Des Autom Embed Syst (2008) 12: 173–183
DOI 10.1007/s10617-008-9027-x

# Embedded systems security—an overview

**Sri Parameswaran · Tilman Wolf**

Received: 16 June 2008 / Accepted: 25 June 2008 / Published online: 17 July 2008


**Abstract** Security is an important aspect of embedded system design. The characteristics of embedded systems give rise to a number of novel vulnerabilities. A variety of different solutions are being developed to address these security problems. In this paper, we provide a brief overview of important research topics in this domain.

**Keywords** Embedded system design · Vulnerabilities · Security

## 1 Introduction

Security in embedded systems is a topic that has received an increasing amount of attention from industry and academia in recent years. Embedded systems are being deployed in a wide range of application areas ranging from control of safety-critical systems to data collection in hostile environments. These devices are inherently vulnerable to many operational problems and intentional attacks due to their embedded nature. Network connectivity opens even more avenues for remote exploits. In response, security solutions are being developed to provide robustness, protection from attacks, and recovery capabilities.

In this article, we provide an overview on embedded system security. We discuss how the characteristics of embedded systems lead to a set of potential vulnerabilities. We also provide a brief survey of attacks on embedded systems and corresponding countermeasures. For other overview articles on embedded system security, see [57, 77].

S. Parameswaran
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
e-mail: sridevan@cse.unsw.edu.au

T. Wolf (✉)
Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA
e-mail: wolf@ecs.umass.edu

## 2 Characteristics and vulnerabilities of embedded systems

Many of the inherent characteristics of embedded systems have direct impact on security-related issues. We discuss some of their implications on vulnerabilities in embedded systems.

### 2.1 Characteristics

Embedded systems are used in special application domains where conventional workstation or 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 defenses against attacks in conventional computer systems (e.g., virus scanner, intrusion detection system).
- *Limited available power* is one of the key constraints in embedded systems. Many such systems operate on batteries and increased power consumption reduces system lifetime (or increases maintenance frequency). Therefore embedded system can dedicate only limited power resources to providing system security.
- *Physical exposure* is typical of embedded systems that are deployed outside the immediate control of the owner or operator (e.g., public location, customer premise). Thus, embedded systems are inherently vulnerable to attacks that exploit physical proximity of the attacker.
- *Remoteness and unmanned operation* is necessary for embedded system that are deployed in inaccessible locations (e.g., harsh environment, remote field location). This limitation implies that deploying updates and patches as done with conventional workstations is difficult and has to be automated. Such automated mechanisms provide potential targets for attacks.
- *Network connectivity* via wireless or wired access is increasingly common for embedded systems. Such access is necessary for remote control, data collection, updates. In cases where the embedded system is connected to the Internet, vulnerabilities can be exploited remotely from anywhere.

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

### 2.2 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 makes them vulnerable to attacks that drain this resource. Energy drainage can be achieved by increasing the computational load, reducing sleep cycles, or increasing the use of sensors or other peripherals.
- *Physical intrusion (tampering):* The proximity of embedded systems to a potential attacker create 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 the same type of remote exploits that are common for workstations and servers. An example is a buffer overflow attacks.
- *Information theft (privacy):* Data stored on an embedded system is vulnerable to unauthorized access since the embedded system may be deployed in a hostile environment. Example of data that should be protected are cryptographic keys or electronic currency on smart cards.
- *Introduction of forged information (authenticity):* Embedded systems are vulnerable to malicious introduction of incorrect data (either via the system's sensors or by direct write to memory). Examples are wrong video feeds in security cameras or overwriting of measurement data in an electricity meter.
- *Confusing/damaging of sensor or other peripherals:* Similar to the introduction of malicious data, embedded systems are vulnerable to attacks that cause incorrect operation of sensors or peripherals. An examples is tampering with the calibration of a sensor.
- *Thermal event (thermal virus or cooling system failure):* Embedded systems need to operate within reasonable environmental conditions. Due to the highly exposed operating environment of embedded systems, there is a potential vulnerability to attacks that overheat the system (or cause other environmental damage).
- *Reprogramming of systems for other purposes (stealing):* While many embedded systems are general-purpose processing systems, they are often intended to be used for a particular use. These systems are vulnerable to unauthorized reprogramming for other uses. An example 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.

