Incorporating Security in Embedded System – A critical analysis

Mirza Aamir Mehmood[1], Amir Shahzad Khokhar[2] and Mazhar Ali[3]

Department of Computer Science[1,2], Department of Information Technalogy[3]

Balochistan University of Information Technology, Engineering & Management Sciences
Quetta, Pakistan

Abstract
Security is becoming a major concern in embedded system designing and development. This paper surveys security requirements, attack techniques and will review countermeasures for these attacks

Keywords: component; embedded systems, security, software attacks, viruses

1. Introduction

Use of embedded systems in diverse and complex applications have relentless importance of secure and fault tolerant system. It is essential for embedded device's to secure sensitive information, ensuring availability and providing secure communication system. Reliability is directly coupled with security in embedded systems thus an unsecured system is also an unreliable system. Traditionally security concaved as an issue related to networks and cryptography and embedded system designers consider it as an additional feature. Growing number of security breaches has dictated that compromise on security can lead to consequences ranging from inconvenience to a complete disaster. Embedded systems are co-design of Software and Hardware. To make a secure and reliable system, it is essential that both components are secure.

2. Embedded System Security Requirements

The functions of embedded system is to access and process sensitive data and provide critical functionality. For example, let's assume that an embedded system has been implanted in a heart patient to monitor heartbeat, blood pressure and sugar level of that person. When it finds any anomaly in blood pressure or sugar level, it send an alarm signal to doctor and when it found that heart beat is gone down a certain limit, it generates a mild shock to save life of that person. In this example, it is very obvious that system should be available twenty four hours a day and seven days a week. It is also important that when communicating with doctor, it should ensure privacy of patient. In general, an embedded system should ensure dependability, confidentiality, integrity and availability to consider as secure system [4] [5]. Figure 1 shows security requirements for embedded system (Taken from [1]).

Figure 1. Security requirements for embedded system

2.1. Confidentiality

To ensure that sensitive information is protected against deliberate or accidental disclosure.

2.2. Integrity

Ensuring that sensitive information is protected against deliberate or accidental corruption and data is protected against illegitimately changes.

2.3. Availability

The ability to protect against deliberate or accidental actions that cause automated information resources to be unavailable to users when needed.

2.4. Dependability

Embedded systems often reside in machines that are expected to run continuously for years without errors/faults and in some cases recover by them-selves if an error/fault occurs [3]. Thus an embedded system should be dependable. Dependability is combination of

2.4.1. Fault-Avoidance

Constructing / developing a mechanism to prevent a fault situation to occur. This is accomplished in designing phase.

2.4.2. Fault-Tolerance

Ensuring that system behaves in normal fashion even when a fault situation arises. This is accomplished through redundancy.

2.4.3. Fault-Removal

In verification phase, it is ensured that system is error/fault free.

2.4.4. Fault-Forecasting

How to estimate, by evaluation, the presence, the creation and the consequences of errors.

3. Challenges in Secure Software Development

Many external factors influence confidentiality and availability of system administrative controls, physical barriers, are few of them. Whereas integrity of the computer system depends on the degree to which vulnerabilities have been eliminated from the system [2]. In addition to the requirements, discussed earlier, embedded system should also be able to face threats like denial of service attacks, system tempering etc. which becomes more complicated in presence of modern techniques for breaking security, such as power analysis and fault analysis. Although Software solutions are not sufficient to keep up with the computational demands of security processing in embedded system but they are major source of security vulnerabilities. Three main factors make development of secure software a challenge i.e. Complexity, Extensibility and Connectivity [1]. Let's briefly examine each of these issues.

3.1. Complexity

With every passing day, software's are becoming more and more complexes. With each new line of code, new bugs and security risks introduced in software. The complexity aggravate when unsafe programming languages (e.g., C or C++) are used, which do not protect against simple kind of attacks, such as buffer overflows.

3.2. Extensibility

New development platforms like .Net and Java developed mainly to provide extensibility. Embedded system now can get updates or extensions from internet for example new mobile phones by Sony Ericsson accept BIOS update through internet. Unfortunately, the very nature of extensible systems makes it hard to prevent software vulnerabilities from slipping in as an unwanted extension.

3.3. Connectivity

Most of modern embedded systems are coming with built in connectivity with internet. This internet connectivity has made it possible that a small problem can propagate and result in substantial security breaches. This also means that an intruder does not require a physical access to embedded system to launch attack and exploit vulnerable software. Thus this ubiquity of networking will result in increase in number of attacks and greater risks from par software security practices.

4. Attacks on Embedded System

Attacks on embedded systems can be categorized in three classes i.e. software attacks, physical attacks and side channel attacks. Software attacks have largest share in total number of attacks on embedded systems and it is most difficult to protect against such attacks. In this article we will focus on software attacks and countermeasure against these attacks. An overview of physical and side channel attacks will be provided.

