Інформаційні технології

UDC 378.112: 004.9 **doi: 10.26906/SUNZ.2019.2.055**

Anoushirvan Rashidinia[1], S. Gavrilenko[1], M. Pochebut[2], O. Sytnikova[2]

[1] National Technical University "KPI", Kharkiv, Ukraine
[2] Kharkiv National University of Radioelektronics, Kharkiv, Ukraine

SOFTWARE SECURITY OVERVIEW

The article analyzes the main threats and problems of software protection. Methods for protecting information, their advantages and disadvantages are considered, and the possibility of using existing tools to protect software is studied. The possibility of improving and using a number of software protection methods against active fraud attacks was brought. Type of attacks exists and why protection is necessary was specified. Furthermore, we discussed several states of the art protection techniques which can be used in software to protect against analysis and tampering attacks. Analyzed such methods: Client-Server Solutions, Code Encryption, Code Diversity, Code Obfuscation, White-Box Cryptography, Tamper Resistant Software, Software Guards, Oblivious Hashing. Although we considered all these possible techniques separately, it is possible to combine these techniques into one solution.

Keywords: software protection methods, type of attacks, threats for software.

Introduction

With the increase of software flaws, there is a rise in the demand for security embedding to achieve the goal of secure software development in a more efficient manner. Any software is intended to recognize, prevent, stop and fix the damage caused by others on your computer or network can be called security software.

Problems of Software Protection. The main problem in the context of software security appears when software is given to remote hosts. Once this is done, the owner practically loses all control over the product. And from that moment on malicious users or malicious software [1, 2] can harm and intervene the local software. Chow et al. called these type of attacks white-box attacks [3, 4] because in this model the attacker has full access to the system. This means that the malicious user or program can execute the program, observe the memory, processor, and registers, and change bytes during execution, etc. Therefore, protection against analysis and tampering of code is necessary.

Attacks, on software. Two common attacks on software are tampering and reverse engineering. Tampering is attacks that aim to change the functionality of the software while reverse-engineering techniques try to analyze the software in order to understand its behavior.

Software attacks can be either static or dynamic. In a white box environment, all these techniques can be used. Due to that, software security requires improvement. The only things that might keep an attacker using these techniques are time and resource constraints. This means that if it takes a lot of memory and computing power to analyze a certain piece of code, this code has higher practical security to resist attacks.

Software Protection. The software can be protected in many ways. It can depend on trusted hardware, which is hardware based protection. Or it can rely on its own implementation and the underlying software, which is called software-based protection. Some techniques are the combination of both. The main benefit of software-based protection techniques is the low cost and compatibility with existing systems. In this Study, our main focus is on software-based protection.

The quality of security in an application consists of the required immunity of the application against reverse engineering (analyze) or tampering attacks. Here, we specify this level in more detail:

- **Vulnerability**: Open systems, such as a desktop, a notebook or a mobile device are much more vulnerable to attacks than closed systems, such as servers behind a firewall.
- **Value of content**: Depending on the kind of application and its content (code and/or data) varies the type of attacks and the number of methods and resources used for attacking the software.
- **Content lifetime**: Content or properties with a longer lifetime require a higher level of security.
- **Security life cycle**: The security of an application can be designed to be periodically renewable. Systems without upgrade possibilities need a higher security level than systems with regular upgrades.
- **Sensitivity for global attacks**: Global attacks are attacks affecting the whole system. This is achievable when the code includes a 'global secret', for example, a constant key or data at a fixed location for each user. In this case, the attacker can develop an automated attack and spread it through the Internet.

The actual security level is always a compromisation between the need for security and the way to implement this security.

Software Security Techniques

In the following sections we try to summarize techniques to protect code against malicious users and programs. This can be protection against either analysis or either tampering.

Client-Server Solutions: One of the earliest methods to protect critical software was to keep it running at the owner side instead of the user side.

