Rethinking Software Piracy: Active Software Rights Verification for Effective Control of Piracy

Steven Chan, Buddhika Kottahachchi, Samitha Samaranayake

ABSTRACT

We present here a novel approach to controlling software piracy. In our proposed framework, software is distributed freely but cannot be executed unless the consumer purchases rights to use it. *Right of use* is linked to a token, such as a Drivers License with an embedded smartcard, that the user would be highly unwilling to share. We claim that this framework is far more promising than current solutions at preventing software piracy.

1. INTRODUCTION

While software piracy is frowned upon in Western society, it is more the norm in other parts of the world. For example, in most parts of Asia you can obtain bootlegged copies of almost any software package for just a few dollars. Even in Western society, the emergence of peer-to-peer file sharing systems has encouraged greater participation in piracy. The Business Software Alliance estimates that this cost the software industry US$ 11.75 Billion in the year 2000 [1].

Given the gravity of the problem, much work has been done in the area, and various ideas to circumvent piracy have been implemented. However, current attempts are not very effective against determined adversaries and inconvenience legitimate users at best. For example, mechanisms such as hardware dongles are a hassle to use and copy-protection schemes prevent users from making backup copies of their software. Even worse, most schemes simply do not work; determined adversaries often find creative ways around them. Furthermore, software vendors tend to be alone in their attempt to prevent piracy. Users simply have no disincentive against engaging in piracy.

We present here a novel framework that tackles this situation by completely rethinking the problem. Users do not buy software, but rather buy the right to use software. Software distribution is not restricted, but encouraged to occur freely. Software rights are not tied to the machine, instead they move with the user. Above all, it is even in the users' best interests to shy away from engaging in piracy.

This proposal is not restricted by current technological constraints. Instead, our approach extends to identify and exploit potentially useful technologies by extrapolating from current technology trends and designing a system that would be feasible in five to ten years.

2. CURRENT APPROACHES

The most common anti-piracy technique used currently requires users to activate software by entering a long product registration code. Software installation cannot be completed without the proper registration codes, which are usually provided with the software. This scheme is popular due to its simplicity and low implementation cost. However, registration codes can be copied easily, and code generators for popular applications are freely available on the Internet. There are many other variants of this scheme — all of which have similar mechanisms and hence weaknesses.

To counter this problem, vendors are moving towards software activation codes that are machine specific [2]. Windows XP, for example, creates a file that identifies the machine by storing information about its configuration. Whenever a copy is being registered, a database is checked to see whether that copy has already been installed on a different machine. This activation system inconveniences the legitimate user. Changing the hardware on your PC could change the machine ID, making the program think it is a different computer and requiring the user to register the software again. Even worse, the effectiveness of this system is questionable since loopholes in the system have already been exposed [3].

One of the more secure anti-piracy systems used today are hardware dongles. This system comprises of an external device (dongle) and software that allows communication between the dongle and the application. Each copy of the application comes with a dongle that is specific to that copy. The application only works when the dongle it was sold with is plugged in. Making backup copies of the software is allowed but only one copy can be used at a time because duplicating dongles is hard. Although this system is more secure, there are many drawbacks. Hardware dongles are expensive devices and it only makes sense to use them for expensive software. Also, each application requires a separate dongle for each application being used simultaneously. With more people using laptop computers, it would be unreasonable to expect users to carry an extra hardware device for each application on their machine. Finally, many implementations of these are known to have been circumvented in software alone.

3. DESIGN GOALS

Given the problems described above, our objective was to propose a universally applicable framework that would address the issues identified above. Therefore, we endeavored

to produce a framework that would prevent users from sharing their rights to software while maintaining convenience for legitimate users.

