LTTP Protection — A Pragmatic Approach to Licensing

Zürich, January 13th, 1994

R. Hauser and K. Bauknecht
Institut für Informatik, Universität Zürich, Winterthurerstr. 190, 8057 Zürich, Switzerland,
hauser@acm.org, baukn@ifi.unizh.ch

Abstract. Licensing is a topic of increasing importance for software publishers and users. More and more, software licensing is performed electronically by mechanisms running on the system on which the licensed software operates. In order to facilitate the use and management of such licensing systems and to enable economic software usage in enterprise-wide computer systems, various organizations are formulating requirements and defining architectures and standard interfaces for license systems. The trustworthiness of these systems is essential because large amounts of revenue can depend on them. A long term solution called Stateful Access Control (SAC) has been proposed, which aims for maximal flexibility in the definition of the license policy and still minimizes the restrictions inferred by the system. To achieve a satisfactory long-term solution like SAC requires a perfectly orchestrated effort by all partners reaching from the hardware and operating system provider to the application developer. Because this cannot be expected to happen all at once, migration strategies must be developed with modes of operation pursuing an optimum for not yet totally satisfactory intermediary states. *Tamper Evidence* is the "weak" security level addressed by this paper. A *Licensing Trusted Third Party* (LTTP) employing a minimum of dedicated security hardware is proposed as such an intermediate solution. It is an "add-on" to currently existing systems without the security pitfalls of today's licensing mechanisms and which assumes less trust in the license administrators.

1. INTRODUCTION

An increasing number of software users is currently gaining experience with network licensed software. Whether the application broadcasts its serial number across the LAN to detect copies of itself or whether a floating license system is installed or other - the experiences are not only pleasant. Also, administrators are unhappy, if they have to administer several license management systems in parallel for each offered software package. The facilitate this, important members of the industry have defined a common licensing application interface [1] which unfortunately only relies on security by obscurity due to secrets hidden in the application code. A long-term solution integrating licensing with access control has therefore been proposed with SAC [2]. This paper expects the ubiquitous bundling of highly trusted hardware with each CPU in distributed systems to be more than one technology generation away and thus this research addresses systems with only a limited amount of trusted hardware.

The purpose of this article is therefore to analyze the gap between such long-term solutions and current approaches and to develop short- and mid-term solutions on this migration path. The proposed minimal trusted hardware called Licensing Trusted Third Party (LTTP) is essentially a "trusted signing and logging robot".

Security purists may remain skeptic about the proposed solutions because their strength cannot be described with orders of magnitude of complexity for the algorithm to be circumvented, but just by informal arguments that it is "hard" to remove all the embedded obstacles and that detection and prosecution is enabled only after the fact for most imaginable attacks.

In section 2, the nature of the problem and the range of applicable assumptions are discussed and a set of comprehensive goals is developed. Section 3 first outlines an architecture based on the mentioned LTTP. Then, it is shown how the system can be made more flexible through cooperation with other interdomain-capable security sub-systems. Last, it is outlined how to accommodate multi-level sales- and distribution methods and some implementation aspects are examined.

2. THE PROBLEM

2.1. Goals and Threats addressed by current Licensing Systems.

In the past, an application developer was forced at an early stage to make the fundamental decision between implementing a licensing subsystem by himself and using an existing generic licensing system which was hard to later abandon. This very inflexible situation lead to the definition of a "License Service Application Programming Interface" (LSAPI[1]) with the following **goals,** among others:

- provide *license system independence* (G1)
- *isolate the product code from the licensing policy* (G2)
- provide the ability to establish, beyond reasonable doubt, when *tampering evidently* has occurred. (G3)

Different licensing systems can in theory be linked to applications without changing the application source if both comply to this API. This tamper evidence is defined as follows:

Code designed to be "tamper evident" requires an overt act of programming to subvert it.

"Such an overt effort can serve as evidence in copyright violation prosecutions by organizations such as the Software Publishers Association." Tampering can be detected by means of strong integrity checksums. The LSAPI authors propose to provide security by using challenges of the applications and responses from the license servers both based on a shared secret. This obsoletes transmitting the secret directly over the networks, thus makes eavesdropping and message alterations useless, and therefore addresses the following threat:

Users cannot subvert the licensing system in a properly configured system without tampering with their applications. This tampering can be made decently hard (see p. 26 of [1]).

