Using Secure Coprocessors to Protect Access to Enterprise Networks

Haidong Xia, Jayashree Kanchana, and José Carlos Brustoloni

Dept. Computer Science, University of Pittsburgh,
210 S. Bouquet St. #6135, Pittsburgh, PA 15260, USA
{hdxia, kanchana, jcb}@cs.pitt.edu

Abstract. Enterprise firewalls can be easily circumvented, e.g. by attack agents aboard infected mobile computers or telecommuters' computers, or by attackers exploiting rogue access points or modems. Techniques that prevent connection to enterprise networks of nodes whose configuration does not conform to enterprise policies could greatly reduce such vulnerabilities. Network Admission Control (NAC) and Network Access Protection (NAP) are recent industrial initiatives to achieve such policy enforcement. However, as currently specified, NAC and NAP assume that users are not malicious. We propose novel techniques using secure coprocessors to protect access to enterprise networks. Experiments demonstrate that the proposed techniques are effective against malicious users and have acceptable overhead.

1 Introduction

Enterprise networks' first line of defense typically consists of firewalls and virtual private network (VPN) gateways. System administrators usually attempt to install such nodes at every point of contact between an enterprise's intranet and the public Internet. However, the security perimeter thereby achieved can be rather leaky. Users can bring to their offices mobile computers infected while used on a trip. Telecommuters can use hacked computers to connect to the enterprise's VPN. Users often install in their offices wireless access points or modems, without consulting system administrators. Users may find such rogue network nodes or mobile computers convenient, but attackers can use them to circumvent the enterprise's firewalls.

These vulnerabilities could be prevented in an enterprise network that verifies that a node's configuration conforms to the enterprise's security policies *before* accepting connection of the node to the network. This type of network policy enforcement has not traditionally been available, but currently there are several initiatives to support it, including Cisco's Network Admission Control (NAC) [1] and Microsoft's Network Access Protection (NAP) [9].

In Microsoft's NAP, before connecting to an intranet, a host sends to a network-designated server a list of the host's software components and configuration. This list indicates, e.g., what operating system and anti-virus software

R. Boutaba et al. (Eds.): NETWORKING 2005, LNCS 3462, pp. 154–165, 2005.

are installed in the host and what version, security patches, and virus definitions they have. If the server finds that the host's software is up-to-date and acceptable, the server allows the host to connect to the intranet. Otherwise, the server confines the host to a restricted network. The restricted network allows the host's software to be updated and brought into compliance with intranet policies. Cisco's NAC architecture extends this concept to control the connection not only of hosts, but also access points and other devices. Networks and nodes that support such access control functionality are expected to become common in the next several years.

However, NAC and NAP provide only weak security. Software component and configuration lists can be easily forged or modified by a malicious user. Using a forged or modified list, an attacker can circumvent NAC and NAP and gain access to a network with a node that can greatly harm the network's security.

This paper examines the question of how *secure coprocessors* could be used to harden architectures such as NAC and NAP. The Trusted Computing Group (TCG) [18] has specified secure coprocessors (called TPMs – Trusted Platform Modules) [19] that have low cost (about $4 per host) and are commercially available in an increasing number of computers by IBM, HP, and other manufacturers [14]. TPMs enable security primitives that were previously unavailable, in particular *attestation*. Attestation reveals to another party the software configuration running on a host, in a securely verifiable manner. Unlike NAC's and NAP's software component and configuration lists, attestations cannot be easily forged or modified by attackers. Section 2 describes attestation in greater detail.

We identify three challenges for building systems that use attestation, and contribute novel solutions for them. First, attestation has to be integrated with network protocols in a manner that does not defeat security. Attestation as specified by TCG is vulnerable to man-in-the-middle (MITM) attacks. We show in Section 3.1 that this vulnerability is not eliminated simply by tunneling attestation packets, e.g. using TLS [2]. We contribute a novel form of attestation, *Bound Keyed Attestation* (BKA), that can be integrated with other protocols easily and without MITM vulnerability.

