Saqib et al LGURJCSIT 2018

LGURJCSIT
ISSN: 2519-7991

LGU Research Jounral for
Computer Sciences & IT

Vol. 2 Issue 4, October - December 2018

Hashing Based Software Watermarking for Source Code Files

Shazia Saqib[1], Asad Raza Kazmi[2], Saleh Alrashed[3]
[1]Department of Computer Science, GCU, Lahore Pakistan, Lahore Garrison University, Pakistan, shaziasaqib@lgu.edu.pk
[2]Department of Computer Science, GCU, Lahore, Pakistan, arkazmi@gcu.edu.pk
[3]University of Dammam, Dammam, KSA, saalrashed@uod.edu.sa

Abstract:

Software is developed and delivered to clients as a routine part of software engineering life cycle . Software is quite an expensive entity. However various attacks are possible on software to make its illegal use. Different solutions are there to prevent piracy. Software watermarking embeds a watermark in the source code so that it is undetectable yet it proves the ownership of the developer. The technique has been tested for C++ source code files, however, it can be applicable on any other language. The proposed techniques scans the code for all possible constants, forms a hash sequence using MD5 algorithm that calculates the watermark and stores in Date & Watermark Value Repository (DWVR).

Key Words: Watermarks, Piracy, Attack, Embedding, Extraction

1. INTRODUCTION

Software piracy has always been around as a big blow to the software industry. Disk or CD has long been used to deliver software to the place required. This controlled illegal software distribution to some extent. However, as high speed Internet is getting available to everyone, physical media based piracy has been reduced a lot. Delivering software in platform independent form makes its reverse engineering quite easy. Even if we detect that the software has been rated, its hard to detect the culprits.

There are organizations like Business Software Alliance (BSA)[21]. These use audits to verify legality of software. Even audit cannot trace the person behind piracy. There are many techniques to discourage software piracy. These techniques have been implemented both in hardware and software. Softwarebased solutions that include code obfuscation, software tamper-proofing, and software watermarking are cheaper but the security provided by them is not that powerful[1].

Although hardware based solutions give more reliable security against piracy but they are more convenient to be used by the software vendor however end-user is not comfortable with hardware.

There are Software-based Methods available as well [1].

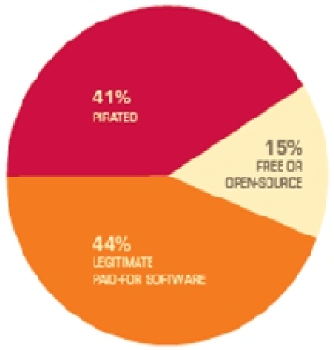

Figure 1: Software Units by Category

Software pirates attack software in three ways. These attacks (additive, subtractive, distortive) are handled by technical defenses. One such solution is Code obfuscation. This approach prevents reverse engineering implementation. In this technique source code is changed to a very difficult form without changing its actual output. This makes the software very hard to reverse engineer[2-4] .

Software tamper-proofing is another way to control piracy. This technique works in such a way that it prevents program execution in case someone tries to change it[3].

A very popular technique to control software piracy is software watermarking. It stops a user to stop illegal distribution of copies of the software. In addition to software watermarking, media watermarking uses intentional errors of images, audio, or videos which are undetectable by the human auditory system. Same way a software watermark is added to the software which results in adding the proof of ownership in source code[21-27].

Software watermarking applies different techniques. It may scan identifiers in the program. It may form a graph of program statement. Sometimes a unique programming style is adopted. Program Flow can also help determine watermark value[3].

2. LITERATURE REVIEW

Many algorithms have been developed in past few years to stop the software piracy. Preda had proposed a Robust data hiding algorithm that adds watermark in random coefficients of Discrete Wavelet Transform. The input image is transformed for increasing efficiency of the watermark [15].

Puncturable pseudo-random functions (PPRF), assume the indistinguishability obfuscation and injective one-way functions[6].

The algorithm uses The codec system for encoding watermark numbers uses reducible permutation flow-graphs. The algorithm uses the self-inverting permutation and the reducible permutation graph[5].

In another research the General Chinese Remainder Theorem helps in dividing the watermark represented as a huge figure into pieces to increase stealth [16]. Zhang et al. uses satisfied parameters to calculate the watermark. Some of the program's constants are replaced by specific level hash function. will lead to the application's undefined behavior[19]. Abstract software watermarking, hides the watermark in the program code and it can be extracted only by an abstract interpretation of the concrete semantics of this code[9].

The survey by Nagra and Thomborson explains the validation mark, licensing mark, authorship mark and fingerprinting mark etc.[12]. In another technique, watermarking mechanism was added at the assembly level and hides the watermark in the power consumption of the device[14].

The work by Collberg et al. differentiates between obfuscation, watermarking and tamper proofing. A defense against reverse engineering is obfuscation[7].

