Hidden in Plain Sight: Quick-and-Dirty Safeguards for Computer-Based Game Licenses

John R. Dickinson
University of Windsor
MExperiences@bell.net

ABSTRACT

Licensing of computer-based games, exercises, and such is a common practice. Prudent it is, then, for the licensor to take steps to ensure that the license conditions are met. Herein are presented several easily implemented devices to effect that prudence.

INTRODUCTION

The license for the *World Beater* simulation game is to expire on June 30, 2023 or 063023. The array of numbers in Exhibit 1 comprises 20 rows and 50 columns and was distributed with the *World Beater* licensed computer program. The file is readable and alterable by the licensee, though he or she has been informed that altering the file would violate the terms of the license. The license expiry date is recorded in the file. Where does 063023 appear in the file? (Don't waste your time. It doesn't, *per se*? Or waste your time and make the point of this study.)

EXHIBIT 1
EXAMPLE CORE FILE

```
5983081GE94L18VbpOS296V66SEW3RCgpU1ENM5W2089bM37
E6295U8043W503N508mE8D1414BUEP0YM7845JTG683CFW42
58352jF6T7SV3L9MR706DR87DYW3Bc16E0LSJ3SD5468446N
7Q410i7NEO3551783MV0DFW95238J9F1MX47M0LFTWY3755N
FM8901HRO8WEV462OU452Ew4C6B7N8M9MMJ49mvGKLI953SE
MT4D3K8V1Z7BK895HRY3NnN2LP975EC285NXWUM35SB7M89L
LLG47VB39HJ6ED26C8N63V4M0D42X2X2RE487N59n3H4J788
PO07NB5F4J8Mc29OPU85JD623SC3B559N8E4w7T7ebb8T999
0O0O7GW09MVv7654SPU12DQa376N7B497Md53K8B35D4S2AA
336mF77ROP84N7997fhet35208MF8RH4P9OUR656XNEF9466
dyE289012MVC534231WA0909MNU631ASWC7M8L04FdEYImV6
49764rEuMIFX29512878mVTLIPSTC45m99103W786VBE0NY2
189J7F53WS1a85V1K1veO07MC639NEI7S8M809KNLOtWS34V
211TRVM54678JKRPBMTGFD34ACXDSfeuTIM53NV8ScROUNRG
XGGG3kDR3VJSSORBnM463TDKY878787MbURVCESWLQO95616
sLdv375RE663528BNFJTUKOYEVXE3556NVeEV49WVEVRrRRR
49653142334ERWDSEcCWEWREG7582424243546576871 21JU
HfyMHUTJFGWCZm683546VBEC35X6V8N2sK3M1LFK97SKEHC9
gmd8EJ4PS2Q4X2R8uk9G35WDUB4M2M8M9M7VDLKeUR48593R
```

Publishers of computer-based games, exercises, learning aids, and such may sell their products with no conditions. Alternatively, the products may be sold with accompanying terms and conditions or the products may be licensed rather than sold.

Conditions of licenses may specify an expiry date, number of times the program may be executed, or other limitations. Additionally, identification information such as license number or password may be specified. (Following, "game" is used generally to represent other types of computer-based tools as well.)

Where the game is Internet-based that Internet-base may be controllable by the publisher. However, there are circumstances where an Internet base is not appropriate. It may be more practicable for students to be tasked with competing against their own personal computers or completing exercises on their own. Management trainers may wish to administer the game at several different locations with it being impractical for a separate license (and accompanying publisher logistics) being granted for each location. In isolated instances, access to the Internet may not be available. And so on.

In such cases some means for monitoring and enforcing applicable license conditions may be desirable. The present study describes several devices for these purposes. The devices, ultimately, may be defeated. Some licensees, though, may lack sufficient guile to defeat them. Others with sufficient guile may find it not worth the investment of their time and effort to accomplish what is an illegal end. From this latter cost-benefit perspective, for many games and exercises any benefit from circumventing license conditions is likely to be local and not great.