As described above, this is known to be a difficult problem and therefore our attempts focused on working from the ground up and rethinking the entire issue. As a result of this exercise, it was felt that a framework with the characteristics we desired would have to meet the following requirements:

1. *Software should be linked to the user and not to a machine:* Traditionally software is purchased for the exclusive use on a single machine. Instead we propose that software should be purchased by individuals who desire to use it. Therefore, they purchase the right to use this software and this right can be exercised anywhere regardless of which machine is being used at the time.
2. *Software should be freely available and distributable:* Since our framework relies on attaching *right of use* to the user, it must be possible for the actual software to be installable on any machine with no restrictions. One of the hardest problems in addressing software piracy is in effectively controlling software distribution. We will attempt to address this issue by simply redefining it and making it a non-issue in our design.
3. *Authenticate users with a unique token they would be unwilling to share:* Current attempts at using tokens to control right of use focus on various mechanisms to achieve unshareability (long serial numbers, hardware dongles etc.). However, these tokens are flawed in that either they fail completely in their task or cause unacceptable levels of inconvenience to the user.

Furthermore, it was felt that we need not restrict ourselves to use existing technology infrastructure since any potential implementation of the proposed framework would not happen in the short term. Therefore, it was decided to identify and exploit any technologies that had the potential to achieve widespread adoption in five to ten years. This would go further in helping us achieve our objective of completely re-thinking the problem.

4. PROPOSED FRAMEWORK

4.1 Software Distribution

As stated in our Design Goals, we desire to make the control of software distribution a non-issue. Therefore, we require a radical new distribution model that completely violates the current set of accepted norms with regards to software distribution. We propose that software be distributed in the following manner:

When a software product is "market-ready" it should be made freely available. We suggest that this be achieved by making the product available for download over the Internet. This concept may seem counter-intuitive initially, especially with regards to determining how a software developer could capture any of the value he creates in developing the product. However, we are not proposing that the product be given away free. Instead, we propose that the downloaded software should be designed such that it is only usable by someone who has purchased the right to do so. This on its own is not a radically new concept. Our scheme differs in that we link *right of use* to a particular individual as opposed to a machine. In this way, we encourage the creation of an environment where it is possible for a legitimate user to use software regardless of the machine being used.

However, we still need to address the issue of how one purchases and is assigned the *right of use* for a given software product. Since we propose that the software itself is obtained over the Internet, it seems a natural extension to propose that the *right of use* be also sold over the Internet. We envision this to be very similar to buying a book from Amazon.com today. However, the transaction envisioned will differ in that the item being purchased is not tangible but rather a *right of use*.

This will be achieved by having the vendor process the monetary transaction in a standard manner before he adds the purchaser to an Access Control List (ACL). The nature of the entry on the ACL is determined by the license purchased and the terms set forth by the vendor. Furthermore, this entry will contain some information that will enable the software vendor's server to authenticate the customer and his rights. This information should be obtained from a token that the user is unwilling to share with others. This is important since the product can only be used by someone who produces valid credentials at the time of execution. These credentials should be verifiable with their respective Issuing Authorities.

Good candidates for these tokens would be IDs such as Passports and Driver's Licenses. These documents could be embedded with a smartcard supporting Public Key Infrastructure (PKI), thus enabling their participation in various cryptographic authentication and encryption schemes. Since people aren't likely to share their Passports or Driver's Licenses, this makes it ideal for our system.

4.2 Infrastructure Overview

The infrastructure of our system is inspired largely by the web services model [5]. In the web services model, a thin client program is distributed to users while all vital software methods remain on the vendors' servers. Since the thin client is lacking certain methods, it carries these out by receiving input from the user, sending this data to the vendor's computers in the form of Remote Procedure Calls (RPCs), and getting back processed data to work with. All this is done transparently as though the methods were residing in the local software package– with the exception that the user is charged for server time on the vendors' computers.

With this web services model, it is obvious that software piracy is virtually impossible since the thin clients are useless on their own, and the server-side methods are impossible to obtain without carrying out some form of industrial espionage. On the other hand, this model is inefficient and does not scale well since most computation occurs on the server's machines, while all the resources on the user's machines remain largely untapped.