Critical software was not disseminated to unstable hosts but maintained on a well-protected server. The protection of the server depends on as well as network, hardware, and software security (the operating system). The code itself is often not guarded by any other techniques. By this setup, the services are distributed not the software itself. From an attacker's perspective, the server will be seen as a black box that can be reached by sending queries and receiving replies.

The main drawback of client-server systems is that the server or the network bandwidth becomes a bottleneck, causing services to be temporarily unavailable. Although this can be resolved by upgrading network infrastructure, a new model has been proposed, called partial client-server. In this design, the sensitive code is divided into a critical and a non-critical part. The critical part needs to be protected and is, therefore, run at the server side, the non-critical part is distributed and is run at the client side. The benefit is that the load of the service is now better spread over the clients and the server. The code running at the server side can also be substantially smaller, although some extra overhead is needed to support communication between the client part and the server part. This directly shows the main problem. At first sight this model seems to unload the server, nevertheless, in practice, the client part and the server part have a highly interactive communication so that once more the bandwidth becomes a bottleneck. Although client-server was one of the first and still commonly used methods to preserve software from attacks, it actually tackles the problem by protecting the server and not the software running on it.

Code signing: Some languages (for example C) have no security mechanisms in line that check code before execution, therefore, these languages, in particular, are very sensitive to tampering attacks changing the program in a way that its computations cannot be trusted any longer.

To bypass the tampering of a program, its code needs to be protected during transmission and storage. Each time the program executes it should check and verify its integrity to detect tampering. Signing techniques [5] are most suitable for this type of checking. The owner can sign the software and the user can validate the signature appended to the software. This is already the case with some Windows drivers that are signed by Microsoft and verified by the user at installation time [6]. One could extend and automate this process so that the signature is verified at each execution of the program. For example, software guards [7] do not sign the code with a key but verify a calculated checksum with a stored one.

The downside is that without extra security measures in place the code and the signature are still vulnerable to intervention. If the signature scheme is known, one could simply change the code to its own needs, recompute the signature and restore the old signature by the new one. The verification module would then just verify the new signature and would not assume any tampering. The main reason for this vulnerability is that the signature and the verification module are not signed themselves.

Code Encryption: Additional to code signing, designers can also encrypt code during transmission and storage [8–11]. Tools such as cryptographic wrappers encrypt the code of a software application in order to avoid attackers gaining access to the software. It protects software against static reverse-engineering and tampering attacks. For example, an attacker cannot see the code and therefore not make any structured changes when the code is stored on a disk or transmitted over a network. Note that an attacker can always flip random bits and what will result in a corrupted application.

During program execution parts of the code will be decrypted 'on the fly' with a secret key. Unfortunately, at that moment the code appears, in memory for example, so that it is able to intercept. The intercepted code can then be debugged, decompiled, etc. This is the main vulnerability of this technique and furthermore makes the presence of a secret key this technique less suitable for distribution.

Even if the code or the data remains encrypted [12], an attacker can recognize what happens during runtime if bits in the encrypted code or data or flipped. This technique is also known as fault analysis [13]. Encrypted and polymorphic viruses [14–16] perform similar techniques. An encrypted virus encrypts at each new generation the body with a unique key. This is essential to avoid detection through string analysis searching for specific byte signatures. In front of the body, a decryption routine is added to secure that the virus body gets decrypted on the fly during execution. Nonetheless, if the encryption routine remains unchanged, scanning for signatures is still possible. For that, encrypted viruses evolved and added a mutation engine ensuring that for each new generation also the decryption routine has changed. This kind of viruses is therefore called polymorphic viruses. Note that the decryption routine can, of course, be protected with other analysis tackling techniques. Once a virus is decrypted and stored in memory, it will choose a new key, encrypt the new variant and add a modified decryption routine.

Code Diversity: The last month's viruses and worms [2] become a hot topic in the media. Triggered by these virus outbreaks discussions often mention the choice of operating system. This actually refers to the problem why viruses spread so successful. One reason could be that the software community is evolving to a 'monotone' distribution, meaning that most people use the same type of operation systems, containing the same type of bugs. This is one of the reasons why viruses, whom most of the time try to exploit only one bug at a time, are so successful.

