A Comparative Analysis of Software Protection Schemes

Muhammad Khan[1], Muhammad Akram[1], and Naveed Riaz[2]
[1]Shaheed Zulfikar Ali Bhutto Institute of Sciences and Technology, Pakistan
[2]College of Computer Science and Information Technology, University of Dammam, Saudi Arabia

Abstract: *In the era of software globalization, the need for securing software is much sought to ensure its smooth functioning for continuous availability of services to the users. Particularly, in cloud computing environment, all the software in the cluster needs to be secured and shielded against unauthorized accesses. Software crackers are always in the search of flaws in the software to obtain access to the software functionally by penetrating into the software skeleton. This paper reviews and critically analyzes various software protection techniques, both software-based and hardware-based, that can help control the software piracy issues in order to determine their efficacy and specific use in different environments and scenarios. The software protection techniques explored in this paper include cryptography, software watermarking, secure access scheme, software aging, guards, obfuscation and multi-block hashing techniques. The paper also discusses the taxonomy of the software protection techniques and the probable attack models that can be launched against each technique to evade the protection mechanism.*

Keywords: *Software tempering, software reversing, cryptography, watermarking, digital rights management.*

Received March 21, 2013; accepted March 17; 2014, published online June 26, 2014

1. Introduction

Software protection plays an important role in the fields of computer science and information technology [26]. In ours Internet age, the use of various types of software has become an essential part of industry, trade and commerce, official routine work and even in our personal daily life activities. The biggest challenge being faced by the software industry is the handy duplication and redistribution of software components and digital documents; hence, the issue of software security due to potential threats of reverse engineering, piracy and tampering has become a matter of prime concern. The piracy rate in some countries has been anticipated to be as high as ninety two percent.

A number of techniques have been formulated to protect software from unauthorized use and make it temper resistant. Due to serious issues of software piracy, reverse engineering, tempering, exploits and virus/Trojan attacks, researchers have developed a number of software protection techniques to augment the security of the computer systems [26]. Security related challenges and problems fall into two broader categories; technical challenges that relate to availability of technology and infrastructure of information structure; and social challenges that relate to the impact of human factor on security of a system [24]. The ideal software protection mechanism is to achieve the concept of 'one machine, one code' [12]. This mechanism can be turned into reality by introducing dynamic registration code that takes into account fingerprint of the hardware e.g., disk serial number, BIOS information, MAC addresses etc.

Software piracy is the foremost problem for the software industry as huge financial losses suffered by the software producers are primarily due to rampant illegal distribution and unauthorized use of products made by them [21]. Software is required to be safeguarded in such a way that it should always retain its functionality as well as protect the intellectual property and frustrate the attempts of reproducing its illegal copies [21]. The earlier proposed methodologies include: Cryptographic techniques and physical tamper-resistant devices like dongles. Software encryption methodologies involve using encrypted code where program instructions are decrypted on the fly prior to their execution. Software diversity is another protection mechanism against different software attacks (e.g., viruses) which can be easily achieved through slightly obfuscating new instances of a program which are functionally comparable to the original code. Moreover, formulating and putting in place effective content protection strategies for Digital Rights Management (DRM) systems are need of the day for the publishing and entertaining industries as volume of the digital contents being produced and distributed are increasing day by day. DRM has become a necessity for managing and distributing digital contents over the Internet as copyright protection is much sought by the digital content industry for its progressive development [29]. DRM technologies not only entail observance of the prescribed rules and policies, but also require application of the cryptographic techniques. Software tamper-resistance methods, like

obfuscation techniques, attempt to restrain unsolicited uses of software [11] by making the code more and more difficult to analyze and comprehend. Legitimate reasons for a program to look for and stop the debugging environment are merely for supporting anti piracy and authorize use of software licenses [9]. DRM technologies consist of a set of protocols and system architectural designs to provide solutions to the software piracy issues. To this end, DRM also require necessary legislative support from the governments to protect intellectual property rights of the researchers and software industry.

The main focus of this paper is to study the latest trends in the software copy prevention techniques. A survey of software protection techniques along with their strengths and areas of application is provided in this paper. The software protection techniques discussed in the literature are also critically evaluated to determine gaps that still remain to be addressed. The gaps, thus identified, may serve as future research directions. This paper is structured into six sections. An introduction to the software protection technique in this section is followed by an insight into reverse engineering process. A brief description of various hardware and software based protection techniques in provided in section 3. The next section summarizes literature review of the current software and hardware based techniques devised over the time to protect digital media and software from piracy and tempering. A comparative and critical analysis of the contemporary protection techniques is enunciated in section 5 and finally we conclude in the last section.

2. Insight into Reversing Process

The main aim of performing reverse engineering is to search for loopholes or security breaches in the software either to exploit or steal the main logic or algorithm behind the functionality of the software [2]. Reverse engineering is the systematic process of analyzing a piece of software code to identify different software objects/components and their interaction with each other to comprehend an abstract view of the functionality of the software system. Such abstract representations of the working mechanism of the software then help either to modify the existing software or recreate a duplicate version of the cracked software to fulfill certain mercantile interests by the software companies or vendors. The associated technologies with antidebugging include exception handling, junk code, checksum, hardware registers and process hiding [12].

