Trust[ed | in] Computing, Signed Code and the Heat Death of the Internet

Jonathan A. Poritz
Hagenbuchenweg 20
8602 Wangen, Switzerland
jonathan.poritz@gmail.com

ABSTRACT

The *Trusted Computing Group* (TCG) is an industry consortium which has invested in the design of a small piece of hardware (roughly a smartcard), called a *Trusted Platform Module* (TPM), and associated APIs and protocols which are supposed to help increase the reliability of TPM-endowed computing platforms (*trusted platforms*). The TCG envisions that boot loaders, OSes and applications programs on trusted platforms will all collaborate in building a cryptographic hash chain which represents the current execution state of the platform, and which resides on the TPM. Remote sites can then verify that the platform in question is "in a trusted state" by requesting the TPM to produce a signed data blob containing the value of this hash chain, which can then be compared against a library of recognized ("trusted") values; this process is called *remote attestation*, and the whole picture is sometimes referred to as *integrity-based computing* (IBC).

We argue that there is a fundamental gap between the stated goals of the TCG's IBC and the central technology that is intended to achieve these goals, which gap is simply that remote attestation asks the attesting platform to answer **the wrong question** – the platform is not attesting to its **security state**, but rather to its **execution state**, and this underlies all of the troublesome use cases, as well as a number of the practical difficulties, of the TCG worldview. One response to this is to replace standard TCG attestation with *property-based attestation* (PBA), which places the emphasis on deriving security properties from (potentially) elaborate trust models and conditional statements of security property dependencies. Herein the central rôle for IBC of trust and deriving consequences from precise trust models becomes clear.

Finally, we claim that the TCG's own remote attestation is most properly viewed in fact as a form of PBA, with a certain simple trust model and database of security properties. From this point of view, it becomes clear that IBC can have a much less restrictive range of applications than envisioned merely by the TCG. In fact, with the right "trust infrastructure" and sufficiently open software using and relying upon this infrastructure, IBC could actually realize some of the portentous early promises of the TCG for truly increasing the reliability of individual users' platforms and pushing back the apocalyptic rise of malware, especially if platforms and OSes virtualize and enforce some kind of signed code contracts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'06 April 23-27, 2006, Dijon, France
Copyright 2006 ACM 1-59593-108-2/06/0004 ...$5.00.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*authentication, cryptographic controls, invasive software*

General Terms

Trusted computing, trust model, hypervisor

Keywords

Remote attestation, binary attestation, property-based attestation, signed code, security properties, trust language, security property language, OS virtualization, web of trust

1. INTRODUCTION

All information security professionals are aware of the exponentially increasing curves of known vulnerabilities in widely distributed software and the similar curves of reported incidents of malware in the wild. While the networked community would be happiest if old vulnerabilities could be eliminated and new ones not introduced, a minimal first step towards improved security of computing platforms would be at least to know if the platforms are running their intended software or if they have instead been infected with malware – something which has itself to date proven difficult to achieve. One approach to realizing this minimal improvement has come to be known as *integrity-based computing* (IBC – or, also, as *trusted computing*), and is built upon two elementary components: first, a platform must keep a running snapshot of its execution environment, usually inside some mechanism of heightened security (such as separate security hardware or protected software partition); second, the platform must be able to report this snapshot reliably (in appropriate ways, to appropriate recipients) over the network.

IBC has an extensive overlap with the hardware, protocols and accompanying infrastructure proposed by the Trusted Computing Group (TCG; see [10]) industrial consortium, but need not necessarily be limited to this; for the TCG,

IBC is hardware assisted, using a component called a *trusted platform module* (TPM). In fact, TCG member companies have already released products containing TPMs: in fact, it is estimated [6] that hundreds of millions of TPMS will have shipped with laptop and desktop computers by 2006; these are various machines from IBM, HP, Fujitsu and others. Furthermore, Microsoft's *Next Generation Secure Computing Base* (NGSCB), a security strategy for future releases of their operating system, has many points of contact with particular features of the TPM, to the extent that NGSCB seems only possible in its fullest, more secure form on a platform which has a hardware TPM. A number of Linux kernel modules and user-space tools which work with TPMs are also being released by several corporate and open-source actors.

It therefore appears that the TCG grand strategy is being given a good trial in the world marketplace, at least in so far as widely distributing the necessary hardware and providing ample operating system support in OSes used by the great majority of users. One last component whose general availability is unclear at the moment is the infrastructure, public-key and other related cryptographic, which is necessary for the full TCG world-view. It remains to be seen if this infrastructure will ever actually be created, but the TCG's inertia makes it worthwhile to examine carefully their true security efficacy, and to prepare for other IBC solutions.

