Journal of Engineering Science and Technology Review 11 (4) (2018) 48 - 51

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

A Subtle Serial Key based Software Protection Scheme

Aderibigbe Israel Adekitan and Abidemi Orimogunje

Department of Electrical and Information Engineering, Covenant University, Ota, Ogun State, Nigeria

Received 11 March 2018; Accepted 19 September 2018

Abstract

Software piracy is a modern day war between malicious software pirates and software developers. Annually, developers lose billions of dollars in revenue to piracy, making it an industry bane that must be controlled by all means. The drive for improved software protection systems has increased the complexity of both proposed and implemented schemes. Some of the methods are cost intensive in terms of development, management and hardware requirement (smart card tokens), while this may be justified for costly software applications, for low cost and basic applications by small scale developers, implementing advanced protection schemes is often economically infeasible and largely an overkill. This study presents a subtle software protection model using serial keys. The model implements a form of obfuscation by using hidden codes, encrypted functions and uses a distraction technique by diverting the attention of potential hackers to the serial key while trickily using coded strings for the actual user authentication.

Keywords: Application software protection, serial key authentication, function hiding, data and system security, unique signature, encryption

1. Introduction

Software is vital for productivity virtually in all spheres of life. Software are deployed for program driven machines in industries, data processing applications and various simulation tools in different operations and research fields. Software development requires a lot of resources and man-hours, and as such, a sizable budget must be available to drive the software development project based on the project scope. After a successful software design and implementation, the developer needs to recoup on the investment, and this is done via sales of various licences for the use of the developed application, but this is often challenged by software piracy. Annually, the software industry loses billions of dollars to piracy [1], and this has triggered efforts toward protecting software applications.

A number of software applications are unprotected while others have weak protection. Software protection is a daunting task particularly with the ever growing army of hackers, skilled at reverse engineering and windows data recovery [2]. This has made it compulsory to keep evolving improved strategy using both software and hardware protection schemes, coupled with legal protection and sanctions. At times illegal software use can also be in the form of multiple installation of legally acquired single user licence, and this may go unnoticed if not monitored.

Software protection may be in the use of license keys that are checked during installation. This requires that the licence validation be embedded in the software creating an opportunity for hackers. The use of hardware signatures is also common and this entails extracting key information from the hardware on which the application is installed and using it to generate a key. This creates a problem for the user if the user replaces his hardware with a new one. The study by [3] proposes the use of function hiding techniques that performs key checks without revealing the method applied, but this is only limited to polynomial functions [1]. Code obfuscation method is another method that tries to hide the program behaviour making it hard for hackers to understand and manipulate [4-6]. Some network based protections schemes have also been deployed, and this performs license checks each time the application is run, making it challenging for users with irregular access to the internet [7].

Further research on software protection led to the development of tamper proof [8] and anti-debugging approaches, and the use of hardware for software protection. According to [9], a perfect, only software based solution to software protection is not realistic. The use of program specific hardware tokens or dongles such as smart cards that uses communication ports create card juggling issues for different applications [1]. The study by [10], proposed the use of smart cards and digital certificates using a license management system. A robust protection scheme was developed by [1] using smart cards and cryptographic techniques.

Computer security and trust management is vital for online services [6, 11], and ultimately software protection has become so complex and costly in terms of skill requirement and the use of hardware devices. For expensive software applications, these advance schemes may be cost effective but for small size applications, particularly for low cost software and small scale software enterprises developing basic applications in developing countries some of these schemes are overkill due to the level of software patronage and quality challenges [12, 13]. Hence, there is a need to develop a functional but easy to implement, and cost effective software protection scheme which does not require an additional hardware token in the form of dongles and

*E-mail address: sde_kitan@yahoo.com

ISSN: 1791-2377

doi:10.25103/jestr.114.06

specialized smart cards thereby reducing implementation cost and also eliminating the inconvenience for users to carry a smart card per software creating the problem of card juggling. Also, there is a possibility of hardware token failure or for the user to misplace the hardware token. Instances where codes were developed to bypass the card verification have also been reported.

In this study, a tricky but effective serial key based software protection model is proposed which can be easily deployed, even on low cost software, and it does not require sophisticated computers or specialized tamper proof processors, and expensive computation using asymmetric cryptosystem to deploy, but the user authentication process is tricky, thereby limiting the chances of the software being hacked and used illegally.