Second, the operating system (OS) has to be modified such that it properly records and reports in attestations any OS modifications since the system has booted. TCG specifies TPM's hardware, boot sequence, and interfaces, but not the inner working of systems that use those interfaces, especially after boot time. Most OSs define privileged users (e.g., root) with authority to modify the OS or its configuration at any time. However, the charter of TCG's relevant working group explicitly excludes what the OS should do to maintain attestation consistency after boot time [20]. In Sections 4.1 and 4.2, we propose *TCB prelogging* and *security association root tripping* for guaranteeing such consistency.

Third, use of a TPM must not harm host safety. In addition to attestation, TPMs provide another security primitive, *sealing*. Data sealed to a given host software and configuration can be accessed only when the host software and configuration are the same. Sealing can be used for *digital rights management*. Before sending content to a receiver, a sender obtains the receiver's attestation.

The sender sends the content only if the sender deems the receiver's software as trustworthy. The receiver's software then seals the content so that it cannot be accessed by untrusted software. However, sealing can be used also for *software lock-in* [4]. An application, e.g. an editor, may seal to itself content created by the host owner. This can make it impossible for the host owner to switch to a different application or to access the data using a future host. In this sense, sealing can make a host unsafe. In Section 4.3, we propose *sealing-free attestation confinement* for securing network access without compromising host safety.

Experimental results in Section 5 show that our proposed mechanisms are effective and have acceptable overhead. We discuss related work in Section 6, and conclude in Section 7.

2 TPM Functions

To secure access to enterprise networks, we use two main TPM functions: *authenticated boot* and *attestation*. We describe these functions in this section. Readers who are already familiar with TCG secure coprocessors may skip this section.

Authenticated boot presupposes that when a host is reset, control of the CPU is transferred to a small, trusted, immutable software component. In a personal computer, this component is the BIOS boot block. The OS's boot sequence is modified such that, before each software component A passes control to another software component B that has not yet been measured, A measures B's digest, appends the digest to a *measurement log*, and compresses the digest into the TPM. A obtains B's digest using SHA-1 (Secure Hash Algorithm) [11]. SHA-1 digests are 20 bytes long, regardless of data length. This algorithm has properties such that it is infeasible to modify B without also modifying its digest, or to find a B whose digest is an arbitrary value. The TPM compresses a digest into one of its registers by concatenating the register's value and the digest, computing SHA-1 on this concatenation, and storing the resulting value back into the register. The TPM's registers are initialized to zero on host reset. They can be read, but cannot be otherwise modified. TPM register values can be used to authenticate the measurement log, whose plaintext is stored in main memory.

Attestation is a protocol that enables a remote party R to obtain and authenticate a host P's measurement log. R sends to P a nonce, i.e., a cryptographically random number that is never reused. P asks its TPM to sign a so-called *quote*, containing the nonce and current values of the TPM's registers. Presence of the nonce in the quote guarantees that a quote cannot be later replayed. The signature uses an attestation identity key (AIK). P sends the corresponding AIK certificate to R together with the quote and measurement log. TPMs generate private and public AIK pairs internally. TPMs use private AIKs only to sign quotes (as defined above), and never reveal such keys externally. The host owner obtains the AIK certificate from a so-called *privacy certifying authority*, which verifies that the host contains a properly attached TCG-compliant TPM. R authenticates the AIK certificate using the certifying authority's public key, which R is assumed to know out-of-band. R then uses the nonce and public AIK to

authenticate the quote and uses the quote to authenticate the measurement log. The authenticated log reveals securely to R what software booted in P.

3 Network Protocol Enhancements

This section describes protocol enhancements for securing network access.