In this paper, we pursue such an examination, looking into the basic security that can really be granted by an IBC approach. After reviewing the TCG's approach to IBC in §2, we take a first step, in §3, by reconsidering the fundamental mechanism of a platform communicating its state to a remote verifier. Our conclusion there is simple but important and profound: the TCG's proposed IBC mechanism **answers the wrong question.** A TCG *trusted platform* (a platform with active TPM) **can** reliably collect and report information about its **execution state** by following the TCG's recipes, but the gap between **execution state** and **security state** is a very large one, and one into which fall all of the concerns of various outside commentators about the consequences of the TCG in terms of privacy and business practices, *etc.* This conceptual gap also motivates the idea of *property-based attestation* (PBA), wherein the real security-related question from a remote verifier is answered, at the cost of additional infrastructure of various kinds. PBA usefully brings out the rôles of complex trust models and databases of security-related statements (offered by particular entities which a verifier may or may not trust). This clarification of rôles within the TCG infrastructure is useful to the community. It is also particularly important given that we shall point out, at the end of §3, that the TCG's remote attestation is in fact a form of PBA, albeit one which attempts to conceal a very simple set of trust models and security assertions.

In §4 we imagine how the ideas and mechanisms of IBC, when informed by the discussions of the previous sections – in particular, that all remote attestation is PBA, and PBA is all about trust and trust models – could lead to better, more reliable and flexible IT security, both remotely and even on a particular local platform. It is in this last context, in particular, that we consider the advantages of such technologies as OS virtualization and signed code, which could, we conjecture, lead to some hopeful battles against the ever-rising malware curves and the eventual heat-death of the Internet which they imply.

2. THE TCG'S (BINARY) ATTESTATION

In this section, we describe the TCG's approach to IBC. As mentioned above, the first fundamental idea of IBC is to keep a running snapshot of the execution state of a given platform. The TCG increased the security of this snapshot by keeping it in a separate piece of hardware, called a *Trusted Platform Module* (a *TPM*). A TPM has a small amount of its own memory (some of which is non-volatile) and computational power, and communicates with the host platform only via a limited, carefully-designed API. The non-volatile memory of the TPM can hold long-term secrets, such as the private part of an asymmetric key-pair which is generated on-chip and never leaves the TPM, and shorter-term data, such as the current execution snapshot.

2.1 PCRs

The mechanism provided by the TCG for this last purpose is a collection of *platform configuration registers* (*PCRs*), which are reset at boot time and then updated every time a new executable is loaded, with the hash of that executable added to the end of a hash chain. After the BIOS executes, the OS loader, the OS itself, and each new executable to be run within the OS are, in turn, hash chained into appropriate PCRs, each hash chaining occurring **before** execution control is passed to the new binary. In this way, while certainly binaries may be loaded which seize control of the OS and refuse to continue the TCG contract of hashing before executing further binaries, these negligent binaries will have been recorded and so the PCR values will have the tell-tale traces of this faulty binary already registered.

As a particular example, if some host has BIOS which hashes to the value β, OS loader to λ, OS kernel to κ, loaded OS modules (such as drivers for specialized hardware) with hashes $\chi_1, \ldots, \chi_{n_m}$, loaded user-space applications with hashes $\chi_{n_m+1}, \ldots, \chi_{n_m+n_a}$, then some PCR (say the jth) will have value

$$PCR_j = H(H(\ldots H(H(H(H(\\ 0^{160} \parallel \beta) \parallel \lambda) \parallel \kappa) \parallel \chi_1) \parallel \ldots) \parallel \chi_{n_m+n_a}) \quad (1)$$

where here $H(\cdot)$ is the hash function SHA1 (for the TCG at present) 0^{160} is a 160-bit string of 0s, and $\parallel$ represents concatenation.

2.2 Binary Attestation

The second basic feature of an IBC solution is the ability of the platform to report reliably its current execution snapshot – which we now see as encoded in PCR values – to a remote site. For our purposes here, it suffices to know that a remote site can request a *TPM Quote*, which is a data blob containing the current values of (some of) the PCRs and followed by a digital signature. This signature is built using an identity the host has previously constructed, in negotiations with a trusted third party (TTP) and using various vendor certificates and other data. The entire protocol for creating and conveying this quote is called *remote attestation* by the TCG, although we shall call it *binary attestation* (PBA) for reasons which will soon become clear (in §3.1).

Various of the details of these protocols, which we have suppressed in our high-level description here, are designed,

for example, to prevent replay or man-in-the-middle attacks.