There are a number of open source and commercial tools that can be used to conduct software reverse engineering process. These tools are generally categorized as de-compilers, disassembler, debuggers, un-packers, Program Executables (PE) editors, hex editors and deobfuscators etc., code obfuscation is a promising technique to protect software from being reverse engineered both in static and dynamic analyses. Code obfuscation is achieved through altering programs in such a way that its original functionality remains intact but its readability and internal interaction of different components become very difficult to comprehend during the reverse engineering analyses. This technique also can be used to create code at the runtime, for instance the self modifying code, thus constraining the malware attacks.

Limiting the code and data leakages within the program is an important aspect to ensure security of the software as most of the attacks launched by the attackers and crackers pertain to exploiting buffer, stack and heap overflow vulnerabilities.

2.1. Categorizing Reverse Engineering Tools

The process of reverse engineering necessitates a range of tools to gather, extract, organize and classify contents of an executable. For Microsoft Windows environment, these tools are generally categorised into four groups [3]: Hex editors, disassemble/debugger, de-compilers and de-obfuscators, PE editors, memory dumpers and unpackers. A brief description of these reverse engineering tools is discussed below.

2.1.1. Hex Editors

Hex editors are used to view and edit binaries (DLLs, EXEs) in hexadecimal format. A user can easily change a string or even an instruction of an executable using a basic hex editor. Some hex editors (e.g., HHD and Ultra Edit) offer file comparison utility. But, if search facility is not available in a hex editor then a debugger or a disassembler is normally used to locate the position of the instruction in the binary file which is required to be modified. Nonetheless, some dissasemblers e.g., OllyDbg, also support the basic features of a hex editors. Advanced hex editors like WinHex, Hackman and Hex Workshop have the ability to edit the memory, view and manipulate physical and logical drives and carry out hash calculations.

2.1.2. Disassemblers/Debuggers

A disassembler converts binary/executable code into assembly language code. Some disassemblers also present heuristic examination of the disassembled code e.g., locating iterations of loops, function calls and data structures used in the program. Debuggers augment disassemblers functionality by providing views of the current state of stacks and registers. Advanced debuggers also allow setting breakpoints inside the assembly code for illustrating runtime state of the programs and help edit the programs. Disassemblers/debuggers are particularly of much worth to unpack software, revealing program structure, decoding password and identifying faults in a program for faults. IDA Pro is quite good at generating the disassembly, but its debugging

capabilities are merely at par with a simplistic debugger. OllyDbg is a better choice as it offers both the disassembly and debugging features.

2.1.3. De-Compilers

De-compilers are specific to programming languages and even specific to compilers within the same language. The purpose of decompilers is to reproduce high-level source code from a given executable file. However, if a decompiler fails to produce the source code then it generates its equivalent assembly code. The output generated by decompilers is nonetheless, vital to functionality of a program. Decompilers aim at reproducing the original source code from the executables, however, Java decompilers are much effective even if obfuscation is applied. This could be due to the reason that Java bytecode is not as complex as the assembly language code is. Decompilers that produce better human readable code for binaries have yet to become reality.

2.1.4. De-Obfuscators

Deobfuscators are designed to reverse the obfuscation applied to a source code. Obfuscators are used to make the readability of the source code difficult to comprehend and they can also operate on binary files. There are a number of deobfuscators that attempt to regenerate the original source code by distilling the obfuscator's effects on the source code. PE editors pull out headers of the binary files and allow changes to the headers to remove any hidden/secret code. Programs that are designed to alter themselves in the memory can be debugged through memory dumpers. An unpacker could be used to restore the original code and are generally a good tool to permeate commercial protection schemes.

3. Software Protection Techniques

The term software protection means to safeguard the contents of application programs from unauthorized use and illegal distribution. There are two broader categories of software protection techniques; hardware based and software based solutions. The hardware based protections are limited to attaching dongle or smart cards with the computer system in order to run a software application. The software based protections are generally available in the form of encryption, license file protection, anti reverse engineering methods and watermarking etc. In addition, the hybrid solutions can also be used like MAC binding or disk serial registration.

3.1. Hardware-Based Protection Techniques

The current trend in software protection techniques is the current trend in software protection techniques is to employ hardware based protection in order to attain a higher degree of copy protection [10]. Hardware based protection involves a tamper resistance trusted processor that constantly scans and substantiates every piece of source code that seeks access to execute. A dongle that usually plug into either a USB or serial port is a hardware based software protection technique that is specifically designed to make sure that only authorized users can use licensed software applications. Normally, dongles are used with expensive applications. Such applications check the presence of dongle on the ports whenever they are start up. Presently, dongle is one of the best reliable techniques to safeguard commercial software from piracy.