IEEE 802.1x [7] is a standard, widely supported protocol that can be configured to require mutual authentication between a node and a local area network (LAN) before the node is allowed to communicate through the network (beyond authentication). The participants to this protocol are the *supplicant*, i.e. the node that requests access, the *authenticator*, e.g. a switch or access point that mediates the supplicant's access to the network, and the *authentication server*, e.g. a RADIUS server that authenticates and authorizes the supplicant's access. A variety of authentication protocols can be used over 802.1x, including PEAPv2 [13]. PEAPv2 begins by creating a TLS tunnel between authentication server and supplicant, with certificate-based authentication of the former by the latter. The server then typically uses MS-CHAPv2 for password-based supplicant authentication. Finally, PEAPv2 binds the TLS and MS-CHAPv2 keys to guarantee that the respective endpoints are the same. 802.1x also enables the creation of a security association between authenticator and supplicant, with cryptographic keys for encrypting and authenticating packets sent between them. On Ethernet, such packet-level cryptography is being standardized by IEEE 802.1ae; on Wi-Fi, packet-level cryptography is enabled by IEEE 802.11i.

PEAPv2 authenticates only the supplicant's user. We propose to combine PEAPv2 with Bound Keyed Attestation (BKA). This combination enables the server to authenticate both the supplicant's host configuration and user before the server authorizes the supplicant's communication. If the server does not trust the supplicant's configuration, the server can deny access to the supplicant or confine the supplicant to a restricted virtual LAN (VLAN, as indicated, e.g., by a RADIUS *access accept* attribute).

3.1 Bound Keyed Attestation

This subsection shows that attestation, as defined by TCG, is vulnerable to MITM attacks. It then proposes BKA, a novel form of attestation that resists MITM attacks.

As shown in Fig. 1, a malicious user can use two computers, one of which is conformant to network security policies, to get past TCG-defined attestation. The user hacks the intermediate computer such that it passes to the user's conformant computer any attestation requests, and passes the corresponding attestation replies back to the authentication server. The latter cannot discern the presence of a MITM, and therefore authorizes access, even though the MITM may have a configuration in complete violation of the network's security policies.

This vulnerability is *not* eliminated by securely tunneling attestation messages, e.g. using TLS. A secure tunnel can protect the confidentiality and integrity of the messages against third-party attacks. However, if the client is not

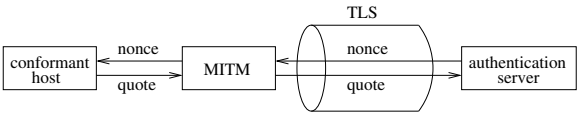

Fig. 1. TCG-specified attestation is vulnerable to MITM attacks. If a malicious user has two computers, one of which is conformant to the server's policies, the user can employ the conformant computer to get a hacked computer accepted by the server. It makes no difference if TLS is used

trustworthy, the tunnel cannot prevent the latter from acting as a MITM that relays attestation or other messages between the tunnel and another computer.

To thwart this type of attack, we propose a novel form of attestation, Bound Keyed Attestation (BKA). It differs from TCG-specified attestation in two important ways. First, it derives new keys shared by the attestation endpoints. Second, it securely binds these keys to a tunnel's keys. This binding guarantees that tunnel and attestation endpoints are the same. Consequently, a MITM attack is not possible.

All BKA messages can be transmitted using a previously established secure tunnel T (e.g., PEAPv2's TLS tunnel) with which the attestation will be bound. It is assumed that T is cryptographically secured with a dynamically generated secret key K_T that is not revealed to users. T can have states *attested* or *not-attested* (default). System policies can prevent communication other than for attestation while the tunnel is not-attested. Like a Diffie-Hellman key exchange [3], BKA uses two publicly known numbers: a prime number q and an integer α that is a primitive root of q. These numbers do not need to change. As shown in Fig. 2, the *attestation initiator* A picks two random numbers: a nonce N_A, which is never reused, and another integer $0 \leq X_A < q$, which may be reused. The initiator computes its public key $Y_A = \alpha^{X_A} \bmod q$. The initiator then sends to the *attestation responder* B a *BKA request* message M_i containing q, α, Y_A, and N_A.