LICENSE-RELATED INFORMATION

Some license-related information presumably remains static over the course of the license period. Such information might include a license number, password, maximum number of executions of the game program, and beginning and expiry dates of the license. Depending on production and distribution decisions, this information may or may not be hardcoded in the game software. This type of information may additionally be placed in a core file (Exhibit 1 and described below) that accompanies the game software. It might, too, only appear in the core file and not be hardcoded.

A second type of license-related information would change over the term of the license. This type of information comprises the number of times the game program has been executed and the dates of each execution and possibly additional items. This type of information, of course, cannot be hardcoded in the game software and must be stored externally. Again, the storage locale of interest here is the core file as follows.

THE CORE FILE

Consider a 100 x 100 array containing 10,000 single alphabetic characters and single numeric digits. The array contents may be stored in a separate file on a flash drive or CD that contains the game or exercise software, possibly packaged with a textbook. That file need not be hidden or protected from alteration. However, the conditions of sale should state that the file must not be altered. The information in those 10,000 locations is the device at the core of the safeguards described here. (A 100 x 100 array requires minimal storage capacity. And if 10,000 locations are deemed by the game programmer to be insufficient those dimensions may be increased.)

Information such as the license expiry date may be hidden among those 10,000 alphabetic characters and numeric digits. Consider, for example, a license expiry date of June 30, 2023 or 063023. The initial “0” may be in location 6281, the “6” in location 42, and so on. (Below it is explained that these digits need not be stored as numbers at all.) Other information may be similarly located.

Again, such static information may or may not be hardcoded in the game software. Where it is hardcoded, with each execution of the program information of these static types can be read from the core file and matched with the same information hardcoded in the program software. Such checks would ensure that the game software and accompanying core file remain paired and, also, are one means of determining whether the file has been tampered with. Additional checking devices are described below.

Information such as number and dates of program executions, of course, changes with each execution of the program but can be retained as the core file is updated at the end of each execution.

At this point, the hiding of such information is due to there being a large number of locations. A licensee seeking to extend the expiry date might be faced with 300 “0”s across the 10,000 locations (or 3,000 or 30,000 etc. in an array of larger dimensions). Or 300 “J”s and 700 “u”s, etc. Actually, the “J”s and “u”s or “June”s and so on may be but decoys.

SURROGATE INFORMATION

As just noted, finding, and thus being able to alter, licence-critical information in the core file is hindered by the large number of locations in that file. Still, for the 063023 license expiry date, finding all instances of “0” and “6” and so on–even some large number of them–in the file is easily done. Too, altering them can be readily accomplished with the find-and-replace function of a word processor. A real risk to the licensee, though, is that the 0s and 6s in some locations may be plants and altering them will result in a warning or worse, i.e., immediate termination of the license.

And search for what? There may be no 0s or 6s at all in the file, or at least none representing the license expiry date. Location 6281 may contain the letter “M” which the game software is hardcoded to recognize as a 0. Location 42 may contain the letter “R” which the software recognizes as a 6. Generally, there is no need for the information in the core file to be literal.

In fact there could be multiple such translation schemes. Say, for example, 31 with the specific applicable scheme being set by the day of the month the core file was most recently written.

DYNAMIC LOCATIONS

The above posits the first digit of the license expiry date, “0,” or its surrogate “M” is in location 6281. As the game or exercise progresses, i.e., the program is successively executed, location 6281 need not always contain that first digit of the expiry date.

The core file in some (hidden) location(s) also retains the number of times the program has been executed. A new location for the first digit of the expiry date may be in location 6281 plus the number of executions. Or 6281 minus the number of executions. Or 6281 plus or minus some *function* of the number of executions. Serving the same purpose, instead of the number of executions, the day of the month the program was most recently executed may be used. Or some combination of number of executions and day(s). Or a random number in location XXXX. Or that random number along with a second random number in location YYYY. Or along with a third random number... The random numbers would be generated at the end of an execution to determine new locations of information in the updated core file with the numbers also being written to the file to be used in the next execution.