3. REAL (PROPERTY-BASED) ATTESTATION

3.1 Mind the Gap

The TCG has received a fair bit of criticism from a number of directions and sources, including fervent protests from privacy advocates, grumblings from free-market advocates, careful analyses of theoretical or practical weaknesses or performance problems, and so on (and on; see, *e.g.*, [2], [1], [5], [9], [3], [7], *etc.*). In some sense, these criticisms are almost all based on a problem with the IBC as we have described it: giving a querying remote verifier some indication of a platform's execution state **answers the wrong question**: the remote verifier has a (quite legitimate) interest in the security of the client platform, but **execution state** is not the same thing as **security state**. It is by driving a wedge into a semantic gap such as this one that unscrupulous actors (from pimply crackers to major corporations) can misuse a technology for self-serving ends.

Just to give an example of TCG use cases which make outsiders nervous, consider that even after great effort to anonymize the actual TPM/platform identity, a particular (ordered) constellation of executed modules may give away an enormous amount of data to the remote verifier – which data, as we have seen, is in fact **not** directly in answer to the verifier's security concerns – to an extent considered privacy-invasive; this is the *big brother scenario*. Another use case which causes nervousness consists of when a vendor with a strong, diversified presence on the network refuses to provide any services to client platforms whose hash chains contains signs of a competitor's software; this vendor lock-in is the *evil empire scenario*.

If we take a step back from reality to consider the high-level view of what is supposed to be going on here by, in particular, pretending that the hash function used in our IBC technology is absolutely perfect (actually a very dubious proposition at the moment, see [7]), then some things become quite clear: We are using the outputs of the hash function $H(\cdot)$ as *universal names for binary executables*, and the value of a PCR's hash chain as a perfectly compressed version of the list of these universal binary names for the executables which have in fact run on the given platform (protected, then, by the PKI-supported signatures on Quote blobs).

3.2 Property-Based Attestation

Perhaps a better approach to IBC would be to answer the right question: a remote verifier should be able to ask a platform for its security state, or, at least, the correctness of certain security properties. These properties need to be expressed in a language which includes Boolean combinations and perhaps even more elaborate compound statements (such as with quantifier, time constraints, *etc.*), whose simple statements must include assertions about the properties of the platform hardware, software which is or is not running, values output by certain processes running on the platform, *etc.* An example statement, which might be of particular interest to a server considering whether to stream digital content to a client platform, might be:

> This platform has a protected path to its screen and speakers. It is running a recent OS. It has run an anti-virus scan in the last 24 hours whose output was "clean". It is not presently running a debugging tool which can examine the contents of arbitrary memory. It is either running no Internet services, or they are in recent, recognized versions and have configuration files with acceptable parameters. There is no system administrator logged in at the moment. The platform user has performed an acceptable authentication protocol with the server to prove she is a paying customer. A tool is running whereby the user has waved the right to change any of these security properties until the next reboot, before which the streamed content will be completely removed from the platform.

There is a some research in the direction of such security property languages; see, for example, [4] for a general and powerful such language used in a different context, or [8] for a very simple language used, however, exactly in the situation we are describing.

A platform can make a direct statement to a verifier of its security properties, but then, of course, the question is why this statement should be trusted. One answer (following [8]) is to have the security statement produced by another agent, probably running on another machine (but see below §4, for another possibility). This machine could be running only certain well-known, simple software to create the property-based security statement and would be trusted, in turn, because the verifier could run binary attestation with it.

Properties of the form of those just mentioned can be verified by (an elaboration of) the TCG's IBC techniques, with a security guarantee similar to that of simple binary attestation. Communicating these properties instead of the raw binary attestation data, however, has several advantages. Privacy is protected by preventing the leakage of any information not pertinent to the relevant security issue. Similarly, restrictive business practices can be forbidden. Actually, a security properties language would certainly permit such privacy-invasive and business-restrictive statements to be made, but at least the server would have to make its practices in this regard explicit and public, when requiring these properties of clients, and a particular client could simply refuse to answer such queries.