The responder then picks a random integer $0 \leq X_B < q$, which may be reused, and computes its public key $Y_B = \alpha^{X_B} \bmod q$. The responder computes the *attestation shared secret* as $K_{AB} = Y_A^{X_B} \bmod q$, and its *attestation binding* $B_B = \text{SHA1}(N_A|K_T|K_{AB}|Y_A|\text{“BKA response”})$, where $|$ denotes concatenation. The responder also computes the *initiator's attestation nonce* $n_A = \text{SHA1}(N_A|K_{AB})$. The responder then gets from its TPM its quote Q_B containing n_A. Finally, the responder sends to the initiator a *BKA response* message M_r containing Y_B, a fresh nonce N_B, B_B, Q_B, and B's measurement log L_B and AIK certificate C_B. Alternatively, if the responder also wishes to obtain the initiator's attestation, the responder sends to the initiator a *BKA response-request* message M_{rr} containing the same fields as M_r.

The initiator processes M_r as follows. First, the initiator computes the attestation shared secret as $K_{AB} = Y_B^{X_A} \bmod q$. The initiator then computes the expected value of B_B using K_{AB}. If the received B_B does not match this expected value, the initiator sends a *BKA error* message to the responder and returns failure (possibly the tunnel and attestation endpoints are not the same,

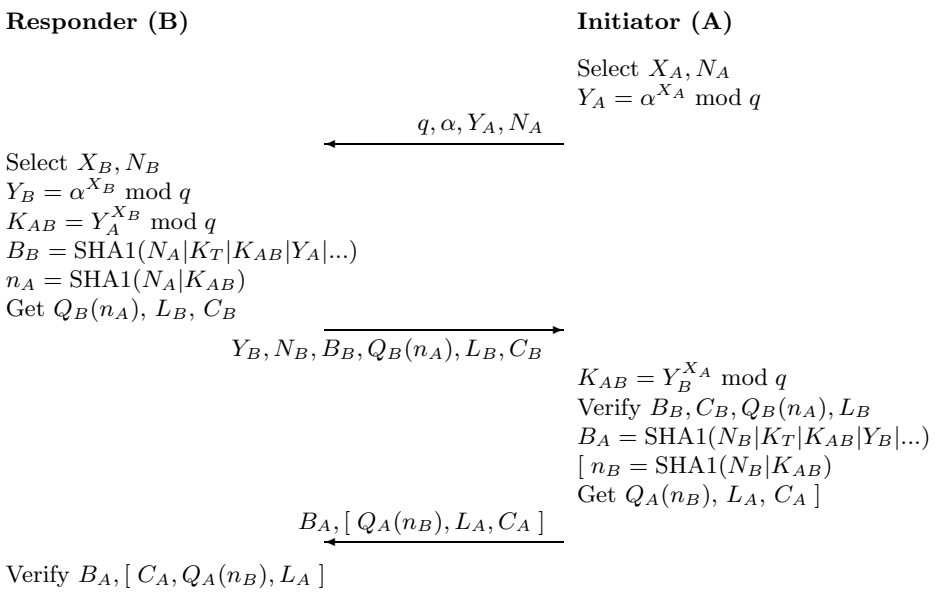

Fig. 2. Bound Keyed Attestation (BKA) thwarts MITM attacks by guaranteeing that the endpoints of the communication tunnel (e.g., TLS) and attestation are the same. Items between brackets are needed only for mutual attestation

and a MITM attack is happening). Otherwise, the initiator authenticates C_B using the respective certifying authority's public key (known securely out-of-band), authenticates the responder's quote Q_B using n_A and C_B, and authenticates the responder's measurement log using Q_B. If any of the authentications fails, or the initiator does not recognize a measurement in the responder's log L_B as that of a trusted software component, the initiator sends a BKA error message to the responder and returns failure. Otherwise, the initiator computes its binding $B_A = \text{SHA1}(N_B|K_T|K_{AB}|Y_B|\text{"BKA success"})$. The initiator then sends to the responder a *BKA success* message M_s containing B_A, transitions T's state to *attested*, and returns success.

Processing of M_{rr} by the initiator is similar to that of M_r, except that (1) the initiator also computes the *responder's attestation nonce* $n_B = \text{SHA1}(N_B|K_{AB})$ and gets from the initiator's TPM its quote Q_A containing n_B, and (2) instead of M_s, the initiator sends to the responder a *BKA success-response* message M_{sr} containing B_A, Q_A, and A's measurement log L_A and AIK certificate C_A.