The authors of the LSAPI, however, only address settings with entirely trustworthy administrators of license servers and they aim for an architecture which *requires tampering on a "per application" basis.* This forces subverters not only to perform one single act of tampering, but forces them to tamper constantly with each new software package and release.

Therefore they enclose the secrets in each application instead of only including it only once, for example in the file system client.

2.2. Threats not yet addressed and Additional Goals

License systems compliant to the LSAPI challenge mechanism do not necessarily counter the following **attacks**:

1) Proper license *requests* might be granted even though they originate *from a foreign domain* that is financially independent from the license serving domain. (*License stealing*)
2) The system administrator can *clone* a license server on a different machine thereby doubling the usage of the legally obtained licenses.
3) The licensing subsystem is readable by the system administrators. Therefore, they can observe its secrets and algorithms and can *write a fake license server* which intercepts all license requests and either positively grants them all or only does so by imminent exhaustion, or reroutes them to extra-domain license servers to be robbed without leaving visible traces in the existing code. Standardized APIs facilitate such attempts.

Attack 1) can happen if the licensing system is installed in an open network[1] that contains several license domains using the same licensing system and the same applications. This is understandably a concern of the buyers of a licensing system and licensed application. At a first glance, this seems, however, not to directly concern the software publishers and the license system provider. But they become aware as soon as different domains start to pool their licenses because according to capacity theory, the denial rate of a pool of 2x is lower than two separated and disconnected pools of x in allocative license policies.

Taking also these three threats into account, the proposed design adopts the goals of LSAPI and aims to also fulfil the following new **goals**:

- To motivate the system operators not to subvert the system to at least the same extent as the users, a reasonable architecture therefore also strives for *tamper evidence on the license server side*. (G4)
- Deny illicit requests from foreign domains. (G5)

2.3. Refinement of Assumptions and New Requirements

The security of current licensing systems relies much on obscurity. The concepts discussed here are not conflicting with approaches which *obfuscate code*. This is assumed to be *useful* to a certain extent for raising the threshold for potential subverters, but once the obscure information is discovered, it can easily be reproduced and is no longer protective. Therefore,

> *"Security by obscurity" measures should never be the only security barrier and therefore should only be applied in addition to other means*

such as tamper evidence. Therefore, this paper does not focus on obfuscation techniques.

To make tamper evidence an effective concept, *inspection of the code* is part of the operating procedures of the proposed licensing scheme. Therefore, the design below is based on the following additional assumptions:

1. No fire-wall machine hinders foreign, illegitimate requests to reach the license server.

- Upon arrival of an inspection team, the system is not changed by some automatic re-configuration potentially hiding subversion, i.e. it can be examined in its normal operation state. (A1)

And:

- It is assumed that the routing in a domain is never subverted in disfavour of the domain from inside that domain. No license system administrators or super-users would subvert the routing of their domain in order to permit foreign entities to obtain licenses illicitly. (A2)
- no hardware is tampered with (this is normally highly evident except for "exchange attacks"[3] where the security enforcing hardware is entirely replaced by some flawed, but from outside seemingly identical device). (A3)

Countering attack 2) becomes more delicate. This means that also the license administrator must be subjected to "tamper-evidence". The cloning of attack 2) can be prevented if the license server is aware of where it is running.

With the above assumption of properly operating hardware (A3) one could envision employing additional hardware identification information to provide server-side tamper evidence. The code of the license subsystem requests from the hardware such unalterable IDs and then compares them with the same information that was hard coded into the license server and client during installation or software distribution. Candidates for this not alterable information are the unique processor ID ("`uname -m`" on AIX machines) or the network adapter providing its physical address (e.g. "`lscfg -v ltok0`"), etc.

This approach to prevent subversions suspected from the administrator side is likely to fail due to several problems:

- Hills[4] advocates reconfigurable system calls for operating systems; `uname` is one such system call. The system call on a machine containing a cloned license server can be reconfigured to instead of reading the real processor ID just returning the ID of a legitimate server machine from which the false server was cloned. To avoid this problem, the server designer could go one step ahead in this "arms race" and not rely on the respective system calls, but query the hardware containing that ID directly (kernel access structures might allow this).
- If virtual memory management is present in the operating system, the software can be subverted not on disk and not even in memory but only when the page which performs the license checks is accessed. The subverter defines a page exception which "on the fly" patches out the relevant code to be transferred to the processor for execution.

