PRACTICAL VERIFICATION OF UNTRUSTED TERMINALS USING REMOTE ATTESTATION

Simone Lupetti
Department of Computer Science, University of Tromsø, N-9037 Tromsø, Norway

Gianluca Dini
Department of Information Engineering, University of Pisa, I-56122 Pisa, Italy

Keywords: Untrusted Terminal Problem, Remote Attestation.

Abstract: We present a technique based on Trusted Computing's remote attestation to enable the user of a public terminal to determine whether its configuration can be considered trustworthy or not. In particular, we show how the user can verify the software status of an untrusted terminal and be securely informed about the outcome of the verification. We present two flavors of this technique. In the first, the user makes use of a personal digital device with limited computing capabilities and a remote trusted server that performs the actual verification. In the second, the personal device is assumed to have enough computing power (as in the case of smart-phones and PDAs) to autonomously perform the verification procedure.

1 INTRODUCTION

One of the fundamental prerequisite to use a computing device is to trust it to correctly perform the requested operations. The owner (or a regular user) of a computer is usually (even if not always rightfully confident about the status of the software of his machine. The prolonged use of a personal computer, possibly including the installation and maintenance of the software, represent a natural way to build a trust relation with it.

But while establishing a trust relation by these means is possible in the case of personal and private computers, there are settings, such as for public terminals in airports, hotel lounges and Internet cafés, where such prolonged relation with a specific computing device is hard (if not impossible) to achieve. In these settings it is hard to know the practices used to maintain the terminal, as well to determine with certainty the set of programs that runs on the platform. Notice that these public terminals not only share all security problems typical of a personal machine but, because they are potentially used by an large number of users, the threats deriving from user's malicious behavior may be exacerbated. The usual way to avoid unintended interactions among different users is to map them into different roles on the machine, but this practice is not a viable option where the user base is not known a priori.

Therefore, when faced with the need to use a public terminal, a user is usually forced to choose between stepping back and potentially endangering his security and privacy. As consequence, most of users will refrain to perform any "critical" operation i.e. any operation that involves sensitive data such as passwords, PINs and cryptographic keys[1]. This means that public terminals may be, for all practical purposes, unfit to do anything from accessing a web-mail service to perform an Internet banking transaction. This dramatically restricts the usefulness of such terminals.

The problem we have described is known as the *untrusted terminal problem* and an handful of very different solutions have been proposed along the years. Some of them are based on personal trusted devices, from smart cards (Abadi et al., 1991; Berta et al., 2005) to camera phones (Clarke et al., 2002; McCune et al., 2005), others require users to make "easy" cryptographic computations (Stabell-

[1]Notice also that, while the notification that a password will be sent on an "untrusted" network is a common feature of most of Internet browsers, the notification that such a password will be typed of an "untrusted" terminal is unconceivable.

Lupetti S. and Dini G. (2007).
PRACTICAL VERIFICATION OF UNTRUSTED TERMINALS USING REMOTE ATTESTATION.
In *Proceedings of the Second International Conference on Security and Cryptography*, pages 402-407
DOI: 10.5220/0002123304020407

Kulø et al., 1999), finally, other are based on AI transformations (King and dos Santos, 2005).

Differently from the previous works on this topic, we do not focus on authenticating a single message or operation but on determining whether the whole terminal meets the security requirements of the user. This includes the authenticity of the messages leaving the terminal but also addresses further security and privacy guarantees as, for example, that user's data, such as typed passwords and browsing history, is not stored in local memory for later unauthorized use by other parties. To this end we propose the use of *remote attestation*, a key feature of Trusted Computing (TC), to give the user confidence on the trustworthiness of the terminal he is using. This technique is designed to allow a local platform to authenticate the hardware configuration and the software stack running on a remote platform to tune its amount of trust in this latter (Bottoni et al., 2006).

We present two variations of the our verification technique. In the first the user is helped by a trusted digital deice that is not powerful enough to perform remote attestation. In this case we assume the availability of a trusted remote server performing the remote attestation on behalf of the user and securely communicating him the outcome. We assume the server is well-known, providing this service as part of its business, for example. The second scenario we consider is one in which the trusted personal digital device is capable of carrying on a remote attestation procedure by itself.

The rest of the paper is organized as follows. In Section 2 we introduce the system model. In Section 3 we give an overview of Trusted Computing and remote attestation. In Section 4 we present the two variations of the verification protocol and argue their resistance to the terminal duplication attack. The protocol is then discussed in Section 5 together with the possible vulnerabilities inherently deriving from the Trusted Computing model. We draw our conclusions in Section 6.