## 3 Attacks and countermeasures

Security threats to embedded systems can be classified by the objectives of the attacks or the means to launch the attack [76, 77]. 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 microprobing, 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 microprobing or analysis under an electron microscope, once acid or chemical means have been used to expose the bare silicon circuitry [70]. Eavesdropping is the intercepting of conversations by unintended recipients which are performed when sensitive information is passed via electronic media, such as e-mail or instant messaging.

Fault injection attacks [12, 13], power analysis attacks [59] (both Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [54]), timing analysis attacks [15] and electro magnetic analysis attacks [75] are examples of side channel attacks. Side-channel attacks are performed based on observing properties of the system while it performs cryptographic operations.

## 3.1 Attacks on embedded systems

### *3.1.1 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. A short list of common crypto and protocol vulnerabilities is given in [77]. Solutions proposed in the literature to counter cryptographic attacks include run-time monitors that detect security policy violations [53] and the use of safe proof-carrying code [68].

Most of the recent security attacks result in demolishing code integrity of an application program [64]. 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 [18, 19]. 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.

### *3.1.2 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 [89]. Adversaries observe side channels such as power usage [59], processing time [15] and electro magnetic (EM) emissions [71] 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 [56, 89]. Kocher et al. [54] first introduced power analysis 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 [11, 22], PDAs [44] and Mobile Phones [88] have microprocessor chips built inside, performing secure transactions using secret keys.

## 3.2 Countermeasures

### *3.2.1 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. Following are the nine groups discussed here:

1. Architecture based countermeasures
2. Safe languages
3. Static code analyzers
4. Dynamic code analyzers
5. Anomaly detection techniques
6. Sandboxing or damage containment approaches
7. Compiler support
8. Library support
9. Operating system based countermeasures

As return addresses of functions are the most attacked target of buffer overflows, there are many hardware/architecture assisted countermeasures that aim to protect these addresses. Some of these techniques are described in [2, 7, 51, 73]. Another technique to counter code injection attack is to ensure code integrity at runtime. The authors in [72] have proposed a microarchitectural technique to ensure program code integrity at runtime and thereby preventing code injection attacks. An embedded monitoring system to check correct program execution is proposed in [60].

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. Examples of such methods are shown in [25, 32, 35].

Static Code Analyzers, analyze software without actually executing programs built from that software [81]. In most cases the analysis is performed on the source code and in the other cases on some form of the object code. The quality of the analysis performed by these tools ranges from those that only consider the behavior of simple statements and declarations, to those that include the complete source code of a program in their analysis. The information collected by these analyzers can be used in a range of applications, starting from detecting coding errors to formal methods that mathematically prove program properties. Examples of static code analyzers are shown in [3, 17, 26].

In dynamic code analysis, the source code is instrumented at compile time and then test runs are performed to detect vulnerabilities. Even though performing dynamic code analysis is more accurate than static analysis (more information of the execution is available at runtime compared to compile-time), dynamic code checking might miss some errors as they may not fall on the execution path while being analyzed. Some well known dynamic code analyzers are shown in [31, 38, 45].

Behavior-based anomaly detection compares a profile of all allowed application behavior to actual behavior of the application. Any deviation from the profile will raise a flag as a potential security attack [48] . This model is a positive security model as this model seeks only to identify all previously known good behaviors and decides that everything else is bad. Behavior anomaly detection has the potential to detect several type of attacks, which includes unknown and new attacks on an application code. Most of the time, the execution of system calls is monitored and is recorded as an anomaly if it does not correspond to one of the previously gathered patterns. A threshold value for the number of anomalies is decided a priori and when the threshold is reached, the anomaly can be reported to the system and subsequent action, such as terminating the program or declining a system call can be taken. On the negative side, behavior anomaly detection can lead to a high rate of false positives. For instance, if some changes are made to the application after a behavior profile is created,

behavior-based anomaly detection will wrongly identify access to these changes as potential attacks. Some examples of this technique are described in [40, 50, 82].