Conclusion:

Achieving strong "tamper evidence" without controlling all software layers between the concerned software and the trusted hardware is infeasible.

Therefore, running a license server as a user mode program will not work, the solution would be to have a self-contained "license server system" without any kernel underneath. This way, system operators cannot configure and extend this system at their discretion. This requires the pertinent code to be loaded from trustworthy permanent storage. Providing also such special storage leaves this approach close to using dedicated tamper resistant license hardware anyway (e.g Palmer's Citadel [5]). If tamper-evident applications were to take the approach of running directly on the hardware only one application could be "booted" on a machine at a time.

Furthermore, tying software to hardware is cumbersome since it greatly reduces the flexibility for allocating hardware resources. Also, each component defect may lead to a significant availability decrease even though enough replacement components may be ready.

Additionally, this whole effort only prevents an administrator from "one-to-one" cloning the server with a brute force approach on a different machine. It still does not prevent the system administrator from building the mentioned "faked license server" based on the observable secrets.

The conclusion is therefore to employ message origin authentication which is not only based on secret within the code, but also on location information. The flexibility to use backup machines during defects of primary machines is restored by allowing a set of locations, i.e. machines, to run the license server code. The trustworthiness of such location information will be discussed when describing the protocol. The logs of the LTTP will be analyzed by trusted authorities and will show evidence whether the primary machines are really idle due to defects or whether both the backup and the primary machines are illegitimately in simultaneous operation. The avoidance of observable secrets in combination with this log analysis are contended to also discourage attacks of type 3).

To this point, the LTTP provides two technical services for tamper-evidence: signing and logging. These two services are only effective, if they are complemented by two administrative actions: The mentioned analysis of the LTTP log and an inspection of the code-user's infrastructure to determine whether the pertinent code has been tampered with. Tampering should become evident by an integrity evaluation *of the inactive code* constituting the system. (***Tamper evidence by observation***).

Tampering, however, can also become evident by inspecting the runtime behaviour of a system. Under the assumption (A1) that the system's behaviour is not changed when inspectors visit, this is a relatively effective approach to verify the compliance to contractually agreed license policies. (***Tamper evidence by simulation*** *of active code*)

A main component of such an inspection certainly is to verify whether a domain with n legitimate instances of a licensed application would still grant the $(n+1)^{th}$ illegitimate license request. Such inspection by simulation has the advantage that intrusive snooping around in the system to find altered code with a different name than the original application or to find cloned servers loses priority. The circumvention of this inspection process would require a totally new class of subversions such as the following one relying on the limited amount of time available to inspectors: "Fake servers are present in the system, but if usage pattern experiences a conspicuous change even the fake clones would follow the proper license policy and deny the $n+1^{th}$ request". Attacks of this level of sophistication are beyond the scope of this paper.

This analysis of the licensing problem leads to the following **new classification of achievable security goals and necessary measures** to get there.

- security can be *enforced* only if highly tamper resistant hardware is connected to each involved CPU by a trusted path. No tampering of software is possible without tampering this hardware.
- security *tampering can always be made evident* if all software between the tamper prone code and the properly executing hardware is trusted.

Any weaker configuration still can make tampering *hard* but no weaker security criterium has been defined to which it reasonably could comply. Also, it is always subject to the following **trade-offs**:

- a layered software architecture provides for portability and flexibility. However each layer also introduces attack points for subverters.
- Policy independence was originally stated as a goal, but this is directly conflicting with the goal to force a subverter to tamper on a per application basis.

3. THE LTTP APPROACH

In this section, a new architecture is proposed which employs some minimal and trusted hardware in just one location of the system. For each license domain, this LTTP is equipped with a different asymmetric key pair.

It forces the subverter with administrative privileges at least to tamper at the client side although not necessarily evident inside the concerned application if the licensed code does not directly run on the hardware. For that purpose, the server side is split in two parts: the license server under full control of the license administrator and this additional licensing trusted third party (LTTP). This LTTP can also be present in the system several times in redundance to provide high availability and thus be resistant, for example to equipment failures. In order to allow other license servers inside the same domain to access the LTTP when for example their collocated LTTP is analyzed or in repair, the LTTP must not only exclusively be addressable locally, but also over the network as if it was a separate host, even though the it might be in the same physical machine. If multiple LTTPs are present in a domain, the logs must be merged for the mentioned analysis.