The watermarks are added in such a way that even minor changes to the software or flow graph have no chance of elude detection by a probabilistic algorithm[18]. A novel dynamic software watermarking design based on Return-Oriented Programming (ROP) adds watermarking code into data and is variant to the attacks based on code analysis[13].

Version Based Software Watermark is used to the LOC attack and Version attack of abstract software watermarking[12]. The research by HämmerleUhl et al. uses the static analysis-based approach which is helpful to identify the watermark even in the presence of few lines of program code without executing it . We illustrate the technique by a simple abstract watermarking protocol for methods of JavaTM classes[9].

Another efficient watermarking technique is based on the dynamic branching behavior of programs. The technique uses tamper-proofing and error correcting algorithms to make path-based watermarks invariant to a wide variety of attacks[3].

Constant String Static Watermarking Algorithm, Bogus Initializer Static Watermarking Algorithm, Bogus Initializer Static Watermarking Algorithm , Bogus Expression Static Watermarking Algorithm, The Davidson-Myhrvold Watermarking Algorithm, Robust Object Watermarking Algorithm, Monden Watermarking Algorithm, Venkatesan's Graph Theoretic Watermarking Algorithm, Execution Path Watermark Algorithm and Thumbprinting .Net Application are also some of the popular algorithms used for watermarking[13-17].

3. WATERMARK EMBEDDING AND EXTRACTION

The watermarking process uses the following algorithm:
1. Add Watermark in the source program
2. Watermark Extraction/Recognition in case doubt of piracy arises Given a program SC, a watermark w, and a key k, a software watermarking system consists of two functions:
Add watermark (SC,w, k) $\rightarrow$ SC'
Extract watermark (SC', k) $\rightarrow$ w[18].

or
• Watermark Embedding and Extraction
– Program + Secret key -> Watermarked Program
– Watermarked Program + Secret key-> Program

3.1. Watermarking Algorithm

This paper is inspired by the work done by Xuesong Zhang, Fengling He, Wanli Zuo. Their work indicates that it is a better technique to hide watermark as a part of the source code as it is quite challenging to separate code and the software watermark even with compiler tools and tampering with them would affect the program correctness and/or performance [11-12].

To apply this algorithm the following are the conditions to be fulfilled:
a. The program should not have watermark stored in program anywhere.
b. Watermark should be calculated using the program contents so that if source code faces any attack, the response of program will be different from what was predicted.
c. Every program must have a unique watermark.

3.1.1. Design

Our algorithm works the following way:
a. Scan the program for all possible constants.
b. Record all possible constants, if needed new constants can be generated.
c. Using these constants form the sequence C1C2…Cn
d. Apply any encryption algorithm like MD5 or SHA. Generate watermark and store in Time Stamping file DWVR. It contains two things: date of watermark stamping and value of the water mark. When the program has been compiled and tested, After completion of a code, it is time stamped initially with 1/1/2999) thus the file will always contain the date and time on which the program was first time time-stamped. Once the program is scanned for all the constants, the sequence of constants is passed to the encryption algorithm.

3.1.2 Watermark Embedding:

Replace all constants with function calls. No attack can change the value returned by a function.

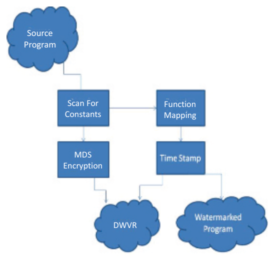

Figure 2: Watermark embedding

3.2 Watermark Extraction

Scan the program, check all function calls, find the candidate functions. If they contain a dummy
parameter whose contents match the contents of a flag , replace the function call with the its returning value. Scan the program for constants, apply encryption algorithm. Compare the result with the watermark stored in the time stamping file.

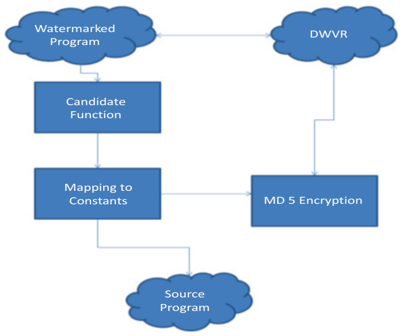

Figure 3: Watermark Extraction

3.2.1. Threat Analysis

Subtractive Attack

As the watermark is not stored anywhere in the program it cannot be removed.

Distortive Attack

It is very hard to distort the watermark as it is very hard to change function result. However replacing the constants with function calls may increase program execution time.

Additive Attack

One solution to additive attack is time stamping with a third party which is an expensive solution.

Moreover Third party facility may not be available especially in underdeveloped countries. One solution which we have used in this algorithm is to timestamp using a external file. The program right at the beginning searches for a particular file, if it finds it, It takes the system time & Date, encrypts it and compare with the encrypted time stored in that file. It stored the older time there, even a history of execution can be maintained.