As a result, the system that we propose is basically web services– with a major twist. Instead of having all computational methods residing on the vendors' machines, we propose that these methods be distributed freely to users in the form of a package we term the *user-side component*. Accordingly, instead of having all the control and coordination of a program happening on the users' machines, this functionality resides on the vendors' machines in the form of a *vendor-side component*. In this way, the user-side component is still dependent on the vendor-side component in order to function; yet most of the computation still happens on the users' machines.

In order for the two components to function as one, the server side component needs to dispatch remote procedure calls (RPCs) to methods residing in the client-side component. We use this requirement as the basis of our authentication model. In our system, this communication must take place through an authenticated SSL channel. We expect that by the time our system is implemented, every person will have his own unique smartcard that supports public key infrastructure (PKI). This smartcard will be used for the client-side of the SSL authentication. When a user purchases a program, the software vendor adds the user's public key to an access control list. This access control list is checked each time the client-side component of a program requests to set up an SSL connection with the vendor's servers. At the same time, the vendor's computers also keep track of the number of instances of the program that each user has running and ensures that this number is not greater than that allowed by the user's purchase license.

4.3 Architecture

In order to provide seamless integration of the client and server sides of the program, as well as to abstract the authentication and network communication away from the actual program, we propose the architecture as shown in Figure 1. The client machines should compose of the actual client-side component of the program, a client-side mediator with which it communicates, and a smartcard reader, which is used by the client-side mediator for authentication. Accordingly, on the server machine we would have the server-side component of the program and the server-side mediator with which the server-side component communicates. The two mediators communicate with each other and effectively act as a relay between the client-side and server-side components of the program. In addition to these two modules, the server should also communicate with a database containing the access control list. This database should be updated each time a new user buys the program or whenever a user's license expires. It should also keep track of how many simultaneous sessions of the program each user is allowed to have.

When a user executes the client-side component of the program, the client-side component sends a start request to the client-side mediator. The client-side mediator then contacts the server-side mediator and establishes an SSL connection using the client's smartcard and the server's SSL certificate. Before this connection can be successfully established, the server-side mediator checks the access control list to ensure that the user has rights to the software. In addition, if the user has a larger number of running instances of the program than is allowed by his license, a message is sent to the client-side mediator which then relays this information to the user. The user is then given a choice of aborting the current attempt to start the program or of terminating a previous session. Once the SSL connection is established, the server-side mediator spawns a new server-side component process on the server. At the same time, the original start request sent by the client-side component is relayed from the client-side component to the server-side mediator, which in turn relays this request to the server-side component. Henceforth, the server-side component takes control and starts its symphony of RPCs to the client-side component via the mediators.

It is interesting to note that the client-side mediator appears to the client-side component of the program as if it were the server-side component. Similarly, the server-side mediator looks like the client-side component to the server-side component of the program. In this way, programmers can write and test the client and server side components of their code independently of the entire access control mechanism. Furthermore, should the software vendor decide to discontinue a particular product, it could merely distribute the server-side component for free so that its original paying customers are not left with a dysfunctional program in the end.

A large part of most programs is comprised of implementations of publicly known algorithms, while the real ingenuity and novelty lies in only a small portion. As a result of this, we propose that programs be divided into two portions: one portion that would be a collection of freely distributable functions or classes, and another portion that would remain the intellectual property of the software vendor. Presumably, the software developers could tune this distribution such that most of the computation and processing resides in the freely distributable components, while most of the coordination and control remains in the private component. In this way, programs could be adapted to function in our model where the client-side component does most of the processing and the server-side component performs the coordination and control.

5. ANALYSIS

5.1 Implementation Feasibility

As stated above, our scheme is, by intent, not suitable for immediate implementation. However, we believe that the key technologies we intend to use are likely to reach the required levels of both proliferation and robustness in five to ten years time. The two technologies we rely on our smartcard PKI and a ubiquitous broadband network accessible by all.