Hardware based software protection techniques usually entail a trusted processor that substantiates every piece of software code that seeks execution. Trusted processor maintains catalogue of the keys needed for digital signature verification and decrypting the license files. This essentially means that the same software will be encrypted differently by each processor by virtue of having unique encryption key. Besides, all the data traffic transmitted over the network is also encrypted. Trusted processor based techniques are quite useful against piracy as the specific hardware is required to run the software. Associating or binding software to a particular machine is another alternative as it discreetly sends its ascribed registration serial number to the relevant software company to ensure that the pirated copy of the software is not being used. Smart cards, digital memory card and dongles, which are generally categorized as portable hardware security devices, are also used to link execution of commercial software subject to connecting them with the computing devices. Hardware-based protection techniques are quite efficient, but additional hardware costs as well as hardware and software versioning dependencies constrict their expediency and widespread usage.

3.2. Software-Based Protection Techniques

Software protection, sometimes called copy protection, necessitates employing requisite safeguards against reverse engineering or tempering of software applications. Password or key check is the simplest and the most commonly used software protection mechanism. It is generally applied at the time of software installation and it is also a popular mechanism in the shareware and the software products launched by the Microsoft.

3.2.1. Multi-Block Hashing Scheme

Multi block hashing schemes employ partitioning an executable into many blocks of independent sizes separated in a way that each block contains set of instruction that pertain to a specific functionally. The instruction blocks are ordinarily stored in encrypted form containing hash key corresponding to the next block; thus making a chain of the entire executable program. A program controller that contains the

decryption procedure is stored at the end of the executable to decipher the blocks. However, the first block, known as the entrance block, is ordinarily not encrypted and whenever a program executes, it passes the corresponding hash value to program controller to successively decode the subsequent blocks. Since, hash values are calculated dynamically, therefore, static reversing of the program becomes nearly impossible.

3.2.2. Cryptography

Cryptography is used to cipher information by using a key. Cryptographic protection techniques entail storing software in encrypted form on digital media which is decrypted just prior to execution. Sometimes, multiple encrypted keys e.g., encrypting DES key with RSA private key are used to further strengthen software protection. For foolproof security, some researchers suggest to burn the decryption key within the machine at the time of manufacturing. Live monitoring of the memory to discover the decryption key could be the only possible attack in such a situation. Min *et al*. [22] address the issue of data security in terms of integrity, availability and confidentiality of data stored on the enterprise networks by using MD5 and AES encryption based machine codes.

3.2.3. Emulation Based Software Protection

Design and implementation errors in the applications (e.g., input driven format strings, stack overflows, integer overflows and buffer overflows etc.,) lead to software exploitation [17]. Emulation based software protection techniques have been proposed in the literature that suggests running the applications in a Sandbox; e.g., Sandboxie [25] is a utility that offers code execution in a protected sandbox layer in the memory.

3.2.4. Modular Approach Employing UMLsec

Security design is generally supported through UMLsec provided in the form of a Unified Modelling Language (UML) extension profile. Constraints associated with UMLsec are used to chalk out criteria for secure data handling and data communication. An attacker model in the form of threats that it poses to the system is ordinarily defined in UMLsec. Therefore, verification routines are required to be defined to verify UMLsec models [15]. UMLsec integrates security related information in UML specifications. For security critical systems, security relevant information is embedded within the system specification diagrams that are primarily based on the notation of the UML [14]. There are tools available that generate code from the Role Based Access Control (RBAC) properties defined through UMLsec [23]. UMLsec is also known as a modular approach and is primarily used to comply with software's authentication, integration and supplementary security requirements.

3.2.5. Code Mutation Scheme

Code mutation techniques scramble the set of instructions at the time of obfuscation and toggle them back into original instructions at runtime. After execution, the instructions are scrambled again. Software mutation makes code inexplicable by adding synthetic data. Likewise, obfuscation makes the job of a reverse too hard to extract semantic information from the code.

3.2.6. Software Aging Techniques

The release of periodic updates of software compatible with the older versions is sometimes known as software aging technique. Such software updates incorporate bug fixes, hot fixes and new features by preserving software synchronization with the earlier versions. Nevertheless, this technique is more beneficial for those applications which are immensely document centric e.g., MS Word that heavily relies on particular formatting.

3.2.7. Protected Access Techniques

Protected access techniques split the software into multiple blocks which are surrounded by the security controller. Security controller filters the access request by corroborating the secure accessing history of the controls and grant access to the software if it is originated from a trusted channel.

3.2.8. Secure Naming Techniques

The secure naming techniques are used to protect software from reversing, piracy and unauthorized alteration. The secure naming techniques can be applied on both functions and files. In secure function naming method, a function is either registered or is assigned a nickname. System allows execution of only the registered functions maintained in its lookup table. In case a nickname of a function is used, then the system replaces the nickname with the original name whenever a function seeks execution. The secure file naming technique is generally used in the web technologies by barring direct access to the files. Files are labelled in such a way that they are only accessible through a function or program and their direct access through a web interface always fails. It is also possible to embed a variable name into a function or file's name so that the system generates the real function of file's name on the fly. This feature increases the piracy and decryption costs so high that it makes it virtually impracticable.

