WHLK: Framework for Software Authentication and Protection

Nishant Gupta¹, Shubhnandan S. Jamwal², Devanand Padha³
¹²Department of Computer Science & IT, University of Jammu
³Department of Computer Science & IT, Central University of Jammu
¹{nis.decent@gmail.com, 9906077855} ²{jamwalsnj@gmail.com, 9419160600}, ³{padha@rediffmail.com, 9419103468}

ABSTRACT
In the last decade, numbers of algorithms were developed to control the piracy. In most of the existing algorithms and software registration schemes, the identity of users and software is not taken into account. After identifying the fundamental weakness of existing piracy control measures, this paper proposes the implementation and testing of WHLK Model [1] in which we introduced an integration of Software Watermarking, Hardware Parameters and License Key that can greatly reduce unauthorized use of software. Unlike traditional static piracy prevention techniques, this new technique would embed the user identity in the software using dynamic watermarking algorithm. The hardware characteristics, which are extracted, and License Key of the software are integrated and then randomizing to generate a unique key dynamically during the software installation on the client machine. In this paper a novel approach is presented which secures the software from being pirated and makes it possible for genuine client to use the same. The proposed model is implemented and tested on 198 machines and the model works with an accuracy of 99.22%.

Keywords: Software security, dynamic software watermarking, control measures of software piracy, harddisk loading, types of software piracy.

African Journal of Computing & ICT Reference Format:

1. INTRODUCTION
In current research, we introduce a low-cost software protection scheme that is secure, flexible and convenient for users. The scheme is based on Software Watermarking [2], Hardware parameters and License Key to avoid the most common attacks to software protection mechanisms such as multiple installations from a single legal license, reverse engineering analysis, and production of unprotected (pirated) copies of the software. We showed that how this new technique can be applied to build secured applications.

The organization of this paper is as follows. In the next section, we have explained the software piracy and the need for the protection of the software. Section 3 elaborates related work in the fields of software protection and watermarking to position our contribution. The Architecture of our Model has been graphically presented in Section 4. Section 5 shows the benefits and improvements of the proposed model, which is followed by the working description of same. The testing and implementation of this model is presented in Section 6. In Section 7, the empirical evaluation of the tested model has been shown. Section 8 analyses the results obtained after empirical tests, and, finally, Section 9 summarizes the conclusions and prospects for future work.

2. SOFTWARE PIRACY
There is a new generation of distributed applications, such as distributed object systems, web services, electronic commerce and grid computing, that have the potential to represent substantial advances in the use of Internet resources. However, security problems become an insurmountable barrier for the widespread deployment of those types of applications. Solving software protection problems is not a trivial task. Several issues of research concerning the piracy of software are:

- Intellectual property protection: The objective is to link the software with information about its author by using techniques such as watermarking [3].
- Protection against function analysis: The objective here is to prevent a malicious host from discovering what function is being computed by a software element. Techniques such as code obfuscation [4] or function hiding [5] are used, sometimes complemented by the use of hardware tokens [6].
- Software use-control: This is aimed at guaranteeing that only authorized users can run the software according to some contractual conditions.
The work presented in this paper focuses on all the above mentioned issues. We also discuss the possibilities and implications of protecting software in order to prevent software piracy, and ensure that the software is installed for an intended user only. Software piracy is the unauthorized utilization, replication, and distribution of commercially available or copyrighted software. A report by the Business Software Alliance (BSA, 2012) noted that for the past five years worldwide, software piracy has hovered between 38% - 42% and losses amounted to billions of dollars [7]. The prevalence and significant growth of software piracy in the past four decades since the advent of computer technology has warranted attention of the researchers, software developers, educators, intellectual property advocates etc. Therefore, the fight against software piracy is becoming an extremely important issue for all these stakeholders.

Surprisingly, the best results against software piracy in recent years have not come only from new software protection mechanisms but the other reasons for controlling the piracy are:

- Software companies have tried to have an effective legal sales presence in all areas of the world.
- Software companies achieved better user support for their legal products.
- The price of software has come down, narrowing the difference between legal and illegal versions.
- Some organizations, like the BSA, promoted high profile legal proceedings against companies using illegal software.
- Governments have cooperated to provide legal protection for intellectual property and to criminalize software piracy.

