered by observing the power dissipation from a chip. Devices like Smart Cards, PDAs and Mobile Phones have microprocessor chips built inside, performing secure transactions using secret keys.

III. COUNTERMEASURES

A. Countermeasures against software attacks

There are several countermeasures proposed in the literature to defend against code injection attacks performed by exploiting common implementation vulnerabilities. These can be divided into nine groups based

on: (1) the system component where the proposed countermeasure is implemented; and (2) the techniques used for the countermeasures. Followingare the nine groups discussed here:

a. Architecture based countermeasures
b. Safe languages
c. Static code analysers
d. Dynamic code analysers
e. Anomaly detection techniques
f. Sandboxing or damage containment approaches
g. Compiler support
h. Library support
i. Operating system based countermeasures

Safe languages such as Java and ML are capable of preventing some of the implementation vulnerabilities discussed here. However, everyday programmers are using C and C++ to implement more and more low and high level applications and therefore the need for safe implementation of these languages exists. Safe dialects of C and C++ use techniques such as restriction in memory management to prevent any implementation errors.

Compilers play a vital role in enabling the programs written via language specifications to run on hardware. The compiler is the most convenient place to insert a variety of solutions and countermeasures without changing the languages in which vulnerable programs are written. The observation that most of the security exploits are buffer overflows and are caused by stack based buffers, has made researchers propose stack-frame protection mechanisms. Protection of stack-frames is a countermeasure against stack based buffer overflow attacks, where often the return address in the stack-frame is protected and some mechanisms are proposed to protect other useful information such as frame pointers. Another commonly proposed countermeasure is to protect program pointers in the code. This is a countermeasure which is motivated by the fact that all code injection attacks need code pointers to be changed to point to the injected code. Since buffer overflows are caused by writing data which is over the capacity of the buffers, it is possible to check the boundaries of the buffers when the data is written to prevent buffer overflow attacks.

Operating system based solutions, use the observation that most attackers wish to execute their own code and have proposed solutions preventing the execution of such injected code. Most of the existing operating systems split the process memory into at least two segments, code and data. Marking the code segment read-only and the data segmentnon-executable will make it harder for an attacker to inject code into a running application and execute it [13].

B. Countermeasures against side channel attacks

There are several countermeasures against side channel attacks. These have been divided into six categories:

a) Masking
b) Window method
c) Dummy instruction insertion
d) Code/algorithm modification
e) Balancing
f) Other methods

To mask code execution and to confuse an adversary, noise can be injected during code execution. Substitution Boxes (SBOXes), often used in cryptology, can also be masked in the execution. A window method can be applied in Public Key Cryptosystems to prevent power analysis based side channel attacks. In the window method, a modular exponentiation can be carried out by dividing the exponent into certain sizes of windows, and performing the exponentiation in iterations per window by randomly choosing the window [14].

Public Key Cryptosystems like RSA and ECC have been severely attacked using Simple Power Analysis (SPA), mainly because of the conditional branching in the encryption. Such vulnerabilities in the program can be prevented by modifying the implementation or replacing with a better new algorithm to perform the same task. Some of the other techniques include signal suppression circuits, which can be used to reduce the Signal-to-Noise Ratio (SNR) to prevent the adversary from differentiating the power profile.

IV. CONCLUSION

This paper provides a comprehensive overview of embedded systems security by describing attacks, vulnerabilities and countermeasures. It enables us to create an attack taxonomy which we used to classify and describe common attack scenarios against embedded systems. The attack taxonomy derived in this paper

provides information on how an embedded system can be attacked. Moreover, the structured knowledge can assist analysis and design of systems including or based on embedded devices during system development lifecycle. The presented attack taxonomy also helps us to forecast trends in embedded-system security.

Considering the attacks and vulnerabilities discussed in this paper and the recent trends in machine-to-machine communications, in our opinion, Internet facing devices will continue to suffer the majority of attacks. Traditional IT systems already have solutions and tools to address these issues. We anticipate that the solutions will be deployed in embedded systems with modifications tailored for the needs of this field. Our next step will be to further validate the taxonomy in realistic settings through different use cases led by industry. Moreover, the taxonomy and the knowledge will be applied to security analysis of cyber-physical systems to identify and enumerate threats in a systematic way with reduced error and uncertainty.

REFERENCES

[1]. R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," Security &Privacy, IEEE, vol. 9, no. 3, pp. 49–51, 2011.

[2]. B. Schneier, "Security risks of embedded systems," https://www.schneier.com/blog/archives/2014/01/security risks 9.html, January 2014.

[3]. S. Parameswaran and T. Wolf, "Embedded systems security – an overview," Design Automation for Embedded Systems, vol. 12, no. 3, pp. 173–183, 2008.

[4]. D. Kleidermacher and M. Kleidermacher, Embedded systems security: practical methods for safe and secure software and systems development. Elsevier, 2012.

[5]. D. N. Serpanos and A. G. Voyiatzis, "Security challenges in embedded systems," ACM Transactions on Embedded Computing Systems (TECS), vol. 12, no. 1s, p. 66, 2013.

[6]. Ravi S, Raghunathan A, Chakradhar S (2004) Tamper resistance mechanisms for secure, embedded systems. In: 17th international conference on VLSI design, January 2004

[7]. Ravi S, Raghunathan A, Kocher P, Hattangady S (2004) Security in embedded systems: design challenges. Trans Embed ComputSyst 3(3):461–491

[8]. Kiriansky V, Bruening D, Amarasinghe SP (2002) Secure execution via program shepherding. In: Proceedings of the 11th USENIX security symposium. USENIX Association, Berkeley, pp 191–206

[9]. Necula GC (1997) Proof-carrying code. In: Conference record of POPL '97: the 24th ACM SIGPLANSIGACT symposium on principles of programming languages, Paris, France, January 1997, pp 106–119

[10]. CERT Coordination Center (2004) Vulnerability notes database. CERT Coordination Center

[11]. CERT Coordination Center (2005) CERT/CC vulnerabilities statistics 1988–2005. CERT Coordination Center

[12]. Kocher P, Jaffe J, Jun B (1999) Differential power analysis. In: Lecture notes in computer science, vol 1666. Springer, Berlin, pp 388–397

[13]. Chew M, Song D (2002) Mitigating buffer overflows by operating system randomization. Technical Report CMU-CS-02-197, Department of Computer Science, Carnegie Mellon University, December 2002

[14]. Nedjah N, de MacedoMourelle L, da Silva RM (2007) Efficient hardware for modular exponentiation using the sliding-window method. In: ITNG '07: proceedings of the international conference on information technology. IEEE Computer Society, Washington, pp 17–24