3.2.9. Software Guards

A software guard is a small piece of code segment that applies checksums on executable binaries to find whether software has been altered or not. Guards are normally placed into the software at different places [21]. In Java bytecode, guards are mostly used in the

program segments including loops. Each loop has a guard associated with it and it is evaluated just before the execution of loop. Guard's value is strictly linked to the coding logic and accordingly alters whenever the program is modified. In case the code gets modified, then program counter also changes which is ultimately detected by the guard. Software guards are usually placed inside the code. Chang and Atallah [5] propose a technique that encompasses software guards specifically programmed to perform arbitrary tasks like code checksum segments to verify integrity of software. Guards also help make software tamper resistant by performing boundary checking. Some software guards also have the limited capabilities of repairing the code, for instance, if a guard detects that certain code segment has become faulty then it automatically downloads/installs fresh copy of the code.

3.2.10. Watermarking

Obfuscation and watermarking are interconnected software protection techniques [30]. Software watermarking is a reactive approach to protect software from piracy and ensure copyright protection for commercial software [26, 28]. Watermarking is the process of embedding a distinctive hidden text that generally pertains to ascertain the ownership of the software into the software code. The owner or copyright holder of the software can later on extract this secret message hidden inside the software to obtain an evidence of piracy and unauthorized use of the software. Watermarking technique was originally devised for specific digital media contents such as video, images and audio files but there has been growing interest in applying watermarking in non media contents such as relational databases and natural language text/document file [1] which protects software through inserting hidden information into software as an identifier of the ownership of copyright for the software. Software fingerprinting is another technique that implants a distinctive user's identification number into each and every copy of the software to track illegal use of software licensees [21]. Collberge and Thomborson [6] suggest that watermarking techniques should possess two properties; stealth making it difficult to discover the watermark and resilience-restraining efforts made by the crackers to remove watermarks.

4. Literature Review

One of the possible solution for software piracy is to use tamperproof hardware tokens that mainly depend on two premises; firstly, ensuring physical security of the hardware based temper resistant device and secondly, by introducing complexity in the software code that makes it hard to analyze by dodging the attempts made by the attackers while looking for presence of the token.

Sasirekha and Hemalatha [26] analyze existing software protection techniques and suggest that cryptography is the more appropriate approach in this regard. The more beneficial technique could be to use code dependencies within the employed cryptographic technique so that the software code can be decrypted and verified at the runtime. The benefit of the proposed technique is that if the code is statically modified, then it would result in producing a corrupted executable indicating the signs of tempering. Though this technique is useful to avert the static analysis and static tempering efforts, but does not provide solution for preventing dynamic analysis when the code becomes available in the memory in its original form.

Guoyuan *et al*. [12] provide a survey of shareware protection schemes by highlighting the need to protect shareware software from antidebugging. Since, shareware are generally free of cost software products, therefore, the only software protection that can be applied is to secure the ownership of the software by thwarting the possibilities of reverse engineering. The proposed solution, therefore, is also limited in scope and is primarily based on melting the protection solution into the development lifecycle. An alternate solution could be to use VMs for securing software. The proposed technique has a limited scope as it only addresses the anti reversing methodology for sharewares.

Jamkhedkar and Heileman [13] studied problems associated with the existing DRM technologies and proposed an open layered framework, that incorporates various interoperating technologies, for developing DRM systems. Rights Expression Languages (RELs) design principles are also studied as part of developing the open layered framework as refactoring RELs is vital to attain a fair degree of DRM interoperability. In this regard, middle ware services for DRM that outline specific tasks and area of operation of the actual DRM system need to be an integrated part of a DRM framework. The strength of the proposed framework is that it ensures a strong mechanism for security of the digital contents but such a system would be too complex as for each middleware service of framework may necessitate implementing different types of security controls and business logic.

Zhang [29] report a survey of the state of the art of DRM systems and suggests employing effective usage control technologies in DRM systems to facilitate user to access, download, transfer and share protected or copyrighted contents. In this perspective, a holistic view of the existing usage control mechanisms and models that take into account RELs, authentication and authorization management security models and secure utilization of end user digital devices. DRM systems should also maintain necessary mechanism to trace sharing of rights among end users as it is particularly much desirable for social networking system.

Maña and Pimentel [19] proposed a software protection scheme based on tamperproof processor by exploiting smart card technology. The proposed technique is based on asymmetric cryptosystem in which private key is embedded on the card. The messages are encrypted with public key and the card that contains the matching private key can only decrypt those messages. The methodology is based on generating unique certificates for each user and requires being burn onto the smart card. The merit of the presented technique is it robustness against attacks as it can bypass code substitution and threats to license management protocols. The core limitation of the scheme is that asymmetric cryptosystem is highly computationally expensive that results in performance degradation. Therefore, it is not an ideal solution as it is imperative to strike a balance between the security and processing speed.