This LTTP is assumed to place a threshold high enough to significantly discourage tampering and thus to justify the extra effort of managing this additional hardware device.

3.1. Architecture

In figure 1, $E_y(X) = Z$ is the notation for the encryption of X by the public key of Y, $D_y(Z)$ is the decryption of cryptogram Z and yields X again. If Z is a plaintext, $D_y(Z)$ is a signature of Z which can be verified employing E_y by the assumed crypto-algorithm.

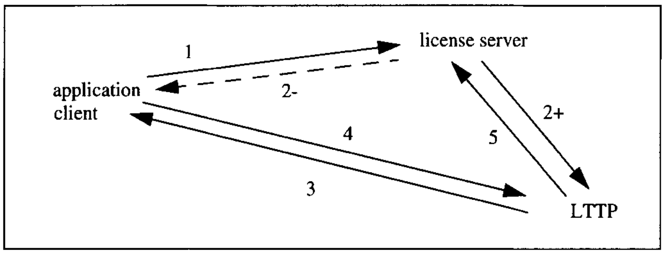
3.1.1. License Request

Flow 1 contains an encrypted nonce produced by the application. This nonce is present to link the license request phase with the license release phase. Only the LTTP and the application itself know this nonce. An attacker can only know this nonce illicitly if the applications nonce generation process is subverted[2] or the application keeping state about this nonce is inspected

2. This nonce generation is implemented maximally avoiding system calls.

and potentially altered in memory at run-time. This approximates a "per instance" tamper evidence. Flow 1 contains furthermore either a previously obtained nonce from the license server or a timestamp of the application which is only partially trustworthy.

FIGURE 1: Protocol including Release



Flow 1: [a, A, $E_{LTTP}(n_a)$, n_s, t_a, ID_a(pr,na,...),X_a], H ([..] , E_{LTTP})

Flow 2: D_s (a, $E_{LTTP}(n_a)$, n_s, L, t_s, s, ID_s(pr, na, ...), X_s)

Flow 3: D_{LTTP}(Flow2, s_{LTTP}, X_{LTTP})

- User Operation -

Flow 4: E_{LTTP}(a, n_a, L) (Release Start)

Flow 5: D_{LTTP}(a, L, a_{LTTP}, t_{LTTP})

Legend:

a	= the application host address (= license client address)
A	= application identifier/name
$E_{LTTP}(n_a)$	= nonce encrypted by the application
L	= the license token
s	= the license server address
t_a	= the application client timestamp
ID(pr,na, ...)	= processor ID/network adaptor ID/... of the application host/license server
X	= optionally further license policy relevant data like expiry etc.
ASP	= application software publisher
H(X,Y)	= a hash of message X integrity-protected by secret Y

License stealing from outside is avoided by authenticating the license request from the application to the license server. This authentication uses location information of the application (A, ID_a(pr, na,...)) and the LTTP's public key. The license server maintains a list of the permitted location information of its clients. This location information and the public key are both not true secrets preventing the attacks 2 and 3, but they can pragmatically be assumed to be relatively secret with respect to "license domain outsiders".

Because everybody inside the domain knows the E_{LTTP}, there is little protection against one insider spoofing another. There is also little incentive to do so except for the license release message as explained below. One insider spoofing another, becomes a problem if license usage is accounted by the domain with an "individual charge-back" system. Therefore, the recommendation is not to individually account for usage unless a system is in place which is capable of strongly authenticating each user as outlined in section 3.2.

Under normal circumstances, the sender address in the packet is sufficient to filter out illegitimate foreign license requests. Outside attackers could forge their packet's sender addresses to be from within a license domain even though in reality they come from outside. This can significantly block licenses until the license server and LTTP recover from their inability to deliver the response. However, the intruder would not gain any direct advantage by this. If no precautions are taken, several other threats exist on this level:

1) an intruder could intercept the routing protocol updates between the routers of the license domain and its neighbouring domain and cause internal routers to believe an external machine to be part of the internal domain.
2) packets from foreign domains could flood the license server and therefore lead to a denial of service situation.

Re 1: There are two approaches to counter this threat:

- the interdomain routing protocol updates can be authenticated if a trust relation exists with some entity in that domain [6].
- if no trust relation exists with outside routers and still connectivity is to be maintained, the domain internal routers must be able to implement rigid semantic checks. This forces routing updates with foreign domains only to concern the availability status and requires topology changes either to be within the scope of sophisticated semantic rules or to be installed by manual configuration.