2. Methodology

To demonstrate the implementation of the subtle, serial key based protection system; an interface was deployed using Visual Basic for collecting vital information about the software user. The submitted data is encrypted, saved on the PC and also uploaded online to the developer's data base with a unique identification for each specific user. A user's email address can only be used for one registration, and users suspected of attempting multiple registrations can be automatically identified for further investigation when the same name or telephone number is submitted for different email addresses. Using function hiding, the collected data is concatenated into strings and divided into multiple parts based on pre-classified specific attributes of the collected data, using an encrypted function, a rule based selection of data segment from each divided data part is done and then data coding is performed to transform the processed data portions into three visually meaningless validation strings which will eventually be used for user verification after a trigger is activated by the developer during online payment.

At the developer's end, a unique serial key is generated for the user using the received encrypted user data. The software developer then sends a link to the user to purchase the unique serial key via online payment. After a successful online payment, with the buyer's attention drawn to the displayed serial key, the subtle user verification process which is independent of the serial key is automatically activated.

3. The proposed subtle software protection scheme

This study presents a basic software protection system that combines a form of obfuscation with online verification. It attempts to direct attention at the serial key but with subtlety, it actually uses a separate series of short strings for actual post registration software validation. Microsoft serial keys typically use an alphabet base of 24 characters which most people are quite familiar with. In this model, a 46 character base is implemented which makes the serial key generated quite different and unfamiliar. For a 20 character serial key, 110.47 bits of data $(\log_2 46^{20})$ will be required. The scheme is described in Figure 1.

Although, according to [1] there is no absolutely secure solution in software protection; in this study an attempt is made to make a simple software authentication process tricky and difficult to hack by subtly misdirecting the focus of potential hackers to the serial key why performing continuous verification with alternative strings. The feasibility of the proposed scheme was demonstrated using windows form. After the protected software is installed, the user data form will pop-up and the user must mandatorily fill it to complete the installation process.

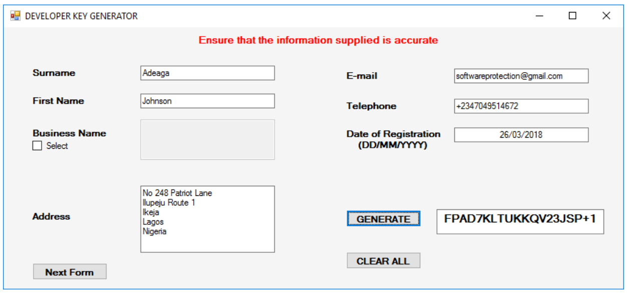
The data collected is encrypted, processed on the user PC and also sent to the software developer for saving on the database and for generating the user specific serial key. Figure 1 shows the generation of a unique serial key by the developer using the decrypted data received from the filled user registration form. A link is sent to the user to complete the transaction by making payment online to purchase a unique serial key. Once the payment is successfully made for the serial key, the software automatically picks the unique serial key and displays it to the user who then clicks FINISH to complete the transaction. By displaying the serial key, the user's attention is drawn to it, and, may consider writing it for possible re-use on another system. Also, a potential hacker's attention will also be drawn to the serial key and the hacker is being tricked to focus on how to re-use or re-generate valid serial keys using the purchased one.

Unknown to any user attempting illegal tampering with the serial key validation process, is the fact that the serial key is only useful at the point of purchase. Once the software purchase is successful and validated, the serial key becomes a dummy variable, and the application initiates the subtle user verification process at each run. During installation on the PC the application uses code obfuscation to develop three coded unique strings (String A, String B and String C) for data validation. Figure 2 shows the three strings generated by performing input filtering, text concatenation and encoding. Each of the three strings will be stored at different locations locally on the PC. The use of online payment and activation directs the attention of potential hackers to the online serial key activation and validation process without being aware of the use of the three strings for the actual user verification as detailed in the flow chart in Figure 3.

Each time the application is opened, the application will run normally for a random amount of minutes and then the validity of string B is checked and if it is not valid the application will keep running for further random minutes and then suddenly crash with an ambiguous error message. If the user restarts the application, the user will be automatically taken to the registration page as a new user. Also, if the initial post start-up check is successful; when the user is now closing the application, before exiting, the application will check the validity of String A and String C, and if either of the two strings are invalid a flag is set such that at the next opening of the application the user is also automatically taken to the user registration form as a new user. A user that changes hardware can be re-authenticated after verification using the stored user data in the developer's central database. The operational model of the subtle serial key is shown in Figure 3.

