Global Journal of Computer Science and Technology
Volume 12 Issue 1 Version 1.0 January 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A Survey on Software Protection Techniques against Various Attacks

By N.Sasirekha, Dr.M.Hemalatha
University, Coimbatore, Tamilnadu, India

Abstract - Software security and protection plays an important role in software engineering. Considerable attempts have been made to enhance the security of the computer systems because of various available software piracy and virus attacks. Preventing attacks of software will have a huge influence on economic development. Thus, it is very vital to develop approaches that protect software from threats. There are various threats such as piracy, reverse engineering, tampering etc., exploits critical and poorly protected software. Thus, thorough threat analysis and new software protection schemes, needed to protect software from analysis and tampering attacks becomes very necessary. Various techniques are available in the literature for software protection from various attacks. This paper analyses the various techniques available in the literature for software protection. The functionalities and the characteristic features are various software protection techniques have been analyzed in this paper. The main goal of this paper is to analyze the existing software protection techniques and develop an efficient approach which would overcome the drawbacks of the existing techniques.

Keywords : Software Security, Software Tampering, Tampering Attacks, Encryption, Cryptography, Decryption.

GJCST Classification: K.6.5

A SURVEY ON SOFTWARE PROTECTION TECHNIQUES AGAINST VARIOUS ATTACKS

A Survey on Software Protection Techniques against Various Attacks

N.Sasirekha [α], Dr.M.Hemalatha [Ω]

Abstract - Software security and protection plays an important role in software engineering. Considerable attempts have been made to enhance the security of the computer systems because of various available software piracy and virus attacks. Preventing attacks of software will have a huge influence on economic development. Thus, it is very vital to develop approaches that protect software from threats. There are various threats such as piracy, reverse engineering, tampering etc., exploits critical and poorly protected software. Thus, thorough threat analysis and new software protection schemes, needed to protect software from analysis and tampering attacks becomes very necessary. Various techniques are available in the literature for software protection from various attacks. This paper analyses the various techniques available in the literature for software protection. The functionalities and the characteristic features are various software protection techniques have been analyzed in this paper. The main goal of this paper is to analyze the existing software protection techniques and develop an efficient approach which would overcome the drawbacks of the existing techniques.

Keywords : Software Security, Software Tampering, Tampering Attacks, Encryption, Cryptography, Decryption.

I. Introduction

Software protection has become one of the attractive domains with high commercial interest, from major software vendors to content providers which also comprises of the movie and music recording industries. The digital data of the software is especially at tremendous risk.

Confidentiality and data authenticity are two important concepts in security. Confidentiality provides data secrecy of a message and data authenticity protects the integrity of the message. Software protection falls between the domains of security, cryptography [30] and engineering among other disciplines.

The software protection technique mainly concentrates on protecting software from various attacks such as reverse engineering by obfuscation, modification by software tamper resistance, program-based attacks by software diversity, and BORE – break-once run everywhere – attacks by architectural design [2].

Protecting content needs protecting the software which processes the content. Copy protection is another form of software protection to the level that it needs several same protections against reverse engineering and software tampering.

Protecting code from attacks such as reverse engineering [32], analysis and tampering attacks is one of the main concerns for software providers. If a competitor succeeds in obtaining and reusing a algorithm, it would result in major issue. Moreover, secret keys, confidential data or security related code are not planned to be examined, extracted, stolen or corrupted. Even if legal actions such as patenting and cyber crime laws are in place, these techniques remain a significant threat to software developers and security expert.

This paper provides a survey on software protection and related areas which would encourage further research. This paper also provides a number of viewpoints, discuss challenges and suggest future directions.

II. Literature Survey

Piracy, reverse engineering and tampering have been the major software threats. Collberg et al. [1] provided a compact outline of the approaches to protect against these threats. Software watermarking for instance focuses on protecting software reactively against piracy. It usually implants hidden, distinctive data into an application in such a way that it can be guaranteed that a particular software instance belongs to a particular individual or company. When this data is distinctive for each example, one can mark out copied software to the source unless the watermark is smashed. The second group, code obfuscation, protects the software from reverse engineering attacks. This approach comprises of one or more program alterations that alter a program in such a way that its functionality remains identical but analyzing the internals of the program becomes very tough. A third group of approaches focuses to make software "tamper-proof", also called tamper-resistant.

