

Software

Strategy & Technology
For Software Executives

BUSINESS

a webcom publication

Software Business Executive Report

May 11th, 2009

Software Attacks and How to Defend Against Them

(Part 1 of a 2 part series)

**Oren Bear, Senior Security Specialist,
Software Rights Management**
Aladdin Knowledge Systems

In the past, software crackers were largely hobbyists who competed against each other for the pure challenge of overcoming the protection, but nowadays a significant number of them are professionals and members of sophisticated distribution networks who make this illegal activity their living. It is important to “know your enemy.” When you are well informed about the types of attacks that a software cracker may make, you will be best able to devise and implement protection strategies that limit or prevent their success.

In general, hardware-based protection keys provide the most robust security, while software-based keys provide support for Trialware and Electronic Software Distribution. This article is part 1 in a 2 part series and focuses on common attacks against software using hardware-based protection keys (dongles) and specific tactics to ensure your software’s protected against these threats. Part 2 will focus on software-based protection keys.

Attack #1: Patching Executable files or Dynamically linked libraries

A software cracker disassembles and/or debugs executable files or dynamically linked libraries to find the protected code. The file is then patched in order to modify run-time flow, or to remove calls in the code. Commonly, the software cracker sends a small, standalone patch executable (crack) that the end-user runs in order to patch your software.

Protection Measure

The goal is to make the process of patching executables more complex and costly than purchasing the software. The

more files that are protected, the longer it takes a software cracker to remove the protection. The most convenient way to protect a large number of executables and dynamically linked libraries is to use an automatic protection tool. Such tools (known as packers) inject the protection code into the compiled binary files, reducing the need to laboriously write protection code for every file. The most effective tools also offer anti-piracy specific measures such as anti-debugging mechanisms, executable file encryption and data file encryption mechanisms that allow the protected application to encrypt and decrypt data files on the fly.

Attack #2: Modifying Key Memory

Licensing data is normally stored in the memory of a protection key. A software cracker attempts to access the key memory in order to modify the licensing terms. For example, the cracker changes a depleted execution-based license into a perpetual license, or enables a feature that was not paid for.

Protection Measure

To prevent key memory modification, high-end protection keys contain Read-only memory (ROM). ROM is a segment of the memory that can contain data that the protection application can access, but not overwrite, unlike regular Read/Write memory segments. This memory segment should only be updated using remote updates, and the creation of such updates should only be possible for the vendor. A well-executed licensing implementation should build upon secure ROM.

Attack #3: Emulating Protection Keys

To emulate the software of a protection key manufacturer, a software cracker creates an application that replays previously recorded calls, as if an actual protection key is returning the

calls. Limited functionality emulators only record and replay calls. Full-functionality emulators also emulate the key, including its encryption. A software cracker requires access to the encryption key to create a full-functionality emulator. Primarily, emulators attempt to replace the device driver.

Protection Measure

The protection keys should employ a secure channel between an application and the hardware key. Data that passes between the protected application and the key should be encrypted. Taking advantage of the secure channel functionality between your application and the key provides you with the strongest possible protection.

The secure channel should employ a different encryption key in every session. This means that a cracker attempting to record data passing through the secure channel would not be able to replay the data, since the encryption key used to encrypt the data will differ from that used to decrypt the data. In addition, the protected application can detect limited-functionality emulators by utilizing the encryption engine of the hardware key to decrypt hard-coded data. The emulator has no way to correctly decrypt such data, as it has no access to the encryption key.

Attack #4: Terminal Servers and Terminal Service Solutions

When using the terminal servers of some operating systems, it might be possible for an end-user with a locally connected protection key to enable software on multiple concurrent terminals.

Protection Measure

The protection should employ mechanisms to determine if a protected application is running on a terminal server. If such an environment is detected, and the license is on a local protection key, the program should not function. If the license is on a network key, one of the concurrent licenses should be consumed.

Attack #5: Cloning Hardware Keys

The software cracker reverse-engineers a hardware protection key, then creates duplicates. This is an extremely difficult and costly attack, both in terms of the reverse-engineering tools and expertise, and in the ongoing production of hardware keys.