Re 2: If trust relations to entities in foreign domains exist, each datagram can be tagged with a visum [7] and be prevented from leaving its home domain by its home gateway. Alternatively, internal routers could prevent external packets from reaching the license server application by essentially removing all external routes to that application. If the application ports are served by the same physical connection and network adaptor as the other ports of the same host, the described sophistication of filtering at the transport level is useless against denial of service attacks. Even though the license server port is theoretically open, the physical resources can be exhausted by the intruder by flooding another port on the same machine which is not filtered out. Therefore, even though not considered by the initial scenario layout of this paper, firewalls preventing all traffic to reach the license servers on the network level may well be a reasonable solution.

Internal location information

If the authority about the license server and about the network are separated, a Network Trusted Computing Base NTCB (see for example "TCP/IP Security par 45" in [8]) can be installed based on a trusted network architecture [9]. However, if these authorities coincide, there is little which can be done against local fake servers or server clones. One can hope that the effort for message interception or kernel alteration[3] to provide wrong network adapter numbers and

3. Especially consistently maintaining such alterations without relinquishing to other services and update benefits depending on the same features.

processor IDs etc. combined with the not negligible probability to be discovered during inspection is discouraging enough.

Flow 2 is signed with the private key D_S of the license server. It is assumed that the license administrator who is the only party able to observe that key will not convey that key to anybody who could use it in disfavour of the domain. The license server decides whether to grant the license very much in the same way as current license servers do. Negative responses are directly sent to the application because the LTTP is supposed to only log positive license events. Negative responses are also signed with D_S in order to prevent denial-of-service attacks from outside. Positive responses are furthermore enhanced by a timestamp, a license token, and potentially some policy information to be evaluated by the application. This license token contains a unique license identifier, the list of locations of permitted license servers and backup machines, and possibly further license policy information such as expiration date of the license etc. This token is evaluated and signed by the application software publisher (ASP) during the sale of the license. In principle, the D_S would identify the license server sufficiently but to force the attacker with administrative privileges more towards tampering, the location information of the license server is added. The attacker therefore has to subvert this location retrieval function to make the one-to-one cloning attack successful.

The LTTP shows the following behaviour when constructing **flow 3**:

1) it analyzes the incoming messages (flow 2):
   - messages *from* a foreign domain are rejected,
   - messages whose license server timestamp is out of the tolerance window are considered replays and are discarded,
   - messages with a mismatch between the license server address in the data part and the protocol information header are discarded,
   - messages *to* a foreign domain are permitted,
2) it then adds to the message:
   - a timestamp
   - the sender address of the license server which it retrieves from its own networking device (thus from the header information of the network packets and not from the data part).

   Then, this message is signed and sent to the application.
3) it contains a write only memory/WORM for an activity log where all relevant data is written to allowing a log analysis after the fact.

Re 1: With this mechanism, the LTTP delegates the decision whether to accept a request from a foreign domain or not to the license server and its authentication and access control infrastructure. Illegitimate requests are discarded by the license server. Legitimate requests from foreign domains are discussed in the section 3.2.

Re 2: The timestamp of the LTTP is present to prevent replays. The LTTP view of the location of the originating license server is provided to enable the application to check whether this is consistent with the list of permitted licenses servers inside the token. If not, the application refuses to accept the license granting message. This signature performed with D_{LTTP} is verified by the client together with the other signatures. This makes it impossible to circumvent the licensing scheme without subverting the client side because the client checks the message with

the public keys hard coded into it (E_{LTTP}, E_s, E_{ASP}, ...). These signatures can be performed with any of the known asymmetric encryption technologies like RSA[10], DSS[11], etc. The private key D_{LTTP} is unretrievably sealed into the LTTP. Questions of key exchange for the LTTP and D_{ASP} are not considered because the LTTP is assumed to be relatively cheap and potentially physically replaced as soon as its storage capacity is exhausted. The procedure to exchange all licenses and all the code with its embedded keys are considered to be too expensive if the LTTP hardware is not exchanged simultaneously.
Re 3: This provides the license policy analysis team with the raw data about past license usage patterns which gives hints about contract-compliance of the participants of an inspected system.