4. RESULTS

4.1 Data Rate

The algorithm has been tested with different programs for programs having different size. The proposed scheme uses encryption algorithm there are several inbuilt algorithms like RSA, Blowfish etc. however we selected MD-5 checksum for calculation of watermark value. We give input text of any size resulting in same size output from these encryption algorithms i.e. 128 bits or 16 bytes.

4.2 Embedding Overhead on Execution Time of Program:

To find the effect of watermark on source code, the watermark was added to the program. The time required to completely run the program with and without watermarking was calculated a number of times with same code. The table and graph give us actual picture of before and after the application of watermark. However results show that there was very little impact of the watermark on the program execution speed.
This shows that the technique used in this paper adds a high quality watermark to the source code. However if the source code has huge collection of many constants too many iterations/repetitions then performance of watermarked program might degrade.

Table 1: Impact on Execution Time

Program Title	Execution Time In milliseconds	
	Original Program	Watermarked Program
sorting of an array	6	6
Linear Search	3	3
pay roll	4	4
direction of Qibla	6	6

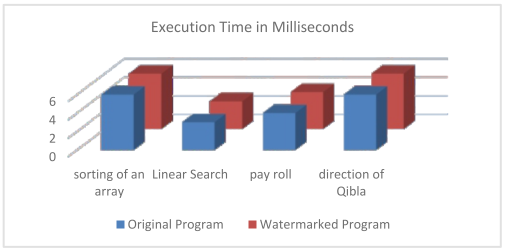

Figure 5 Embedding overhead in program execution time

4.3 Embedding Overhead in terms of Size of Program

The effect of the watermarking technique was also determined in terms of the source code size. The size ws determined before applying the watermark and after applying the watermark. For the sake of accuracy we repeated the process again and again. However the code is least affected by the size of the code. In case of too many repetitions or too many constants there is slight increase in code size but still ignorable.

Table 2:Program Size before and after Water Mark

Program Title	Program Size	Watermarked Program Size (KB)
sorting of an array	40.7	41.0
Linear Search	15	16
pay roll	40.7	40.9
direction of Qibla	44.7	59.1

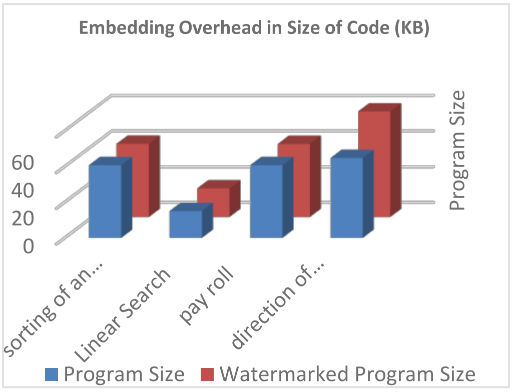

Figure 6: Impact of watermark on program size

4.4 Stealth

When we measure efficiency of this technique , Stealth is a key deciding factor. This theory of software watermarking says that the watermark should be very carefully embedded in program so that it is not visible and it does not change the end product of our source code at all. The watermark calculated by our technique can not be removed by the attacker as it is not stored anywhere in the program and is calculated at runtime. The kind of watermark used in the program makes it hard to locate the watermark within the program.

5. CONCLUSION

The algorithm represented here is a very safe way to watermark the code, it is resilient to all types of attacks. In addition watermark can be easily extracted from the source code. Use of algorithm like MD-5 ensure that every program has a unique watermark. Another added benefit is to concatenate user information along with the program constants. So this algorithm is resistant to various forms of transformation attacks.

6. FUTURE DIRECTIONS

Many improvements can be made to this approach. One such initiative would be to store watermark and time stamping information somewhere in the operating system However, this makes this technique operating system structure dependent. Another improvement would be to include all type of constant (String constants, Character Constants, floating point constants etc.) in addition to numeric constants.

References

[1] D.Seetha Mahalaxmi, D. S. (n.d.). Arboit, G. (2002). A method for watermarking java programs via opaque predicates. The Fifth International Conference on Electronic Commerce Research (ICECR-5).

[2] Becker, G. T., Burleson, W., Paar, C., & Horst, G. (2011). Side-Channel Watermarks for Embedded Software, 478–481.