Sandboxing is a popular method for developing confined execution environments based on the principle of least privilege, which could be used to run untrusted programs. A sandbox limits or reduces the level of access its applications have to the system. Sandboxes have been of interest to systems researchers for a long time. Butler Lampson, in his 1971 paper [58], proposed a conceptual model highlighting properties of several existing protection and access-control enforcement mechanisms. Other examples are given in [36, 37, 53].

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. Solutions proposed as compiler support for bounds checking are also discussed in this section. Some examples of the techniques are given in [4, 14, 16].

Safe library functions attempt to prevent vulnerabilities by proposing new string manipulation functions which are less vulnerable or invulnerable to exploitations. In [65] the authors propose alternative string handling functions to the existing functions which assume strings are always NULL terminated. The new proposed functions also accept a size parameter apart from the strings themselves. In [61], another safe string library is proposed as a replacement to the existing string library functions in C. Other examples are shown in [5, 10, 27].

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 segment non-executable will make it harder for an attacker to inject code into a running application and execute it [24, 33, 41].

### *3.2.2 Countermeasures against side channel attacks*

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

1. Masking
2. Window method
3. Dummy instruction insertion
4. Code/algorithm modification
5. Balancing
6. Other methods

To mask code execution and to confuse an adversary, noise can be injected during code execution. Examples of masking techniques are presented in [78, 86, 87]. Substitution Boxes

(SBOXes), often used in cryptology, can also be masked in the execution. Some examples for the SBOX masking techniques are presented in [28, 42, 46].

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 [69].

Dummy instructions can be placed to provide random delays. This confuses the adversary when attempting to correlate the source implementation with the power profile. Chari et al. [20] claimed that countermeasures involving random delays (i.e., dummy instructions used to provide random delays in an execution) should be performed extensively, otherwise they can be undone and re-ordered, causing a successful attack. Several dummy instruction approaches are presented in [1, 6, 43].

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. Key code modification techniques to prevent power analysis are explained in [6, 20, 23].

The software code can be modified in such a way that complementary events are coded to negate the effects of the actual computations. Examples of such code balancing techniques are presented in [21, 29, 79]. Evidently, balancing at the gate level is the most appropriate solution to prevent power analysis, since the power is consumed/dissipated depending on the switching activities in gates. Hardware balancing is primarily performed by placing two gates in parallel, one complements the other when switching. Various hardware balancing techniques are given in [30, 34, 39].

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. Examples for the suppression circuits are given in [52, 67, 74]. Software level current balancing approaches are performed by modifying the source and inserting *nop*s to keep the current constant [66].

May et al. [62] proposed a non-deterministic processor design, where the independent instructions are identified and executed out-of-order in a random choice by the processor. This infringes the conventional attack rule removing the correlation between multiple executions of the same program, thus preventing the adversary from comparing different runs for power analysis. Several other improved versions of the non-deterministic processor architecture are proposed in [49, 63].

Randomizing the clock signal [83] for the secure processor to confuse the adversary is another countermeasure proposed to prevent power analysis. This prevents the adversary from analyzing the clock signals to identify certain significant instruction executions in the power profile. More examples on handling the clock signal to prevent power analysis are presented in [8, 9, 21, 29, 55].

Power analysis can also be prevented by designing special instructions whose power signature is difficult to analyze [46] or whose power consumption is data independent [80]. Several examples of creating extensible instructions are given in [39, 47, 84, 85]. Such extensible instruction designs can also be adapted to prevent power analysis attacks.

## 4 Summary

In summary, embedded systems require special security considerations due to their inherent characteristics and unique usage scenarios. Research work in the field of embedded system

security is in the process of identifying attack scenarios, developing counter measures, and novel system designs with inherent security properties.