2 MODEL

A user U wants to gain confidence about the trustworthiness of an untrusted public terminal T. This means that an attacker can control the complete software status of T. However we exclude threats in which the attacker have physical access to T and is able tamper with its hardware. With this ability a malicious party can always gain *complete* control of a machine and, under this assumption, no verification process would be able to detect tampering.

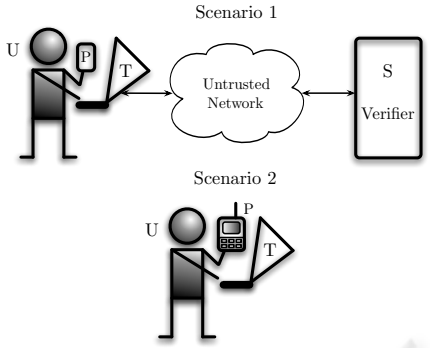

Figure 1: Representation of the two scenarios for the verification of an public terminal.

T's hardware must be conform to the Trusted Computing specifications and therefore supports the remote attestation procedure as described in (Pearson, 2002). If this is not the case, our verification procedure will fail and the user U will not be able to consider T as trustworthy regardless its actual software status.

We assume a public terminal that supports our verification mechanism to clearly show a label (for example taped on its screen frame) stating a human readable ID (more on this in Section 4.3). Notice, that because we offer security under the assumption of software-only attacks, the assumption of ID labels as a secure source of information for the user is sound (McCune et al., 2005).

To perform the verification of T's software status we assume that the user carries with her a trusted personal device P. Initially we will consider this device to have a simple screen and minimal computing capabilities. This could be a smartcard with a alphanumeric display that can be connected to the terminal T in order to exchange data with it. We assume that while P is not powerful enough to perform the whole verification procedure, it is able to compute some simple cryptographic functions such as message authentication code and to display the user a string of characters. In this scenario we assume the presence of a remote trusted server S connected to the terminal T (Figure 1, Scenario 1). The communication channel between T and S is considered also untrusted. In particular we assume that an attacker is able to read, modify, delete and create new messages to and from T.

Later we will relax the constrains on the personal digital device considering a more powerful one, such a smart-phone or a PDA, that is able to carry on the verification process autonomously. In this scenario the trusted server S is not needed anymore (Figure 1, Scenario 2).

3 TRUSTED COMPUTING AND REMOTE ATTESTATION

A Trusted Computing Platform (TPM) is a computing platform entailing a specialized co-processor for security operations, such as to perform digital signatures and data hashing or to provide protected storage. We consider the Trusted Computing Group platform's specifications as reference model (Pearson, 2002). This provides guidelines for developers to enable the design of security-related functionalities on a TPM-based system.

Remote attestation is a process with which a host can authenticate the software stack present in a remote platform. Intuitively, a trusted component called *Core Root of Trust Management* (CRTM) starts the boot process on the remote platform *measuring* the BIOS. Then the BIOS measures the boot loader, that, in turn, measures the operative system's kernel and so on, up to user's applications. A measure is usually performed by taking a SHA-1 hash of the image of the software that is being measured. This hash is stored in a TPM's protected set of registers called *Platform Configuration Registers* (PCRs). A software measurements describe therefore the whole platform's configuration (Sailer et al., 2004). In this way it is possible to detect the run of malicious (or simply undesired) software.

To prevent tampering, access to the PCRs is hardware-restricted and their value can be modified only in two ways: the first is cold-initializing the whole platform (*TPM_Startup()*), operation that set all PCRs to all zeroes but requires physical access to the platform (in case the initialization follows instead from a sleep-mode, the state of all PCRs is restored to the ones previous the sleep). The second way to modify the value of a PCR is to use the *TPM_extend()* function. This latter takes a 160 bit string and target PCR as input values. The selected register is then updated as follows: the input string is concatenated with the current value of the register, then the result is hashed in 160 bit digest to fit the register length and there stored. The value for the *i-th* update of a PCR (indicated as PCR[i]) is therefore calculated as:

$$PCR[i] = SHA\text{-}1(input\_string \mid PCR[i-1])$$

The value of a PCR can be reported either by a *TPM_PcrRead()* that produces a non-cryptographic description of the content of a set of PCRs or by the *TPM_Quote()* function. This latter requests the TPM to produce a signed message that contains the value of a set of PCRs and some externally supplied data, (typically a nonce to confirm the freshness of the signed message)[2]. The resulting data structure can be used to implement remote attestation because it assures the integrity and authenticity of the content the PCR to remote parties.