Without arguing about safe operating system design and implementation, we can state that just as in nature diversity is stronger to resist threats such as viruses and worms. It also offers extra protection against global attacks because once software images are diversified, a common attack might be a lot harder to set up and only parts of the software community might be vulnerable.

Forrest et al. sketch the analogy between diversity in computer systems and diversity in biological systems [17]. Guided by this idea computer code could be

randomized, without changing the functionality or losing much user-friendliness or performance. Their paper presents some preliminary results on randomizing stack layouts by increasing certain slots with a random time 8 bytes. Such a simple modification could harden a program instance better against standardized buffer overflow attacks.

Another technique to battle buffer overflow attacks, called address obfuscation, is also based on the idea of code diversity [18]. This technique randomizes the code and data sections on the stack by randomizing all the base and start addresses, locations of routines and static data and introducing gaps between objects. More on buffer overflow protection techniques can be found in [19, 20].

Code Obfuscation: Object-oriented programming is used everywhere because it offers various advantages to read, adapt or extend the code. However, this way of programming in modules leaves many traces into an executable and reverse-engineers will exploit these traces as good as possible to reconstruct the original source code [21]. Therefore, programmers developed several techniques to maximally obscure the internals of a program so that analysis becomes very difficult. The most common technique to do is code obfuscation. This technique applies one or more transformations to code that make the code more resistant to analysis and tampering but preserve its functionality. Obfuscated code can then be distributed to untrussed hosts without risking to be reverse engineered soon.

Code obfuscation is used more and more due to the need for embedded software protection. It is originally designed for languages such as Java because Java bytecode is very sensitive for code analysis. This means it is easy to recover original Java source files out of Java bytecode files. Many Java obfuscators [22, 23] (and deobfuscators) have therefore been designed. Also .NET obfuscators [24] are becoming common on the Internet. Nevertheless, C/C++ obfuscators are very rare and difficult to find. Although, C and C++ are very common and widely used languages.

Wroblewski [25] and Mambo et al. [26] propose code obfuscation on an instruction level, e.g. Assembly code. This has certain advantages. First, the code does not have to be compiled anymore, which facilitates integrity checks and hashing of code. This is one of the reasons why software guards [7, 27] are implemented on an assembly level. Second, transforming on an instruction level instead of on a high level is often preferred for watermarking [28].

White-Box Cryptography: In past few years, attacks have been performed to extract key information out of RSA and even DES implementations. Boneh, Demillo, and Lipton have come up with a method for RSA [25], Biham and Shamir have continued this method for DES [13]. These new attacks focus on the extraction of the secret key embedded in a cryptographic implementation and are a new threat in security.

In August 2002, Chow et al. defined this new thread model, the white-box attack context or malicious host attack context as follows:

- Full-privileged attack software shares a host with cryptographic software, having complete access to the implementation of algorithms;
- Dynamic execution (with instantiated cryptographic keys) can be observed;
- Internal algorithm details are completely visible and alterable at will.

The attacker's objective is to extract the cryptographic key, e.g. for use on a standard implementation of the same algorithm on a different platform. Obfuscation alone does not help against this threat, because obfuscated cryptographic algorithms store parts of the secret key in the malicious hosts' memory and can thus be extracted.

Chow et al. proposed a new technique to secure cryptographic algorithms against white-box attacks, called white-box cryptography. This technique is based on the idea that an encryption function E_K with key K can be replaced by an equivalent function $E'_K = G.E_k.F^{-1}$ in which F is an input encoding and G is an output encoding. The strength of this substitution is that none of the implementation components computes the function E_K for a key K. An attacker would first have to analyze E_K and isolate the encoding functions F and G before he can analyze E_K to find the secret key K.

Due to the introduced functions F and G, it is possible to inject sufficient 'randomness' in the implementation so that finding and extracting the key is becoming hard. So far the only practical disadvantages of white-box cryptography are the code size and the extra execution time.