Another problem is the generation and installation of the applications and license servers which must be "domain-individualized" by hard-coding the public keys and location information into them. The ASPs can choose whether they want the license domain administrator to give them the domain's E_{LTTP} in order to place it themselves in the code or whether this can be delegated to some installation program. Such an installation program is certainly another target for subversion attacks. In the first case, the ASP is trusted not to disclose that public key (E_{LTTP}) to others than members of that domain because of the described approach to defeat license stealing. The same decision has to be taken for the list of legitimate license server processor IDs, license server network adaptor IDs, etc., if they are to be included into the application at all.

3.1.2. Release Phase

Flow 4: The license to be released is identified by its token. The nonce of the application is the same which already traveled flow 1-3 in the original request encrypted under E_{LTTP}. Not encrypting it alone anymore proves to the LTTP that the same application instance which originally requested the license also released it. Without this protection, an inside attacker could have the application regularly request the license, but shortly after the granting period the attacker would assemble a release message and immediately free the license even though it is still in use. Problems may arise, if the application eventually tries to renew its license after the grace period[4] and all licenses are exhausted at that time, but this is not different from the situation a user might experience after a temporary, involuntary segmentation of the license domain's network.

If the LTTP is more intelligent than just being a signing robot and it could retrieve logged messages and evaluate whether the application's nonce has already been released in another release message. If yes, it would log the request and raise an alarm, if not it could retrieve the license token and add it to the release message. The application client in this case no longer needs to send back the license token.

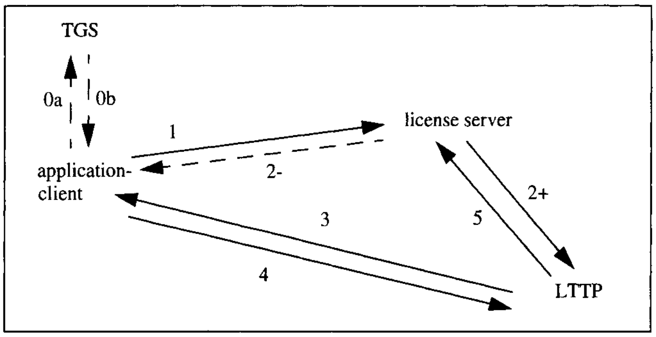
Flow 5: The LTTP also logs the release message (flow 4) together with the sender address retrieved from its own network infrastructure. Then, the LTTP signs the content plus a timestamp of its own with D_{LTTP}. If the grace period is very long, a 6th flow is recommended with which the license server acknowledges the receipt of the release message to the

4. The grace period determines for how long the application is permitted to continue operation after it fails to contact the license server. It is also the period after which the license server is permitted to hand out a license again even though it never received a release message.

application. Otherwise, an attacker could remove from the network all attempts to send a flow 5 to create a denial-of-service situation - thus, each granted license would be maximally blocked from the license server's point of view.

3.2. Kerberos Security with Generic API Available

FIGURE 2: Protocol including Release in the Presence of a System Authentication Service



Flow 1:	$K(a,s)(a, A, E_{LTTP}(n_a), t_a, X_a), T(s,a)$
Flow 2:	$D_S(a, E_{LTTP}(n_a), L, ID_s(pr, na, ...), s, t_s, X_s)$
Flow 3:	$D_{LTTP}(Flow2, s_{LTTP}, X_{LTTP})$
	- User Operation -
Flow 4:	$E_{LTTP}(a, n_a, L)$ (Release Start)
Flow 5:	$D_{LTTP}(a, L, t_S)$

Legend:

a	=	the application host address (= license client address)
A	=	application identifier/name
$E_{LTTP}(n_a)$	=	nonce encrypted by the application
L	=	the license ID token
s	=	the license server address
ID(pr,na, ...)	=	processor ID/network adaptor ID/... of the application host/license server
t_s	=	the application client timestamp
X	=	optionally further license policy relevant data like expiry etc.
ASP	=	application software publisher
K(a,s)	=	Shared Kerberos session key between a and s
T(s,a)	=	Ticket for s containing K(a,s)
TGS	=	Ticket Granting Server

If Kerberos[12] or equivalent security (e.g. KryptoKnight [13]) is available, two improvements of the protocols are possible. First, it can be used to fully authenticate the traffic between the application and the license server without relying on the partial secrecy of the LTTP public key and the location information of the application. In Kerberos parlance, the license server can be configured to be a regular application server (a ticket for this application server will be obtained with flow 0a and 0b). Second, this provides additional flexibility to a user having access to several domains. When operating in one domain, he can still obtain a ticket for the license server of his other domain and submit a request there. As described before, the signing LTTP is allowed to send responses to outside clients, but never to receive requests from outside servers and therefore supports this scenario. The license server is now allowed to accept license requests from outside domains if appropriate interdomain authentication protocols are available[14, 15] for deciding whether access is to be granted or not.