Smartcards, especially outside the US are gaining wider acceptance and popularity. The amount of smart cards being shipped each year is growing in the order of 30% and currently stands at 1.5 billion [6]. This fits in ideally with our outlook of proposing a scheme that is feasible for implementation in five to ten years. Furthermore, according to Nua.com as of August 2001 approximately 8.46% of the world population was connected to the Internet; a growth of 40% over the year.

It is also encouraging to note that countries in the South

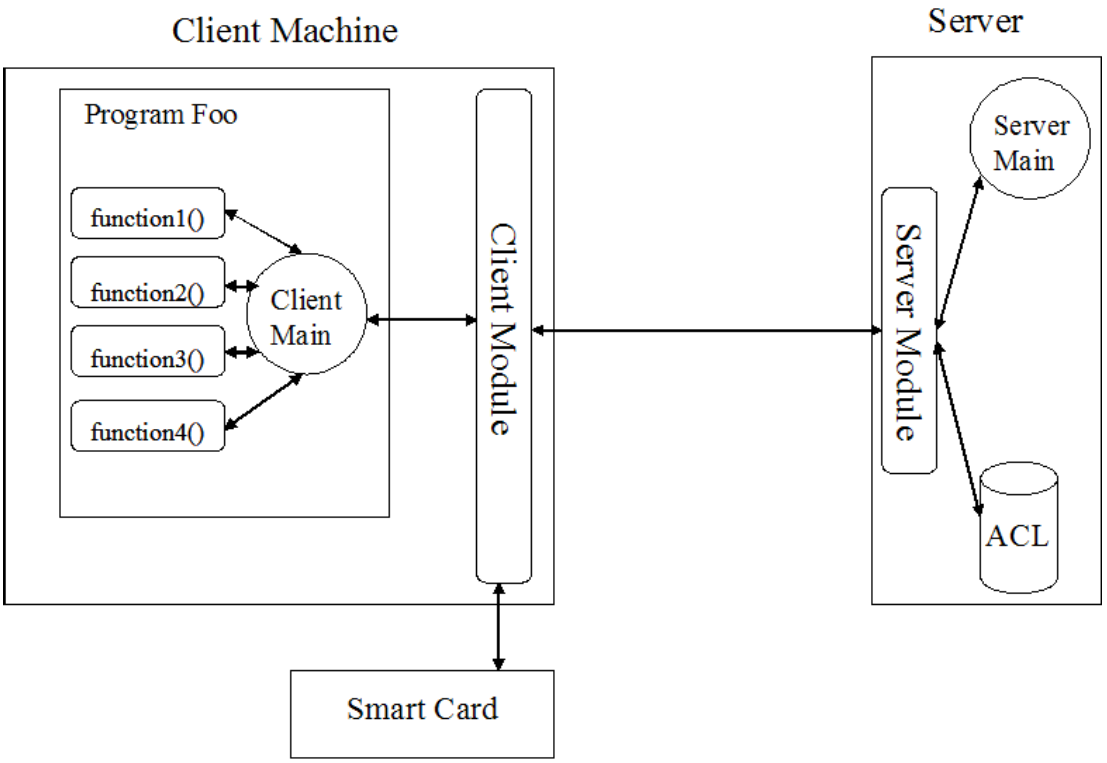

Figure 1: Architecture Overview

East Asian region, where software piracy is most significant, are adopting both technologies. For example, the government of Malaysia recently introduced a National Multipurpose Card which is intended to include the following applications: National ID, Drivers License, Immigration, Health Card, eCash and PKI. [7]

Given these trends, we are confident the scheme we propose in this paper will be feasible within five to ten years.

5.2 Security

Securing the user's rights to software and preventing it from being shared is a key goal of our system. Most traditional methods of circumventing anti-piracy schemes focus on hacking the code to bypass the anti-piracy mechanism, which is usually a single decision point in the code. Software cracks of this nature do not work in the scheme we propose due to the strong reliance on the server-side component. Our scheme relies on having an active rights verification scheme that checks for the user ID by connecting to a server each time the application is run. This check cannot be bypassed because the software cannot run without server coordination. Our system can be circumvented if an adversary can obtain the private key of a legitimate user. However, we believe that it remains secure because private keys cannot be easily read from smartcards. Furthermore, since the number of instances of an application running at a given time are restricted by the server the user has a strong disincentive to extract his private key and share it.