The responder processes M_s by computing the expected value of B_A and verifying that the received B_A matches it. If so, the responder transitions T's state to *attested*. Otherwise, T's state remains *not-attested*. Processing of M_{sr} is similar, except that, if the received B_A matches its expected value, (1) the responder also authenticates C_A, Q_A, and L_A, and (2) if all authentications succeed, and the responder identifies each measurement in the initiator's log as

that of a trusted software component, then the responder transitions T's state to *attested*; otherwise, T's state remains *not-attested*.

Nonces N_A and N_B guarantee quote freshness and allow bound keyed attestations to be obtained frequently, without burdening the processors each time with the expensive computation of Y_A, Y_B, and K_{AB}. Performance can be improved also by caching the authentication of C_A and C_B and the values of common intermediate expressions that change infrequently. The attestation nonces n_x are one-way functions of not only the nonces N_x but also the attestation shared key K_{AB}. Therefore, the quotes are bound to the attestation shared key, which in turn is bound to the tunnel key K_T by B_x, where $x \in \{A, B\}$.

4 Operating System Enhancements

As explained in the previous section, BKA requires hosts to have TPMs. However, use of TPMs poses several OS challenges. We propose in the following subsections three novel solutions to these challenges: *TCB prelogging*, *security association root tripping*, and *sealing-free attestation confinement*.

4.1 TCB Prelogging

TCG documents specify that a system's BIOS, master boot record, boot loader, and OS kernel should be measured and their digests included in the system's measurement log. However, not only these components, but any member of the system's *Trusted Computing Base* (TCB) needs to be measured. A system's TCB is defined as the set of components whose malfunction (due, e.g., to a bug or attack) would allow the system's policies to be compromised. Therefore, the TCB includes not only the TCG-mentioned components, but also their configuration files and any privileged applications that could modify them, e.g. root-owned scripts or daemons and setuid applications. On the other hand, unprivileged user applications that are not part of the TCB can be created, configured, modified, or destroyed without compromising the system's ability to enforce policies.

We propose that each system maintain a configuration file listing the system's TCB components and respective digests. A bug in this TCB list (e.g., absence of a TCB component) could enable system policies to be violated. Therefore, the TCB list needs to include an entry for itself. By convention, the digest of the TCB list is calculated by making this entry's digest equal to zero, and the result is stored in this entry. At boot time, after the kernel mounts the file systems, the kernel appends the TCB list to the measurement log and compresses the list's digests into the TPM. Thereafter, whenever a file that is a TCB component, root-owned script or daemon or setuid application is opened or exec'ed, the kernel measures the file's digest. If this digest is different from the one last logged for the file, the kernel appends the new measurement to the log and compresses the new digest into the TPM. Thus, an authenticated measurement log will reveal whether a tampered or untrusted TCB component has run in the system since the system booted.

4.2 Security Association Root Tripping

Unprivileged users cannot cause the OS kernel or daemons to compromise the system's policy enforcement. (If that is not true, the system has a bug that needs to be fixed.) However, privileged users (e.g., root) *can* easily violate the system's policy enforcement, e.g. by using commands such as sysctl or ifconfig or by using a debugger to attach and modify privileged processes, after boot time. It can be difficult or impossible to guarantee that all such configuration modifications are captured in the system's measurement log. Moreover, TCG does not specify what to do when such modifications occur.

We propose to modify the OS such that it detects and takes appropriate action when a privileged user attempts to gain interactive access to the system (e.g. by logging in or using the su command). If there are any attestation-based security associations, the system warns the user that, if the user wants to continue, the system will immediately drop those security associations. Thus, in the case of secure intranet access, the system will destroy the keys used for packet-level cryptography, making access impossible. Furthermore, if the user wants to continue, the system appends this event to the measurement log with a well-known digest and compresses the latter into the TPM. Therefore, subsequent attestations will show that a privileged user has logged in interactively. System administrators can configure authentication servers to deny access to such supplicants, who will need to reboot before again gaining access to the network (reboot erases the measurement log, as well as any non-persistent configuration changes; persistent changes are captured by the new measurement log after reboot). These mechanisms do not preclude remote system administration or help, as long as these are performed using daemons that the network's authentication server is configured to trust.

