Problems of Tamper Resistant Software

Petr Hanáček[1]
hanacek@dcse.fee.vutbr.cz

Abstract: The article deals with the problems of tamper resistant software. Tamper resistant software is software that is resistant to analysis and modification. This enables to a certain extent and within bounds to trust that the software operates properly even when under a malicious attack. This property of software is very useful especially in mobile code applications and in software licensing schemes.

Key Words: tamper resistant software, tamper resistance

1 Running code in untrusted environment

Running code in untrusted environment means that programs or program fragments are exchanged between computers. Programs are thus executed on computers that could have different interests than an author of the code. Such programs are also called mobile code. Mobile code however, suffers from considerable security problems. Because it is based on the execution of mobile programs on remote and possibly untrusted computers, many observers question its fitness. The main areas for mobile code include [5]:

Software distribution: Electronic software distribution schemes usually need the ability of the program to restrict the functionality of program depending on different parameters. The example could be: time-limited functionality of program, restriction of several functions, restriction of number of runs, locking to a single computer etc. These restrictions are usually "unlocked" by licensing schemes, that allows a user to increase the functionality of program by entering a "license number".

Mobile Computing: Computing with mobile devices poses problems to networking because if the changing physical location requires continuous reconfiguration of the data links. If connectivity cannot be always maintained it also requires applications to handle extended off-line periods. Mobile software agents are very useful because they carry a request to a server, cause its execution and bring back the result as soon as connectivity is re-established. Because mobile agents can also preprocess the result (e.g. extract only the required text sections), they make better usage of the usually slow communication link between the mobile device and the network.

Active Networks: Current computer networks are based on the exchange of passive data packets. In active networks, each data packet is replaced by an active mobile code packet that carries not only data but also the instructions telling a network node how to process this packet. Although it induces some processing overhead, this instructional mode of communications gives freedom for realizing customized communication architectures.

E-commerce: Mobile agents are an enabling technology for the automation of electronic commerce. Beyond simple information gathering tasks, mobile agents can take over all tasks of commercial transactions, namely price negotiation, contract signing and delivery of (electronic) goods and services. A typical scenario is that a mobile shopping agent is sent to

1 Department of Computer Science and Engineering, Božetěchova 2, CZ-612 66 Brno

the servers of various airlines for checking the availability and finding out the best price for a given itinerary and date.

Music and video distribution: The systems for distributing music and video (sometimes called Pay Per View - PPV) allows the user to buy a limited number of music and video performances. Because the user has an unrestricted access to whole file with musical record or video, the only way how to limit the number of performances is the software mechanism, that runs on behalf of the music/video producer.

In following text we will concentrate on the problem of online software distribution.

2 Software distribution problem

The goal of on-line software distribution schemes is usually to limit the functionality of distributed software for trial users and then allow to extend the functionality for users that pay the distributor of software. The limitations of trial version of software can include:

- Disallow some functions of program for non-registered users
- Add the expiration date to the program (after expiration program stops its functionality)
- Allow only functionality for limited number of days after installation
- Allow only limited number of runs of the program after installation
- Limit a number of simultaneous execution of program on the network

All above functions can be easy implemented in the software code, but the must be protected against tampering. So the program is logically divided into two separate parts (see Fig 1) - into the application itself that perform the normal work of the program and into the *license enforcing code* that enforces the license politics of the program distributor. The main problem of the license enforcing code is the ability of code to persist an attack, in other words to be *tamper resistant*.