We are again witnessing an increase in software piracy because of the popularity of the internet. Despite the use of deterrent and preventive measures to combat software piracy this problem seems unabated and it is observed that previous measures will not achieve better results.

3. MOTIVATION AND BACKGROUND

In today’s computing world, maintaining security is a challenging issue. Customers are expecting security to be delivered out of the box, even on programs that were not designed with security in mind. The challenge is even greater when legacy systems must be adapted to networked/web environments, while they are not originally designed to fit into such high-risk environments. Tools and guidelines have been available for developers for a few years already, but their practical adoption is limited so far. Nowadays, software maintainers must face the challenge to improve program security and are often under-equipped to do so.

Addressing issues related to software piracy is an on-going challenge. While software companies and lawmakers have developed and attempted corrective measures to fight the trend of piracy through security design patterns, birthmark, protection schemes, application virtualization, exterior component, secure coding, copy-protected software, product identification key, hardware dongle, the problem of piracy remains prevalent and significant.

As for security engineering, it aims at considering security early into the development life cycle of software. A number of Security Design Patterns (SDP) is available in order to guide software engineers in designing their security models and securing their applications at the design phase. When it comes to security hardening against piracy, these proposed security design patterns are not really relevant. The reason is that we are dealing with already developed applications that are, in many cases, deployed.

Spyros T. Halkidis et.al [8] tried to achieve this goal by applying specific patterns in software architecture. Here, the well-known design patterns for building well-structured software have been defined, a new kind of patterns called security patterns have emerged. These patterns enable to incorporate a level of security at the design phase of a software system. This proposal is not relevant in the setting of security hardening against piracy since we are dealing with already developed software.

The research by Hyun-il Lim et.al [9] proposes a method of detecting the theft of Java programs through analysis of the flow paths (called as FP) of the program, specifically of the FP birthmark which can be used to identify the origin of software by comparing the inherent characteristics of the program. The proposed birthmark is a measure for detecting programs while copying, and can lessen the time and effort required for manual reverse engineering in the identification of software theft. But this method is ineffective if the basic blocks have been decomposed or if the instructions in programs are reordered or changed via modifications.

Ibrahim Kamel [10] proposed a technique that detects malicious attacks with high probability on real datasets using reasonable node sizes and attack model. This paper also presented the increase in detection rate of data alteration at the cost of the attack detection. But this scheme was based on a watermarking scheme to be used only for R-tree data structures instead of designing the watermark for other database indexes like B-trees and Quad-trees. Bin Zhao et.al [11] proposed an enhanced watermarking scheme in the encrypted domain with flexible watermarking capacity. This paper demonstrated that the enhanced watermarking scheme eliminates the drawbacks of other watermarking schemes in the encrypted domain. But there is no use of error correcting codes to correct bit errors and improve the robustness of watermark sequence against other severe attacks in the encrypted domain.

Several watermarking schemes have been proposed distinctively whether it is viable to hide the information in the program/code itself or dynamically. Xuesong Zhang et.al in his paper [12] proposed a hash function based dynamic software watermarking algorithm which is able to withstand a variety form of transformation attacks. But still there is lack of effective feature code matching and extraction feature to identify the hash function.
Rafael Augusto Teixeira et al. [13] presented a work for an on-demand software deployment system based on application virtualization concepts which eliminates the need of software installation and configuration on each computer. As many a commercial software still uses the license per installation, it is a real problem in this application streaming solution. Qiang Liu [14] used Exterior Component method which allows only authorized users to run the software. So without this exterior component software cannot be run, and thus they will protect against illegal copying and use of software. Although this method can be applied to some commercial software, it is still at high cost to apply to the personal computer users.