Another possible method of breaking the system would be by simulating the vendor server. An adversary could monitor all the transactions between the user and the server and map these transactions to the operations performed. We believe that this is as hard as reverse engineering the software if the software is written properly. Software vendors should make sure that an adversary cannot easily monitor which client side functions are executed after each response from the server. For example, the software can mislead an adversary by pretending to call functions that are not needed. Vendors could also decide to have some of the critical functions remain on the server side, thus preventing an adversary from accessing it. This makes the program secure even if the coordination part is re-engineered by the adversary. Furthermore, if malicious users were to undertake the task of reverse engineering a product, they may as well write a competing version.

The attack our system is most vulnerable to is a denial of service attack. An adversary could attempt to flood the vendor server with authentication requests and prevent legitimate users from using the server. Vendors should have many servers distributed throughout the network so that a single denial of service attack cannot stop all the users from using that application.

5.3 Privacy

Admittedly, privacy is an issue in our system. Firstly, software vendors are fully aware of their customers' usage patterns. Sadly, this knowledge cannot be avoided since it is plays a key role in the billing and access control aspects of our system.

Secondly, while the program is being run, data is exchanged between clients and servers as arguments of the remote procedure calls. Hence, it is possible that an over-inquisitive software vendor could make use of this to peek at its users' data. We propose that function calls use pass-by-reference instead of pass-by-value as far as possible. This would limit the exposure of users' data as well as minimize network overhead of the system. However, this would be left to the software developer to implement since it cannot be easily enforced.

It is apparent that our system places a great deal of faith in software developers protecting their own users' privacy. This is not too unreasonable since it is, after all, in the best interests of any self-respecting software vendor to build a good reputation among its users.

On the other hand, the degree of this trust is not much greater than what is currently required. Any software developer today can already place backdoors in their programs to secretly peek at their users' data and the average user would be none the wiser. It is up to the few technically adept vigilantes to police software and post public advisories about errant practices. Similarly, our system could rely on such social mechanisms to keep developers' practices in check.

5.4 Scalability

Scalability is an important issue when dealing with large-scale systems. Our primary concern regarding scalability is the network and server load created when many users use the application simultaneously. We strongly recommend that the vendors write their software in a manner that minimizes client-server communication. Since the server deals mostly with function coordination this should not be a problem. With the expected increase in overall network bandwidth, network capacity should not be a problem especially with a well-distributed collection of servers. Software vendors also need to deal with the server load of individual servers by providing enough servers to handle the server requests. Vendors should be able to do this effectively because the number of registered users and their usage statistics are known.

It is also important that the system scales well with users running many applications at the same time. Certain anti-piracy systems such as hardware dongles require a piece of hardware for each application being used. This does not scale well because there are a limited number of device ports on a machine. Also, it is not convenient to carry around a dongle for each piece of software that is on your machine. Our system allows many applications to be run on a single client module that can be part of the operating system and scales well with running many applications simultaneously.

5.5 Ease of use

Ease of use is an important part of any successful anti-piracy system. Traditional systems inconvenience users by requiring them to carry the original CD, use a hardware device or go through a complicated product registration process. Our system simply requires the user to insert a smart card during use and the underlying system transparently takes care of all the piracy protection without inconveniencing the user. A single smart card gives the user access to all the applications that the user owns.

Our system also makes it easy for the user to access the application from anywhere since the client side software can be downloaded freely. Therefore, users do not need to have the application on the hard disk or carry around the product CD.