Zhang [28] proposes a software watermarking technique that employs hash function which contains watermark signature into it. Hash function extracts the embedded watermark at the run time through the predefined parameters. The hash function is calculated through manipulating certain programmatic constants defined within the program and any alteration/ tampering with the values of constants would lead to erratic behavior indicating the signs of software tampering. The main distinguishing feature of the proposed hashing technique from its counterparts is that it calculates watermark dynamically.

Ghosh *et al*. [11] present a software tamper resistant approach that employs obfuscation in the forms of encryption and checksum guards through process level virtualization. The idea is to build software application in a way that it only runs in Virtual Machine (VM) environment where the Just In Time (JIT) compiler performs the necessary decryption and executes the code. The decrypted code is periodically discarded to avert attempts to analyze the application code or taking snapshot of its memory dump. Despite the proposed technique carries certain advantages of protecting the software from unauthorized use, but the periodic discarding of the code from the memory results in decrypting the original code again and again for a single execution of the program which will definitely result in slowing down the application performance. Furthermore, the specialized VM executable would also require to be supplied with the software and no mechanism is suggested to protect the VM software.

Kimball [17] proposes two emulation based software protection techniques that are especially designed to protect software from reverse engineering. The techniques employ page granularity code signing and encrypted code execution methodologies which are executed within the trusted emulators (sandbox). An application code needs to incorporate anti debugging, anti disassembly and obfuscation methods in addition to encrypting the code. The proposed techniques though minimize the chances of reverse engineering as they run in a sandox (or an emulator), but still the encrypted code needs to be decrypted before execution.

Erlingsson *et al*. [8] propose a software guards model, named as XFI, to protect user mode and kernel mode address spaces. The XFI executes code without creating any additional software in a type safe language as well as without creating a new process. XFI supports low level architectural features (e.g., language based protection). XFI addresses the issue of running the native plug in code safely through interposition of system calls, thus isolating the un-trusted code. The proposed methodology segregates all the kernel extensions in a detached protection area to inhibit chances of faults to occur. Though such a software guard facilitates safe execution of the code, but its overheads are considerably higher as it keeps watching both the user mode and kernel mode address spaces and needs administrative privileges to execute.

Zhu *et al*. [31] provide an overview of software watermarking techniques supplemented with watermarking attack models, its taxonomy and algorithms. The four types of watermarks are identified as: Preventive, assertion, permission and affirmation marks. Prevention marks are used to restrict unauthorized software use. Assertion marks in fact symbolize a legitimate claim to the software ownership. Permission marks authorize limited changes to be made to software and affirmation marks are used to ascertain authenticity of an end user.

Lin *et al*. [18] emphasize that lack of self protection against anti debugging is the main source for encouraging reverse engineering. In this regard, benefitting hardware virtualization could serve the purpose. One possible solution could be to monitor the debug events in the higher privilege level instead of the conventional kernel space.

Dedic *et al*. [7] propose a probabilistic program transformation algorithm to make software tamper resistant by mimicking the series of steps taken by a hacker during the course of reversing a program in the form of a flow graph. By sequencing the walk of a hacker made on a program and depicting it in the shape of a graph not only provide a vivid picture of the modus operandi, but also is useful to pinpoint the possible areas of the code segments to be protected. The proposed approach entails inserting a number of tamper detection checks at various locations within the program. Each tamper check has specific scope and the predefined piece of program fragment to monitor. These checks are required to be homogeneous to detect any sign of tempering. The proposed methodology is useful to be incorporated in DRM systems. However, such algorithms may suffer from exponential or polynomial computational time complexity.

Birrer *et al*. [2] argue that static obfuscation techniques alone are not robust enough to protect the

software and suggest adding a metamorphic layer of protection in the form of program fragmentation on top of the traditional obfuscation techniques. The proposed program fragmentation technique amalgamates outlining and jump tables obfuscation that place different sections of the code into disparate locations in the memory in order to reduce the program's locality. For this purpose a jump table is maintained that links different sections of the program in accordance with its actual flow. The proposed technique adds further complexity to the already obfuscated code and makes the job of a reverse engineer more difficult as understanding and tracing the actual execution of the programs becomes too complicated.

Min *et al*. [22] address the issue of data security in terms of integrity, availability and confidentiality of data/software stored on the enterprise networks by using MD5 and AES encryption based on generating the unique machine codes. The key idea proposed is based on exploiting unique machine code (i.e., MAC address) feature. The methodology suggests generating a unique registration code for each installation of the software that includes MAC address as well as time of installation followed by MD5 encryption of the generated code. In case, the software is modified or replicated on another machine then the variation in the registration code will automatically halt the execution of the software. The proposed methodology can only work within an enterprise network where machines are interconnected and will fail if the same software is replicated on other isolated networks or standalone machines.

