International Journal of Engineering Education Vol. 28, No. 6, pp. 1393–1398, 2012
Printed in Great Britain

0949-149X/91 $3.00+0.00

A Non-Deterministic Self-Checking Mechanism to Enhance Tamper-Resistance in Engineering Education Software*

YULIER NÚÑEZ MUSA
Superior Polytechnic Institute 'José Antonio Echeverría' – Informatics Engineering Faculty FII ISPJAE, Calle 114 No. 11901. e/ Ciclovía y Rotonda, Marianao, La Habana 19390, Cuba. E-mail: jnunezm@ceis.cujae.edu.cu

ROBERTO SEPÚLVEDA LIMA
Superior Polytechnic Institute 'José Antonio Echeverría'—Informatics Engineering Faculty FII ISPJAE, Calle 114 No. 11901. e/ Ciclovía y Rotonda, Marianao, La Habana 19390, Cuba. E-mail: sepul@ceis.cujae.edu.cu

SERGIO CUENCA ASENSI
Computer Technology Department, University of Alicante, Carretera San Vicente del Raspeig S/N, 03690 San Vicente del Raspeig, Alicante, Spain. E-mail: sergio@dtic.ua.es

ITZAMÁ LÓPEZ YÁÑEZ
National Polytechnics Institute – Interdisciplinary Professional Unit on Engineering and Advanced Technologies IPN UPIITA, Av. Instituto Politécnico Nacional No. 2580, Mexico City 07340, Mexico. E-mail: ilopezy@ipn.mx

LUIS OCTAVIO LÓPEZ LEYVA
National Polytechnics Institute—Superior School of Computing IPN ESCOM, Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal, ESCOM Building, Mexico City 07738, Mexico. E-mail:octavioll@gmail.com

Obfuscation and code encryption applied to ensure confidentiality achieve some degree of tamper resistance due to the complexity of the analysis required to break these protection schemes. However, there are very few proposals that combine high integrity and confidentiality levels at the same time. In the context of engineering education software, this article presents a new mechanism for increasing the software tamper resistance applications based on non-deterministic integrity self-checking network.

Keywords: software piracy; engineering education software; software tamper-resistance; integrity; confidentiality; integrity self-checking network.

1. Introduction

There are several situations in which becomes desirable to protect a software component form malicious attacks by inspection or modification, once that component has been distributed to third parties. An example of such instances is software piracy. It is common for software attacker to make dynamic tests on applications to inspect and modify the code by applying reverse engineering techniques and tools [1]. Generally, software piracy is formed by illegal distribution of applications, theft of intellectual property or privilege system escalation. The attacker modifies the application and executes it in order to demonstrate if he has made a successful attack. If the confidentiality application attribute is ensured, the attacker has to resign himself to make a black-box security application analysis watching the pair sets of inputs and outputs inferring the application behavior. Moreover, the protection of the application integrity [2] is intended to detect or prevent attacks which alter the application behavior. It becomes necessary to use protection mechanisms that ensure the confidentiality and the application integrity under an untrusted host threat model [3].

In this sense, several techniques can be used in order to guarantee the privacy of the application, such as: obfuscation [1, 15, 16], cipher [9, 17–19], and self-modification of code [20–22]. On the other hand, the program integrity may be secured by using integrity self-verification techniques [2, 7, 23] or even fragile watermarks [24].

This proposal successfully combines an integrity self-checking mechanism and obfuscation in the same protection mechanism using non-deterministic behavior. The main contribution is to address the integrity self-checking mechanism, from a practical security point of view, by using non-deterministic detections and responses to integrity violation, rather than a theoretical security model which is difficult to define in this kind of context [4]. Note that the protection mechanism is designed to be used primarily in the area of software protection, against software piracy.

* Accepted 2 August 2012.

2. Previous work

To ensure the application confidentiality, several techniques can be used such as obfuscation [5] or encryption [6]. On the other hand, the integrity is usually assured by integrity self-checking mechanisms (ISCM) [2], [7]. These self-checking mechanisms are focused on augmenting the strength of the program before attacks by modification [14]. In general terms, these techniques are made up by two components: a tamper detection component, and a tamper response component [25]. These mechanisms are defined as self-checking because the application is in charge of protecting itself, being the control components mentioned before a part of said application.