We use the term *property-based attestation* (PBA), then, for a protocol to query and report encoded security property statements of the form just mentioned (perhaps via a trusted third party). Features of binary attestation to prevent replay or man-in-the-middle attacks must be repeated for PBA, but there is one extremely significant additional data flow:

3.3 It's all about trust

Execution history can be retrieved by the method of the TCG, and elaborations of this method allow all of the above-mentioned simple (non-compound) security property statements. But, in the end, a mapping must be made by some agent between the available binary data and the semantics represented in a security property statement, which mapping a particular remote verifier may or may not choose to trust. It thus becomes clear that the trust model which the verifier wants to apply, or the model which our platform must apply in order to evaluate a particular security property, is

completely determinant. Additionally, the mapping between binary data and security properties must be supported by knowledge databases of certificates created and maintained by certain agents, which may or may not be trusted by the verifier. In fact, some certificates could be produced "on the fly", by some analysis tool running (on the platform, or the TTP) with input a particular binary executable (and perhaps other data) about which one desires to know the security properties – but, then, certificates of the reliability of this particular tool in a particular situation are needed as well.

Hence one extra data flow in the PBA protocol of [8] communicates the trust model, which must also be encoded in some language, either from the verifier to the platform, when the verifier wishes to insist upon a particular model, or the other way, when the verifier wishes to know if a particular property can be verified at all by the platform (in which case the platform will reply essentially with the most restrictive trust model under which the property could be verified, if it can at all be). Another flow is between the PBA TTP and various certificate databases, in order to drive the derivation of properties statements from binary data.

Introducing yet another complex component of the whole security picture, in particular one which requires a specific language to describe, may seem to be a net loss to the community. But we believe it is impossible to achieve any real security without making explicit the various underlying and sometimes hidden assumptions from which security statements are to be made. Furthermore, there is room for a great deal of flexibility in this new component, the trust model, which allows even for very naïve users: a typical home user might simply choose to trust all statements made by their OS and hardware vendor, or, if she were somewhat more ambitious and security conscious, she could pay a third party security firm and choose only statements they make (certify) and statements made by agents to whom the security firm has delegated authority. A corporate intranet (or value-added ISP subnet) could have home-grown certificate databases and pro-active tools and updates in addition to, or instead of, the main vendors. Finally, hackers and security experts could self-certify and choose to delegate mutual trust in the way of current open-source community webs of trust.

3.4 BA *IS* PBA

With the perspective of PBA now at our disposal, let us briefly consider the simple binary attestation of the TCG. When a particular verifier asks for a TPM Quote and then decides whether or not to trust the platform based on the returned data, it is clear that in fact the verifier is merely deciding upon a BIOS, OS loader and kernel and modules, all with hashes of β, λ, κ, and $\chi_1, \ldots, \chi_{n_m+n_a}$, that, for it, define the security property "acceptable". That is, the security property is

> Platform ran BIOS with hash β, OS loader with hash λ, OS kernel κ, and then modules with hashes $\chi_1, \ldots, \chi_{n_m+n_a}$ (or some subset thereof), in some order.

In addition to this very simple security property, the verifier has an equally simple trust model – beyond the basic and universal issues of trusting the TCG itself, its CAs and other parts of the TCG infrastructure, and hardware vendors) – where only the specific hash values the verifier has pre-loaded into this security property statement are to be trusted, and so no certificate database need be consulted (or trusted).

Pre-loading hash values into this simple security/trust specification resident in the verifier seems at first simpler, but in fact merely conceals all the difficulties. The values are not likely to remain static for very long, so each verifier must update its specification internally with some frequency. The process of such updates is then exactly what we are describing above in a more general, automatic and reliable form, as verifying the trustworthiness (with regards to some existing, if unstated, trust model) of a certificate which, when unwrapped, contains a new hash value to be trusted.

In fact, doing the computation inherent in the above security property (*any subset* and *any permutation* are allowed, so hash chains must be verified on the fly, as the number of potential acceptable PCR values grows factorially with the number $3+n_m+n+a$ and so they cannot be pre-computed) is probably better done nearer to the verified platform than the verifier – it should be in the platform's interest to do this work, more so than for the verifier.

Hence binary attestation can be seen as a form of PBA where the computation is done in a less appropriate place, checking, using an almost tautologically simple trust model, the validity of an extremely simple property. This simplicity makes for excessive rigidity and renders updates (a large component of which would be essentially a human-operated version of the automated PBA described above) both very awkward and very frequently necessary.

4. BETTER IBC

Let us imagine that sufficient PKI is in place, well-designed security-property and trust-model languages are created and deployed, along with comfortable tool support and commercial enterprises exist to serve as TTPs or existing companies (security firms, hardware vendors, ISPs, *etc.*) are willing to offer these services as product differentiators or for further revenue. What then should be the real maximum possible security benefits for platforms, locally and in term of trustworthy remote e-commerce? We consider briefly in this section a possible answer, and how IBC can help it to come about.