Protecting the reliability of software platforms, particularly in unmanaged customer computing systems is a tough task. Attackers may try to carry out buffer

Author [α] : Doctoral Research Scholar, Karpagam University, Coimbatore, Tamilnadu, India,
E-mail : sasirekha.research@gmail.com, Telephone: number here
E-mail : here@here.com
Author [Ω] : Head, Department of Software Systems, Karpagam University, Coimbatore, Tamilnadu, India.
E-mail : hema.bioinf@gmail.com

overflow attacks to look for the right of entry to systems, steal secrets and patch on the available binaries to hide detection. Every binary has intrinsic weakness that attackers may make use of. In this paper Srinivasan et al., [3] proposed three orthogonal techniques; each of which offers a level of guarantee against malware attacks beyond virus detectors. The techniques can be incorporated on top of normal defenses and can be integrated for tailoring the level of desired protection. The author tries to identify alternating solutions to the issue of malware resistance. The techniques used are adding diversity or randomization to data address spaces, hiding significant data to avoid data theft and the utilization of distant evidence to detect tampering with executable code.

This paper focuses on the protection of a software program and the content that the program protects. There have been billions of dollars spent each year by the industries especially for software piracy and digital media piracy. The achievement of the content/software security in a huge segment is based on the ability of protecting software code against tampering and identifying the attackers who issue the pirate copies. In this paper, Hongxia Jin et al., [4] concentrates on the attacker identification and forensic examination. The author discussed about a proactive detection approach for defeating an on-going attack before the cooperation has occurred. The author also describes another detection approach for post-compromise attacker identification. Especially, the author takes into account the real world scenarios where the application programs connect with their vendors every so often, and where a discovery of attacking can bar a hacker user from further business.

Code obfuscation focuses to protect code against both static and dynamic study and there exists another approach to protect against code analysis, namely self-modifying code. This approach provides the opportunity to create code at runtime, rather than changing it statically. Practically, self-modifying code is highly restricted to the monarchy of viruses and malware. Yet, some publications regard self-modifying code as an approach to protect against static and dynamic analysis. Madou et al., [5] for instance regard dynamic code generation. The author proposed an approach where functions are generated earlier to their first call at runtime. Moreover, clustering is presented in such a way that a general template can be utilized to generate each function in a cluster, carrying out a least amount of alterations. In order to protect the constant `edits' against dynamic analysis, the authors suggested the usage of a Pseudo Random Number Generator (PRNG). The decryption at runtime technique is equal with code generation, apart from the fact that the decryption key can depend on other code, rather than on a PRNG. Moreover, it lessens re-encryption the viability of code during execution, while Madou et al. do not clearly protect a function template after the function executed.

Protecting code against tampering can be regarded as the issue of data authenticity, where 'data' refers to the program code. Aucsmith [6] explained an approach to implement tamper resistant software. The approach protects against analysis and tampering. The author utilizes small, armored code segments, also called Integrity Verification Kernels (IVKs), to validate code integrity. These IVKs are protected via encryption and digital signatures in such a way that it is tough to modify them. Morover, these IVKs can communicate with each other and across applications via an integrity verification protocol.

Chang et al. [7] proposed an approach depending on software guards. The protection technique of the author is chiefly based on a composite network of software guards which mutually validate each other's consistency and that of the program's critical sections. A software guard is a small segment of code carrying out particulars tasks, e.g. check summing or repairing. When check summing code discovers a modification, repair code is capable to undo this malevolent tamper challenge. The security of the approach depends partly on hiding the obfuscated guard code and the complexity of the guard network.

Horne et al. [8] described on the same idea of Chang et al. [7] and proposed `testers', small hashing functions that validate the program at runtime. These testers can be integrated with embedded software watermarks to result in a unique, watermarked, self-checking program. Other related research is unconscious hashing [9] which interweaves hashing instructions with program instructions and which is capable of proving whether a program is operated correctly. Recently, Ge et al. [10] presented a research work on control flow based obfuscation. Although the authors contributed to obfuscation, the control flow data is protected with an Aucsmith-like tamper resistance approach.