Usually, different protection mechanisms utilize techniques that provide confidentiality to ensure integrity collaterally because the attacker may not understand the application behavior. In these cases, they cannot satisfy their intentions by modifying the target application. Otherwise, designing an integrity control mechanism which ensures confidentiality in an explicit way may be more difficult to achieve. Note that the integrity algorithms are not designed to prevent an attack on the application confidentiality. Different authors [2], [7] propose combination of techniques that ensures integrity with others that provide confidentiality, either obfuscation [8] or code encryption [9]. Due to some negative results in the area of obfuscation [4] and code encryption [10], it is necessary to identify an integrity control mechanism that takes into account the obfuscation, not so much from the structural dimension, but rather from a functional point of view [11].

There is a clear tendency to design and implement integrity self-checking systems that include some random elements in their operation, exhibiting a non-deterministic behavior. Under an untrusted host threat model, the primordial aspect is to detect the attack, yet both detection and response may be non-deterministic without stopping the protection mechanism from being effective.

Also, some concepts from the areas of trusted and fault tolerant systems are being included in order to evaluate the security of the protection mechanisms. More specifically, the mechanisms of interest are those that offer resistance to modification, given the analogy between the concept of failure and attack, both of which affect integrity in the end.

3. Proposed method

We propose a self-checking mechanism that is characterized by the following security attributes:

- Distributed security. Instead of having a single point of integrity checking, a distributed security scheme through various integrity control nodes scattered throughout the application is defined.
- Non-deterministic redundant verification. Each node verifies the integrity of a node subset, therefore, redundant verification exists. The revisions performed by the nodes are non-deterministic; they may detect a change in each application execution with some probability.
- Diverse and equiprobable response. This attribute is reached by introducing different response mechanisms, for example, progressive data corruption, application performance degradation, etc. After the detection of attacks, the application responses are selected in a random and equiprobable way, ensuring a low relationship between attacks and responses.
- Non-deterministic response time. The response time to an attack is non-deterministic. Thus, the response may be issued several executions after modification detection.

Suppose the following scenario. An attacker tries to compromise the integrity control mechanism application in order to illegally distribute and allow the application execution by unauthorized users. The integrity control mechanism application under attack is designed by taking into account the security attributes described above. The attack consists in executing the application to locate and deactivate each integrity control node. The attacker will have succeeded, if the integrity control mechanism never gives a response to the change. Otherwise, the attacker will have to repeat the process again and again. Note, one of the ways that the attacker has to locate a node is based on the response after an attack, and by finding the part of the application which generated the response. Thus, the attacker learns the integrity control mechanism by monitoring the attack-response relationship. If the responses to the same attack are different, equiprobables, and the response time is non-deterministic in every execution, then the amount of information in terms of entropy [12] that provides the integrity control mechanism for the attacker will be less than that given by traditional protection mechanisms. As there is more chaos or confusion in an attack-response relationship, the entropy, given by the mechanism, increases, so the attacker will need to perform other application executions to get the necessary amount of information in order to understand the integrity control mechanism behavior.

After each node modification, the integrity control mechanism cannot give a response for two reasons: (1) the attacker successfully defused each node or, (2) the protective mechanism did not perform the verification during the current execution. Although, there are probabilities that the verification be carried out, in subsequent performances. For this reason, the attacker has to perform a high number of attempts to obtain an expected response to his attack.

Based on the assumptions outlined above, both time location and time node deactivation could increase. Thus, it is necessary to establish, in an objective way, the following observations:

- From the attacker point of view. The fact that the revisions and responses to attacks are non-deterministic provokes attacker to consume a great effort to be certain that the protection mechanism has been overcome. Moreover, the attacker can accept that the target application works according his interests and the application execution fails with a low probability (e.g. one in a hundred executions on average), is considered a successfully attack under these conditions.
- From the illegitimate user point of view. Protection through a non-deterministic integrity control mechanism generates distrust in potential illegitimate users, lacking the certainty that the results generated by the application are always correct. The effort and cost required to correct the errors caused by protection may outweigh the cost for obtaining a legal application. The time between failures may be accepted as valid for an attacker, but not necessarily for an illegitimate user. The uncertainty of the proper application functioning, together with the cost of correcting flaws caused by the protection mechanism, can force the illegitimate user to acquire a legal application.

