An Analysis into Software Piracy Prevention

Peter A. Ciszak, Jack Lai, Nathan Tai
EECE 412 – Computer Security – Group 2
University of British Columbia

***Abstract* – The issue of software piracy has existed for several decades. Full-time developers dedicate their lives towards creating software in order to make a living. Many insist that software is a form of intellectual property and that the quality of certain products could not continue to be produced if the developers were not adequately compensated. Regardless of the social factors it's clear that there is a financial motivation involved when developers employ copy-protection schemes. This report analyzes many of the faults with traditional methods and explores the capabilities of online authentication and its power when paired with software diversity. Although no copy-protection technique is completely bullet proof, there is still significant room for practical improvement.**

I. Introduction

There was little need for copy protection in the early days of computing. At the time, most software was custom-developed for in-house applications. It wasn't until the early 1960s that computer applications were being actively marketed. According to the Copyright Office, the first deposit of a computer program for registration was on November 30th, 1961. As computers have become more and more popular, many have invested their lives in the advancement of their capabilities. As a result, there is a significant financial motivation behind software development while more and more development houses begin to enforce license agreements and apply technical protection schemes to deter unauthorized copying. A software license agreement is an attempt at a contract between a producer and a user of computer software, which typically grants the user the right to only execute the software.

II. Previous Efforts

There was a project was written in the previous cohort regarding software cracking. It primarily focused on common techniques to exploit statically secured applications. This static nature means that the entire process can be executed any number of times with different testing criteria with no previous consequence. For instance everything could be simulated in a refreshable virtual machine.

The focus of this report is specifically about how online authentication techniques could be used to secure software to a much greater extent then simple static protection schemes.

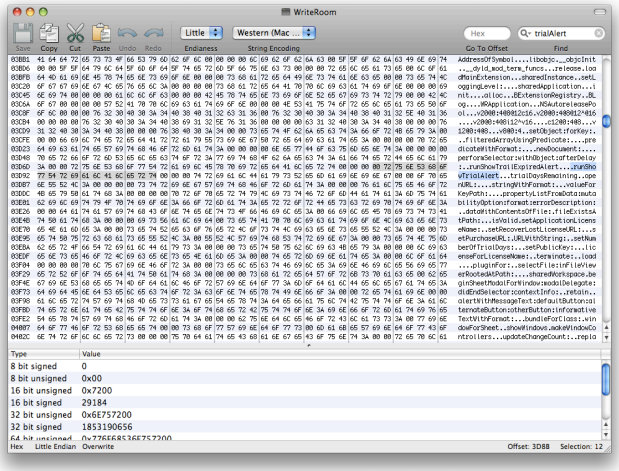
III. Consider The Problem

It's a common consensus in computer security that an attacker will have complete control over the machines which they have physical access. This is because with static local software, the whole world, as it is relevant, exists entirely inside the computer on the attackers desk. Naturally it follows from this that any authentication mechanisms would essentially exist entirely within the attackers computer. This is a major disadvantage.

The typical toolkit of a software cracker would certainly include a binary disassembler. A disassembler would convert the machine code of the application binary back into assembly language. Although, this is seldom a perfect process; often code and data are difficult to distinguish and may generate large blocks of garbage code. Nevertheless, with this, an attacker can attempt to isolate particular important algorithms in order to gain an understanding in how to exploit the software. An attacker will often also use a debugger to step through the resulting assembly to see exactly where the program is as it executes in real time. Once the attacker has pinpointed what they believe to be an integral part of the protection algorithm, they can make an educated modification using a hex editor. If anything goes wrong, they can repeat the process by running everything inside a virtual machine and simply revert states. This is the major problem with static defenses.

Although it may sound difficult to perform such tasks, we were actually able to do it ourselves with a lightweight word processor for Mac OSX called WriteRoom. WriteRoom used no online authentication scheme and in addition to this, it employed a third party protection mechanism to attempt to secure itself. Ironically we were able to exploit WriteRoom by completely bypassing its 3rd party mechanism with a simple tweak inside a hex-editor.