Various strategies have been employed to make unauthorized duplication and use of software more difficult [15]. One such approach is to provide a hardware “key” which is typically installed in the parallel port or USB of the computer to provide a software interlock. If the key is not in place, the software will not execute. This method is relatively expensive for the developer and cumbersome for the authorized user while remaining vulnerable to the theft conducting piracy by duplication of the hardware key. Another approach requires the user to enter a serial number or customer identification number during the installation of the software. Missing or invalid registration information prevents the installation of the software. This approach is easily defeated by transferring the serial number or customer identification number to one or more unauthorized users. Yet another approach requires registering the software with the manufacturer or distributor to obtain an operational code or password necessary for the installation of the software. Again, once the operational code or password is obtained, it may be perpetually transferred along with pirated copies to numerous unauthorized users.

Concerning the secure coding approach, it presents either safe programming techniques, or a list of programming errors together with their corresponding solutions [16]. Several publications compiled common errors and vulnerabilities in code production languages such as C/C++. Their intent is to instruct software developers to avoid these errors. Such proposals are not relevant in the setting of security hardening against piracy since we are dealing with already developed software. Moreover, these secure coding practices are very often manually applied and our aim is actually to elaborate a systematic and even preferably automatic approach to security against piracy.

Yawei Zhang et al. [17] proposes a software-splitting technique which put the split contents on the client instead of the remote trust server. This new technique would encrypt the extracted contents from the software by a key relating to the hardware characteristics, and then decrypt them dynamically during the main program running. In spite of the usefulness of this technique, it is not stimulating for some explanation programming languages such as VB, JAVA because it’s nearly impossible to directly manipulate in the memory for these programming languages.

As a result, integrating security into software to prevent piracy is becoming a very challenging and interesting domain of research. Above mentioned research projects motivated to create methods and solutions to integrate systematically secure components into software. Our proposition is inspired by the best and most relevant methods and methodologies found in each one of the aforementioned concepts and approaches. This research provides secure software by hardening framework for installation and registration. The experimental results presented explore the efficiency and relevance of our approach.

4. ARCHITECTURE OF THE WHLK MODEL

The model provides an integration of the privacy of users, security of information and license key, together for the control of the piracy. Our system enjoys a modular design and can be implemented by any machine with flexible configurations and windows X operating systems. Furthermore, the proposal allows flexible registration information definition. This Model not only makes it harder to create an additional available copy based on diversity, but also prevents illegal uses on the copy. The architecture of our Model has two phases:
Client-Vendor Interaction Design (Phase I): Fig. 1
Installation and Registration Design (Phase II): Fig. 2
5. BENEFITS AND WORKING OF WHLK MODEL (IMPROVED)

5.1 Benefits of WHLK Model
The current research is carried out after collecting data from academic as well as industrial sector for finding the most common cause of piracy. Numbers of causes are discussed in the literature and most common cause of piracy found is HardDisk Loading. It is a method of software piracy where hard disk of the new customer is loaded by the vendor of the hardware with the different software available with him. It has been observed that the hard disk loading and softlifting are the common practice of making an illegal copy of the software.

We have studied different control measures for curbing the software piracy in Section 3 and no such measure has been found for doing so. We have implemented our technique on different machines with variant configurations and versions of windows X operating system which shows that it is flexible with any of the machines and is not limited to some specially configured machines.

An integrated (dual) approach is applied to secure the software from being pirated. Our Model has been designed in two phases including Client-Vendor Interaction Design (Phase I) as shown in Figure 1 and Installation and Registration Design (Phase II) as shown in Figure 2. In phase I, Dynamic Watermarking Algorithm is implemented to annotate, embed, tracer and recognize a watermark (as a component) in the software using SANDMARK, which was proposed by Collberg and Thomborson [18]. Also, the Obfuscation algorithm is applied to encrypt this watermarked file. It helps in encrypting the user identification.

An algorithm is used for encrypting the information fetched from the machine and the user in coded form. The unique keys are generated in a randomizing pattern by applying these algorithms in the Installation phase, thus, securing them from being revealed to secondary users. Registration Code (RGCN), in encrypted form, is generated randomly on the basis of unique keys generated through algorithms unlike other registration schemes where registration is done storing the information as it is on the server thereby inviting a risk of being hacked. User information is not at any risk of being misused or misinterpreted elsewhere.