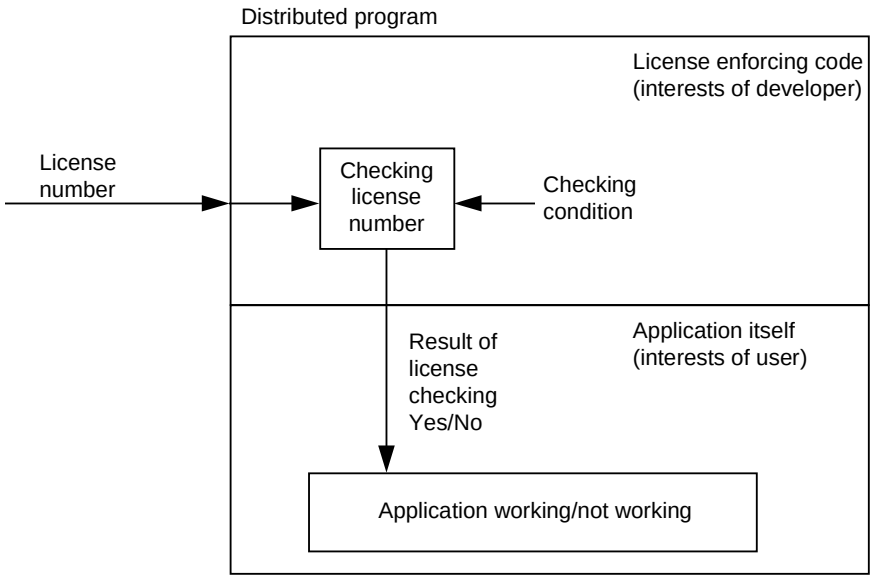

Fig. 1 General program protecting and licensing architecture

Evaluating the level of tamper resistance offered by a given product is important problem, but one which has been neglected by the security research community. One of the articles that

discuss the subject proposes the following taxonomy of possible attackers [2]:

- **Class I (clever outsiders):** They are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment. They often try to take advantage of an existing weakness in the system, rather than try to create one.
- **Class II (knowledgeable insiders):** They have substantial specialized technical education and experience. They have varying degrees of understanding of parts of the system but potential access to most of it. They often have highly sophisticated tools and instruments for analysis.
- **Class III (funded organizations):** They are able to assemble teams of specialists with related and complementary skills backed by great funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use Class II adversaries as part of the attack team.

By analyzing the problem of program distribution we concluded that the system must be resistant at least against Class II attackers.

3 Software tamper resistance

Ideally, the tamper resistant programs must be protected in such a way that an attacker executing a program it didn't create:

- Cannot read, or at least cannot understand, the program's code and data, and
- Cannot modify the program's code and data.

Many of the problems listed above required only program tamper resistance, but some rest up on a program's resilience to reverse engineering - on its ability to resist understanding. The construction of programs which are difficult to analyze is valuable to this kind of protection. The protection of the programs against tampering usually consists of a number of techniques, that include: anti-disassembling techniques, anti-debugging techniques, anti-modification techniques, and anti-decompilation techniques. The mechanism used for these techniques are following:

Anti-disassembling techniques

- Code encryption
- Nebelbombing
- Virtual machine

Anti-debugging techniques

- Hardware specific (trapping, processor errors, ports)
- Platform specific (e.g. “stack smashing”)
- Time-limited blackbox security

Anti-modification techniques

- Code encryption
- Integrity checking

Anti-decompilation techniques

- Code encryption
- Obfuscation

3 Tamper resistance mechanisms

Some of the mechanisms we will discuss more detailed in following paragraphs.

3.1 Anti-disassembling

Protection against reverse engineering seems to be the most difficult task since the program contains all the necessary information to reconstruct its algorithms.

The most known anti-disassembling method is so called nebelbombing (based on german word “Nebelbomb”). The main principle of nebelbombing is to perform assembly language jumps into operands of other instructions. These operands are then interpreted as other instructions, invisible in simple disassembly of the code. Although this technique is very simple, a normal disassembler is usually totally confused by this trick. If the disassembler is smarter and performs a control flow graph analysis, indirect jumps are used. This measure completely thwarts attackers that use only conventional tools, because the only suitable way is to write a program, that locates nebelbombs in the code and replaces them with no operation instructions. Writing of such program is not a trivial task, because a creator can use rich library of nebelbombs (with thousands nebelbombs) and he can also use parametrized nebelbombs or mutating nebelbombs that have in every occurrence different binary image.

Another tool against disassembling is to generate a part of program as a non-native code, interpreted by included virtual machine or represent a part of algorithm as a DFA (Deterministic Finite Automaton). If a code or DFA is machine-generated and is and is obscure enough, it can significantly increase the disassembling time.