For that matter, location 6281 need not be a consistent base at all. At the termination of a given execution, a random number may be generated and placed in hardcoded locations. Those numbers, then, indicate where vital information is to be stored and that information, then, is stored in those locations. At the next execution of the program, the numbers in those hardcoded locations are first read and the vital license information thereby read from those locations. This approach may be made further complex with a stepped protocol. The random numbers may not indicate where the vital information is stored. They may indicate other locations with those other locations indicating where the vital information is stored.

The specific location algorithm, of course, is hardcoded in the game software. In sum, the effect is to make it ever more difficult to locate and alter vital licence information.

HIDDEN CONTROLS: WARNINGS, EXTENSIONS, AND RECOVERY

As the date of license expiry or the licensed maximum number of program executions approaches, a notice to such effect can be automatically displayed. This is standard operating procedure. At some point, such a notice might be replaced with a warning that reaching a license limit is nigh or, eventually, that a limit has in fact been exceeded, or that the core file has been illegitimately altered.

Possibly, the licensee’s circumstance is innocuous. For example, the limit may be reached during the course of a competition or within an uncompleted series of exercises. In such circumstances the publisher may wish to provide some accommodation to the licensee. The displayed warning may provide instructions for contacting the publisher, perhaps including an email address via which the licensee can provide an explanation. Within the software *something* triggered the warning to be displayed. The displayed warning, then, could also instruct the licensee to quote a code that would inform the publisher of the specific event that triggered the warning.

In return communication, the publisher might provide a codeword and instruct the licensee to display a specific screen and hover the mouse over the lower left-hand corner of the screen. Though no command button is visible (its “visible” property being set to “false” in the software), clicking the mouse would cause a text box to be displayed into which the publisher-provided codeword could be entered by the licensee. The particular codeword would extend the license expiry date or the maximum number of executions. The specific extension length or number would–you guessed it–be retained somewhere in the core file.

This hidden command button having been revealed to the licensee, what is to prevent him or her from exercising it in the future? That the accommodation has been exercised once or some other number of times is also retained at some location in the core file. The game or exercise software reads this information from the file and accordingly disables the hidden command button.

IF ACCOMMODATION IS NOT PROVIDED OR FAILS

Above, procedures for providing some extension of the license limits are described. Possibly, though, the publisher may not wish to provide such accommodation or the exceeding of extended limits continues to occur. In these cases, the game or exercise program can simply erase the core file. When the program is executed, one of its first tasks is to check that the core file exists (and then to read it). The core file not existing causes the program to terminate.

COMPUTER DATE (NO, NOT THAT KIND OF DATE!)

The core file may be used to thwart manipulation of the date and time in the licensee’s computer operating system. The beginning and ending dates and times of the license period may be hardcoded in the program software. The same information, too, importantly would be present in the core file.

A licensee may set his or her computer date for some time in the past, thinking that that date will never pass the expiry date of the license. Or during the term of the license, he or she may periodically reset the date so that it does not progress to the ending date of the license.

To thwart the former strategy, on the first ever execution of the program, a check can be made whether the computer date is after the beginning date of the license. If it is not, a message may be displayed advising the licensee that the program will not execute until the computer date is reset. (Note that the core file may also record that such a message has been displayed and how many times.)

Subsequently, the date and time of each execution of the program may be retained in the core file. When the reset computer date is earlier than the most recent execution date, again, a message to this effect may be displayed. (As with other types of information, the most recent execution date may be stored in multiple locations that can be checked for consistency.)

The licensee, though, might reset the computer date and time to be at very small increments so that it is past the most recent execution date but approaches the license expiration date slowly. If the license does not stipulate a maximum number of executions, then this strategy would indicate compliance with the license. If the license does stipulate a maximum number of executions then the number of executions, as described above, is already stored in the core file at virtually impossible to identify locations and this will override the "minimally incremented date/time" strategy.