Temper resistant code encryption technique has been proposed by Cappaert *et al*. [4] that employs bulk encryption over the software code and applies on demand decryption as and when required. The proposed technique is an effective safeguard against both static and dynamic analyses of software code. The technique uses various chunks of the program codes to encrypt and decrypt the other segments of the code. Such code segment dependencies act as a software guard and make it very difficult to tamper the original code as tampering in certain part will eventually result in malfunctioning of the other code segment.

The software protection techniques discussed in this paper are either used individually or as a blend of multiple techniques. This section draws an abridged comparison of these techniques in terms of their area of usage and benefits/ limitations.

Product key protection is somewhat a weak method of software copy protection as it can be circumvented through key generators or associating a patch with the executable. Though multi block hashing techniques impede static decompilation and fairly lessen the chances of reversing the code, but they are not economical in terms of their computational complexity. Likewise, hardware based protection approaches eliminate the chances of replicating software through a specialized form of protection mechanism, but are pretty expensive and lack user friendliness. Digital watermarking would only be effective if there is no perceptible difference between the watermark and the original contents. Watermark embedded in the information should be inseparable so that a reverser could not remove or alter it without damaging the object. Code mutation, that makes readability of the code complex, is a promising alternative as it absolutely transforms the veneer of code without changing its functionality. Guards also help tackle software piracy issue, but their usage is relatively limited. Software aging techniques are only effective for document and data centric applications. Cryptographic techniques are still a better choice, but the software vendors need to supply decrypting routines and the decryption key with the software [21]. However, once the decryption code snippet and the decryption key are distributed, the software can easily be reverse engineered. Watermarking is generally used when it is not always possible to preclude reversing attacks. Though the protected access methodologies can restrain the chances of direct accesses to the secure data, but an intelligently designed reversing process can still dig out data during the course of data streaming; therefore, these techniques are typically suitable for open source web technologies where source code is not encrypted. Watermarking and fingerprinting are two extensions of DRM technology that help enable content monetization across several media - centric applications. Khan [16] suggests that Bayesian techniques are more promising than other conventional machine learning techniques for timeline reconstruction which can help copyright protection and authentication purposes. In general, data protection and security is categorised as a non functional requirement [27]. Masoumi and Amiri [20] proposed a digital video watermarking scheme based on scene change analysis which embeds a digital watermark into an electronic document.

5. Comparative and Critical Analysis of Software Protection Techniques

A critical review of the various software protection techniques described in the contemporary literature is provided in Table 1, see appendix.

6. Conclusions

Digital assets are under growing threat of damages and comprises. There is a pressing need to protect these assets from piracy and unauthorized use. This paper reviewed a number of software protection techniques that help eliminate chances of unauthorized access, tampering/ destruction and making illegal copies of the software applications. The prime focus of these techniques is to ensure software security against virus attacks, making the process of reverse engineering more and more difficult, cost-intensive and time consuming, baring the software piracy and making software temper-resistant. Despite the presence of several hardware and software based

protection techniques, there is still no guarantee of totally software security against the aforesaid threats as the PE and binaries can always be reverse engineered. However, we believe that a mixture of the different hardware and software based protection techniques can help further eliminate the chances of software misuse and to achieve nearly total software security. Surprisingly, legal protection means like patents, copyrights and trademarks have not been much adapted as an additional cover for software patents. We believe that legal protection means could also help further curtail software piracy and misuse issues.

As a future dimension to this research, we intend to make a watermarking based software protection technique that not only embed obfuscation but also ensure authentication from the vendor server.

References

[1] Atallah J., "A Survey of Watermarking Techniques for Non-media Digital Objects," *in Proceedings of the 3rd Australasian Information Security Workshop*, Australia, pp.73-73, 2005.

[2] Birrer D., RainesA., Baldwin O., Mullins E., and Bennington W., "Program Fragmentation as a Metamorphic Software Protection," *in Proceedings of the 3rd International Symposium on Information Assurance and Security*, Washington, USA, pp. 369-374, 2007.

[3] Canzanese J., Oyer M., Mancoridis S., and Kam M., "A Survey of Reverse Engineering Tools for the 32-bit Microsoft Windows Environment," available at: https://www.cs.drexel.edu/~spiros/ teaching/CS675/asmrceFINAL.pdf, last visited 2005.

[4] Cappaert J., Preneel B., Anckaert B., Madou M., and DeBosschere K., "Towards Tamper Resistant Code Encryption: Practice and Experience," *in Proceedings of the 4th International Conference on Information Security Practice and Experience*, Berlin, UK, pp. 86-100, 2008.

[5] Chang H. and Atallah M., "Protecting Software Code by Guards," *in Proceedings of the 1st ACM Workshop on Digital Rights Management*, London, UK, pp.160-175, 2002.

[6] Collberg C. and Thomborson C., "Watermarking Tamper-roofing and Obfuscation-tools for Software Protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 735-746, 2002.

[7] Dedic N., Jakubowski M., and Venkatesan R., "A Graph Game Model for Software Tamper Protection," *in Proceedings of the 9th International Workshop Information Hiding*, Saint Malo, France, pp. 1-15, 2007.