The main goal of the proposed mechanism is to ensure that both, detection and response to an attack are non-deterministic, with the aim of creating an uncertainty as to whether or not the attacker was able to disable the protection mechanism. Moreover, equiprobable responses to attacks increase the entropy of the protection mechanism to prolong the reverse engineering process applied by the attacker, achieving some degree of obfuscation.

We denote the application P to be protected by its Control Flow Graph, $G_{CF} = (V,A)$ where G is a directed graph in which $V(G_{CF})$ is the vertex set of the graph and $A(G_{CF})$ is the arcs set that indicates the transition between two vertices during the execution of P. The integrity control mechanism, structurally speaking, consists of two functional components: the tamper detection component and the tamper response component:

- Tamper detection: Is composed by a set of verifiers $C = \{c_1, \ldots, c_n\}$ those are responsible for detecting unauthorized modifications on the objects set $O = \{o_1, \ldots, o_n\}$ belonging to the program P. Each verifier c_i has a verification subset $O' \subseteq O$ and verifies the integrity of each $o_k \in O'_i$ with a given probability at each execution of P.
- Tamper response. This component provides a response r_j, selected randomly among the l different responses of the corresponding set R = $\{r_1, \ldots, r_l\}$, against a change detected by a verifier c_i on some object o_k. The response component may choose many diverse responses or no response when an attack is detected repeatedly in different application executions. For this reason, the number of responses r_j is much larger than the number of verifiers c_i.

Since both the verification and the responses may be non-deterministic, the same c_i during different executions can take one of the following three states:

- Non-verification/non-response: the verifier c_i cannot verify the integrity of some o_k that has been deliberately modified by the attacker, so no response is issued. At this stage, the attacker does not receive any response, so he considers that a successful attack has been obtained during the execution in progress.
- Verification/non-response: the verifier c_i detects the change made on some o_k, but no response is issued. At this stage, the attacker receives no response, so he could consider a successful attack has been obtained.
- Verification/response: the verifier c_i detects the change made on some o_k and issues a response r_j randomly selected. At this stage, the attacker receives a response to this attack.

In order to insert the integrity self-checking network into program P, several transformations must be done to the program, which is described below.

- Verification graph insertion into program P.
  1. Add the verification graph vertices to the vertex set of the Control Flow Graph of P. The new vertex set is $V(G_{CF}) = V(G_{CF}) \cup V(G_V)$.
  2. Reorganize the arcs $A(G_{CF})$ such that the verifiers $C \subseteq V(G_V)$ are uniformly distributed throughout the G_{CF} of P. For this, an arc subset $A_c \subseteq A(G_{CF})$ is selected randomly such that $|A_c| = |C|$. Each arc $(a_i, a_j) \in A_c$ is substituted by two new arcs (a_i, c_k) and (c_k, a_j) for each of the $c_k \in C|C \subseteq V(G_V)$ selected randomly.
  3. Configure randomly the Verification Graph G_V in such way that the input degree $d_{\bar{G}_V}(o_i)$ of each $o_i \in V(G_V)$ is approximately proportional to the rest. Even though the Verification Graph has a 'fixed' configuration when it is inserted into program P, a different part of it

will be activated at each execution, given the non-deterministic verifications.

- Response graph insertion into program P.
  1. The response vertices $R(G_R)$ are inserted (like the graph G_v is inserted into G_{CF}), such that $V(G_{CF}) = V(G_{CF}) \cup R(G_R)$.
  2. Reorganize the arcs $A(G_{CF})$ such that the responses $R(G_R)$ are uniformly distributed throughout the G_{CF} of P. The procedure is the same as the one used for the verifiers, only that a new arcs subset $A_R \subseteq A(G_{CF})$ is selected, keeping the responses from being to spacially close to the verifiers.
  3. Configure the Response Graph G_R such that it is a complete bipartite graph. This ensures that each $c \in C(G_R)$ may activate any $c \in C(G_R)$ during the execution of P.