Tamper Resistant Software: Tamper resistant software requires very skilled programmers working on a binary or source code level to embed 'booby traps' for tamper detection in software. A good tamper resistant code always has a dual function. First, the code needs to identify undesired changes and second the program needs to fail in case of tampering.

Aucsmith came up with one of the first papers on tamper-resistant software (TRS) [30]. He proposed a tamper-resistant software architecture which bundles many of the previously mentioned techniques in order to realize a tamper-resistant software implementation. His technique is a combination of four principles:

1. Disperse secrets in both time and space.
2. Obfuscation of interleaved operations.
3. Installation unique code.
4. Interlocking trust.

These principles have also been applied as a base for ideas such as code diversity, software guards, code obfuscation, etc. Aucsmith's architecture consists of two parts namely integrity verification kernels (IVKs) and an interlocking trust mechanism. An IVK is a small, armored section of code to embed in a larger program. The IVK has mainly two functions:

1. Verifying the integrity of code segments of programs.

2. Communicating with other IVKs in order to accomplish these functions securely.

It is organized in cells, which are decrypted at runtime and thus define the smallest level of granularity

which is ever exposed unencrypted. The encryption of cells is made in a pseudo-random order based on generator function. Moreover, each IVK contains one or more keys. A secret key to sign and a public key to verify signatures made on other code segments.

The second part of the TRS architecture is the interlocking trust mechanism. It consists out of IVKs, an integrity verification protocol, and a system integrity program. These three parts operate together in an interlocking trust mechanism based on mutual integrity verification [30].

Software Guards: Chang et al. defined small pieces code that checksum code fragments [7]. Measuring an integrity checksum can be done by for example CRC [31]. Using a complex, nested network, these guards are able to verify each other's

Code plus the program code itself and repair it if necessary. In this way, tampering of the program is extended to detecting the complete agent network, this means identifying, localizing and eliminating the whole network of guards and then tapering the actual program code itself. A guards graph and its placement in a control flow graph (CFG).

The drawback of this method of software protection is that it is hard to automate and thus depends on one's programming skills. As a result, the support cost will be very high. Moreover, this technique does not offer any protection against dynamic analysis attacks.

The new study from Horne et al. attempts to extend and automate this technique [27] to enhance tamper resistance of programs. Their techniques are based on testers and correctors. The testers, code in Assembly, are included at the source code level, while the correctors are included in the object code. The values of the correctors and some watermark values are computed at installation time, ending in a watermarked, self-checking, fully functional program [27, 32, 33].

Oblivious Hashing: As a response to the idea of software guards checking only static code, Chen et al. insinuated an oblivious hashing (OH), a method that allows certain computation of a hash value of the actual execution [34, 35]. The approach is to hash the execution trace of a piece of code, enabling to confirm the run-time behavior of the software. Hashing instructions are mixed with the original code and take results of previous instructions and apply them to hash values stored in memory. Assignment results and control flow results achieve most of the dynamic behavior of a program, for that, it is adequate to hash only assignments and control flows.

Oblivious hashing has two important application domains. First, it is able to be used to give local software tamper resistance and second, it has the capacity to be used for remote code authentication. In a white box model, local software should render its own security so that remote code authentication is not an option [35].

Conclusion

In this study, we went through the problem of software protection. Type of attacks exists and why protection is necessary was specified. Furthermore, we discussed several states of the art protection techniques which can be used in software to protect against analysis and tampering attacks. Although we considered all these possible techniques separately, it is possible to combine these techniques into one solution.