An example of simple nebelbomb

Source code

```
jmp   $+3     ; Nebelbomb
db    81h     ; Nebelbomb
push  ds
push  di
mov   di,2869
push  cs
```

Disassembled code

```
jmp   29FE
sbb   word ptr [BF57],2869
push  cs
```

An example of parametrized nebelbomb

Source code

```
jmp   $+4     ; Nebelbomb
dw    $+8281h  ; Nebelbomb
push  di
mov   di,2869
push  cs
push  di
xor   ax,ax
push  ax
call  6287:06B4
call  6287:05E5
call  6287:0246
mov   di,0AA8
push  ds
```

Disassembled code

```
jmp   29FF
or    byte ptr [BF57],0069
sub   [3157],cl
rcl   byte ptr [bx+si-66],B4
push  es
xchg  [bp+si-66],sp
in    ax,05
xchg  [bp+si-66],sp
inc   si
add   al,[bx-409E]
test  al,0A
push  ds
```

3.2 Obfuscation

Obfuscation uses heuristic techniques to modify the program by transforming it in to a "messy" form that is very hard to understand, but performs the same function as the original program.

Obfuscation transformations are usually classified into four general types: layout, control, data, and preventative transformations. A key tool in implementing these transformations is the idea of opaque variables and opaque predicates which have some property that is known a priori to the obfuscator, but difficult for a deobfuscator to deduce. However, the effectivity of proposed techniques remains to be seen. The main weakness of obfuscation is the difficulty of measuring how much these transformations really make it harder for a human user (fully-armed with deobfuscation tools) to understand the obfuscated code. Often it is not necessary to discover all the rules and break all the opaque predicates to successfully attack a piece of code.

3.3 Encrypted Computation

Encrypted computation seeks to provide true blackbox security by employing provably secure cryptographic techniques to allow an untrusted host to perform useful computation without understanding what it is computing.

3.3 Time-limited tamper resistant software

Time-limited blackbox security recognizes the shortcoming of obfuscation by restricting the definition of blackbox security as follows: programs must be protected in such a way that a host executing a program it didn't create:

- Cannot read, or at least cannot understand, the program's code and data, and
- Cannot modify the program's code and data - for a certain known time interval.
- After this time interval, attacks are possible, but these attacks would have no effects.

4 Designed protocol

In the course of researching the area of tamper resistance and software licensing was designed an experimental system for licensing and protecting the software. The designed system comprises of cryptographic protocol for distributing and checking license cryptograms and a number of measures that make the protocol (and important parts of application) tamper resistant.

The cryptographic protocol is based on the strong cryptography, including one-way functions and asymmetric cryptography based on subset sum problem, that is NP-complete.

The tamper resistant package consist of combination of above techniques. The techniques are combined in order to support cryptographic protocol and make the difficulty of breaking comparable to rewriting the whole application.

Further research is now concentrated to the evaluation of resistance of designed system in comparison with similar systems.

5 Conclusion

Motto: "*There is no such thing as tamper-resistant software on a general purpose computer. If your computer can see the instructions, then you can see them, too.*"

Bruce Schneier

Finally, a successful solution to this problem (if we will commit that a something like successful solution exists) will probably be achieved by combining all presented techniques - that is, by using good cryptographic protocol for communication protection, obfuscation for program structure hiding, encryption scheme to prevent the malicious host from understanding and manipulating the code, and by using integrity tests to detect attempts to change the code.

This work was done within the research intention No. CEZ:J22/98: 262200012 - "Research in Information and Control Systems" and it was also supported by the Grant Agency of Czech Republic grant No. 102/98/0552 " Research and Application of Heterogeneous Models".

References

1. Anderson, R.: 1993, Why cryptosystems fail, First ACM Conference on Computer and Communications Security, Fairfax, VA, pp. 215-227. A shortened version of this paper appeared in Communications of the ACM, 11/94.
2. Abraham, D.G., Dolan, G.M., Double, G.P., Stevens, J.V.: Transaction Security System, in IBM Systems Journal v 30 no 2 (1991) pp 206-229
3. Matsumoto, C.: , Intel ID Protection Scheme Called Insufficient, EE Times, 01/28/99
4. Pentium III Processor Serial Number Feature and Applications, Content Group, Intel Corp., Intel Technology Journal, 1999, http://developer.intel.com/technology/itj/q21999/articles/art_3.htm
5. Sander, T., Tschudin, Chr.: Towards Mobile Cryptography, In the Proceedings of the 1998 IEEE Symposium on Security and Privacy.
6. Aucsmith, D.: Tamper Resistant Software: An Implementation, Information Hiding, First International Workshop, Lecture Notes in Computer Science, Vol. 1174, 1996
7. Protecting Programs from Hostile Environments: Encrypted Computation, Obfuscation, and

Other Techniques by Luis Sarmenta Dengfeng Gao,October 4, 1999