A timeframe has been set using an algorithm for those software users who have either formatted their machine or purchased new hard disk or machine. The client can re-use the software on new machines using existing Registration Code (RGC) provided to them during the first installation of watermarked software. No other user installs or uses the software on their machines.

The cost of this technique is rather low as compared to others. We don’t need any hardware i.e. dongles for the implementation of secured software installation. It is rather convenient for the developers to implement this strategy.

5.2 Working of WHLK Model
Noting the weakness in existing approaches of software registration, we incorporated the new processes in the process of software registration. First, instead of preventing from illicit copying, we force the software to be only used by an authorized running environment. In order to accomplish this task, the user is required to submit his identification details which are being applied through algorithms to generate random unique key (string) and this key is embedded as a watermark in the program. Also, an automated program fetches the uniquely identifying characteristics of running environment such as the physical sequence numbers of the CPU, the main board, processor and other hardware information. These hardware characteristics and the License key are integrated to be used as parameters. All of these parameters are encrypted using a pre-defined code generating a new random unique number called as Registration Code (RGCN) which is acknowledged to the user.

The working of WHLK has been categorized in two phases:

5.2.1 Phase I
In this phase, the client is required to provide his personal details to the software vendor including his Name, Affiliation and Social Security Number / ID. This process is the first process for the authorization of software user and represented as Process P in Fig. 1. The details are entered by the user and fetched through Process P1. This Phase enters into a new Process P2. Software vendor applies an automated algorithm on these details to generate a Unique Key (UK). This Unique Key was generated on the basis of programming code extracting certain alphanumeric characters of each of these details randomly. UK is used as a parameter for Sandmark watermarking tool. The Collberg-Thomborson (CT) software watermarking algorithm is applied to embed this key (watermark) into the class files of the software program. This algorithm embeds the watermark into the topology of graph structure built at runtime in response to a sequence of special user actions.

To use this algorithm, the calls are embedded to sandmark.watermark.ct.trace.Annotator.smmark(*) into the software program. These points represent the locations where watermark code can be inserted. Then run a trace with a special input sequence in SandMark and finally watermark is embedded. The software is watermarked with the key (watermark) and called as Watermarked Software (WS). Before distributing this watermarked software WS to the Client, it is obfuscated to remove the risk of being susceptible to collusive attacks. Then WS is supplied to the client as per his requirement.
5.2.2 Phase II

During the process of installation, an automated process P6 fetches the hardware characteristics of this machine which includes the CPU Identification (ID), Hard Disk ID, MAC ID and Processor Serial Number. These Hardware parameters (HW) are being forwarded to the Hardware Database (D1) for checking their existence in the database. D1 validates the HW and responds in different ways.

If all or any of the HW already exists in the database, it means that the user is already registered for the software on the machine with same HW and that he had already received the Registered Code (RGC). In this case, a denial note is acknowledged to the user through process P7.1 asking him to provide with the existing RGC available with him. User provides the RGC and submits it to the Registration Database (D2). Installation processes completes its cycle and the software is ready on machine for use.

If user forgets RGC, he has to check E-Mail inbox for the mail received sent to him during the first installation. In case of user forgetting his E-Mail ID, he has to use the License Key given by the vendor for re-registration. In extreme case of non-availability of License key, he has to contact the vendor who can trace the RGC using the user details given to him by the user while purchasing the software.

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the

If HW does not exist in D1, it means that the user has not registered yet and a new registration process begins. An acceptance note has been sent through Process P7.2 to pass the hardware parameters to P8.1. Here the process generates a Unique Key (UK1) taking HW to be used as parameters in programming. This Unique Key was generated on the basis of hardware parameters to P8.1. Here the process generates a registered yet and a new registration process begins. An each of these hardware parameters randomly. UK1 is being sent programming code extracting certain alphanumeric characters of acceptance note has been sent through Process P7.2 to pass the
6.1.4 The next phase is Obfuscation phase, where scprog_wm.jar is being obfuscated using the Class Encrypter Obfuscating Algorithm. This algorithm encrypts the file using an encryption key and causes them to be decrypted at runtime. The output of the obfuscator is a new watermarked jar-file scprog_obf.jar. This program is served as input to the recognizer.