It appears that the developers used string constants to specify application wide configuration settings within the code. These strings also turned out to manage the registration of the software. We simply changed the strings according to what they described. As can be seen in the figure below, the *runShowTrialAlert* strings as well as the trialDaysRemaining constants are exposed in plaintext.



In general one of the great design problems with copy-protection is that developers commonly program what is referred to as a SPOT or single point of truth into their designs. In essence it is a single bit that software looks at to determine if it should run or not. While in terms of software engineering many developers consider the SPOT rule to be good design, in terms of piracy security this single point of truth makes it all too simple for an attacker to isolate and flip the appropriate bit.

Some programs employ obfuscation techniques inside the application binary designed to make the attackers job more time consuming. Nevertheless the bottom line is that developers will always have the more difficult job since an attacker always has complete control over the code running on their own machine.

In spite of what appears to be a futile exercise, what exactly could a developer do to secure their software? One option is to run the application completely remotely, which would naturally imply the deployment of web applications. This is what a lot of developers are actually doing. However this style of deployment is obviously not suitable for every form of application. In addition web applications create an extra set of security vulnerabilities that need to be considered when designing for the Internet.

An alternative is to consider running only part of the application remotely. Today more and more software is using what is called online authentication or online product activation. The idea is that a developer should keep the authentication logic in the client application simple and keep the real protection logic far away from the attackers. This is an example of applying an economy of mechanism.

One of the great advantages with online product activation is the fact that the attacker does not have access to the remote authentication servers. This means that a developer does not have to worry about obfuscating the authentication logic.

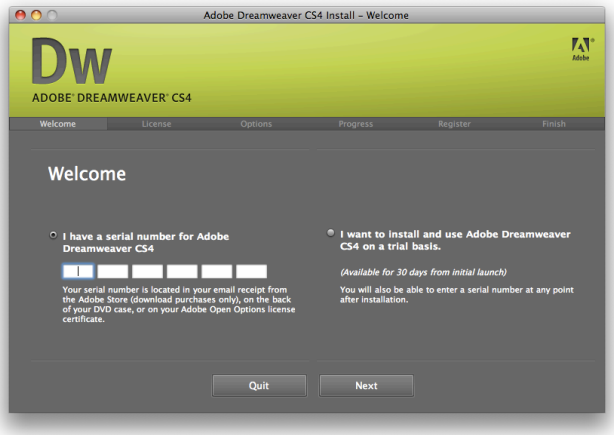
IV. Online Authentication

Online authentication allows developers to move the authentication mechanism to a remote computer. Immediately it is clear that this will create particular usability tradeoffs. A developer does not want to lock legitimate paying customers out of their software during either a failing of their service or the inability for the customer to connect to the Internet.

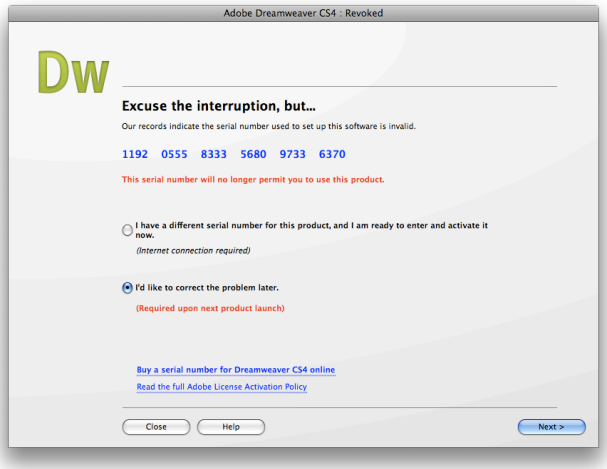
A typical online authentication routine usually involves the following steps. The software will demand the serial number from the user. After the user enters the number, the information is sent to the activation servers for remote processing. Some common information in addition to the serial number includes a checksum of the application binary as well as a hash of unique machine signatures. This allows the activation severs to generate a machine dependent activation code and ensure that the binary has not bee modified. The activation code it then returned to the client. The software now has to verify the correctness of this activation code. It's important that the developer still ensure not to use a single point of truth on the client side. Like all copy-protection mechanisms, a single weakness could defeat the entire system regardless of how much protection was behind that SPOT. This is why it's very important to practice using defense in depth when designing a copy-protection scheme.