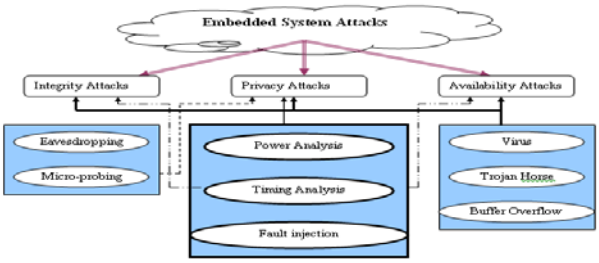

Figure 2. Embedded System attacks [6]

4.1. Physical Attacks

Embedded system can be divided in two categories
1. System on circuit board
2. System-on-chip

On circuit board embedded systems, attacks can be launched by probe to eavesdrop on inter-component communications. Whereas when launching attack on system-on-chip, micro-probing techniques are required. Physical attacks are relatively hard because they require expensive infrastructure and very complex techniques are used.

The most important part of any embedded system is microcontroller as it essentially controls all the operation of embedded system. The attacks on the microcontroller can be possible via JTAG, it is necessary to disable access to the microcontroller's internals via JTAG before fielding the finished product

4.2. Side Channel Attacks

Side channel attacks are based on observing system properties e.g. time, power consumption while system is performing computations e.g. cryptographic operations. In certain systems timing information can lead to entire secret key, though it seems that timing information can give very little information but it has found that with proper study of timing sequence entire secret key can be found.
Along with this power consumption can also lead to the entire secret key, well equipped labs have the equipment that can measure the changes in the power consumption with about 1% accuracy and are very inexpensive. To overcome timing attacks one may add random timing delays to various operations, in similar manner we can overcome power consumption attacks by adding random noise or by proper shielding of the equipment but it leads to increased cost of the equipment.

4.3. Software Attacks

Software attacks are very common in embedded systems capable of downloading application from internet or have some means of communication to interact with external world. As compare to physical and side channel attacks, software attacks are very cheap and does not requires any big infrastructures thus making it an immediate challenge for embedded system design. These attacks could further classified in three categories 1. Virus attacks 2. Buffer Overflow and 3. Exploiting Software Vulnerabilities.

4.3.1. Virus, Worm and Trojan

These attacks executed through malicious agents like virus, worm, Trojan. Following table 1 lists some viruses and respective embedded operation system.

Table 1 Viruses and Respective Embedded Operation System

Operating system	***Creator***	***Known viruses (including variants)***	***First virus appearance***
BlackBerry OS	Research In Motion	1	August 2006
Embedded Linux	GNU Project, Linus Torvalds and al.		
Mac OS X	Apple Inc.	0	-
Palm OS	PalmSource, Inc.	4	September 2000
Symbian OS	Symbian Ltd.	83	June 2004
Windows Mobile	Microsoft	2	July 2004

4.3.2. Vulnerability Exploitation

These agents exploit weaknesses in end-system architecture [7, 8, 9, 10]. "A vulnerability allows the attacker to gain direct access to the end-system, while an exposure is an entry point that an attacker may indirectly exploit to gain access" [6]. Following table lists latest list of vulnerability published by CERT.

Table 2 . Vulnerability List Published By CERT [10]

Vulnerability ID	**Description**
VU#298521	SonicWall NetExtender NELaunchCtrl ActiveX control stack buffer overflow
VU#446897	CUPS buffer overflow vulnerability
VU#180345	Microsoft Kodak Image Viewer code execution vulnerability
VU#342793	RSA Keon cross-site scripting vulnerabilities
VU#871673	RealPlayer playlist name stack buffer overflow
VU#559977	Mozilla products vulnerable to memory corruption in the browser engine
VU#755513	Mozilla products vulnerable to memory corruption in the JavaScript engine
VU#349217	Mozilla XUL web applications may hide the titlebar
VU#230505	Cisco IOS LPD buffer overflow vulnerability
VU#179281	Electronic Arts SnoopyCtrl ActiveX control and plug-in stack buffer overflows

4.3.3. Buffer Overflow

The buffer overflow is a common flaw in OS and Application software. It's the inability of the buffer to store the information, which results from adding more information to a buffer than it was designed to hold. An attacker may exploit this vulnerability to take over a system". This problem arises when buffers used with poor boundary checks. Buffer bounds may be violated due to incorrect loop bounds, format string attacks, etc. Buffer overflow effects can include overwriting stack memory, heaps, and function pointers. Intruders sue buffer overflow to overwrite program addresses stored nearby. "This may allow the attacker to transfer control to malicious code, which when executed can have undesirable effects. A good high-level introduction to the challenges involved in writing secure code can be found in [10]". In figure 3, the left-hand chart shows, for
Each year, the total number of CERT-reported vulnerabilities and the number that can be blamed primarily on buffer overruns whereas right-hand chart graphs the percentage of CERT-reported vulnerabilities that were due to buffer overruns for each year (taken from [15]).