6.1.5 In the recognition phase, the scprog_obf.jar is traced again (using the same secret input), and the branch sequence is checked for the watermark.

6.2 Testing
Our technique has been tested on different machines with different configurations and versions of Windows X operating system to check its reliability, accuracy and integrity. Firstly the watermarked software has been installed on the client machine and then the process of validation on the server for its authentication and security are carried out.

6.2.1 Client-Side Installation
The testing of our technique starts with an executable file running a sequence of steps for installing the software on the client machine. Hardware parameters are extracted from the machine and submitted to the server for validation. This process has been done with an algorithm designed for executing these validation checks on server. In next step, the user identification and License Key, provided with the software pack, is collected, fetched, integrated and processed at different steps. An algorithm, which is designed for generating unique keys randomly, is being applied on this integrated information. Then this information is being submitted and stored at the server and registration code (encrypted code) is generated which is being acknowledged to the client. The process of installation finishes after completing all its steps and the software is ready for use. This is an automated program designed with different algorithms and program codes. This phase secures the authenticity and usage of the software.

6.2.2 Server-Testing
Server side testing algorithms are designed to test and validate the data provided by the user, generated on client-side machine during installation process.

- Algorithm for validating the hardware parameters fetched by the client machine
- Algorithm for securing the registration codes submitted by the client machines
- Algorithm for acknowledging the newly generated registration code to the e-mail of client in encrypted code securing it from being tampered with or being hacked and cracked.

7. EMPIRICAL EVALUATION

7.1 Evaluation In Phase I
We evaluated the watermarking scheme for Java bytecode described in Section 6.1 using an implementation built on top of SandMark, a collection of obfuscation and watermarking algorithm implementation for Java bytecode. The system reads in Java archives (jar-files), applies one obfuscation and watermark, and writes the resulting code to another Java archive. We used 270 jar files with CT Dynamic watermarking algorithm for our experiments. A high percentage of the instructions in these jar files are executed frequently. After all the programs have been transformed, we extracted the watermarks from the programs. We expect that many watermarks will be lost during the transformations and attempt to find the affect of transformations on watermarks.

7.1.1 The Watermarker
We are testing CT dynamic watermarking algorithm from Sandmark watermarking systems. Sandmark is an academic framework and it is the only available system for watermarking Java programs.

7.1.1.1 Sandmark
SandMark is a tool [19] developed by Christian Collberg et al. at the University of Arizona for research into software watermarking, tamper-proofing, and code obfuscation of Java bytecode. The project is open-source and both binaries and source-code can be downloaded from the SandMark homepage [19]. We used version 3.4.0 (Mystique) released in 2004.

7.1.2 The Jar Files
All the jar files that we use in the tests are plugins for the text editor jEdit [20]. These files are fairly small (average 40KB) but represent a collection of real-world Java software. The range of plugins represents a variety of code, and were all written by different programmers but as they are plugins they share some characteristics. For example, some classes may subclass jEdit’s abstract plugin classes to use jEdit’s plugin API. All the test files were obtained by installing jEdit and then using the built-in plugin manager to download the plugin jar files. The average number of classes per jar is 10, while the average number of methods per jar is 77. The average number of fields is 25 and the average number of local variables is 175. The biggest program jar was 742 KB while the smallest was 1.7KB. The largest program jar had 171 classes and the smallest had only 1. Two programs had no fields while the largest program contained 523. The largest program contained 3004 local variables.
7.2 Evaluation in Phase II
We evaluated the installation process of the software dynamically as described in Section 6.2 through the empirical analysis of the algorithms for validation checks on server, generating the unique keys, concatenating the unique keys generated and, lastly, generating the Registration code. We have installed the software on 198 machines with different configurations, versions of windows X operating system and platforms.

7.2.1 Validating Hardware Information
We have tested Validate algorithm for checking whether the hardware parameters fetched exists in the database or not. In some machines, they exist on the database, the instructions have been sent to the client to enter the already provided registration code and submit to the server for Re-registration. The remaining machines complete the rest of installation process.