V. Dreamweaver Case Study

We investigated Adobe Dreamweaver CS4 to determine how copy-secure the latest and greatest software from Adobe actually was. Before going further its important to understand Adobe's interesting distribution model. Adobe allows their customers to download full complete versions of all their products from their website with no restrictions aside from a thirty day trial limit. After the limit has expired, the user is required to activate. This was one of the reasons we decided to go with Dreamweaver since it's a popular and well known software package and its latest version CS4, just came out of beta about a month before this writing.



To initially test what Dreamweaver's behavior would be, we downloaded and installed a clean copy. When we launched the app, just as expected, it told us that it was time limited until we purchased a license and activated with Adobe. Instead of purchasing a license, we decided to find an existing serial by searching on Google. A Dreamweaver serial was very easy to find, as there were many results. In fact the serial number we chose was actually the beta serial number, which was publicly available. We entered the serial into Dreamweaver and sure enough there were problems.

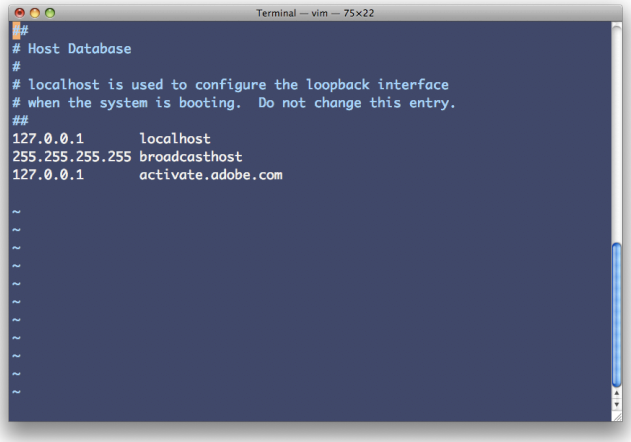


Before going any further, it is always a good idea in computer security and as well when attempting to exploit software to learn as much as you can before attempting too much. We went to Adobe's website and found an interesting article regarding troubleshooting during the activation process. As quoted directly from the site:

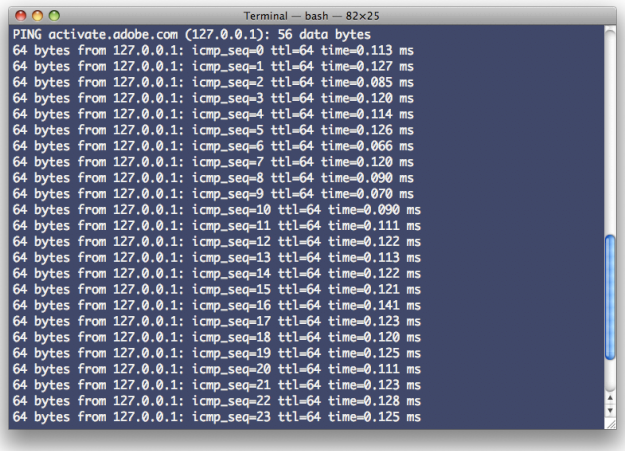
> *"Product activation is a technical measure that helps protect Adobe against unauthorized use and copying of its software. Activation runs silently in the background and occurs when the application detects an Internet connection."*

This short paragraph actually gives an attacker a great deal of information. The first important point is that it happens in the background when the application detects an Internet connection. We tried a simple firewall to block the whole Dreamweaver process and this was not sufficient to change anything. Dreamweaver still wanted us to activate when we had a connection. To analyze the matter further we fired up Wireshark, a popular packet analyzer, and determined that Dreamweaver was in fact attempting to send packets to activate.adobe.com. The interesting thing in this study is that instead of cracking Dreamweaver by modifying it's binary, we wanted to alter it's environment to trick it into thinking it's servers were offline although it was aware that it was connected to the Internet.