REFERENCES

1. Intro to spyware. http://www.spywareguide.com/txt_intro.php.
2. R. E. Mahan. Malicious Software, http: //www.tricity.wsu.edu/htmls/cs427/public_html/ Ch%2013%20Malicious% 20Software.pdf.
3. H. J. S. Chow, P. Eisen and P. van Oorschot. A White-Box DES Implementation for DRM Applications. In Proceedings of 2nd work ACM Workshop on Digital Rights Management (DRM 2002), November 18 2002.
4. H. J. S. Chow, P. Eisen and P. van Oorschot. White-Box Cryptography and an AES Implementation. In Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002), 2002.
5. A. Menez, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, Inc., 1997.
6. Microsoft Corporation. Digital signature benefits for windows users, 2002.
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. Amin Salih M., Yuvaraj D., Sivaram M., Porkodi V. Detection And Removal Of Black Hole Attack In Mobile Ad Hoc Networks Using Grp Protocol. *International Journal of Advanced Research in Computer Science*. Vol. 9, No 6. P. 1–6, DOI: http://dx.doi.org/10.26483/ijarcs.v9i6.6335
9. Saravanan S., Hailu M., Gouse G.M., Lavanya M., Vijaysai R. Optimized Secure Scan Flip Flop to Thwart Side Channel Attack in Crypto-Chip. *International Conference on Advances of Science and Technology*, ICAST 2018. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Vol 274. Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-15357-1_34
10. Porkodi V., Sivaram M., Mohammed A.S., Manikandan V. Survey on White-Box Attacks and Solutions. Asian Journal of Computer Science and Technology. Vol. 7, Is. 3. pp. 28–32.
11. Manikandan V, Porkodi V, Mohammed AS, Sivaram M, "Privacy Preserving Data Mining Using Threshold Based Fuzzy cmeans Clustering", ICTACT Journal on Soft Computing, Volume 9, Issue 1, 2018, pp.1813-1816. **DOI:** 10.21917/ijsc.2018.0252
12. T. Sander and C. F. Tschudin. On Software Protection via Function Hiding. In Proceedings of the Second Workshop on Information Hiding, LNCS 1525:111–123, 1998.
13. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. Advances in Cryptology: Crypto '97, LNCS 1294:513– 525, 1997.
14. Symantic. Understanding and Managing Polymorphic Viruses. http://www.symantec.com/avcenter/reference/striker.pdf.
15. P. Szor and P. Ferrie. Hunting for Metamorphic, September 2001. http://www.peterszor.com/metamorp.pdf.
16. T. Yetiser. Polymorphic Viruses. http://vx.netlux.org/texts/html/polymorf.html.

17. S. Forrest, A. Somayaji, and D. H. Ackley. Building Diverse Computer Systems. In Proceedings of the Sixth Workshop on Hot Topics in Operating Systems, pages 67–72, 1997.
18. D. C. D. Sandeep Bhatkar and R. Sekar. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In Proceedings of the 12th USENIX Security Symposium, pages 105–120, August 2003.
19. C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. http://www.immunix.org/StackGuard/discex00.pdf.
20. I. Simon. A Comparative Analysis of Methods of Defense against Buffer Overflow Attacks, January 2000. http://www.mcs.csuhayward.edu/~simon/security/boflo.html
21. C. Cifuentes and K. Gough. Decompiling of binary programs. Software – Practice & Experience, 25(7):811–829, 1995.
22. Z. KlassMaster. The second generation java obfuscator. http://www.zelix.com/.
23. P. Solutions. Dasho - the premier java obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dasho/.
24. P. Solutions. Dotfuscator - the premier .NET obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dotfuscator/.
25. G. Wroblewski. General Method of Program Code Obfuscation. PhD thesis, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.
26. M. Mambo, T. Murayama, and E. Okamoto. A tentative approach to constructing tamper-resistant software. In Proceedings of New Security Paradigms Workshop, pages 23–33, 1997.
27. B. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic SelfChecking Techniques for Improved Tamper Resistance. In Proceedings of Workshop on Security and Privacy in Digital Rights Management 2001, pages 141–159, 2001.
28. J. P. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater. Robust object watermarking: Application to code. In Information Hiding, pages 368– 378, 1999.
29. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. Journal of Cryptology: the journal of the International Association for Cryptologic Research, 14(2):101–119, 2001.
30. D. Aucsmith. Tamper resistant software: an implementation. Information Hiding, 1174:317–333, 1996.
31. R. N. Williams. Welcome to the Sci.Electronics. A painless guide to CRC error detection algorithms, 1993. http://www.repairfaq.org/filipg/LINK/F_crc_v3.html
32. Yogesh Awasthi, R P Agarwal, B K Sharma, "Intellectual property right protection of browser based software through watermarking technique", International Journal of Computer Applications, vol. 97, no. 12, 2014, pp. 32-36.
33. Yogesh Awasthi, R P Agarwal, B K Sharma, "Two Phase Watermarking for Security in Database", International Journal of Computing, vol. 4, no. 4, 2014, pp. 821-824.
34. Kuchuk G.A. An Approach To Development Of Complex Metric For Multiservice Network Security Assessment / G.A. Kuchuk, A.A. Kovalenko, A.A. Mozhaev // Statistical Methods Of Signal and Data Processing (SMSDP – 2010): Proc. Int. Conf., October 13-14, 2010.– Kiev: NAU, RED, IEEE Ukraine section joint SP, 2010. – P. 158 – 160.
35. Y. Chen, R. Venkatesan, M. Cary, R. Pang, and S. S. an Mariusz Jakubowski. Oblivious hashing: a stealthy software integrity verification primitive. In Information Hiding, 2002.