7.2.2 Generating the Unique Keys
We have tested Gen algorithm for generating the unique keys fetching the characters position wise randomly as programmed in the source code. Hardware information has been fetched and an algorithm has been tested on this information generating a unique key (random number) called as UK1. UK1 has been submitted and stored on SQL Server Key Generation Database D3. Then the algorithm has been tested for E-mail verification. Another unique key UK2 has been generated and stored on D3. Also, the E-Mail ID (in original form) has been sent and stored on Registration Database D2. Lastly, the License Key has been fetched from the user (provided by the software vendor). Further, the algorithm has been tested on this information and a unique key UK3 has been generated which is submitted to the database D3.

7.2.3 Generating the Registration Code
We have tested Con algorithm for concatenating these three unique keys UK1, UK2 and UK3. After this, we have applied the Gen algorithm on this new parameter derived after using the Con algorithm. A new registration Code called as RGCN has been generated and stored on the database D2.

7.2.4 Acknowledging RGCN
We have tested the Sec algorithm on RGCN to be encrypted and then this encrypted RGCN has been acknowledged to the E-Mail ID of the user. After testing all these steps, we have successfully installed software on the machine for a genuine user in a secured environment. All tests were run using Sun’s JVM version 1.4.0 and Windows X operating systems.

8. RESULTS

8.1 Observed Results of Evaluation Phase I
After embedding watermarks we obtained 259 out of an expected 270 watermarked jars. 11 watermarked jar files failed to embed the specified watermark, due to error or incompatible program jar. The Collberg-Thomborson (CT) algorithm managed to correctly embed watermarks in all 270 test programs - they were embedded, obfuscated and recognized correctly. Exactly 95.92% of the expected watermarked jar files were actually produced (Fig. 3). Out of the 259 watermarked jar files all 257 contained watermarks which were successfully obfuscated and recognized. 2 out of 259 watermarked jar files could not be recognized as the input key is not available with the client. This means that 99.22% of the watermarks in the watermarked jar files produced were actually recognized (Fig. 4).

8.2 Observed Results of Evaluation Phase II
After testing Validate algorithm on 198 machines, we found that 38 machines have been already registered for this particular software which means that hardware parameters of 38 machines were found on the server. Instructions were sent to these 38 machines to re-submit their registration code already available with them. Statistically, 19.19% clients were found already registered for using this software (Fig. 5). When the rest of installation process was being run through the 160 machines, we found that 6 clients have forgotten the details of their E-Mail IDs and 13 clients have lost their License Key. These 19 (adding 13 and 6) clients were not able to complete their installation process. Only 11.87% of clients were not successful in their installation process. A high percentage of 88.12% of clients were able to receive RGCN and install the software on their respective machines (Fig. 6).
9. CONCLUSION AND FUTURE SCOPE

9.1 Conclusion
In this paper we present an approach that illustrates our proposition and methods to harden security into applications. This approach, which is based on Software Watermarking, simplifies security hardening by maintainers and allows developers to perform security hardening of software by providing an abstraction over the actions required to improve the security of programs. We have taken this approach as a measure to control the software piracy which is very commonly prevalent in our technological world. Our model consists of dual aspects. First aspect, occurs at the software vendor side, includes the implementation of dynamic software watermarking in the software program. WHLK offers easy embedding of watermark without affecting the data and control structure of original program and also shield the watermarked program from collusive attacks using obfuscation algorithm.

Second aspect, which occurs at the client machine, fetches the information, processes them using encryption algorithms designed and then submits them to the server for validation. Server validates and puts acceptance to the client by registering its details and acknowledges it to him for software use. We conclude that our watermarking system would mitigate and nullify the threat of unauthorized distributions of copyright software by its secondary users.
9.2 Future Scope

In future, we would like to focus on the following work. Our Model, in its second phase, needs Internet connectivity to complete its process which can be constraint. The techniques can be developed to remove this constraint of Internet. Our scheme works for only HardDisk Loading type of software piracy. We would like to extend the scheme for other types of software piracy like OEM Unbundling, Softlifting, and Corporate Software Piracy. We would like to apply and extend our scheme to other platforms like .NET framework and to carry out experimentation on these machines.

REFERENCES