4. Results and Discussion

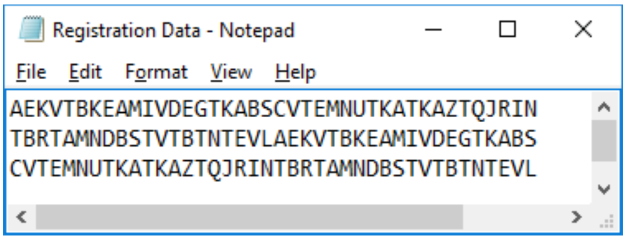
The model demonstration using windows form confirms the feasibility of the implementation on a commercial scale. Using sample user data, serial keys were generated using the serial key generator to depict the developer's end of the process while the three verification strings were also successfully generated from the user data and verified during operation. When an attempt was made to use the serial key for one user to validate the application form for another user, an invalid serial key error was triggered because the serial key supplied is not a coded equivalent of the submitted user data. The non-use of the serial key after payment, and the verification of String B few minutes after startup, and String A and C when closing the app creates a rear verification process that potential hackers are not used to. Also, by performing the verification checks only after running the application for a random period of time at each run makes it difficult to predict the operation, and it also prevents forming an easy general conclusion of precisely saying that a tampered or hacker modified installation keeps crashing after running consistently for e.g. ten minutes which might make targeted reverse code debugging easily possible.



Ensure that the information supplied is accurate

Surname: Adeaga
First Name: Johnson
Business Name
☐ Select
Address: No 248 Patriot Lane
Ilupeju Route 1
Ikeja
Lagos
Nigeria
E-mail: softwareprotection@gmail.com
Telephone: +2347049514672
Date of Registration (DD/MM/YYYY): 26/03/2018
FPAD7KLTUKKQV23JSP+1

Fig. 1. Generation of serial key by the software developer using the user data



```
AEKVTBKEAMIVDEGTKABSCVTEMNUTKATKAZTQJRIN
TBRTAMNDBSTVTBTNTEVLAEKVTBKEAMIVDEGTKABS
CVTEMNUTKATKAZTQJRINTBRTAMNDBSTVTBTNTEVL
```