**Acknowledgements** We would like to thank Roshan Ragel, Angelo Ambrose and Jorgen Peddersen for their contributions in putting together this article.

## References

1. Aciiçmez O, Koç ÇK, Seifert J-P (2007) On the power of simple branch prediction analysis. In: ASIACCS '07: proceedings of the 2nd ACM symposium on information, computer and communications security. ACM, New York, pp 312–320
2. Arora D, Ravi S, Raghunathan A, Jha NK (2005) Secure embedded processing through hardware-assisted runtime monitoring. In: Proceedings of the design, automation and test in Europe (DATE'05), vol 1
3. Ashcraft K, Engler D (2002) Using programmer-written compiler extensions to catch security holes. In: SP '02: proceedings of the 2002 IEEE symposium on security and privacy. IEEE Computer Society, Washington, p 143
4. Austin TM, Breach SE, Sohi GS (1994) Efficient detection of all pointer and array access errors. In: PLDI '94: proceedings of the ACM SIGPLAN 1994 conference on programming language design and implementation. ACM, New York, pp 290–301
5. Baratloo A, Singh N, Tsai T (2000) Transparent run-time defense against stack smashing attacks. In: Proceedings of 9th USENIX security symposium, June 2000
6. Barbosa M, Page D (2005) On the automatic construction of indistinguishable operations. In: Cryptography and coding. Lecture notes in computer science, vol 3796. Springer, Berlin, pp 233–247
7. Barrantes EG, Ackley DH, Palmer TS, Stefanovic D, Zovi DD (2003) Randomized instruction set emulation to disrupt binary code injection attacks. In: CCS '03: proceedings of the 10th ACM conference on computer and communications security. ACM, New York, pp 281–289
8. Benini L, Macii A, Macii E, Omerbegovic E, Pro F, Poncino M (2003) Energy-aware design techniques for differential power analysis protection. In: DAC '03: proceedings of the 40th conference on design automation. ACM, New York, pp 36–41
9. Benini L, Micheli GD, Macii E, Poncino M, Scarsi R (1999) Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers. ACM Trans Des Autom Electron Syst 4(4):351–375
10. Bhatkar S, DuVarney DC, Sekar R (2003) Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In: 12th USENIX security symposium, Washington, DC, August 2003
11. Biham E, Shamir A (2003) Power analysis of the key scheduling of the AES candidates. In: Second advanced encryption standard (AES) candidate conference, pp 343–347
12. Boneh D, DeMillo RA, Lipton RJ (1997) On the importance of checking cryptographic protocols for faults. In: Lecture notes in computer science, vol 1233. Springer, Berlin, pp 37–51
13. Boneh D, DeMillo RA, Lipton RJ (2001) On the importance of eliminating errors in cryptographic computations. J Cryptol 14(2):101–119
14. Bray B (2002) Compiler security checks in depth. Available at http://www.codeproject.com/tips/seccheck.asp, February 2002
15. Brumley D, Boneh D (2003) Remote timing attacks are practical. In: Proceedings of the 12th USENIX security symposium, August 2003
16. Bulba and Kil3r (2000) Bypassing stackguard and stackshield. Phrack Mag 10(56)
17. Bush WR, Pincus JD, Sielaff DJ (2000) A static analyzer for finding dynamic programming errors. Softw Pract Exp 30(7):775–802
18. CERT Coordination Center (2004) Vulnerability notes database. CERT Coordination Center
19. CERT Coordination Center (2005) CERT/CC vulnerabilities statistics 1988–2005. CERT Coordination Center
20. Chari S, Jutla C, Rao JR, Rohatgi P (1999) A cautionary note regarding evaluation of AES candidates on smart-cards. In: Second advanced encryption standard (AES) candidate conference, Rome, Italy. http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm
21. Chari S, Jutla CS, Rao JR, Rohatgi P (1999) Towards sound approaches to counteract power-analysis attacks. In: CRYPTO, pp 398–412
22. Chaumette S, Sauveron D, New security problems raised by open multiapplication smart cards
23. Chevalier-Mames B, Ciet M, Joye M (2004) Low-cost solutions for preventing simple sidechannel analysis: side-channel atomicity. IEEE Trans Comput 53(6):760–768