4.3 Sealing-Free Attestation Confinement

TPMs include a function, sealing, that can be abused by applications. As discussed in the Introduction, sealing enables software lock-in, making hosts unsafe.

To enable BKA without compromising safety, we propose that the OS (1) support authenticated boot and attestation, but not sealing, and (2) allow attestation only in conjunction with network access control protocols (such as 802.1x and IPsec's IKE). Abusive applications then cannot encrypt and bind to themselves file contents, as necessary for software lock-in. When the host is disconnected, the contents would need to be sealed, but the OS does not provide this service. On the other hand, when the host is Internet-connected, abusive applications, instead of sealing, could attempt to store file encryption keys in a remote server that reveals the keys only to abusive applications in attested clients. However, the OS allows attestation only in conjunction with protocols that typically cannot go through enterprise firewalls. Because the OS confines attestation to the intranet, it thwarts software lock-in also in this case.

Table 1. Breakdown of authentication and authorization latency and projected throughput (standard deviations represented between brackets: [σ])

Step	PEAPv2	PEAPv2 + LOG	PEAPv2 + BKA
TLS	39.6 ms [0.2]	39.9 ms [0.8]	38.9 ms [0.9]
LOG		24.4 ms [0.2]	
BKA			2758 ms [263]
MS-CHAPv2	20.6 ms [0.5]	17.4 ms [0.2]	17.9 ms [0.2]
Binding TLS/MS-CHAPv2	7.0 ms [0.2]	6.8 ms [0.2]	6.8 ms [0.1]
Total	67.2 ms [0.5]	88.4 ms [0.5]	2822 ms [263]
CPU busy	22.6 ms [0.2]	23.9 ms [0.2]	116 ms [10]
Projected throughput	2650 supp/min	2510 supp/min	519 supp/min

5 Experimental Evaluation

This section evaluates the performance impact of the proposed mechanisms. Reported results are averages of six measurements.

We implemented the proposed OS enhancements on FreeBSD 4.8 and installed this OS on an IBM ThinkPad T30 computer with 1.8 GHz Pentium 4 CPU, 256 MB RAM, TPM version 1.1b, TPM-aware BIOS, and built-in 802.11b interface. The master boot record and GRUB were modified for measuring digests and compressing them into the TPM as specified by TCG. We measured a total boot time of 20.08 s ($\sigma = 0.12$) before and 20.15 s ($\sigma = 0.17$) after our modifications. Although TCB prelogging and file digest measuring impose overheads, they are completely dominated by other boot costs. During system operation, each file digest measurement can be cached, and need not be repeated while the file is not modified [15]. Because TCB components change infrequently, file digest measurements can be expected to have little impact on steady-state performance. Security association root tripping affects only certain commands (e.g., login and su) and only when used by privileged users. Therefore, it also has negligible performance impact, as does sealing-free attestation confinement.

We integrated PEAPv2/802.1x with BKA on the FreeRADIUS authentication server [5] and Open1x supplicant [12]. In order to estimate a comparison with NAP, we alternatively integrated PEAPv2/802.1x with a NAP-like LOG protocol. Both in BKA and in LOG, the supplicant sends its list of software components and configuration to the authentication server, who may approve it or not. However, unlike BKA's list, LOG's list can be forged by the supplicant. We installed FreeRADIUS on a Dell Dimension 4550 computer with 2.4 GHz Pentium 4 CPU, 256 MB RAM, and unmodified FreeBSD 4.10. We installed Open1x on the aforementioned IBM T30 computer. As authenticator, we used a Cisco Aironet 1100 802.11b access point connected to the authentication server via Fast Ethernet. We used BKA with precomputed modulus (1024-bit) and primitive root. Table 1 shows that the time it takes for the supplicant to connect to the network increased from about 67 ms to 88 ms with LOG or 2.8 s with BKA. To understand the source of BKA's large overhead, we measured the

time it takes for the supplicant to obtain a quote from its TPM, and found it to be 2.5 s ($\sigma = 0.2$). Therefore, most of the latency added by BKA is due to the TPM. However, even with a low-cost, slow TPM, such as the one we used, the connection latency is acceptable.