Fig. 2. The three strings generated for user validation

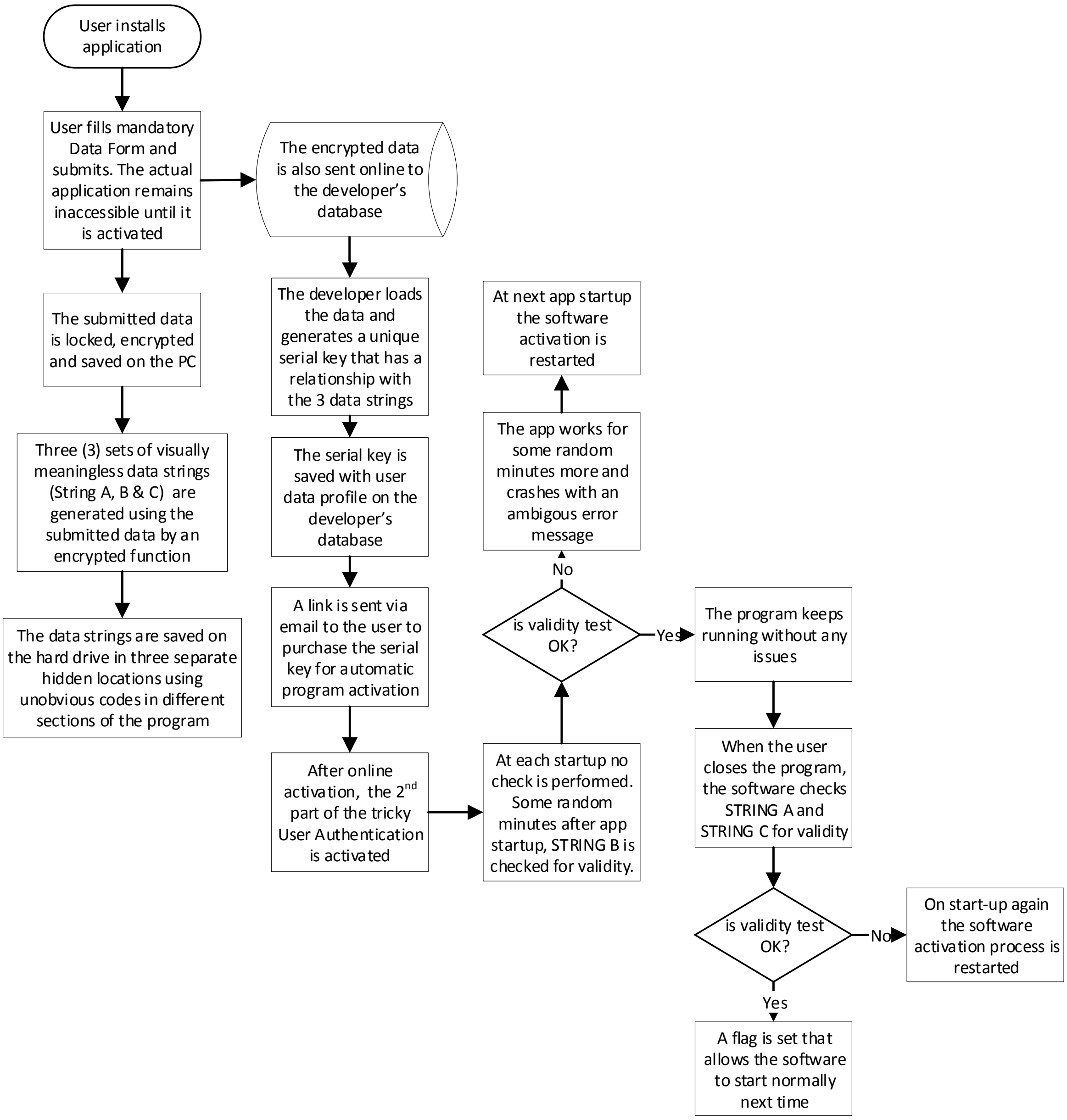

User installs application
User fills mandatory Data Form and submits. The actual application remains inaccessible until it is activated
The encrypted data is also sent online to the developer's database
The submitted data is locked, encrypted and saved on the PC
Three (3) sets of visually meaningless data strings (String A, B & C) are generated using the submitted data by an encrypted function
The data strings are saved on the hard drive in three separate hidden locations using unobvious codes in different sections of the program
The developer loads the data and generates a unique serial key that has a relationship with the 3 data strings
The serial key is saved with user data profile on the developer's database
A link is sent via email to the user to purchase the serial key for automatic program activation
After online activation, the 2nd part of the tricky User Authentication is activated
At each startup no check is performed. Some random minutes after app startup, STRING B is checked for validity.
is validity test OK?
No
The app works for some random minutes more and crashes with an ambigous error message
At next app startup the software activation is restarted
Yes
The program keeps running without any issues
When the user closes the program, the software checks STRING A and STRING C for validity
is validity test OK?
No
On start-up again the software activation process is restarted
Yes
A flag is set that allows the software to start normally next time

Fig. 3. Flow chart of the proposed software protection scheme

5. Conclusion

In this study, a subtle easy to implement software protection scheme was proposed and implemented using windows form. The proposed model takes the use of serial key as a means of protection, a step further by combining online activation with subtle obfuscation strategy as a distraction strategy that will make potential hackers spend time effortlessly on breaking the serial key generation and validation process without being aware that actual user validation is performed using three hidden coded strings which is activated when a matching serial key is supplied.

Acknowledgments
The Authors appreciate Covenant University Centre for Research, Innovation and Discovery for supporting the publication of this research study.

References

1. Maña, A. and E. Pimentel. An Efficient Software Protection Scheme. in Trusted Information. 2001. Boston, MA: Springer US.
2. Olajide, F. and S. Misra, Forensic investigation and analysis of user input information in business application. Indian Journal of Science and Technology, 2016. **9**(25).
3. Sander, T. and C.F. Tschudin. On software protection via function hiding. in International Workshop on Information Hiding. 1998. Springer.
4. Jiutao, T. and L. Guoyuan. Research of software protection. in 2010 International Conference on Educational and Network Technology. 2010.
5. Wroblewski, G., General method of program code obfuscation (draft). Citeseer, 2002.
6. Osemwegie, O., et al., On issues, strategies and solutions for computer security and disaster recovery in online start-ups. International Journal of Applied Engineering Research, 2017. **12**(19): p. 8009-8015.
7. Carlsson, K., Developing an efficient software protection and licensing scheme, in Department of Computer Science and Engineering. 2014, Chalmers University of Technology.
8. Tan, G., Y. Chen, and M.H. Jakubowski. Delayed and controlled failures in tamper-resistant software. in International Workshop on Information Hiding. 2006. Springer.
9. Goldreich, O. Towards a theory of software protection. in Conference on the Theory and Application of Cryptographic Techniques. 1986. Springer.
10. Aura, T. and D. Gollmann. Software License Management with Smart Cards. in Smartcard. 1999.
11. Govindaraj, P. and N. Jaisankar, A review on various trust models in cloud environment. Journal of Engineering Science and Technology Review, 2017. **10**(2): p. 213-219.
12. Sowunmi, O.Y. and S. Misra. An empirical evaluation of software quality assurance practices and challenges in a developing country. in Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015, 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015, 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015 and 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015. 2015.
13. Sowunmi, O.Y., et al., An empirical evaluation of software quality assurance practices and challenges in a developing country: A comparison of Nigeria and Turkey. SpringerPlus, 2016. **5**(1): p. 1-13.