24. 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
25. Condit J, Harren M, McPeak S, Necula GC, Weimer W (2003) CCured in the real world. In: PLDI '03: proceedings of the ACM SIGPLAN 2003 conference on programming language design and implementation. ACM, New York, pp 232–244
26. Cousot P, Halbwachs N (1978) Automatic discovery of linear restraints among variables of a program. In: POPL '78: proceedings of the 5th ACM SIGACT-SIGPLAN symposium on principles of programming languages. ACM, New York, pp 84–96
27. Cowan C, Barringer M, Beattie S, Kroah-Hartman G, Frantzen M, Lokier J (2001) Formatguard: automatic protection from printf format string vulnerabilities. In: Proceedings of the 10th USENIX security symposium. USENIX Association, Berkeley, pp 191–200
28. Daemen J, Peeters M, Assche GV (2001) Bitslice ciphers and power analysis attacks. In: FSE '00: proceedings of the 7th international workshop on fast software encryption. Springer, London, pp 134–149
29. Daemen J, Rijmen V (1999) Resistance against implementation attacks: a comparative study of the AES proposals. URL: http://csrc.nist.gov/CryptoToolkit/aes/round1/pubcmnts.htm
30. Danil S, Julian M, Alexander B, Alex Y (2005) Design and analysis of dual-rail circuits for security applications. IEEE Trans Comput 54(4):449–460
31. Deeprasertkul P, Bhattarakosol P, O'Brien F (2005) Automatic detection and correction of programming faults for software applications. J Syst Softw 78(2):101–110
32. DeLine R, Fahndrich M (2001) Enforcing high-level protocols in low-level software. SIGPLAN Notes 36(5):59–69
33. Designer S (1997) Non-executable stack patch. Available at http://www.usenix.org/events/sec02/full_papers/lhee/lhee_html/node7.html
34. Dhem J-F, Feyt N (2001) Hardware and software symbiosis helps smart card evolution. IEEE Micro 21(6):14–25
35. Dhurjati D, Kowshik S, Adve V, Lattner C (2003) Memory safety without runtime checks or garbage collection. In: LCTES '03: proceedings of the 2003 ACM SIGPLAN conference on language, compiler, and tool for embedded systems. ACM, New York, pp 69–80
36. Erlingsson Ü, Schneider FB (2000) SASI enforcement of security policies: a retrospective. In: NSPW '99: proceedings of the 1999 workshop on new security paradigms. ACM, New York, pp 87–95
37. Evans D, Twyman A (1999) Flexible policy-directed code safety. In: IEEE symposium on security and privacy, pp 32–45
38. Fink G, Bishop M (1997) Property-based testing: a new approach to testing for assurance. SIGSOFT Softw Eng Notes 22(4):74–80
39. Fiskiran A, Lee R (2004) Evaluating instruction set extensions for fast arithmetic on binary finite fields. In: Proceedings of the 15th IEEE international conference on application-specific systems, architectures and processors, pp 125–136
40. Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A sense of self for Unix processes. In: SP '96: proceedings of the 1996 IEEE symposium on security and privacy. IEEE Computer Society, Washington, p 120
41. Frantzen M, Shuey M (2001) StackGhost: hardware facilitated stack protection. In: 10th USENIX security symposium, pp 55–66
42. Gebotys C (2006) A table masking countermeasure for low-energy secure embedded systems. IEEE Trans Very Large Scale Integr Syst 14(7):740–753
43. Gebotys CH, Gebotys RJ (2003) Secure elliptic curve implementations: an analysis of resistance to power-attacks in a DSP processor. In: CHES '02: revised papers from the 4th international workshop on cryptographic hardware and embedded systems. Springer, London, pp 114–128
44. Gebotys CH, White BA (2006) Methodology for attack on a Java-based PDA. In: CODES+ISSS '06. ACM, New York, pp 94–99
45. Ghosh AK, O'Connor T (1998) Analyzing programs for vulnerability to buffer overrun attacks. In: Proceedings of the 21st NIST-NCSC national information systems security conference, pp 274–382
46. Goubin L, Patarin J (1999) Des and differential power analysis (the "duplication" method). In: CHES '99: proceedings of the first international workshop on cryptographic hardware and embedded systems. Springer, London, pp 158–172
47. Großschädl J, Savas E (2004) Instruction set extensions for fast arithmetic in finite fields $gf(p)$ and $gf(2^m)$. In: CHES, pp 133–147
48. Hofmeyr SA, Forrest S, Somayaji A (1998) Intrusion detection using sequences of system calls. J Comput Secur 6(3):151–180
49. Irwin J, Page D, Smart NP (2002) Instruction stream mutation for non-deterministic processors. In: ASAP '02: proceedings of the IEEE international conference on application-specific systems, architectures, and processors. IEEE Computer Society, Washington, p 286