[3] C. Collberg, E.Carter, S.Derbray, A. huntwork. (2013). Dynamic Path-Based Software Watermarking. Journal of Chemical Information and Modeling, 53(9), 1689–1699. https://doi.org/10.1017/CBO9781107415324.004

[4] Chroni, M., & Nikolopoulos, S. D. (2012). An Embedding Graph-based Model for SoftwareWatermarking. https://doi.org/10.1109/IIH-MSP.2012.69

[5] Cohen, A., Holmgren, J., & Vaikuntanathan, V. (2015). Publicly Verifiable Software Watermarking. Cryptology ePrint Archive, Report 201, 1–38. Retrieved from https://eprint.iacr.org/2015/373.pdf

[6] Collberg, C. S., Society, I. C., Thomborson, C., & Member, S. (2002). Obfuscation Tools for Software Protection.

[7] Cousot, P., & Cousot, R. (2004). An Abstract Interpretation-Based Framework for Software Watermarking. Principles of Programming Languages, 31, 173–185. https://doi.org/10.1145/982962.964016

[8] Dalla Preda, M., & Pasqua, M. (2017). Software Watermarking: A Semantics-based Approach. Electronic Notes in Theoretical Computer Science, 331, 71–85. https://doi.org/10.1016/j.entcs.2017.02.005

[9] Hämmerle-Uhl, J., Raab, K., & Uhl, A. (2011). Robust watermarking in iris recognition: application scenarios and impact on recognition performance. ACM SIGAPP Applied Computing Review, 11(3), 6–18. https://doi.org/10.1145/2034594.2034595

[10] "Fourth Annual Global Software Piracy Study", w3.bsa.org/globalstudy//upload/2007-

Piracy-study-Findings.pdf, – May 2007.

[11] Ma, H., Zhang, H., Lu, K., Ma, X., Jia, C., & Gao, D. (2015). Software Watermarking using Return-Oriented Programming. Asian ACM Symposium on Information, Computer and Communications Security, 369–380. https://doi.org/10.1145/2714576.2714582

[12] Nagra, J., Thomborson, C., & Collberg, C. (2002). A functional taxonomy for software watermarking. Aust. Comput. Sci. Commun., 24(1), 177–186. https://doi.org/10.1145/563857.563822

[13] Singh, S., Singh, H. V., & Mohan, A. (2015). Secure and Robust Watermarking Using Wavelet Transform and Student t-distribution. Procedia Computer Science, 70, 442–447. https://doi.org/10.1016/j.procs.2015.10.071

[14] Tang, Z. (2011). A Tamper-proofSoftware Watermark using Code Encryption, (61070176), 12 | P a g e
156–160.

[15] Tyagi, S., Singh, H. V., Agarwal, R., & Gangwar, S. K. (2016). Digital watermarking techniques for security applications. 2016 International Conference on Emerging Trends in Electrical Electronics & Sustainable Energy Systems (ICETEESES), 379–382. https://doi.org/10.1109/ICETEESES.2016.7581413

[16] Venkatesan, R., Vazirani, V., & Sinha, S. (2001). A graph theoretic approach to software watermarking. Information Hiding, 157–168. Retrieved from http://link.springer.com/chapter/10.1007/3-540-45496-9_12

[17] Zhang, X., He, F., & Zuo, W. (2008). Hash Function Based Software Watermarking. 2008 Advanced Software Engineering and Its Applications, (20070533), 95–98. https://doi.org/10.1109/ASEA.2008.57

[18] Danicic, J. H. (2011). A Survey of Static SoftwareWatermarking. IEEE.
[19] Lucila Maria Souza Bento, D. B. (2013). Towards a provably resilient scheme for.

[20] Sukriti Bhattacharya, A. C. (2010). Zero-knowledge SoftwareWatermarking. IEEE.

[21] ACT Accountancy Policy - Software http://www.treasury.act.gov.au/accounting/download/AP_05.pdf, – February 2002.

[22] Christian S. Collberg and Clark Thomborson "Watermarking, Tamper-Proofing and Objuscation- Tool for Software Protection". In IEEE Transaction on Software Engineering Vol 28 in August 2002.

[23] William Zhu, Clark Thomborson, and Fei-Yue Wang "A Survey of Software Watermarking". In Springer-VerlagBerlin ISI 2005, LNCE 3495, pp. 454 - 458 2005.

[24] Monden, A., Iida, H., Matsumoto, K., Torii, K., and Inoue, K. 2000. A Practical Method for Watermarking Java Programs. In 24th International Computer Software and Applications Conference (October 25 - 28, 2000). COMPSAC. IEEE Computer Society, Washington, DC, 191-197. 13 | P a g e

[25] D. Curran,N. J. Hurley,M. ´O Cinn´eide, 2001, Securing Java through Software Watermarking

[26] Genevi`eve Arboit, "A Method for Watermarking Java Programs via Opaque Predicates"

[27] Hash Function Based Software Watermarking, Xuesong Zhang, Fengling He, WanliZuo, College of Computer Science and Technology, JilinUniversity, Changchun, 130012, P.R.China xs_zhang@126.com, Hefl@jlu.edu.cn, wanli@jlu.edu.cn