One starting point is the fundamental contract which underlies the security of hash chained PCRs for the TCG – each successive binary which has the potential of spawning further future binaries must agree to extend an appropriate PCR with the hash value of its child processes *before every* such child process is executable. But this contract could also be more pro-active: as each child's hash value is added to the chain, the security property consequences could be computed, according to a trust model that the user has chosen on her platform (and/or one which she knows will be relevant in a future remote negotiation), the executing binary could choose whether or not actually to spawn the child process depending upon whether or not the new security properties would meet certain targets the user had registered.

Consider what this pro-active contract would mean for the OS itself: every new binary would be linked to a security property certificate, by means of the hash of the executable as a handle to look in PBA-TTP administered databases – which databases and how to combine properties depend-

ing upon the registered trust model; then, depending upon the computed security consequences, execution would be granted or not. Hence, from this point of view, what we are suggesting is a "signed code" OS solution to security problems, however integrated into the larger picture of the entire platform, elaborate trust models, complex derived security properties and hardware assisted security computations.

There remains of course the issue of the reliability of the PBA property derivation computation, which we have described in §3.2 as happening on a TTP with which a verifier can run binary attestation. When moving to the pro-active approach, a TTP is of course possible, for a networked platform. If we want to increase individual, stand-alone platforms' security, we need to do this with a "virtually-remote TTP". There are a number of groups looking into virtualizing OSes *and their TPMs*, so we suggest that the cleanest approach is for our platform to run applications in separate hypervised partitions, and for one partition to run a stripped-down, simple PBA TTP. In fact, this approach could be used in general with PBA, and also it makes sense always to run several instances of a desired client OS (each with virtualized TPM access) so that individual partitions can have IBC-guaranteed security properties appropriate to the user's needs, and if one partition fails to have some desired property, it can immediately be terminated and another be used.

We have seen here, then, that particular with the assistance of secure and efficient OS and TPM virtualization, PBA can be used conveniently, even in a pro-active mode where the user can have confidence in the security status of *their own machine*; oddly enough, this last feature is totally ignored in the current TCG IBC instantiation, as the TPM Quote blob could indeed be displayed to the user sitting at her platform, but unless she can do public-key crypto in her head, she can have less confidence than a remote but computationally active verifier.

5. CONCLUSIONS

We have motivated PBA by pointing out a fundamental flaw in the paradigm of the TCG's IBC: it conflates the *security state* of a platform with its *execution state*. Teasing these ideas apart makes room for more subtle and flexible security models – as well as increasing scalability and privacy – and shows the central rôle of trust models. Moving to PBA could increase the flexibility, scalability, openness and even the security of IBC, and explicit, public security property targets and trust models will not prevent the abuses critics fear from TCG technology, but *will* help to make them difficult to conceal. Finally, we suggested a future direction for IBC – based on OS virtualization, code signing, and an IBC which not only *verifies* security properties, but *enforces them* – which could have the advantages just attributed to PBA, and which also could help slow the terrifying rise of malware on the 'net today.

6. REFERENCES

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

[2] B. Arbaugh, *Improving the TCPA specification*, IEEE Computer **35** (2002), no. 8, 77–79.

[3] S. Bechtold, *The present and future of DRM – musings on emerging legal problems.*, Digital Rights Management (Eberhard Becker, Willms Buhse, Dirk Günnewig, and Niels Rump, eds.), Lecture Notes in Computer Science, vol. 2770, Springer, 2003, pp. 597–654.

[4] A. Bernard and P. Lee, *Enforcing formal security properties*, Carnegie Mellon School of Computer Science report, 2001, `http://reports-archive.adm.cs.cmu.edu/anon/2001/CMU-CS-01-121.pdf`.

[5] Edward W. Felten, *Understanding trusted computing: Will its benefits outweigh its drawbacks?*, IEEE Security and Privacy **1** (2003), no. 3, 60–62.

[6] R. Kay, *The future of trusted computing*, Commissioned Research Report, on TCG web site, 2005, `https://www.trustedcomputinggroup.org/home/IDC_Presentation.pdf`.

[7] J. Poritz, *Hash chains: a weak link for trusted computing*, 2005, submitted to Eurocrypt'06.

[8] J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner, *Property attestation – scalable and privacy-friendly assessment of peer computers*, IBM research report RZ3548, 2004.

[9] S. Schoen, *Trusted computing: promise and risk*, a position paper of the Electronic Frontier Foundation, 2003, `http://www.eff.org/Infrastructure/trusted_computing/20031001_tc.php`.

[10] Trusted Computing Group consortium, *Promotional literature and specifications available on public web site*, `http://www.trustedcomputinggroup.org/`.