As well, the license expiry date and bogus computer date may be included in reports to game players, the bogus date perhaps raising suspicion among players.

SEARCHING THE CORE FILE

A farsighted licensee might store copies of the core file after each execution. Toward discovering license-related information, successive files might be searched–and the searches thwarted–as follows.

INFORMATION THAT REMAINS THE SAME IN A GIVEN LOCATION

Expecting, say, the expiry date to be in fixed locations in the core file, a search might be made to identify locations where the information remains the same after updating of the core file (which would occur at the end of each program execution). First, in light of the dynamic locations described above, the expiry date would not remain in fixed locations. Second, also as described above, the expiry date may not be recorded as numbers at all, but as alphabetic or other characters. Third, among the 10,000 locations, perhaps 200 would be programmed to contain constant (meaningless) information across all executions. Fourth, instances of the numeric expiry date could be planted as decoys throughout the file, but be ignored by the software. Should the licensee alter either of the last two examples, the game software could check for such tampering and react accordingly.

INFORMATION THAT CHANGES PREDICTABLY FROM ONE EXECUTION TO THE NEXT

The number of times the game program has been executed would be present in the core file and could be expected to increase by one after each execution. Successive files, then, could be searched for locations where this occurs. (Actually, the number might occupy three locations: one for the hundreds digit, one for the tens digit, and one for the units digit.) But, as with the date decoy above, perhaps 400 locations are programmed to increase by one, those locations being ignored by the software (though being checked for tampering). In keeping with the general surrogate information device, there may be no relevant numeric digits at all or numbers that increase by one but not recognizably so, e.g., Roman numerals, numbers other than to the base 10.

A similar device could be applied to the date the program was most recently executed.

RANDOM INFORMATION

The hundreds or thousands of locations that serve to safeguard the game licence–containing both real and planted information–still leave several thousand locations whose only purpose is embedment for the safeguarding information. At the end of a program execution the information in these locations should be randomly set. From one execution to the next, some of the information would change and some would not, further stifling the illegitimate search for both constant license information and changing license information. More controlled, a mix of locations could be programmed to contain the same constant meaningless information for variable numbers of program executions.

DECOMPILING

Software for computer games, exercises, learning aids, and such is created using a programming language. That original programming language is, indeed, a *language* unto itself. The code may not be *plain* English, but it does resemble English. Distributing the game software in its original language allows the licensee with not-too-difficult-to-learn knowledge of the language to readily alter the program. (Though not the norm, this may be an express allowance in the license.)

More commonly, the software is distributed in compiled form. Whatever the original programming language, that original code is “compiled” or interpreted into some form of machine language. (As a form of symbiosis, while computers “understand” machine language, they cannot operate on the original code that is understood by human programmers.) Technically, machine language is readable by computer programming experts. In light of earlier notes, that skill (guile) is not widespread and where present may not be worth the investment of time and effort.

Also requiring computer expertise (and investment), it is possible to decompile compiled code into a form that is recognizable to, and alterable by, humans. The original programmer may purposely write code that is difficult to follow, but ultimately decompiling is a means of overcoming the safeguards described here.

Prohibiting decompiling is a standard clause in license contracts, but the devices described here target licensees who would not honor the contract.

DISCUSSION

Defeating the devices described here is possible though this would require considerable premeditation and expertise.

The licensee would have to copy the accompanying core file, planning to reinstall the program software along with the copied core file and to reset his or her computer date to accomplish this. In the case of a competitive business game provision is usually made within the program for the start of a new competition. The administrator need do no more than to select this option and enter the parameters, e.g., number of industries and companies, importance weights of decisions, etc., of the new competition. The administrator need not reinstall the program software and there would be no reason for him or her to think that it should be.

The purpose of the devices described herein is to discourage circumvention of game or exercise license parameters. This discouragement can accurately be characterized as (extreme) inconveniencing rather than outright prevention. The devices, or kindred types of devices devised by programmers, can be readily implemented.