[8] Erlingsson U., Abadi M., Vrable M., Budiu M., and Necula C., "XFI: Software Guards for System Address Spaces," *in Proceedings of the 7th Symposium on Operating Systems Design and Implementation OSDI'06*, Seattle, USA, pp. 75-88, 2006.

[9] Gagnon M., Taylor S., and Ghosh A., "Software Protection Through Anti-debugging," *Security Privacy, IEEE*, vol. 5, no. 3, pp. 82-84, 2007.

[10] Genov E., "Designing Robust Copy Protection for Software Products," *in Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, New York, USA, 2008.

[11] Ghosh S., Hiser D., and DavidsonW., "A Secure and Robust Approach to Software Tamper Resistance," available at: http://link.springer.com/chapter/ 10.1007/ 978-3-642-16435-4_3#page-1, last visited 2010.

[12] Guoyuan L., Jiutao T., and Bing G., "A Survey of Shareware Protection Schemes," *in Proceedings of the 2nd International Conference on Signal Processing System*, Dalian, China, pp. 697-700, 2010.

[13] Jamkhedkar A. and Heileman L., "Digital Rights Management Architectures," *Computers and Electrical Engineering Archive*," vol. 35, no. 2, pp. 376-394, 2009.

[14] Jurjens J., "UMLsec: Extending UML for Secure Systems Development," *in Proceedings of the* 5th *International Conference on the Unified Modeling Language*, Berlin, UK, pp. 412-425, 2002.

[15] Jurjens J. and Shabalin P., "Automated Verification of UMLsec Models for Security Requirements," available at: http://www. verisoft.de/.rsrc/PublikationSeite/uml04.pdf, last visited 2004.

[16] KhanA., "Performance Analysis of Bayesian Networks and Neural Networks in Classification of File System Activities," *Computers and Security*, vol. 31, no. 4, pp.391-401, 2012.

[17] Kimball W., "Emulation-Based Software Protection. Black Hat DC," available at: http://www.blackhat.com/presentations/bh-dc-0 9/Kimball/BlackHat-DC-09-Kimball-Emulation-Software-Protection.pdf, last visited 2009.

[18] Lin Q., Xia M., Yu M., Yud P., Zhu M., Gao S., Qi Z., Chen K., and Guan H., "SPAD: Software Protection Through Anti-debugging using Hardware Virtualization," *in Proceedings of ACM Symposium on Applied Computing*, Taichung, Taiwan, pp. 623-624, 2011.

[19] Maña A. and Pimentel E., "An Efficient Software Protection Scheme," *in Proceedings of the 16th International Conference on Information Security: Trusted Information*, Paris, France, pp. 385-401, 2001.

[20] Masoumi M. and Amiri S., "Content Protection in Video Data Based on Robust Digital

Watermarking Resistant to Intentional and Unintentional Attacks," *the International Arab Journal of Information Technology*, vol. 11, no. 2, pp. 204-212, 2014.

[21] Memon M., Khan A., Baig A., and Shah A., "A Study of Software Protection Techniques," *Innovation and Advanced Techniques in Computer and Information Sciences and Engineering*, pp. 249-253, 2007.

[22] Min Z., Qiong-mei L., and Cheng W., "Practices of Agile Manufacturing Enterprise Data Security and Software Protection," *in Proceedings of the 2nd International Conference on Industrial Mechatronics and Automation*, Wuhan, China, pp. 318-321, 2010.

[23] Montrieux L., Jürjens J., Yu Y., Haley B., Schobbens Y., and Toussaint H., "Tool Support for Code Generation from a UMLsec Property," *in Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, New York, USA, pp. 357-358, 2010.

[24] Mouratidis H., Jurjens J., and Fox J., "Towards a Comprehensive Framework for Secure System Development," available at: https://www-secse.cs.tu-dortmund.de/jj/ publications/ papers/ caise06.pdf, last visited 2006.

[25] Sandboxie., available at: www.sandboxie.com, last visited 2012.

[26] Sasirekha N. and Hemalatha M., "A Survey on Software Protection Techniques Against Various Attacks," *Global Journal of Computer Science and Technology*, vol. 12, no. 1, pp. 53-58, 2012.

[27] Umar M. and Khan A., "A Framework to Separate Non-functional Requirements for System Maintainability," *Kuwait Journal of Science and Engineering*, vol. 39, no. 1B, pp. 211-231. 2012.

[28] Zhang X., He F., and Zuo W "Hash Function Based Software Watermarking," *in Proceedings of the Advanced Software Engineering and its Applications*, Hainan Island, pp. 95-98, 2008.

[29] Zhang Z., "Digital Rights Management Ecosystem and its Usage Controls: A Survey," *International Journal of Digital Content Technology and its Applications*, vol. 5, no. 3, pp. 247-263, 2011.

[30] Zhu F., "Concepts and Techniques in Software Watermarking and Obfuscation," *PhD Thesis*, University of Auckland, New Zealand, 2007.

