Anti-piracy system: Legal Enforcement/licensing Executive Summary

Pradeep Varma
Buffnstaff Software and Services Private Limited
634 Sector 21, Gurgaon Haryana 122016, India
sales@buffnstaff.com

1. INTRODUCTION

Intellectual Property may be the most valuable property in your organization. Each time you sell, lease, rent, or share your property with customers, corporates, or the public, you run the risk of the property being copied and used without permission in ways and means unacceptable to you. Copyright protection may not be enough for your purpose. Further, copyright assertion, without enforcement, may not be enough for your purpose.

Businesses almost always have to run within a legal and auditory framework, wherein electronic support, e.g. website, intranet, must obey prescribed regulations. It is important then, to build the electronic infrastructure in a manner where demonstrable compliance with regulations is manifest.

Buff 'N' Staff® has developed a proprietary, comprehensive, extremely lightweight law or rule enforcement system for the intranet and internet presence of companies, including websites. The system comprises a software licensing system that can be made available on any interface or platform of choice. Using the tooling, BNS® can build or further develop a website to enable licensing or legal enforcement capabilities for software deliverables.

An open source subset of the system, called Paint is available at www.buffnstaff.com as a part of the Buffnstaff's Greetings offering. The Greetings/Paint offering is available with all client-side software provided both in executable and source forms for careful review by customers under the Gnu General Public License (GPL). Paint is an illustrative system; it does not contain patented or proprietary technology that the company only offers to customers under proprietary licenses. A customer may enter into a non-disclosure or confidentiality agreement to know further about the proprietary system.

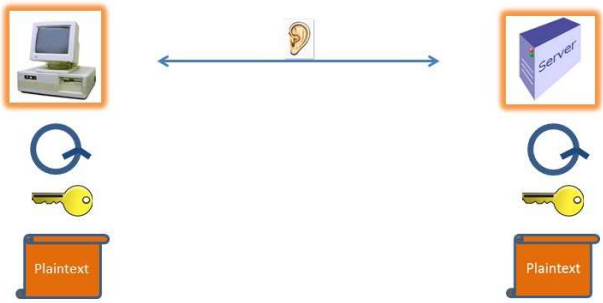

Figure 1: Nice World Assumptions – Limited or No Law Enforcement Needed for Client or Server.

2. THE PAINT ANTI-PIRACY SYSTEM

Laws on copyright (similarly many other laws) are well known throughout the world. In a nice world assumption, shown in Figure 1, the law is understood by all parties and followed to the letter. Thus enforcement of law is neither necessary, nor carried out and copyright protection is obtained. In the client server dialog shown in Figure 1, no adverserial action takes place, hence a company's server stores data in plaintext form on the machine, including secret keys (or passwords), and runs any software necessary (shown as a looping circle). A client machine, not in control of the company, works similarly, and safely, since adversity is simply not present. Indeed, in such a world, the need for secret keys and cryptography may be questioned. However, since laws may not apply uniformly everywhere on the globe such as privacy laws, one may assume that in the communication path from the client machine to the server, there may be eavesdropping on the communication line, requiring data encryption during transport to protect against intruders. Thus adverserial action, if any, is contained and limited and assumed to be only during transport and not directly on the machines of interest to a company and its customers.

Such nice world assumptions however are impractical and haloed computers with no adverserial intent or action cannot be assumed. Figure 2 therefore shows a practical world context, in which the company server is assumed to be secure. The glow in its halo is less, since security is actively maintained by measures such as firewalls, secure languages and

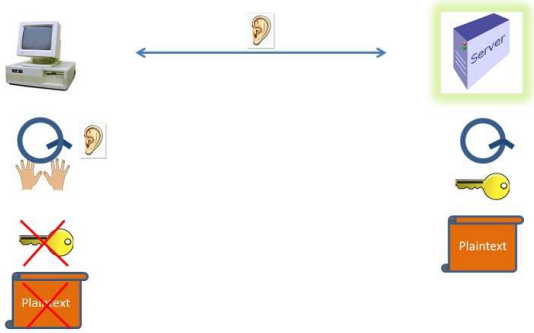

Figure 2: Pragmatic World Assumptions – Law Enformcement Needed for Client (also Server).

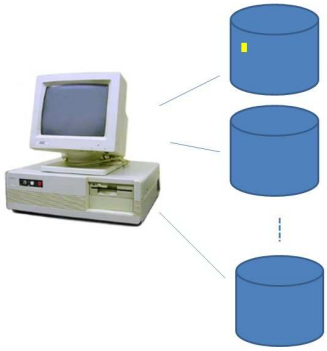

Figure 3: Basic Paint Architecture.

cryptography, but still, under company control and protection, the server handles adverserial action in its stride. The client computer however, cannot be assumed to have any halo at all. Company software deployed on the client may be snooped on, modified hands on (hacked), and acted upon with any malicious intent. Secret keys cannot be shared with the client, as the keys may be duplicated and used in disallowed contexts, such as running more than the licensed number of copyrighted software instances, by repeatedly reusing a license key. Company software should not store confidential data on the computer in plaintext form, as it can be maliciously read, copied and/or modified. Indeed, company software has to assume complete adverserial control of the client machine, since a client can well be a pirate, purchasing a legal copy of the software only with a plan to break and re-distribute it.