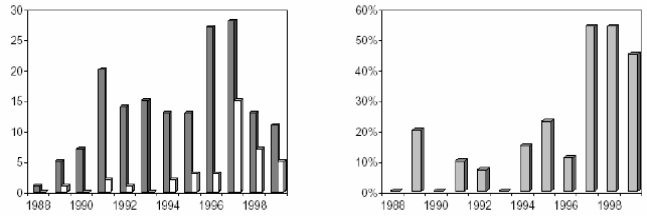

Figure 3. Frequency of buffer overrun vulnerabilities, derived from a classification of CERT advisories[16].

5. Countermeasure for software attacks

When designing countermeasures against software attacks, confidentiality and integrity of data and code is ensured, all security loopholes that make system vulnerable are removed. "A common feature of these countermeasures involves regulating the accesses of various software components (operating system, downloaded code, *etc.*) to different portions of the system (registers, memory regions, security co-processors, *etc.*) during different stages of execution (boot process, normal execution, interrupt mode, *etc.*), through a combination of hardware and software changes. Since an effective countermeasure must allow the system to provide guarantees about the security of the system starting from the powered-on state, most measures define notions of trust or *trust boundaries* (also referred to as *security perimeters*) across the various hardware and software resources. This allows the system to detect infringements of trust boundaries (such as illegal accesses to memory regions) and enforce recovery mechanisms (such as zeroing processor registers and memory regions). Thus, a trust boundary provides a natural and convenient foundation for the system to make judicious decisions about its security (or compromise, thereof)" [6].

5.1. Hardware Support

At hardware level security commonly implemented by using secure co-processor module [17, 18,19,22] this co-processor processes sensitive information. Information that needs to be send out of the co-processor is encrypted. Many embedded system also have secure memory areas. These secure memory areas are accessible to trusted system components only.

5.2. Secure Bootstrapping

Integrity check could be implemented at boot process level. After power is switched on, system should only be able to access the next layer if all integrity check found intact. This is done by comparing the securely saved value with hashed value of boot process component.

5.3. Operating System (OS) Enhancements

Secure Operating Systems provide features like process isolation, process attestation and secure storage. Secure storage is ensured by using of cryptographic file systems (CFSs) [20,21].

6. Security Pyramid

Embedded systems have three main components, Hardware, Software and communication mechanism / communication protocols. It is important that all three components should have built in security mechanism. Following figure shows suggested security pyramid for embedded systems.

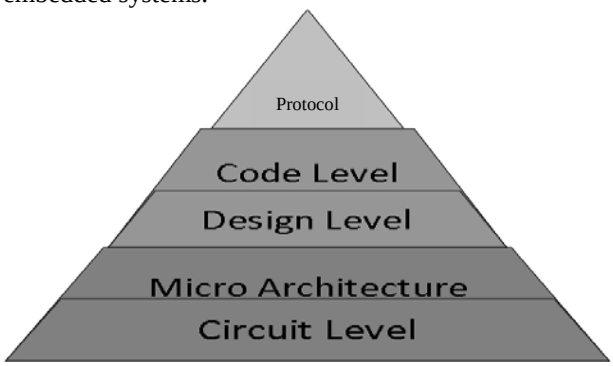

Figure 4. Security Pyramid for embedded systems

6.1. Communication / Protocol Level

Various security protocols have been developed to address security at different layers of network protocols, the most popular being IPSec, a network layer protocol, and TLS/SSL which works on transport layer [13,14]. Security protocols use algorithms (asymmetric or public-key ciphers, symmetric or private-key ciphers, hashing functions, etc.) to make communication secure.

Figure 5. A simple embedded system under protocol attack [16]

6.2. Software Level

To ensure that the software for embedded system meets security requirements, it is important that security should be implemented on various levels.

6.2.1. Code and Algorithm Level
Static analysis tool could be used to scan code to uncover common vulnerabilities. These tools can detect bugs at code level so that they could be fixed [14].
6.2.2. Design and Architecture Level
System must be coherent and present a unified security architecture that takes into account security principles (such as the principle of least privilege) [14].
6.2.3. Requirements Level
Security requirement should cover functional security.
6.2.4. Hardware Level Security:
At hardware level, security should be implemented on Micro-Architecture Level and at Circuit Level.
6.2.5. Micro Architecture Level
Incorporating security at hardware design of the modules (the processors and coprocessors) which is specified at the architecture level
6.2.6. Circuit Level
Implementing security at this level means use of techniques at transistor level and package-level to prevent various physical-layer attacks.