[31] Zhu F., Thomborson C., and Wang Y., "A Survey of Software Watermarking," *in Proceedings of IEEE International Conference on Intelligence and Security Informatics*, Berlin, UK, pp. 454-458, 2005.

Muhammad Khan obtained DPhil degree in computer system engineering from the University of Suusex, UK. His research interests include software engineering, cyber administration, information security policies, digital forensic analysis and machine learning techniques.

Muhammad Akram is presently pursuing for his MS degree in computing at Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, Pakistan. He is working in the field of information technology for a long time. His research interests are in the fields of software engineering, information security policies and embedded architectures.

Naveed Riaz received PhD degree from Graz University of Technology 2008 in software engineering and an MS degree in software engineering 2005 from National University of Sciences and Technology, Pakistan. His research interests include model-based and qualitative reasoning, theoretical computer sciences, distributed computing and digital image processing.

Appendix

Table 1. Critical analysis of software protection techniques.

Ref #	Technique/ Methodology	Strengths	Scope/Limitation
Sasirekha and Hemalatha [26]	Cryptography based software protection technique.	Thwarts static analysis and static tempering of the program.	Is not effective against dynamic analysis when the original code becomes available in the memory.
Guoyuan *et al.* [12]	Shareware protection schemes to protect software from anti-debugging by thwarting the possibilities of reverse engineering.	-	The scope is limited as it is based on melting the protection solution into the development lifecycle and it only addresses the anti-reversing methodology for sharewares.
Jamkhedkar and Heileman [13]	Open layered framework that incorporates various interoperating technologies, for developing DRM systems.	It ensures a strong mechanism for security of the digital contents.	It is a complex solution as for each middleware service of framework it requires implementing different types of security controls and business logic.
Zhang [29]	Suggests employing effective usage control technologies in DRM systems to facilitate user to access, download, transfer and share protected or copyrighted contents.	It employs usage control mechanisms and models that allows RELs, authentication and authorization management security models and secure utilization of end-user digital devices.	-
Maña and Pimentel [19]	Software protection scheme based on tamperproof processor by exploiting smart card technology. It is based on asymmetric cryptosystem in which private key is embedded on the smart card.	It is a robust technique against attacks as it can bypass code substitution and threats to license management protocols.	The limitation of the scheme is that asymmetric cryptosystem is computationally expensive which results in performance degradation.
Zhang [28]	Watermarking technique which employs hash function that contains watermark signature. Hash function extracts the embedded watermark at the run-time.	The strength of this technique is that watermark is calculated dynamically through hash function.	-
Ghosh *et al.* [11]	Employs obfuscation in the forms of encryption and checksum guards through process-level virtualization.	It is an effective technique for protecting the software from unauthorized use with enhanced security.	Scope of the proposed technique is limited as it requires periodic discarding of the code from the memory which results in decrypting the original code again and again for a single execution resulting in performance degradation. The specialized VM executable is also limitation for each software. It also doesn't suggest mechanism to protect the VM software itself.
Kimball [17]	Emulation-based software protection techniques to protect software from reverse engineering by page-granularity code signing and encrypted code execution within emulators (sandbox).	It minimizes the chances of reverse engineering as it requires sandbox or an emulator to execute the software.	Is not efficient because the encrypted code needs to be decrypted before execution which will cause performance degradation.
Erlingsson *et al.* [8]	A software guards model named XFI is proposed to protect user-mode and kernel-mode address spaces.	XFI supports low-level architectural features (e.g., language-based protection) which facilitates safe execution of the code.	The proposed technique has overhead of watching both the user-mode and kernel-mode address spaces with administrative privileges.
Zhu *et al.* [31]	Software Watermarking techniques supplemented with watermarking attack models, its taxonomy and algorithms.	Four types of watermark models ensure security, ownership, user authenticity and unauthorized uses of software.	-
Lin *et al.* [18]	Suggests Hardware virtualization for self-protection against anti-debugging/reverse engineering.	-	-
Dedic *et al.* [7]	Program-transformation algorithm by simulating hacker's steps to reversing a program in the form of a flow graph.	This proposed technique is useful for DRM systems.	The program-transformation algorithm may suffer from exponential or polynomial time complexity which limits scope of the proposed technique.
Birrer *et al.* [2]	Suggests adding a metamorphic layer of protection in the form of program fragmentation on top of the traditional obfuscation techniques.	The proposed technique adds further complexity to the already obfuscated code making reverse engineering more difficult.	-
Min *et .al.* [22]	Methodology uses encryption of the MAC address and generates a unique registration code for each installation of the software.	-	The proposed methodology can only work within an enterprise network where machines are interconnected, and will fail if the same software is replicated on other isolated networks or standalone machines.
Cappaert *et al.* [4]	The technique employs various chunks of the program codes to encrypt and decrypt the other segments of the code.	The proposed technique is an effective safeguard against both static and dynamic analyses of software code as it employs code dependencies.	-