In order to evaluate the impact of LOG or BKA on the CPU utilization of the authentication server, we instrumented the OS's idle loop and interrupt vector. The instrumented OS uses the CPU's built-in cycle counter to measure CPU idle time, excluding interrupts. We also instrumented FreeRADIUS to measure the total time necessary to authenticate and authorize supplicant access. Table 1 shows how long the authentication server's CPU was busy while processing a single supplicant, and the projected throughput (load for saturating the CPU). LOG and BKA reduced projected throughput from roughly 2650 to 2510 or 519 supplicants/minute, respectively. BKA's large overhead is due to its use of public-key algorithms for key exchange and authentication of certificates and quotes, as well as repeated applications of SHA-1. Although BKA's overhead is substantial, the projected throughput may be acceptable for medium-size networks. Note that, in our scheme, a supplicant imposes no load on the authentication server after the latter authorizes access. In larger networks, load can be distributed simply by using multiple low-cost authentication servers, such as the one we used.

6 Related Work

NAC [1] and NAP [9] are the architectures most closely related to our work. Because NAP uses DHCP for controlling access to enterprise LANs, malicious users can circumvent it simply by using a static networking configuration. We use 802.1x instead of DHCP to avoid this security loophole. NAC is currently implemented on certain routers, using a proprietary access control protocol. Unlike our solution, such an implementation does not prevent unauthorized access to LANs that a malicious user might directly connect to.

Our TCB lists are similar to what Tripwire [17] uses to verify host integrity. However, unlike Tripwire, we enable administrators to verify host integrity remotely. Bear [8] is a Linux-based OS that supports TPMs and uses similar lists, signed by trusted system administrators. Bear's Enforcer verifies at load time whether a file's digest matches its value on the signed list. In case of a mismatch, Enforcer takes the action specified on the list (e.g., return failure). Bear does not support attestation, relying instead on certificates with special semantics, and has no protection against privileged users. Security association root tripping (Section 4.2) would significantly improve Bear's security.

TcgLinux [15] is another OS with TPM support. Because tcgLinux does not have a TCB list, it logs and compresses into the TPM digests of *all* files that are executed, and requires shells and other programs to be modified to do the same with their security-sensitive scripts and configuration files. This design makes it harder to verify that all TCB configuration files are being measured. It also unnecessarily exposes in attestations the execution of unprivileged programs, reducing user privacy. TcgLinux has mechanisms to prevent privileged users

from making system modifications that might not be detected by attestation. It does allow, however, any modifications that would be detected by attestation. TcgLinux attestations have been used to secure VPN access [16]. However, that solution has several shortcomings. First, TCG-specified attestation is vulnerable to MITM attacks, as shown in Section 3.1. Second, tcgLinux would be vulnerable to MITM attacks even if BKA were used, because it does not prevent privileged users from reading secret keys and other parameters of VPN tunnels. In contrast, security association root tripping would close the VPN tunnels before such reading would be possible. Third, the VPN gateway has to verify fresh attestations of each client frequently. If attestation frequency is low, users may be able to connect an insecure node to the VPN long enough to cause harm. On the other hand, our experimental results suggest that high attestation frequencies could severely limit the throughput of the VPN gateway. In contrast, security association root tripping achieves security with a single attestation at the beginning of each client's session.

Microsoft's NGSCB architecture [10] uses TPMs and divides the system into trusted and untrusted halves. The untrusted half runs a conventional OS, including file system and network protocol stack. On the other hand, the trusted half secures user credentials and keys for digital rights management. It uses storage and communication services provided by the untrusted half. NGSCB requires special CPUs with Intel's LaGrande Technology (LT). Terra [6] is an OS with similar architecture, but uses a virtual machine monitor instead of LT. It is unclear how NGSCB or Terra would be used for securing network access, because those systems cannot guarantee the integrity of the untrusted half's configuration.