Рецензент: д-р техн. наук, проф. С. Г. Семенов,
Національний технічний університет «Харківський політехнічний інститут», Харків
Received (Надійшла) 04.02.2019
Accepted for publication (Прийнята до друку) 21.03.2019

Обзор программного обеспечения безопасности

Аноушиван Рашидіна, С. Ю. Гавриленко, М. В. Почебут, О. А. Ситникова

В статье проведен анализ основных угроз и проблем защиты программного обеспечения. Рассмотрены методы защиты информации, их достоинства и недостатки, а также проведены исследования возможности использования существующих средств для защиты программного обеспечения. Показана возможность усовершенствования и использования ряда методов защиты программного обеспечения от активных атак и фальсификации. Для каждого существующего типа атаки указаны необходимые меры защиты. Кроме того, рассмотрены несколько современных методов защиты, которые можно использовать в программном обеспечении для защиты от атак анализа и взлома программы. Проанализированы такие методы: клиент-серверные решения, шифрование кода, разнесение кода, обфускация кода, криптография White-Box, программное обеспечение для защиты от несанкционированного доступа, защита программного обеспечения, остаточное хеширование. Хотя все эти методы рассмотрены отдельно, можно объединить их для совместного использования для программного обеспечения безопасности.

Ключевые слова: методы защиты программного обеспечения, тип атак, угрозы программному обеспечению.

Огляд програмного забезпечення безпеки

Аноушиван Рашидіна, С. Ю. Гавриленко, М. В. Почебут, О. О. Ситнікова

У статті проведено аналіз основних загроз і проблем захисту програмного забезпечення. Розглянуто методи захисту інформації, їх переваги і недоліки, а також проведені дослідження можливості використання існуючих засобів для захисту програмного забезпечення. Доведена можливість удосконалення і використання ряду методів захисту програмного забезпечення від активних атак та фальсифікації. Для кожного існуючого типу атаки вказані необхідні заходи захисту. Крім того, розглянуті кілька сучасних методів захисту, які можна використовувати в програмному забезпеченні для захисту від атак аналізу і злому програми. Проаналізовано такі методи: клієнт-серверні рішення, шифрування коду, рознесення коду, обфускація коду, криптографія White-Box, програмне забезпечення для захисту від несанкціонованого доступу, захист програмного забезпечення, залишкове хешування. Хоча всі ці методи розглянуті окремо, можна об'єднати їх для спільного використання для програмного забезпечення безпеки.

Ключові слова: методи захисту програмного забезпечення, тип атак, загрози програмному забезпеченню.