Buffer overflow utilization is a one of the notable threat to software security. In order to lessen the threat, Visual studio C/C++compiler facilitates to randomize the addresses of the compiled program in initialization time and to implant security stack guards by the compiled program in run time. Yongdong Wu [11] upgrades the compiler by raising the compiled program's abilities in the following features:

i. Protects a frame pointer from tampering without additional cost;
ii. Defeats the attack which tampers 1-2 bytes of a protected region at a very low cost;
iii. Checks the indirect function call against the prologue pattern so as to lessen the probability of software crash in case of being attacked.

The experiments demonstrated the enhancement on Microsoft Visual Studio in generating secure and robust software.

Cappaert et al., [12] presented a partial encryption approach depending on a code encryption approach [12], [13]. In order to utilize the partial encryption approach, binary codes are partitioned into small segments and encrypted. The encrypted binary codes are decrypted at runtime by users. Thus, the partial encryption overcomes the faults of illuminating all of the binary code at once as only the essential segments of the code are decrypted at runtime.

Jung et al., [14] presented a code block encryption approach to protect software using a key chain. Jung's approach uses a unit block, that is, a fixed-size block, rather than a basic block, which is a variable-size block. Basic blocks refer to the segments of codes that are partitioned by control transformation operations, such as "jump" and "branch" commands, in assembly code [12], [13]. Jung's approach is very similar to Cappaert's scheme. Jung's approach tries to solve the issue of Cappaert's approach. If a block is invoked by more than two preceding blocks, the invoked block is duplicated.

Unauthorized reverse-engineering of algorithms is a major issue for the software industry. Reverse-engineers look for security holes in the program to make use of competitors' vital approaches. In order to discourage reverse-engineering, developers use a wide range of static software protections to obfuscate their programs. Metamorphic software protections include another layer of protection to conventional static obfuscation approaches, forcing reverse-engineers to alter their attacks as the protection changes. Program fragmentation incorporates two obfuscation approaches, over viewing and obfuscated jump tables, into a novel, metamorphic protection. Segments of code are eliminated from the chief program flow and placed throughout memory, minimizing the locality of the program. These fragments move and are called using obfuscated jump tables which makes program execution hard. This research by Birrer et al., [15] evaluates the performance overhead of a program fragmentation engine and offers examination of its efficiency against reverse-engineering approaches. The experimental results show that program fragmentation has low overhead and is an effective approach to obscure disassembly of programs through two common disassembler/debugger tools.

Song-kyoo Kim [16] deals with the stochastic maintenance approach for the software protection through the closed queueing system with the untrustworthy backups. The technique shows the theoretical software protection approach in the security viewpoint. If software application modules are denoted as backups under proposed structural design, the system can be overcome through the stochastic maintenance model with chief untrustworthy and random auxiliary spare resources with replacement strategies. Additionally, the practical approach of technology improvement in software engineering through the technology innovation tool called TRIZ.

Zeng Min et al., [17] considered the supple manufacturing venture networks data security and software protection and proposed an enterprise classified data security and software protection solution, to describe the enterprise data storage, transmission and application software installation authorization, license and so on, presented a time and machine code depending on MD5, AES encryption algorithm dynamic secret key the encryption approach, to protect the enterprise data confidentiality, integrity and availability, to attain the software installation restrictions and using restrictions.

Kent [18] proposed a software protection technique which deals with the security needs of software vendors like protection from software copying and modification (e.g. physical attacks by users, or program-based attacks). Techniques proposed to handle these requirements include physical Tamper-Resistant Modules (TRMs) and cryptographic techniques. One approach comprises of using encrypted programs, with instructions decrypted immediately preceding to execution. Kent also observed the dual of this issue like user needs that externally-supplied software be confined in its access to local resources.

Gosler's software protection survey [19] investigates circa-1985 protection technologies which comprise of hardware security tools (e.g. dongles), floppy disc signatures (magnetic and physical), analysis denial approaches (e.g. anti-debug approaches, checksums, encrypted code) and slowing down interactive dynamic analysis. The main goal is on software copy prevention, but Gosler observed that the potency of resisting copying should be balanced by the potency of resisting software analysis (e.g. reverse engineering to study where to alter software and for protecting proprietary approaches) and that of software modification (to bypass security checks). Useful tampering is generally headed by reverse engineering.