In order to do this, we tweaked the hosts file which is usually found on UNIX based systems inside the /etc directory. What this file does is map static name resolutions to IP addresses before going to a DNS server, so in our case we wanted to map activate.adobe.com to something else such as localhost. As we can see below, a ping test shows us that when the system requests activate.adobe.com, we are certainly not resolving the address that Adobe intended.



What we have done is created a unique third case where the Dreamweaver process is still connected to the Internet except that when it tries to resolve an address for activate.adobe.com it gets our dummy address instead. This gives the appearance that there may be something wrong with Adobe's servers. Upon re-launching Dreamweaver it does not ask for anything anymore. Adobe has made a particular tradeoff to act non-intrusively when its activation servers are possibly offline. This behavior also could have been a consequence of the fact that Dreamweaver just came out of beta. It is necessary that the attacker perform the modifications to the hosts file before the initial launch of Dreamweaver. If this is not the case, Dreamweaver will store the invalid serial deep inside the attackers system and will not run either way.



Interestingly, this particular exploit does not work on the

latest version of Dreamweaver; it's possible that it only existed initially to smooth the product launch in the event of activation problems. Nevertheless it's clear that online authentication alone is not enough. A single point of truth on the client side can still be flipped in order to make the software continue to execute.

What we need is that client requires a service that can only be provided remotely. This brings us to the exciting topic of software diversity.

VI. Software Diversity

The concept of software diversity may be new to copy-protection, however it has been applied in networking applications with some success. Server diversity helps to reduce the number of shared vulnerabilities between different servers. Shared vulnerabilities can result in a large number of attacks that can affect an entire installation base. Software diversity is an approach to mitigate the risk of correlate failure and lower the resulting consequence of a single attack and it's effectiveness of repeated application. Many researchers have been focusing on introducing diversity by using varying kinds of techniques on a system-by-system basis. Therefore, diversity must be introduced at all levels of system design, including any scheme that is used to introduce the diversity itself.

There does not yet exist a good example of a product that has successfully deployed a practical diversity technique. Yet there are may methods that could significantly increase the amount of time an attacker must spend. A particularly interesting combination is using online authentication with software diversity to link a remotely obfuscated module after installation. The module would be obfuscated using hash of the unique hardware signatures of the client machine. Vendors could deploy their software without particular libraries preventing the software from functionally being able to run no matter what an attacker does. This would require that the product call home to get these libraries which could be diversified case by case. In addition the libraries could employ a form of software ageing which could force periodic activation. This helps us because a single hack will no longer work universally. What we have now is a dynamic nature defense and no longer a static defense.

VII. Other Techniques

a. Dynamic Nature of Defense

Another common routine for implementing a dynamic nature of defense often involves five phases: the purchasing phase, the downloading phase, the installation phase, and the execution and update phases. At first, a user has to purchase the product key and unique installer through the server. The server will authorize the user to download the unique installer with a key accompanied. Then the user will install the software, which includes a uniquely compiled executable binary, so the crack cannot be generalized. In the execution phase, the execution path contains an embedded parity check to once again decrease the likelihood of successful malicious activities. Finally, an unscheduled update allows the software to age based on the fragmentation, accumulation of errors and the exhaustion of operating system resources.

b. Encryption

Encryption is the most common method used to hide digital data. So how could it be used to protect software? The problem is that everything still needs to be decrypted on the attackers computer before it can be of any use. In light of this weakness, several practical cases have shown that using encryption to hide part of the software binary is a significant deterrent for an attacker. An increase in the number of encrypted entities will certainly increase an attackers work. Encrypting all application state data including the serial numbers, server-destined packets and parts of the application binary itself will always add to the time and difficulty of exploiting the software.