7. Future Work

Active research is going on to develop secure platforms for embedded systems. One approach is trusted computing, a project of Trusted Computing Group The embedded processor community has also presented some design alternatives to achieve security e.g. ARM has developed new security architecture [23]. The security is achieved by portioning all the software and hardware resources so that they exist in one of two worlds - the secure world for the security subsystem, and the Normal world for everything else. .This Architecture is deployed at both the levels i.e. the hardware level and the software level.

At hardware level they have designed the extended bus; the most significant feature of this bus is extended control signals that will restrict non- secure masters to access secure slaves. ARM has developed new architecture by using its “ARM Trust Zone Technology”

MIPS has also developed a secure processor core that includes secure memory management, protection against side-channel attacks, and uses an architecture that provides fast software cryptography using the SmartMIPS extensions to the MIPS32 architecture..

8. Conclusion

In this survey paper, we have examined security requirements and challenges in development of embedded systems. We have also examined how an embedded system can be attacked and their counter measures are also examined. We believe that implementing security pyramid discussed in this article will enable developers and designers of embedded system to develop more secure system.

9. References

[1]. Srivaths Ravi , Paul Kocher , Ruby Lee , Gary McGraw , Anand Raghunathan, Security as a new dimension in embedded system design, Proceedings of the 41st annual conference on Design automation, June 07-11, 2004, San Diego, CA, USA
[2]. McCoy, J. A. An embedded system for safe, secure and reliable execution of high consequence software. In HASE 2004: The 5th IEEE International Symposium on High Assurance Systems Engineering.
[3]. Online encyclopedia available at www.wikipedia.com
[4]. Summers, Rita C. Secure Computing: Threats and Safeguards. New York: McGraw-Hill, 1997
[5]. LAPRIE, J.C. Dependable computing and fault tolerance: concepts and terminology. In Proceedings of the 15th IEEE Symposium on Fault Tolerant Computing Systems (FTCS-15, June 1985). IEEE Computer Society Press, Los Alamitos, CA, 2-11.
[6]. Srivaths Ravi , Anand Raghunathan , Srimat Chakradhar, Tamper Resistance Mechanisms for Secure, Embedded Systems, Proceedings of the 17th International Conference on VLSI Design, p.605, January 05-09, 2004
[7]. Common Vulnerabilities and Exposures. Available at cve.mitre.org
[8]. Latest Virus Threats. Symantec Corporation. Available at http://www.symantec.com/avcenter/vinfodb.html
[9]. Virus Information. Computer Security Resource Center, National Institute of Standards and Technology. Available at http://csrc.nist.gov/virus/
[10]. Vulnerability notes database. CERT coordination center Available at http://www.kb.cert.org/vuls/
[11]. M. Howard and D. LeBlanc. Writing Secure Code. Microsoft Press, 2001.
[12]. Buffer over flow techniques in computer viruses. Symantec white paper. Available at http://securityresponse.symantec.com/avcenter/whitepapers.html
[13]. W. Stallings, Cryptography and Network Security: Principles and Practice. Prentice Hall, 1998.
[14]. B. Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C. John Wiley and Sons, 1996.
[15]. John Viega, Gary McGraw , Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley Professional, 2001
[16]. Lawrence P. Ricci , Larry B. Mcginness , White Paper Embedded System Security Designing Secure Systems with Windows CE.
[17]. B. Yee, Using Secure Co-processors. PhD thesis, Carnegie Mellon University, 1994.
[18]. Secure Co-processing. IBM Inc. Available at http://www.research.ibm.com/scop/
[19]. The IBM PCI Cryptographic Coprocessor. IBM Inc. Available at http://www-3.ibm.com/security/cryptocards/
[20]. M. Blaze, “A Cryptographic File System for UNIX,” in Proc. ACM Conf. on Computer and Communications Security, pp. 9–16, Nov. 1993.
[21]. T. Garfinkel, M. Rosenblum, and D. Boneh, “Flexible OS Support and Applications for Trusted Computing,” in Proc. 9th Wkshp Hot Topics in Operating Systems, May 2003.
[22]. E. Goh, H. Shacham, N. Modadugu, and D. Boneh, “SiRiUS: Securing Remote Untrusted Storage,” in Proc. ISOC Network and Distributed Systems Security (NDSS) Symp., pp. 131–145, 2003.
[23]. Jaggar, D.; , "Arm Architecture And Systems," Micro, IEEE , vol.17, no.4, pp.9-11, Jul/Aug 1997