Gosler also described that one should anticipate that an opponent can execute dynamic analysis on the target software without discovery (e.g. using in-circuit emulators and simulators) and that in such scenario, due to repeated experiments, one should anticipate the opponent to win. Thus, the main goal of practical resistance is to construct such experiments "enormously arduous". Another proposal [19] is cycling software (e.g. through some forced obsolescence) at a rate faster than an opponent can break it; this expects the model of forced software renewal (Jakobsson and Reiter [20]), who suggested hopeless pirates via forced updates and software aging). This technique is suitable where protection from attacks for a restricted time period suffices.

Herzberg and Pinter [21] focused on the issue of software copy protection and presented a solution needing CPU encryption support (which was far less possible when presented almost 20 years ago, circa 1984-85). Cohen's research [22] on software diversity and obfuscation is directly concentrated to software protection and offers a wide range of algorithms.

The subsequent practical tamper resistance system of Aucsmith [23] handled similar problems by an integration of just-in-time instruction decryption, and re-arranging instruction blocks at run-time to vigorously change the deals with the executing statements during program execution.

Several researchers have proposed techniques on software obfuscation using automated tools and code transformations [24, 25]. One idea would be to employ language-based tools to transform a program (most easily from source code) to a functionally equivalent program which presents greater reverse engineering barriers. If implemented in the form of a pre-compiler, the usual portability issues can be handled by the back-end of standard compilers.

Collberg et al. [26] provides more information regarding software obfuscation which includes descriptions about:

- Categorizing code transformations (e.g. control flow obfuscation, data obfuscation, layout obfuscation, preventive transformations)
- Identification of controlflow changes using opaque predicates (expressions not easy for an attacker to predict, but whose worth is recognized at compilation or obfuscation time)
- Preliminary suggestions on metrics for code transformations
- Program slicing tools
- The usage of (de)aggregation offlow control or data

Essential suggestions in software protection are done by Aucsmith [6], in combination with Graunke [23] at Intel. Aucsmith provides tamper prevention software which prevents inspection and change, and it is highly dependent to work accurately in unfriendly situations. Architecture is suggested according to an Integrity Verification Kernel (IVK) that checks the reliability of vital code segments. The IVK architecture is self-decrypting and includes self adjustment code.

Software tampering prevention using self-checking code was described by Horne et al. [27]. The integrity of segments of code is tested using some code known as testers. This can be a linear hash function and a predictable hash value. If the integrity condition is not satisfied, suitable actions will be carried out so as to make the integrity condition satisfied. The attackers can be confused and it is difficult for them to hack the testers if more number of testers is used.

Chang and Atallah [28] presented a technique with fairly extensive capacity containing a set of guards that can be programmed to perform arbitrary processes. An illustration for this is the check sum code segments for integrity checking which provides resistance against software tamper. An additional described guard function is repairing code (e.g. if a spoiled code segment is identified, downloading and installing a new version of the code section). The author also presents a technique for automatically keeping protections within code.

Chen et al. [29] put forth oblivious hashing that engages compile-time code alterations which outcomes in the calculation of a running trace of the execution history of a complete code. In this approach a trace are considered as increasing hash values of a subset of expressions that happens inside the usual program execution.

Gutmann [30] put forth an apparent conversation of the security concerns facing cryptographic usage in software under general-purpose operating systems, and analyzes the design difficulties in nullifying these concerns faced by using secure cryptographic co-processors.