In this way, program P becomes a protected program $P_{protect}$ which is able to verify its integrity in a non-deterministic way.

4. The attacker's activities

The attacker will try to overcome the mechanism by removing the self-checking nodes. During the process of locating nodes, the attacker is facing two main problems:

1. Since both the verification and the responses are non-deterministic, the attacker is forced to make a high number of attempts until a response to their attacks is issued. It is necessary to take into account that the attacker initially does not know the minimum number of executions to be carried out in order to observe a response, which increases the uncertainty of his attacks.
2. The attacker receives little information from the protection mechanism after a modification, since the mechanism emits equiprobable responses. This forces the attacker to perform a thorough analysis of all possible answers after each modification, because he doesn't know which response may be issued.

The attacker must confirm that target application has an expected behavior after modifications that remove each of the verifiers and protection mechanism that does not issue any response.

To validate their modifications, the attacker must execute the application repeatedly with different input data to analyze the application responses. If the target application does not emit any response after a certain number of executions, the attacker may infer that: (1) he modified a non-protected application zone, (2) the data set applied has not activated the execution path that contains the verifier that checks the modified zone or (3) he doesn't apply a sufficient number of executions to obtain a response.

To guarantee an exhaustive running of all network nodes, the attacker must execute the target application by different data sets or test cases. That is, he needs to define sufficient data sets that assure high code coverage.

If a verifier c_i is inserted into an application zone that doesn't activate the data set that the attacker has applied, the application would not emit a response. In this case, the attacker might conclude that the protection mechanism was eliminated completely.

The attacker may decide to distribute the target application to a third party. Illegal users may execute the target application with different data sets that could activate non-removed verifiers.

The attacker must confirm that the target application has an expected behavior after modifications, that remove each of the verifiers and protection mechanism that doesn't issue any response.

By previous situations, the attacker must identify the data sets by ensuring 100 percent code coverage, but this could be a complex task. On the other hand, the attacker's intentions would be to obtain highest code coverage with minimum data sets, because they must execute the target application for each data set. To define a minimum amount of data, to guarantee code coverage a 100 percent, is an NP-problem due to the fact that this is a particular cover case problem [13]. The attacker may apply heuristics techniques to avoid the explained complexity, which supposes an additional effort.

5. Experimental results

This proposal is situated from a practical security point of view [14] related with experiments and validations. A theoretical approach that ensures 100 percent attack detection is not considered. It may be an impossible scenario due to the untrusted host threat model that has been considered [3].

The experiments simulate the attacker activities, which modify the target application and collect the attacks-responses pairs to decide if they have performed a successful attack.

The proposed mechanism security analysis is based on estimating the effort required by an attacker to locate, modify and test the result of their attack on each of the nodes in the network. To validate the proposal, we applied a set of preliminary experiments. We used as a target application, the gzip application, included into Benchmark Spec CPU2000 (http://www.spec.org/cpu2000/).

The process to inserting a self-checking network inside the target application was done by automating the compilation process using Microsoft Phoenix Framework (https://connect.microsoft.com/Phoenix) as a main tool. The target application was modified by the insertion of one thousand verifiers.

The first experiment consists in simulating the attacker behavior. It consists in generating the data sets (test cases) to reach bigger code coverage.

For the target application (gzip), we generated 58 test cases and a 85.1 percent of code coverage was obtained. No technique to minimize the data sets was applied in this case.

In general terms, for any test case, 24 percent of verifying nodes were not executed. This implies that the attacker will not detect the presence of these nodes, because he will not obtain any related response.

Another aspect to take into consideration is the impact on program performance produced by the protection mechanism. From 58 test cases, 49 did not suffer any impact on program performance. The worst test case consumed 2.8 times longer for executing than the original application, without the self-checking mechanism.

6. Conclusions and future work

A mechanism for software protection is proposed by means of a non-deterministic self-checking network. This proposal combines the integrity controls and confidentiality in a unique protection mechanism. A non-deterministic behavior, forces the attacker to perform a high number of application executions to be sure that the protection mechanism has been overcome. This represents a considerable effort by the attacker and delays the reverse engineering process considerably.