To produce this data structure,a TPM has exclusive access to an attestation identity key pair (AIK^{-1}; AIK) securely created and kept inside the TPM for its whole lifetime. Its private part AIK^{-1} is protected in a way that it can be used only by the TPM itself while a certificate issued by an external certification authority binds its public part to the identity of a specific TPM.

4 VERIFICATION PROTOCOL

We describe here how the verification of T's status is carried out by the user first with the help of a trusted personal device with limited computing capabilities P and the remote server S, and how the user is securely informed about the outcome. Then we will show how the same verification procedure can be greatly simplified by relaxing the constraints on the personal device P.

In both scenarios with T_{ID} we indicate the identifier of the untrusted terminal T. We assume this identifier to be derivable, in some way, from the public identity key AIK of T by a compression mechanism such an hash function. For example, if SHA-1 is used we have:

$$T_{ID} = SHA\text{-}1\{AIK\}$$

While details about this are still scarce, we expect a TPM identity key pair certificate to contain (at least) serial number and a series of attributes uniquely identifying the TPM (the *distinguished name*, in X.509 jargon). This means that the serial number (or any other unique attribute) can be used as T_{ID} as well. Regardless of what is used as T_{ID}, a human readable version of this must clearly visible on the public terminal.

4.1 Verification by a Remote Trusted Server

In this first scenario we assume the user to be helped in the verification process by a trusted personal device with limited computing capabilities and a minimal display and by a remote trusted server S that performs the verification on user's behalf. In particular we assume P to be able to perform message authentication encoding and decoding. In this scenario P

[2]The caller must have the necessary credentials to use the signing key to generate this message

shares a secret string of characters K with S. A high level description of the procedure is given below.

Step 1: After user's request, P produces the following one-time login token:

$$OTP = MAC_K(Time)$$

by computing a symmetric message authentication code such as HMAC under the key K. The actual data to be authenticated is a timestamp $Time$ indicating when the login attempt is being performed.

Step 2: $P \longrightarrow S : OTP, Time$
P forwards the one-time password and the timestamp to the trusted server S (using the connectivity offered by T) to initiate the verification of the local terminal. The use of the one-time password OTP is necessary because otherwise the untrusted terminal could record and reply a login token to the server S to asking a remote attestation and therefore potentially exposing S to a DoS attack. OTP serves also as a guarantee for user's anonymity because it creates unlinkability of different login attempts from the same (or different) terminal.

Step 3: $S \longrightarrow T$: *"Attest"*, N
The trusted server checks that $Time$ is properly authenticated. Notice that timestamps are here used only to prevent replay attacks and there is therefore no needs for finely synchronized clocks. The only verification done on the value of $Time$ is that this latter must indicate a subsequent instant in respect with the last login attempt from the same user. If so, the trusted server S initiates the remote attestation protocol by challenging the untrusted terminal. The challenge should include a nonce N to prevent the host under verification to re-use an old response.

Step 4: $T \longrightarrow S : \{PCRset, N\}_{AIK^{-1}}$
At this point the terminal can respond to the attestation challenge, issuing a *TPM_Quote()* to produce an answer message containing a set of PCRs describing the platform status and the nonce from the attestation challenge to prove the freshness of the reply. This message is then sent back to S. Notice that in some implementations the sole list of PCRs may not be enough for S to derive the T's configuration, maybe because the value of $PCRset$ for too many acceptable configurations should be known by S. In this case, to allow direct inspection by S of the T software status, the log of all executable activations should be sent along with $PCRset$. This latter would be also sent to S as it serves as integrity measure of the log.

Step 5: By the analysis of $PCRset$ (or $PCRset$ and the activation log, if necessary) S determines whether T is in a trustworthy configuration or not. The set of acceptable configurations may vary both in respect of the service that the user want to access (different services may require different configurations) and of the users themselves (different users may have different requirements about T).

Step 6: $S \longrightarrow P : Outc, T_{ID}, HMAC_K(Outc|T_{ID})$
S sends the terminal the attestation outcome $Outc$. We indicate here that both a positive or a negative outcome is sent back to T but, in the case the attestation fails, to send a negative outcome to T may be completely irrelevant since T, not being in a trusted configuration could drop this message. This means that $Outc$ can simply be a confirmation bit in case of positive attestation and nothing otherwise. What is important, however, is that T will not be able to produce a phony positive outcome because it does know K. S sends this message to T that forward it *ad verbatim* to P. We omitted this additional step for brevity.