Approaches	Functionalities
[1]	Outline of the approaches to protect against these threats. Software watermarking for instance focuses on protecting software reactively against piracy
[2]	Proposed three orthogonal techniques; each of which offers a level of guarantee against malware attacks beyond virus detectors.
[4]	Concentrates on the attacker identification and forensic examination. The author discussed about a proactive detection approach for defeating an on-going attack before the cooperation has occurred
[5]	an approach in which functions are generated earlier to their first call at runtime
[6]	The author utilizes small, armored code segments, also called Integrity
	Verification Kernels (IVKs), to validate code integrity
[7]	The protection technique of the author is chiefly based on a composite network of software guards which mutually validate each other's consistency and that of the program's critical sections.
[12]	Presented a partial encryption approach depending on a code encryption approach

[14]	Presented a code block encryption approach to protect software using a key chain
[16]	Deals with the stochastic maintenance approach for the software protection through the closed queueing system with the untrustworthy backups
[12]	Focused on the issue of software copy protection and presented a solution needing CPU encryption support
[27]	Software tampering prevention using self-checking code

III. Problems and Directions

The theoretical results to date on software obfuscation provide software protection of considerable practical value. The impracticality of constructing a program to find out whether other software is malicious does not preclude highly valuable computer virus detection technologies, and a feasible, anti-virus industry. It is still early in the history of research in the domains of software protection and obfuscation and that several discoveries and innovations lie ahead particularly in the domains of software diversity (which are utilized are less in the present scenario), and software tamper resistance. Increased number of secure techniques for software protection is very much needed which involves public scrutiny and peer evaluation. Cappaert proposed a tamper-resistant code encryption scheme, and Jung proposed a key-chain-based code encryption scheme. However, Cappaert's scheme did not meet the security requirements for code encryption schemes, and Jung's scheme had an efficiency problem. Moreover, time cost and space cost should also be taken into consideration. To improve efficiency, support from the compiler and operating system is needed [19].

More open discussion of particular approaches is very much needed. Cryptography is observed to be the technique that can be incorporated in the software protection technique for improved protection. Past trends of proprietary, undislosed techniques of software obfuscation approaches similar to the early days in cryptography have to be altered.

For decades encryption has provided the means to hide information. In this research, the self-encrypting code is used as a means of software protection. In this research work, the concept of efficient code encryption techniques, which offers confidentiality and a method to create code dependencies that implicitly protect integrity need to be established. Moreover, several dependency schemes based on a static call graph which allow runtime code decryption simultaneous with code verification can also be used. If code is modified statically or dynamically, it will result in incorrect decryption of other code, producing a corrupted executable. Better and efficient cryptographic techniques can be integrated for better results. This research uses the encryption technique to secure software static analysis and tampering attacks.

IV. Conclusion

This paper presented and discussed a survey on the protection of software because of various attacks. Several software protection techniques available in the literature are analyzed and examined. The characteristic features of the existing algorithms are thoroughly investigated in this paper. This study would facilitate in development of efficient software protection techniques. Encryption techniques can be incorporated with the existing software protection techniques to improve the overall security of the software. Code encryption schemes for protecting software against various attacks like reverse engineering and modification. Therefore, novel and efficient code encryption scheme have to be established based on an indexed table to guarantee secure key management and efficiency.

References Références Referencias

1. Collberg, C.S.; Thomborson, C.; "Watermarking, tamper-proofing, and obfuscation - tools for software protection", IEEE Transactions on Software Engineering, Volume: 28 , Issue: 8, Page(s): 735 – 746, 2002.
2. T. Ogiso, U. Sakabe, M. Soshi, A. Miyaji, "Software Tamper Resistance Based on the Difficulty of Interprocedural Analysis", 3rd Workshop on Information Security Applications (WISA 2002), Korea, August 2002.
3. Srinivasan, R.; Dasgupta, P.; Iyer, V.; Kanitkar, A.; Sanjeev, S.; Lodhia, J.; "A Multi-factor Approach to Securing Software on Client Computing Platforms", 2010 IEEE Second International Conference on Social Computing (SocialCom), Page(s): 993 – 998, 2010.
4. Hongxia Jin; Lotspiech, J.; "Forensic analysis for tamper resistant software", 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.
5. M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, and K. De Bosschere. Software protection through dynamic code mutation
6. D. Aucsmith. Tamper resistant software: an implementation. Information Hiding, Lecture Notes in Computer Science, 1174:317-333, 1996.
7. H. Chang and M. J. Atallah. Protecting software codes by guards. ACM Workshop on Digital Rights Managment (DRM 2001), LNCS 2320:160-175, 2001.
8. B. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic Self-Checking Techniques for Improved Tamper Resistance. In Proceedings of Workshop on Security and Privacy in Digital Rights