On the other hand, the authors are currently working on identifying or developing (where necessary) metrics which allow the quantification of both integrity and privacy reached by the proposed model under a single model. For this, some concepts will be taken from areas like trusted computing and fault tolerant systems.

Acknowledgements—The authors would like to thank the ICyTDF (grants PIUTE10-77 and PICSO10-85), the ISPJAE, the CITI, the University of Alicante, the Instituto Politécnico Nacional (Secretaría Académica, COFAA, SIP, CIC, UPIITA, and ESCOM), the CONACyT, and SNI for their economical support to develop this work.

References

1. C. S. Collberg and C. Thomborson, Watermarking, Tamper-Proofing, and Obfuscation—Tools for Software Protection, *IEEE Transactions on Software Engineering*, **28**(8), 2002, pp. 735–746.
2. B. Horne, L. Matheson, C. Sheehan and R. Tartan, Dynamic self-checking techniques for improved tamper resistance, *Lecture Notes in Computer Science*, **2320**, 2002, pp. 141–159.
3. A. Main and P. C. van Oorschot, Software protection and application security: *Understanding the battleground, International Course on State of the Art and Evolution of Computer Security and Industrial Cryptography*, Heverlee, Belgium, June, 2003, pp. 1–19.
4. S. Goldwasser and Y. T. Kalai, On the impossibility of obfuscation with auxiliary input, *46th Annual IEEE Symp. Foundations of Computer Science*, Pittsburgh, PA, USA, 23–25 Oct., 2005, pp. 553–562.
5. S. Goldwasser and G. N. Rothblum, On best-possible obfuscation, *Lecture Notes in Computer Science*, **4392**, 2007, pp. 194–213.
6. W. Michiels, Opportunities in white-box cryptography, *IEEE Security & Privacy*, **8**(1), 2010, pp. 64–67.
7. M. Jacob, M. H. Jakubowski and R. Venkatesan, Towards integral binary execution: implementing oblivious hashing using overlapped instruction encodings, *9th Workshop on Multimedia & security*, Dallas, Texas, USA, September 20–21, 2007, pp. 129–140.
8. H. Jin, G. Myles and J. Lotspiech, Towards better software tamper resistance, *Lecture Notes in Computer Science*, **3650**, 2005, pp. 417–430.
9. J. Cappaert, B. Preneel, B. Anckaert, M. Madou and K. De Bosschere, Towards tamper resistant code encryption: Practice and experience, *Lecture Notes in Computer Science*, **4991**, 2008, pp. 86–100.
10. L. Goubin, J.-M. Masereel and M. Quisquater, Cryptanalysis of white box DES implementations, *Lecture Notes in Computer Science*, **4876**, 2007, pp. 278–295.
11. N. Dedic, M. Jakubowski, and R. Venkatesan, A graph game model for software tamper protection, *Lecture Notes in Computer Science*, **4567**, 2007, pp. 80–95.
12. M. Alvim, M. Andrés and C. Palamidessi, Entropy and attack models in information flow, *Theoretical Computer Science*, **323**, 2010, pp. 53–54.
13. S. Yoo and M. Harman, Using hybrid algorithm for pareto efficient multiobjective test suite minimisation, *The Journal of Systems and Software*, **83**, 2010, pp. 689–701.
14. V. D. Gligor, Architectures for practical security, *ACM symposium on access control models and technologies SACMAT*, Pittsburgh, USA, June 9–11, 2010, pp. 161–162.
15. C. Linn and S. Debray, Obfuscation of executable code to improve resistance to static disassembly, *10th ACM conference on Computer and communications security*, Washington D.C., USA, October 27–30, 2003, pp. 290–299.
16. N. Kuzurin, A. Shokurov, N. Varnovsky and V. Zakharov, On the concept of software obfuscation in computer security, *10th Information Security Conference*, Valparaiso, Chile, October 9–12, 2007, pp. 281–298.
17. S. Chow, P. A. Eisen, H. Johnson and P. C. van Oorschot, White-Box Cryptography and an AES Implementation, *Lecture Notes in Computer Science*, **2595**, 2003, pp. 250–270.
18. S. Chow, P. Eisen, H. Johnson and P. van Oorschot, A white-box DES implementation for DRM applications, *Lecture Notes in Computer Science*, **2696**, 2003, pp. 1–15.
19. W. Michiels and P. Gorissen, Mechanism for software tamper resistance: an application of white-box cryptography, *ACM workshop on Digital Rights Management*, Alexandria, Virginia, USA, October 29, 2007, pp. 82–89.
20. B. Anckaert, M. Madou and K. D. Bosschere, A Model for Self-Modifying Code, *8th International Workshop on Information Hiding*, Alexandria, VA, USA, July 10–12, 2006, pp. 232–248.
21. H. Cai, Z. Shao and A. Vaynberg, Certified self-modifying code, *ACM SIGPLAN conference on Programming language design and implementation*, San Diego, California, USA, June 11–13, 2007, pp. 66–77.
22. Y. Kanzaki, A. Monden, M. Nakamura and K. Matsumoto, Exploiting Self-Modification Mechanism for Program Protection, *27th Annual International Conference on Computer Software and Applications*, Dallas, Texas, USA, November 3–6, 2003, pp. 170–179.
23. H. Chang and M. J. Atallah, Protecting Software Code by Guards, *Workshop on Security and Privacy in Digital Rights Management*, Philadelphia, USA, November 5, 2001, pp. 160–175.