50. Joglekar SP, Tate SR (2004) Protomon: embedded monitors for cryptographic protocol intrusion detection and prevention. In: Proceedings on the international conference on information technology: coding and computing (ITCC'04), vol 1. IEEE Computer Society, Washington, p 81
51. Kc GS, Keromytis AD, Prevelakis V (2003) Countering code-injection attacks with instruction-set randomization. In: CCS '03: proceedings of the 10th ACM conference on computer and communications security. ACM, New York, pp 272–280
52. Kessels J, Kramer T, den Besten G, Peeters A, Timm V (2000) Applying asynchronous circuits in contactless smart cards. In: Advanced research in asynchronous circuits and systems (ASYNC 2000), pp 36–44
53. 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
54. Kocher P, Jaffe J, Jun B (1999) Differential power analysis. In: Lecture notes in computer science, vol 1666. Springer, Berlin, pp 388–397
55. Kocher P, Jaffe J, Jun B (1999) Using unpredictable information to minimize leakage from smartcards and other cryptosystems. US Patent 6327661
56. Koeune F, Standaert F-X (2006) A tutorial on physical security and side-channel attacks. In: Foundations of security analysis and design III: FOSAD 2004/2005, pp 78–108
57. Koopman P (2004) Embedded system security. Computer 37(7):95–97
58. Lampson BW (1971) Protection. ACM Oper Syst 8(1):18–24
59. Mangard S (2003) A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: Lee PJ, Lim CH (eds) Proceedings of the 5th international conference on information security and cryptology (ICISC 2002). Lecture notes in computer science, vol 2587. Springer, Berlin, pp 343–358
60. Mao S, Wolf T (2007) Hardware support for secure processing in embedded systems. In: Proceedings of 44th design automation conference (DAC), pp 483–488, San Diego, CA, June 2007
61. Messier M, Viega J (2005) Safe C string library. Available at http://www.zork.org/safestr/
62. May D, Muller HL, Smart NP (2001) Non-deterministic processors. In: ACISP '01: proceedings of the 6th Australasian conference on information security and privacy. Springer, London, pp 115–129
63. May D, Muller HL, Smart NP (2001) Random register renaming to foil dpa. In: CHES '01: proceedings of the third international workshop on cryptographic hardware and embedded systems. Springer, London, pp 28–38
64. Milenkovic M, Milenkovic A, Jovanov E (2005) Hardware support for code integrity in embedded processors. In: CASES '05: proceedings of the 2005 international conference on compilers, architectures and synthesis for embedded systems. ACM, New York, pp 55–65
65. Miller TC (1999) Strlcpy and strlcat—consistent, safe, string copy and concatenation. In: 1999 USENIX annual technical conference. USENIX Association, Monterey, pp 175–178
66. Muresan R, Gebotys CH (2004) Current flattening in software and hardware for security applications. In: CODES+ISSS, pp 218–223
67. Muresan R, Vahedi H, Zhanrong Y, Gregori S (2005) Power-smart system-on-chip architecture for embedded cryptosystems. In: CODES+ISSS '05: proceedings of the 3rd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis. ACM, New York, pp 184–189
68. Necula GC (1997) Proof-carrying code. In: Conference record of POPL '97: the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages, Paris, France, January 1997, pp 106–119
69. Nedjah N, de Macedo Mourelle 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
70. Wikipedia Foundation Inc. (2006) Pirate decryption definition. The free encyclopedia. http://en.wikipedia.org/wiki/Pirate_decryption
71. Quisquater J, Samyde D (2001) Electro magnetic analysis (EMA): measures and counter-measures for smart cards. In: E-smart, pp 200–210
72. Ragel RG, Parameswaran S (2006) IMPRES: integrated monitoring for processor reliability and security. In: Proceedings of the design and automation conference 2006 (DAC'06). ACM, San Fransisco, pp 502–505
73. Ragel RG, Parameswaran S, Kia SM (2005) Micro embedded monitoring for security in application specific instruction-set processors. In: Proceedings of the international conference on compilers, architectures, and synthesis for embedded systems (CASES'05). ACM, San Francisco
74. Rakers P, Connell L, Collins T, Russell D (2001) Secure contactless smartcard ASIC with DPA protection. IEEE J Solid-State Circuits, pp 559–565
75. Rao JR, Rohatgi P (2001) Empowering side-channel attacks. Cryptology ePrint Archive, Report 2001/037
76. Ravi S, Raghunathan A, Chakradhar S (2004) Tamper resistance mechanisms for secure, embedded systems. In: 17th international conference on VLSI design, January 2004