c. Code Obfuscation

Code obfuscation is the process of intentionally making the object, machine or source code difficult to understand. The purpose is to deter reverse engineering, disassembly or de-compilation. Therefore, this can prevent unauthorized access to the source code, so that it is difficult to duplicate or modify the code to achieve a desired outcome.

d. Legal Measures

Even though the Copyright Act is very common in many countries, its exploitation is often not enforced on a person by person basis. It's not difficult to find illegal copies of software online for even the technically un-savvy individual. Software vendors have cumulatively lost billions of dollars every year because of this. As a result, many companies have suggested that the government increase the level of discipline so that they can more successfully take legal action if necessary. Some believe that through better enforcement to deter piracy by creating a fear of consequence could possibly prevent a significant number of piracy cases. This pessimistic solution is not universally accepted.

VIII. Financial Considerations

Developing copy-protection schemes requires an experienced development team and specialized knowledge.

Not all developers are able to adequately apply such measures. A developer should consider the additional costs required to invest in a protection scheme vs. the actual estimated losses from duplication. For small time developers it's often not worth the investment. In addition, deploying a system with online authentication and diversity requires substantial sophistication in the server-side software as well as the resources to keep the cluster running constantly. Nevertheless, its been shown to be quite a reasonable investment for larger corporations which would actually see a return on their investment.

IX. Conclusion

When designing any secure system, copy-protection included we must consider the principles of secure systems. This means we need defense in depth, not a single point of truth. We want a least common mechanism, we should use encrypted channels whenever communicating with servers. We can apply complete mediation by removing important libraries from shipped software to prevent it from initially running without calling home. Applying a good economy of mechanism implies that we should keep it simple on the client side and try to put most of the complexities of the scheme if any on the remote machine where an attacker cannot analyze them. As well we must consider the psychological acceptability of the entire solution. This is the compromise that Adobe had to make when they chose not to block us out of Dreamweaver when it was unable to contact its activation servers. Regardless of all these techniques, the harsh reality is that software diversity and online authentication only make software exploits slower. It's easy for a developer to overlook something and if it's serious enough it can rip a hole in the entire mechanism. It's still very much a game of cat and mouse and the attackers have the head start. Nevertheless, there is still a lot that a developer can do that is still not commonly practiced. This includes many of the techniques described above and the pairing of online authentication with software diversity. Applying more then just one single point of truth significantly increases the deterrence of such attacks.

References

[1] Bertrand Anckaert, Bjorn De Shutter and Koen De Bosschere, "Software Piracy Prevention through Diversity". [Online]
Available: http://portal.acm.org/citation.cfm?id=1029146.1029157

[2] Gareth Cronin, "A Taxonomy of Methods for Software Piracy Prevention"
Available: http://www.croninsolutions.com/writing/piracytaxonomy.pdf

[3] Michael Folk, "Software Piracy Prevention"
Available:
http://www.cs.allegheny.edu/~gkapfham/research/RICSS/RICSS_folk.pdf

[4] Autodesk and AutoCAD, "Product Activation"
Available: Activation_Whitepaper_v2.pdf

[5] Software & Information Industry Association, "Anti-Piracy"
Available: http://www.siia.net/piracy/whatis.asp

[6] Ankit Jain, Jason Kuo, Jordan Soet and Brian Tse, "Software Cracking"
Available: http://courses.ece.ubc.ca/412/previous_years/
2007_1_spring/modules/term_project/reports/2007/software_cracking.pdf

[7] Heather Meeker, "Only in America? Copyright Law Key to Global Free Software Model"
Available: http://www.linuxinsider.com/story/50421.html

[8] Canadian Intellectual Property Office
Available: http://www.cipo.ic.gc.ca/epic/site/cipointernet-internetopic.nsf/en/wr00090e.html