7 Conclusions

Firewall-based network security perimeters can be leaky. Architectures such as NAC and NAP attempt to plug perimeter leaks by preventing nodes that do not conform to a network's security policies from joining the network. However, those architectures are vulnerable to malicious users. TCG has specified inexpensive secure coprocessors that could harden those architectures. However, several challenges need to be overcome, including how to prevent MITM attacks, undetected changes to system files, tampering by privileged users, and software lock-in. We proposed novel solutions to these challenges: bound keyed attestation, TCB prelogging, security association root tripping, and sealing-free attestation confinement. We integrated these mechanisms with FreeBSD and PEAP over 802.1x. Experimental results show that our techniques allow secure coprocessors to protect access to enterprise networks robustly, safely, and with acceptable overhead.

References

1. Cisco: Network Admission Control. [Online] `http://www.cisco.com/en/US/netsol/ns466/networking_solutions_sub_solution_home.html`

2. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. IETF, RFC 2246, Jan. 1999. [Online] ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt
3. Diffie, W., Hellman, M.: New Directions in Cryptography. In *Transactions on Information Theory*, IEEE, 1976, 22:644-654.
4. Felten, E.: Understanding Trusted Computing. In *Security and Privacy*, IEEE, May/June 2003, pp. 60-62. [Online] http://www.princeton.edu/~echi/ele572/Felten\%20-\%20Understanding\%20trusted\%20computing.pdf
5. FreeRADIUS: Homepage. [Online] http://www.freeradius.org/
6. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proc. 19th Symposium on Operating System Principles*, ACM, 2003. [Online] http://www.stanford.edu/~talg/papers/SOSP03/terra.pdf
7. IEEE: Port-Based Network Access Control. 802.1x Std. (2001) [Online] http://standards.ieee.org/getieee802/download/802.1X-2001.pdf
8. Marchesini, J., Smith, S., Wild, O., Stabiner, J., Barsamian, A.: Open-Source Applications of TCPA Hardware. In *Proc. 20th Annual Computer Security Applications Conference*, ACSAC, Dec. 2004. [Online] http://www.cs.dartmouth.edu/~carlo/research/bearapps/bearapps.pdf
9. Microsoft: Network Access Protection. [Online] http://www.microsoft.com/windowsserver2003/technologies/networking/nap/default.mspx
10. Microsoft: Next Generation Secure Computing Base – Technical FAQ. July 2003. [Online] http://www.microsoft.com/technet/security/news/ngscb.mspx
11. NIST: Secure Hash Standard. Federal Information Processing Standards Pub. 180-1, Apr. 1995. [Online] http://www.itl.nist.gov/fipspubs/fip180-1.htm
12. Open1x: Homepage. [Online] http://www.open1x.org/
13. Palekar, A., Simon, D., Salowey, J., Zhou, H., Zorn, G., Josefsson, S.: Protected EAP Protocol (PEAP) Version 2. IETF. Internet Draft, Oct. 2004. [Online] ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-josefsson-pppext-eap-tls-eap-10.txt
14. Pearson, S. (ed.): Trusted Computing Platforms – TCPA Technology in Context. Prentice Hall, 2003.
15. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proc. Security Symposium*, USENIX, Aug. 2004. [Online] http://www.usenix.org/publications/library/proceedings/sec04/tech/sailer.html
16. Sailer, R., Jaeger, T., Zhang, X., van Doorn, L.: Attestation-based Policy Enforcement for Remote Access. In *Proc. 11th Conference on Computer and Communications Security (CCS)*, ACM, Oct. 2004. [Online] http://portal.acm.org/citation.cfm?id=1030083.1030125
17. Tripwire.org: Homepage. [Online] http://www.tripwire.org/
18. Trusted Computing Group: Homepage. [Online] https://www.trustedcomputinggroup.org/home
19. Trusted Computing Group: Trusted Computing Platform Alliance (TCPA) Main Specification Version 1.1b. [Online] https://www.trustedcomputinggroup.org/downloads/Main_TCG_Architecture_v1_1b.zip
20. Trusted Computing Group: Work Group Charter Summary. 2004. [Online] https://www.trustedcomputinggroup.org/downloads/Work_Group_Charters_Summary.pdf