Tamper Resistant Software Design and Implementation

David Aucsmith et al
Presented by : Mohan Rajagopalan

November 17,1999

1 Abstract

We introduce the notion of Tamper Resistant Software. Tamper Resistant Software is software that is resistant to observation and modification. This enables to a certain extent and within bounds to trust that the software operates as intended even when under a malicious attack. We provide tamper-resistance using Integrity Verification Kernels. These are segments of code which are self-modifying, self decrypting and installation unique. This code segment communicates with other such code, thus creating an interlocking trust model.

2 Classification of Threat

Threat to a software system due to malicious users may be broadly classified into three categories based on the nature of malicious agent. The first one, the simplest to detect and handle is when the perpetrator breaches communications access controls to attack the system. The malicious agent is still under the restrictions of the communication protocol. A good example of this genre would be a standard hacker type attack. Robust access control mechanism that isolates the system hardware and software totally form the user is enough to coutner this attack.

The second, comparitively more serious attack would be like a computer virus. Such attacks originate as software running on the platform. Thought the attackers have breached the communcations, they must still depend on the BIOS and OS interfaces. Such attacks are generally the aftermath of a class I attack, and forebode class 3 attacks. Detection of these types of attacks is facilitated by the fact that the perpetrators attacks classes of files.

Class 3 attacks are the most difficult to resolve and detect as the perpetrator is an insider.The perpetrator may modify software or hardware for the system at will. The only form of prevention for this type of attacks is to raise the technoogical bar to such an extent that the perpetrator would end up wasting so much time and resources that it would be a poor investment. The technical bar coulbe be varied from no-specialized-analysis-tools to specialized-hardware-analysis tools.

This presentation is aimed at discussing techniques applied to counter class 2 & 3 type threat.

3 Design Principles

In order to make software immune to observation and modification we try to *smudge* and generally mess up the readability of the code. Another aspect would be to introduce a unique secret component to each software item which would be crucial to the running of the software. Conventaionally 4 mutually exclusive principles were developed.

3.1 Dispersion of secrets in space and time

Secret components are evenly distributed throughtout the workspace. This prevents the perpetrator from "being lucky" and discovering the entire secret component in a single attempt. Another transform would be temporal changes i.e. certain secrets are observed at certain times.

3.2 Obfuscation and Interleaving

Converting a structured, modular program to a less readable state. The general complexity of the program is increased by interleaving tasks and rewriting commonly occuring code in more un-common format.

3.3 Installation Unique Code

To prevent class attacks we must ensure that code has a unique component. This would deter a virus type class attack.

3.4 Interlocking Trust

Each segment depends not only on itself but also on other segments to effectively perform its task. Thus each segment of code would be made responsible for maintaining and verifying the integrity of other segments.

4 Integrity verification kernel

An integrity verification kernel is a small,embedded, armored segment of code which either individually or in conjunction with other similar segments guarantees integrity of state. The IVK conforms to and implements all the previously mentioned design principles. Tasks are interleaved, distributed within the IVK. Corresponding code is obfuscated, and made installation unique. A encryption or security standard will always be cracked if it follows a deterministic algorithms. IVK tries to simulate non-deterministic behaviour as far as possible.

4.1 Architechture

The IVK would be modelled as a collection of equal sized cells. All cells except for the first are encrypted usinga pseudo random generator function which takes as input one of the parameters passed in throught the entry point. The first cell is called *entry point.* The cells are not executed sequentially but in a pseudo random order which is generated using the result of execution of the cells.

The process of encrypting all cells but the first one is termed as *setting up the initial state.* Once this is done a decrypt and jump function is executed to XOR each cell in upper memory with a partner in lower memory and a substitution key. The partners are selected based on a transposition key. Thus the value of each lower memory cell is the result of a substitution-transposition operation. The decrypt and jump ensures that at any given time only one of the cells in either lower of upper memeory is in plaintext format. This operation takes in two random bit strings which are the substitution and transposition keys.

Once the decrypt and jump function has completed execution it may now use an accumulator function to store a comuter hash table value. This value can then be used by the next cell to verify its and the creators integrity.

4.2 Creation

IVKs are created using specialized tools. One of the tools is used when generating the IVK and another just prior to installation. The first step would be generation of code to compute a "cryptographic hash follwed by modular exponentiation". The public keys would be hard coded into this code.

The next step would be the aggregation with standard pre-written C code. The pre-written code would contain IVK's entry code, the generator code, accumulator code, and other tamper detection code.

This code is compiled with a standard compiler to produce relocatable object code. This object code processed and subjected to peephole randomization, branch flow analysis, cell creation and obfuscation engine creation.

The last step is to copy the IVK code to the reserved part in the compiled program.

5 InterLocking Trust

Although an IVK provides sufficient deterence to a perpetrator, we interleave integrity checking among various IVKs so as to provide a trust relationship between those. The references assume that there is a System Integrity program containing a special IVK called the entry-IVK. The eIVK has a published interface and can be used by any other IVK using the IK protocol. The IVK protocol is the mechanism by which the IVK maintains the interlocking trust amongst IVKs.

The effect of going this extra step further and adding the IVK is to increase the ammount of the code the perpetrator will have to modify in order to make a small change. Also since it will be really difficult to make changes to the system IVK we guarantee enough deterence to the client.

6 Technological Expansions

This presentation provides an insight into the functioning of tamper proof software with the use of Integrity verfication kernels. The design may be extended into two broad categories : active defense and hardware assisted protection. Hardware assisted some part of the hardware (eg. ROM) to participate in the integrity maintenence routines. The active defense approach tries to fight back when it monitors a breach in security.