Protection Measure

The hardware key provider should reduce the likelihood of this attack by implementing unique or customized hardware components that are not available for purchase off-the-shelf, and that contain measures against tampering. The firmware should be protected against extraction.

Attack #6: Clock Tampering

Clock tampering relates to the end-user setting the system clock of the machine on which the protected software is running to an earlier date, thus re-enabling a time-limited license that has expired.

Protection Measure

Use hardware keys that contain their own real-time clock (RTC) when implementing time-based licenses for your software. The hardware key provider should ensure that these clocks are highly accurate rather than offering a user interface or an API for resetting their time. An interface could potentially be exploited by end-users or crackers to extend the software license indefinitely. Without such an interface, the RTC remains inaccessible to end-users and virtually impossible to modify.

Attack #7: Tampered License Requests And Updates

If requests for licenses and the license update application are verified at the software level, (e.g. by the device drivers of the hardware key), crackers may attack this code and force all licenses to be granted, or force license updates to be applied. This even applies to expired licenses and license updates that have already been applied.

Protection Measure

The granting of licenses and the application of license updates should be performed by the hardware key itself, outside the reach of debuggers. Failing to obtain a valid license should result in locked functionality. For example, the encryption engine should remain disabled until a license is successfully granted. Similarly, license updates should only be applied after the hardware key itself verifies the signature.

General Protection Strategies

These additional general software protection strategies are also recommended for publishers' employing hardware-based protection keys:

Use Both the API-Based Implementation and Automatic Protection

Maximize security by using the API to implement calls to the software protection keys, and protect the program with an automatic protection tool that offers anti-debugging and anti-reverse engineering measures. Using one protection method does not preclude the use of the other.

Insert Multiple Calls in Your Code

Inserting many calls throughout the code to the protection key and binding data from the key with the software functionality frustrates those attempting to crack your software. Multiple calls increase the difficulty in tracing a protection scheme.

Encrypt/Decrypt Data Using the Protection Key's On-Chip Encryption Engine

The encryption process, which takes place before distributing the software, and the decryption processes, which take place during run-time of the protected application, should be performed inside a hardware protection key, beyond the reach of any debugging utility. The protection key should employ a highly secure public encryption algorithm such as AES. Encrypting data with protection keys using such algorithms considerably enhances software security. By imple-

menting a protection key API scheme in which data is decrypted by a protection key, the association between the protected application and the protection key cannot easily be removed. Cracking the software also necessitates the software cracker decrypting the data. The encryption key length represents a trade-off between security and performance. A short encryption key provides better performance, but a key that is too short may reduce the level of security. 128-bit AES provides a well balanced choice, as it is faster than 256-bit AES, yet still impractical to attack.

It is important to remember that protection is only as strong as the weakest link. An effective implementation must build upon a software protection solution that offers all the building blocks for a secure implementation:

- An encryption engine that employs a proven public encryption algorithm such as AES
- Automatic protection with anti-debugging detection and other anti-reverse engineering measures
- A secure communication channel
- Read-only memory
- And the additional protection strategies mentioned above

A software publisher's own protection implementation must be thorough, complete and, to a certain extent, even creative. Only when both the security infrastructure and its implementation are maximized is the software truly protected. This may

sound like an enormous task to accomplish, and while it is by no means trivial, it is definitely achievable.

Implementing effective protection can not only increase your products' longevity, but it can also significantly increase your company's revenue, making it well worth the effort.

Watch for Part II of "Software Attacks and How to Defend Against Them" in the June issue of the Software Executive e-Report.

Oren Bear is a Senior Security Specialist for the Software Rights Management Business Unit at Aladdin Knowledge Systems, an affiliate of SafeNet, Inc. Oren is responsible for security intelligence, threat analysis and response, and has contributed to the design of Aladdin's award-winning HASP SRM software protection and licensing solution. For nearly a decade, he has researched a variety of security solutions and supported a range of software developers in implementing custom protection for their products. Bear serves as a member of the Israeli Standard Institute technical committee for Hebrew in Computing Systems. He holds a practical engineering degree in software development from Tel Aviv University and is a certified CISSP.

Contact Aladdin at www.aladdin.com.