Step 7: $P \longrightarrow U : T_{ID}$
The trusted digital device D verifies the integrity of $Outc$ and T_{ID}, then checks if the verification outcome $Outc$ is positive. If this is the case, it displays to its screen the terminal identifier T_{ID} in the same format as printed on T's label.

Step 8: The user compares T_{ID} that he sees on the terminal with the one displayed on his personal device. If they are equal then the user assume that the machine that S has verified is really T. Else S has been tricked into verifying another machine on T's behalf and the terminal is considered not trustworthy.

4.2 Verification by the Trusted Personal Device

In this second scenario we relax the constraints on the trusted personal device that is now assumed to able to perform the remote attestation autonomously. The verification process can this way avoid to resort on a remote trusted server resulting in a greatly simplified procedure.

Step 1: $P \longrightarrow T$: *"Attest"*, N
The trusted personal device directly initiates the remote attestation protocol by challenging the untrusted terminal. As in the previous scenario the challenge includes a nonce N to prevent the host under verification to re-use an old response. Notice that, since this message does not bring any

user identification, the privacy of the user is preserved also in this version of the protocol.

Step 2: $T \longrightarrow P : C_{AIK}, \{PCRset, N\}_{AIK^{-1}}$
The terminal responds to the attestation challenge. It first issue a *TPM_Quote()* to produce an string containing a set of PCRs describing the platform status. Then attaches to this string the nonce from the attestation challenge to prove the freshness of the reply and send it back to P. A digital certificate C_{AIK} vouching for the authenticity of the key AIK is also send along with this message.

Step 3: P verifies the digital signatures of $\{PCRset, N\}_{AIK^{-1}}$ using C_{AIK}[3]. Then P analyzes the $PCRset$ to determine whether T is in a trustworthy configuration or not. P takes this decision exactly as S does in the previous scenario.

Step 4: $P \longrightarrow U : T_{ID}$
If P finds that the terminal T is in a trusted configuration it retrieves or calculates T_{ID} and displays it on its screen.

Step 5: The user compares T_{ID} that he sees on the terminal with the one displayed on his personal device exactly as in the previous scenario.

4.3 Terminal Duplication Attack

The previous protocol (in both variants) makes use, among other things, of physical labels containing a cryptographic digest of the terminal's public attestation key AIK. In its last step the user is required to compare the value on this label with the value, that is displayed on his personal device P. This is necessary to establish a link between the terminal that the user is using and the machine that has been actually attested. Without this provision the protocol would be vulnerable to a *terminal duplication attack* where a malicious host M in an acceptable configuration is measured in place of the terminal T and the outcome of the attestation is replayed to the user as it would come from T. At this point the user would believe that the machine that has been attested is T while, in fact, is not.

Notice that the fact that the malicious host M is in a trusted configuration (otherwise the remote attestation would fail) does not help to confine the verification outcome ($Outc$ in the first scenario and $\{PCRset, N\}_{AIK^{-1}}$ for the second) to M.

While the host that performs the remote attestation (S or T) can make sure (by direct verification) that M's configuration is such that it does not allow the message containing the verification outcome to be transfered on another machine, the mere fact that it needs to outputted somehow to the user (in our case by sending it to his personal device P) requires this message to escape the (trusted) domain of the machine that has just been verified (for example by an USB or infrared port, etc.). This makes the attacker able to capture the outcome as soon as it exits M and reply it at will to any other machine.

[3]We assume that P is able to verify and use C_{AIK}

5 DISCUSSION

Our technique allows the user to establish the trustworthiness of the whole platform she is willing to use. This means that, once the verification protocol has successfully run on a public terminal, the user does not need anymore to be concerned about possible malicious behavior of this latter. While most of other works in this field deal on how authenticate a single message coming from an untrusted terminal, we focus instead in how determine once-for-all if the terminal can be safely used by the user (at least within the same session). We believe this to represent a clear advantage over the other available methods in terms of usability. The downside of using remote attestation is that hardware compromization would render tour method ineffective, while the approaches aiming to authenticate each single message are immune to such a threat.

While the trustworthiness of the public terminal T is established by measuring its software state using Trusted Computing's remote attestation, some concerns have been expressed about TC from the technical point of view.