5.6 Distribution

We have observed that controlling distribution is one of the key elements in many traditional schemes intended to circumvent software piracy. The problem with such an approach is that once a single instance of compromise in distribution control occurs, the entire scheme is rendered useless. Furthermore, the software product in its complete functional form becomes freely available to anyone who desires it. Making a truly secure distribution scheme is a difficult problem, and perhaps this is why no universally accepted mechanism for this exists.

We believe that our approach in making this a non-issue is more feasible. And it is certainly viable in our proposed framework. Furthermore, it brings together a range of other benefits. For example, peer-to-peer file sharing systems and pirated software vendors can be considered assets to the software industry in our framework. The cost of distribution to the software vendor becomes negligible, and both the vendor and the consumer will benefit from this (eg. reduced prices, higher availability etc.). Also, since every time the software is used a network connection has to be made, trivial additions can be made to enable the software to upgrade itself dynamically with newer versions. However, the actual functionality made available is based on the rights of the user currently executing the program.

As a result of our scheme of software distribution, we can also support different billing and licensing models. For example, usage based billing like what is done in mobile phone services currently becomes a possibility. Furthermore, support for letting users "try out" software and even rent software is possible. Such flexibility in pricing allows the vendor to effectively price discriminate. Therefore, the vendor is better equipped to meet the individual reservation prices of consumers. We believe this is an important requirement for any attempt at circumventing software piracy, since it reduces the incentive for users to engage in piracy.

For example, take a novelist and a graphics designer. If both were to purchase word processors, who would benefit more from the software? Is it reasonable to charge them both the same price? If this were the case, wouldn't the graphics designer be more inclined to pick up a pirated copy of the software?

Furthermore, consider the income disparity amongst different regions of the world. Is it fair to charge the same prices from consumers in the US and those in Indonesia? Is it not surprising then that a consumer in Indonesia is more inclined to engage in piracy?

These are some of the issues our scheme addresses, simply by making pricing flexibility more readily available to the vendor. Since vendors can price discriminate more effectively now, the incentive for consumers to engage in piracy can be reduced and an economically efficient equilibrium can be

achieved.

6. CONCLUSION

We believe the framework proposed here provides a better alternative to current solutions. Firstly, it effectively prevents software piracy by both incorporating an infrastructure that is difficult to bypass and simultaneously discouraging users from engaging in piracy. Secondly, we bypass the distribution control problem by making it a non-issue. Furthermore, as a result of this, vendors have greater flexibility in pricing and that promises benefits for all. Most importantly, all processes in the framework occur transparently with minimal inconvenience to the user.

While the network and PKI requirements meant that this system would be rather impractical if it were implemented today, we envision that such infrastructure will be available in five to ten years time and the environment will be especially conducive for the implementation of our framework.

7. REFERENCES

[1] Business Software Alliance. Sixth Annual BSA Global Software Piracy Study. [web page] May 2001; <URL: http://www.bsa.org/resources/2001-05-21.55.pdf>. [Accessed 10 Dec 2001]

[2] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages 1999*, San Antonio, TX, January 1999.

[3] ZDNet News. Hackers bypass Microsoft copy protection. [web page] Oct 2001; <URL: http://www.zdnet.com/zdnn/stories/news/0,4586,2821260,00.html>. [Accessed 10 Dec 2001]

[4] Aladdin Knowledge Systems. The Privilege Software Commerce Platform. [web page] Dec 2001; <URL: http://www.ealaddin.com/privilege/default.asp?cf=tl>. [Accessed 10 Dec 2001]

[5] IBM Corporation. Web Services architecture overview. [web page] Sep 2001; <URL: http://www-106.ibm.com/developerworks/webservices/library/w-ovr/>. [Accessed 10 Dec 2001]

[6] IDenticard. What's New in Smart Card Technology. [web page]; <URL: http://www.identicard.com/idideas/newsmartcard.htm>. [Accessed 10 Dec 2001]

[7] Multimedia Development Corporation. National Multi-Purpose Card. [web page] Nov 2001; <URL: http://www.msc.com.my/mdc/flagships/mc.asp>. [Accessed 10 Dec 2001]