3.3. Further Issues

3.3.1. License Production Delegation

If the software publisher chooses to distribute the software by a multilevel sales organization, again the use of an LTTP is proposed to strengthen the integrity of the reseller's license creation tool. The range of licenses delegated to the reseller is specified as follows:

D_{ASP}(license ID*X* - license ID *Z*)

In order to create a valid license, the reseller has to chose one number out of the available license ID range [X..Z] and combine it with this delegated range and have it signed by an LTTP.

Two aspects must be observed by this approach:

1) Which LTTP is to be chosen for this "witnessing operation"?
   - If the reseller also maintains the local license domain, the same D_{LTTP} can be used.
   - More likely, the reseller is not located in the local domain. Then, the reseller's E_{LTTP} can either be communicated out-of-band to the installation program which puts it in the application code or the delegator always includes this key in the delegation certificate as in the example below.
2) Having a complete log is now very important in order to discourage multiple creation of the same license. However, such a log still does not prevent the reseller from copying the created license and selling it several times to different and independent license domains. This can be prevented by having the reseller's LTTP sign some identification of the targeted license domain (domain's IP mask, E_{LTTP}, etc.). A non subverted application can thus detect if it is served with an illegitimate license intended for another domain.

A generic model for implementing similar delegation structures is proposed by Gasser et al. [16], or Neuman's restricted proxies [17].

FIGURE 3: Example of a Nested License Token for a three-level Distribution Hierarchy

$$L = D^2_{LTTP}(D^2(D^1_{LTTP}(D^1(E^1, D_{ASP}(200\text{-}300, 30min, E^1_{LTTP}), E^2_{LTTP}, 260\text{-}270)), 267, test.com))$$

This is the license number 267, with 30 minutes grace period and usable in the license domain "`test.com`". D_{ASP} is equivalent to D^0. The E_{LTTP} of the target domain needs not being enclosed in the token because it is installed in the application code. If many levels of distribution are present, the tokens may grow to a size which is difficult to handle. Taking current cryptanalysis-techniques into account, each of these public keys and explicit signatures will be of at least 512 bit length. This leads to at least 4 KBit signature data for this example.

3.3.2. Implementing Tamper Evidence

First, any library to provide tamper evidence must not be implemented as a shared library nor a dynamically linked library. If the library is not statically linked with each application, the system loses the "per-application" tamper property because the users normally have the right to reconfigure and exchange such shared or dynamically linked libraries.

Furthermore, the design should support tamper evidence by simulation by supporting runtime observation of the license events. This means that the application must be able to show the license request as well as the response in a window. The same real-time observation facilities should also be available in the license server and the involved LTTPs. Therefore the mentioned simulation of the n+1th illegitimate request can be monitored at each involved entity and any message interception and alteration or omission of one of the three involved entity becomes evident.

4. CONCLUSION

The purpose of this paper was to provide an intermediate level of trust for licensing architectures without requiring special purpose security hardware for each involved entity. It was found that the tamper-evidence by observation property can be provided if the system administrators are trustworthy or the license relevant code runs directly on trustworthy hardware.

However, if the administrator's trustworthiness is arguable, only "tamper evidence by runtime simulation" of license denial situations is achievable as a "hard security criterium" under the somewhat cumbersome assumption that a system does not change its behaviour due to the arrival of inspectors.

The then proposed architecture is a compromise which is contended to be a pragmatic optimum between providing incentives to properly follow license agreements on the one hand and additional hardware and other inflexibilities burdened on the system maintenance on the other hand. This additional hardware called "Licensing Trusted Third Party" LTTP is described and the necessary protocols to effectively employ this LTTP are explained. If the LTTP is minimally configured to be only a signature and logging robot, it can even provide tamper evidence properties to applications which were originally not proponents the LTTP approach. The trusted third party analyzing the log could still detect infringements of the licensing policy of such an application if its publishers adhere to a generally intelligible policy specification.