While this remote attestation gives strong assurance on what runs on a platform, it is still unable to cope with trojans exploiting the inevitable vulnerabilities present even in trusted software (Oppliger and Rytz, 2005). Also, the requirement that *all* code containing a runtime that could potentially run in an general purpose platform (including interpreters, virtual machines, etc.) must be instrumented to allow measurements of the code it runs, may be hard to meet. Vendors of closed source, proprietary applications may not be motivated to provide an instrumented version of their software (Reid and Caelli, 2005). However, both these concerns are less pressing in the case of a kiosk-like public terminals because these latter usually run a minimal version of the operative system and limited number of applications (typically a just web browser). This would reduce on one hand the overall size of the software running on the machine, hence leading to expect less vulnerabilities and, on the other hand these particular settings

may motivate vendors to provide ad-hoc version of their software (e.i. web browsers, virtual machines etc.) instrumented for remote attestation to enter the market of public terminals.

To date it remains unclear, however, how the link between a certified platform configurations and it trustworthiness should or could be established. While a given set of PCR's values (or a log whose integrity is guaranteed by the PCRs) must be certified to be used in remote attestation, it is still uncertain on who will provide such a certification and which criteria will be used for this purpose.

Other concerns have been expressed about the ethical and social implication of the complete verification of the software state of machine typical of TC (Anderson, 2003). However we believe these to be here less relevant because what is measured is a public terminal, not a general purpose, private machine where remote attestation may expose user habits and private data. We argue therefore that the measurement of a public terminal does not threat the user's privacy just because of its public availability.

6 CONCLUSIONS

We have presented a mechanism to allow the user of a public terminal to gain confidence about its software state using Remote Attestation a technique provided under the umbrella of Trusted Computing. Our technique comes in two flavors: in the first one, the limitations due to a minimal personal device are overcame by using a trusted remote server that performs the terminal verification on behalf of the user. In the second, we relax the assumptions on the personal digital device and assume that it can autonomously verify the public terminal without the help of a trusted remote server.

By the use of identification tags posted on the public terminal and the trusted personal device, the user is presented a reliable confirmation that the attestation process has been successfully carried out for the machine she is actually using.

Our solution also takes into account user privacy by enabling her to use a different identification token to initiate each verification process. This makes hard for a set of colluding terminals to link the login attempts of the same user to the verification service.

We believe that our mechanism may effectively enable the use of public terminals also for handling sensible user data without threatening the privacy of user's data.

REFERENCES

Abadi, M., Burrows, M., Kaufman, C., and Lampson, B. W. (1991). Authentication and delegation with smartcards. In *TACS '91: Proceedings of the International Conference on Theoretical Aspects of Computer Software*, pages 326–345, London, UK. Springer-Verlag.

Anderson, R. (2003). Cryptography and competition policy: issues with 'trusted computing'. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 3–10, New York, NY, USA. ACM Press.

Berta, I. Z., Buttyán, L., and Vajda, I. (2005). A framework for the revocation of unintended digital signatures initiated by malicious terminals. *IEEE Trans. Dependable Secur. Comput.*, 2(3):268–272.

Bottoni, A., Dini, G., and Kranakis, E. (2006). Credentials and beliefs in remote trusted platforms attestation. In *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 662–667, Washington, DC, USA. IEEE Computer Society.

Clarke, D. E., Gassend, B., Kotwal, T., Burnside, M., van Dijk, M., Devadas, S., and Rivest, R. L. (2002). The untrusted computer problem and camera-based authentication. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 114–124, London, UK. Springer-Verlag.

King, J. and dos Santos, A. (2005). A user-friendly approach to human authentication of messages. In *FC 2005: Proccesings of the 9th International Conference on Financial Cryptography and Data Security*, volume LNCS 3570/2005, pages 225–239. Springer Berlin / Heidelberg.

McCune, J. M., Perrig, A., and Reiter, M. K. (2005). Seeing-is-believing: Using camera phones for human-verifiable authentication. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 110–124, Washington, DC, USA. IEEE Computer Society.

Oppliger, R. and Rytz, R. (2005). Does trusted computing remedy computer security problems? *IEEE Security and Privacy*, 3(2):16–19.

Pearson, S. (2002). *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Reid, J. F. and Caelli, W. J. (2005). DRM, trusted computing and operating system architecture. In *ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 127–136, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Sailer, R., Zhang, X., Jaeger, T., and Doorn, L. V. (2004). Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 223–238.

Stabell-Kulø, T., Arild, R., and Myrvang, P. H. (1999). Providing authentication to messages signed with a smart card in hostile environments. In *Proceedings of the 1st USENIX Workshop on Smartcard Technology*.