The company software has to store and handle confidential data in encrypted form only on the client machine. Secret keys have to be available to the company software without the client or client machine handling them in un-encrypted form. Secure cryptography, carried out under such adverserial assumptions of the client machine is often referred to as whitebox cryptography. Unlike classical cryptography, which assumes secure machines and safe-keeping of secret keys on secure machines, whitebox cryptography does not make such an assumption. Whitebox cryptography seeks mathematical foundations of secure cryptography, such as reducing key discovery to prime factorisation, while allowing cryptography to be carried out on a machine under the control of an adversary.

Classical cryptography crystallizes secret information to a key separate from the algorithm or implementation of cryptography, which is widely studied for establishing quality of the algorithm and enabling widespread implementation. The key represents one specific choice from a large combinatorial space of key choices, such that guessing a key by enumerating the space becomes a prohibitive exercise (e.g. 2^{1024} choices for a 1024-bit key). Classical cryptography fails in a whitebox context, because the key becomes a stumbling block of vulnerability, and hiding the key by encryption only reduces the problem to hiding the second key used for encryption, which in effect, leaves the problem unsolved. Given the large momentum of work in classical cryptography, whitebox cryptography is nevertheless attempted often with a goal of abstracting absent information from a whitebox implementation, which makes the implementation secure, while seeking alternative means of delivering the key information so that cryptography indeed can be carried out.

Paint is a software licensing system for enforcing copyright protection that does minimal, illustrative whitebox cryptography by hiding its cryptography key within the implementation itself. Thus the key delivery problem is solved by tying it to its use context. The paint system is provided in open source form, so the key can be reverse engineered by studying the code, but that is intentional as the system is only meant to be illustrative. Whitebox cryptography is meant to be robust enough to survive adverserial action on reverse engineered code (e.g. Java source code reverse engineered from bytecode) so that a hidden key is not discovered without prohibitive cost.

In our proprietary research, we have developed several highly effective cryptography techniques for a whitebox context, wherein regardless of a licensed program being reverse engineered, run in a harness or debugger or virtual machine, or modified (e.g. dynamically-linked library (DLL) substitution), the cryptography implementation cannot be undermined.

Paint is unique in its capability to handle masquerade attacks by adversaries. A masquerade attack can be carried out by an adversary in substituting the library/system functions on the client machine, for example, to offer spurious serial number or identity information of the computer. Thus a licensing system that identifies an installation by such information can be misled into allowing pirated copies that are fed with forged identity information to obtain allowance. Paint works by not relying on system fed information thus. Instead, Paint paints the computer with a specific random stroke, that is unique to the computer and later identifies the computer with its chosen unique stroke as opposed to system generated information. Figure 3 shows the basic Paint system, that marks the computer with a stored, encrypted paint file hidden somewhere in the filesystem on the computer so that neither does the adversary know what is contained in a paint file, nor where it is stored.

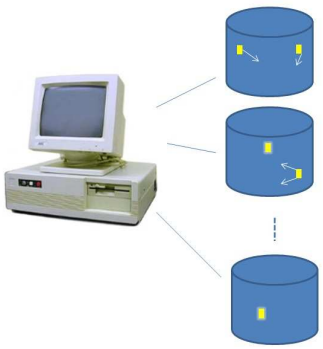

Figure 4: Paint Premium. A Mobile Architecture.

Since a whitebox environment is targeted, when company software linked to Paint is downloaded (e.g. the Greetings software), the download does not arrive with a random seed that can be observed by an adversary and duplicated. Instead, the random seed is generated online within the computer environment. Examples of the random seed are, time of the day, GPS position of the computer, the number of running processes on the computer at the time, any random fact about the computer. Specfically, the llustrative Paint system uses the length of the PATH variable to be the random seed. This is likely to be sufficient to discriminate distinct client machines and prevent piracy and yet be generatable again if needed later, as discussed below.

Basic Paint suffers from a problem that the hidden paint file can be overwritten inadvertantly. So in Figure 4, Paint Premium, (or Paint Mobile) paints the computer with more than one stroke, so that at any time, even if a subset of the strokes survive, the licensing system can work. The paint files are evolved over time so that they move from their locations, update their contents, and dissipate or replicate in a manner that makes them hard to track by an adversary. The evolving paint system migrates from the installation time state of the computer so effectively, that a piracy attempt based on the installation time state becomes infeasible.

The open-source paint system used in the Greetings software uses Basic Paint only. It overcomes inadvertant overwriting by regenerating the paint information dynamically. Since the PATH variable is not changed very often, the random seed can be generated again later, if the paint mark disappears and the system provides a best-efforts licensing capability in the long run, if the situation so arises.