The LTTP approach, however, lacks the following features provided by SAC:

- non-executing, "passive" data such as fonts cannot be license-protected.
- the user cannot copy/rename/patch the code available to him at his discretion without opening the possibility of escaping the license control.
- the code implementing the licensing system is redundantly present in each application binary. Therefore, it cannot afford to be very sophisticated without significantly increasing the overall system's storage devoted to license control.

For the cost of the "per application" property, instead of each application, the file-system daemon could be implemented in a tamper-evident way. Such a file system client could then implement these SAC features missing in the proposed LTTP architectures.

If the per application property is to be maintained, an application programming interface (API) similar to the LSAPI and the pertinent libraries implementing it are very desirable.

Acknowledgments: The comments from Michael Steiner, Michael Waidner, Gene Tsudik, and Phil Janson have been very helpful.

REFERENCES

[1] *License Service Application Programming Interface.* Lester Waters, 1 Microsoft Way, 5/2024, Redmond, WA 98052-6399, May 1992. available from lsapi@microsoft.com.

[2] R. Hauser. "Does licensing require new access control techniques?" *Communications of the ACM,* 37(11):48–55, November 1994. Originally presented at the 1st ACM Conference on Computer and Communications Security, Nov. 3-5, 1993, Fairfax, Virginia.

[3] W. Mayerwieser and R. Posch. "Prohibiting the Exchange Attack calls for Hardware Signature." In B. Fortrie, editor, *10th International Information Security Conference IFIP SEC'94,* Curacao, Dutch Antilles, May 1994.

[4] T. Hills. "Structured interrupts." *ACM OSR,* 27(1):51–68, Jan. 1993. A note on it in ACM OSR 28,2 p. 88ff.

[5] S. R. White, S. H. Weingart, W. C. Arnold, and E. R. Palmer. "Introduction to the citadel architecture: Security in physically exposed environments." Technical Report RC 16672, IBM Research Division, Thomas J.Watson Research Center, Yorktown Heights, NY 10598, May 1991. Version 1.4.

[6] B. Kumar and J. Crowcroft. "Integrating security in inter domain routing protocols." *ACM SIGCOMM Computer Communication Review,* 23(5):36–51, October 1993. Dept. of Comput. Sci., Univ. Coll. London, UK.

[7] D. Estrin, J. C. Mogul, and G. Tsudik. "Visa protocols for controlling interorganizational datagram flow." *IEEE Journal on Selected Areas in Communications,* 7(4):486–497, May 1989.

[8] IBM. "AIX version 3.2 hypertext information base library." CD-ROM, October 1993. 10th Edition, Bar Code for ordering: SC 23-2163-10.

[9] B. Thomson, E. S. Lee, P. I. P. Boulton, M. Stumm, and D. M. Lewis. "A trusted network architecture." Technical report, Computer Systems Research Institute CSRI, University of Toronto, Ontario, Canada M5S 1A4, October 1988.

[10] R. L. Rivest, A. Shamir, and L. M. Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Journal of the ACM,* 21(2):120–126, February 1978.

[11] U. S. National Institute of Standards and Technology NIST. "Digital Signature Standard (DSS), Federal Register 56," August 1991.

[12] J. T. Kohl and B. C. Neuman. "The Kerberos network authentication service (V5)." RFC 1510, 1993.

[13] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti. "*KryptoKnight* authentication and key distribution system." In *1992 European Symposium on Research in Computer Security*, pages 155–174, 1992.

[14] F. Piessens, B. de Decker, and P. Janson. "Interconnecting domains with heterogeneous key distribution and authentication protocols." In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 66–81, Oakland, CA, May 1993. The Institute of Electrical and Electronics Engineering IEEE Inc.: Computer Society Press.

[15] J. T. Kohl, B. C. Neuman, and T. Y. Ts'o. "The evolution of the Kerberos authentication service." In *Proceedings of the Spring 1991 EurOpen Conference*, 1991.

[16] M. Gasser and E. McDermott. "An architecture for practical delegation in a distributed system." In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 20–30, Oakland, CA, May 1990. The Institute of Electrical and Electronics Engineering IEEE Inc.: Computer Society Press.

[17] C. B. Neuman. "Proxy-based authorization and accounting for distributed systems." In *13th International Conference on Distributed Computing Systems*, pages 283–291, May 1993.