24. W. Zhu, C. Thomborson and F.-Y. Wang, A Survey of Software Watermarking, *Lecture Notes in Computer Science*, **3495**, 2005, pp. 283–331.

25. G. Tan, Y. Chen and M. H. Jakubowski, Delayed and controlled failures in tamper-resistant software, *Lecture Notes in Computer Science*, **4437**, 2007, pp. 216–231.

Yulier Núñez Musa obtained his Bachelor degree and MSc degree as Informatics Engineer at the Superior Polytechnic Institute 'José Antonio Echeverría' (ISPJAE). He is currently a Professor at the Informatics Engineering Faculty in the ISPJAE, and studying his Doctorate in Knowledge Society Technologies at the University of Alicante. Areas of interest: Cryptographic Applications and Public Key Infrastructure, Application and Intellectual Property Protection, and Information Systems Audit and Evaluation.

Roberto Sepúlveda Lima received his PhD degree (1998) on Technical Sciences at Superior Polytechnic Institute 'José Antonio Echeverría' (ISPJAE). He is currently the President of the National Commission for the Informatics Engineering Career and Member of the National Group for the Development of Cryptography, in Cuba. Also, he is a Titular Professor at the Informatics Engineering Faculty in the ISPJAE, of which he is a former Dean. Areas of interest: Cryptography and Information Security, Artificial Intelligence, Software Engineering, and Computer Networks.

Sergio Cuenca Asensi is an associate professor in the Computer and Technology Department at University of Alicante, Spain. He received the B.S. degree in electronic physics in 1990 from University of Granada, Spain. He received PhD in Computer Engineering from the University Miguel Hernández of Elche, Spain in 2002. His current research interests are reconfigurable computing, hardware/software codesign and soft error mitigation in embedded systems

Itzamá López Yáñez received his Bachelor degree as Information Systems Engineer (2003) at Monterrey Institute of Technology and Superior Studies (ITESM), while the MSc (2007) and PhD (2011) degrees on Computer Sciences at National Polytechnics Institute (IPN) Center for Computing Research (CIC), both with mention of Honor. He was granted the Lázaro Cárdenas 2012 Award by the President of the Republic. Currently he is both a Professor at, and the President of the Telematics Academy at IPN Interdisciplinary Professional Unit on Engineering and Advanced Technologies (UPIITA). Areas of interest: Associative Memories, Neural Networks, Software Engineering, and Pattern Classification, in particular the Gamma classifier.

Luis Octavio López Leyva obtained his Bachelor degree as Communications and Electronics Engineer (1983) at National Polytechnics Institute (IPN) Superior School of Mechanical and Electrical Engineering (ESIME), while the MSc degree (2003) on Computer Engineering and the PhD degree (2008) on Computer Sciences, both at IPN Center for Computing Research (CIC). He currently is a Researcher Professor, Titular C, at IPN Superior School of Computing (ESCOM). Areas of interest: Associative Memories, Neural Networks, Support Vector Machines, and Software Engineering.