77. Ravi S, Raghunathan A, Kocher P, Hattangady S (2004) Security in embedded systems: design challenges. Trans Embed Comput Syst 3(3):461–491
78. Rostovtsev A, Shemyakina O (2005) AES side channel attack protection using random isomorphisms. Cryptology ePrint Archive, Report 2005/087
79. Sakai Y, Sakurai K (2006) Simple power analysis on fast modular reduction with generalized mersenne prime for elliptic curve cryptosystems. IEICE Trans Fundam Electron Commun Comput Sci E89-A(1):231–237
80. Saputra H, Vijaykrishnan N, Kandemir M, Irwin MJ, Brooks R, Kim S, Zhang W (2003) Masking the energy behavior of des encryption. 01:10084
81. Wikipedia Foundation Inc. (2006) SCA definition. The free encyclopedia. http://en.wikipedia.org/wiki/Static_code_analysis
82. Sekar R, Bendre M, Dhurjati D, Bollineni P (2001) A fast automaton-based method for detecting anomalous program behaviors. In: SP '01: proceedings of the 2001 IEEE symposium on security and privacy. IEEE Computer Society, Washington, p 144
83. Sprunk E (1999) Clock frequency modulation for secure microprocessors. US Patent WO 99/63696
84. Tillich S, Großschädl J (2006) Instruction set extensions for efficient AES implementation on 32-bit processors. In: CHES, pp 270–284
85. Tillich S, Großschädl J (2007) Power-analysis resistant AES implementation with instruction set extensions. In: Paillier P, Verbauwhede I (eds) Proceedings of the 9th international workshop on cryptographic hardware and embedded systems (CHES 2007), Wienna, Austria, September 10–13. Lecture notes in computer science, vol. 4727. Springer, Berlin, pp 303–319
86. Trichina E, Seta DD, Germani L (2003) Simplified Adaptive Multiplicative Masking for AES. In: CHES '02: revised papers from the 4th international workshop on cryptographic hardware and embedded systems. Springer, London, pp 187–197
87. Wayner P (1998) Code breaker cracks smart cards' digital safe. In: New York Times, p C1
88. Wolf W (2005) Multimedia applications of multiprocessor systems-on-chips. In: DATE '05: proceedings of the conference on design, automation and test in Europe. IEEE Computer Society, Washington, pp 86–89
89. YongBin Zhou DF (2005) Side-channel attacks: ten years after its publication and the impacts on cryptographic module security testing. Cryptology ePrint Archive, 2005/388