Management 2001, pages 141-159, 2001.
9. Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. Jakubowski. Oblivious hashing: a stealthy software integrity veri_cation primitive. In Information Hiding, 2002.
10. J. Ge, S. Chaudhuri, and A. Tyagi. Control flow based obfuscation. In DRM '05: Proceedings of the 5th ACM workshop on Digital rights management, pages 83{92, 2005.
11. Yongdong Wu; "Enhancing Security Check in Visual Studio C/C++ Compiler", Software Engineering, 2009. WRI World Congress on WCSE '09. Volume: 4 , Page(s): 109 – 113, 2009.
12. J. Cappaert et al., "Toward Tamper Resistant Code Encryption: Practice and Experience," LNCS, vol. 4991, 2008, pp. 86-100.
13. J. Cappaert et al., "Self-Encrypting Code to Protect Against Analysis and Tampering," 1st Benelux Workshop Inf. Syst. Security, 2006.
14. D.W Jung, H.S Kim, and J.G. Park, "A Code Block Cipher Method to Protect Application Programs From Reverse Engineering,"J. Korea Inst. Inf. Security Cryptology, vol. 18, no. 2, 2008, pp. 85-96 (in Korean)
15. Birrer, B.D.; Raines, R.A.; Baldwin, R.O.; Mullins, B.E.; Bennington, R.W. Program Fragmentation as a Metamorphic Software Protection, Third International Symposium on Information Assurance and Security, 2007 , Page(s): 369 – 374, 2007. IAS 2007.
16. Song-kyoo Kim; "Design of enhanced software protection architecture by using theory of inventive problem solving", IEEE International Conference on Industrial Engineering and Engineering Management, 2009. IEEM 2009.
17. Zeng Min; Liu Qiong-mei; Wang Cheng; Practices of agile manufacturing enterprise data security and software protection, 2010 2nd International Conference on Industrial Mechatronics and Automation (ICIMA).
18. S. Kent, Protecting Externally Supplied Software in Small Computers, Ph.D. thesis, M.I.T., September 1980.
19. J. Gosler, "Software Protection: Myth or Reality?", Advances in Cryptology – CRYPTO'85, Springer-Verlag LNCS 218, pp.140–157 (1985)
20. M. Jakobsson, M.K. Reiter, "Discouraging Software Piracy Using Software Aging", Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001), Springer LNCS 2320, pp.1–12 (2002).
21. A. Herzberg, S.S. Pinter, "Public Protection of Software", pp.371–393, ACM Trans. Computer Systems, vol.5 no.4 (Nov. 1987). Earlier version in Crypto'85.
22. F. Cohen, "Operating System Protection Through Program Evolution", Computers and Security 12(6), 1 Oct. 1993, pp. 565–584.
23. D. Aucsmith, G. Graunke, Tamper Resistant Methods and Apparatus, U.S. Patent 5,892,899 (filed June 13 1996; issued Apr.6 1999).
24. C. Collberg, C. Thomborson, D. Low, "Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs", Proc. Symp. Principles of Programming Languages (POPL'98), Jan. 1998.
25. C. Collberg, C. Thomborson, D. Low, "Breaking Abstractions and Unstructuring Data Structures", IEEE International Conf. Computer Languages (ICCL'98), May 1998.
26. C. Collberg, C. Thomborson, D. Low, "A Taxonomy of Obfuscating Transformations", Technical Report 148, Dept. Computer Science, University of Auckland (July 1997).
27. B. Horne, L. Matheson, C. Sheehan, R. Tarjan, "Dynamic Self-Checking Techniques for Improved Tamper Resistance", Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001), Springer LNCS 2320, pp.141–159 (2002).
28. H. Chang, M. Atallah, "Protecting Software Code by Guards", Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001), Springer LNCS 2320, pp.160–175 (2002).
29. Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, M. Jakubowski, "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive", Proc. 5th Information Hiding Workshop (IHW), Netherlands (October 2002), Springer LNCS 2578, pp.400–414.
30. P. Gutmann, "An Open-source Cryptographic Co-processor", Proc. 2000 USENIX Security Symposium.
31. E. Eilam, Reversing: Secrets of Reverse Engineering, Wiley Publishing, Inc., 2005.