January 28, 2003

RT0507
Security 6 pages

Research Report

Linux with TCPA Integrity Measurement

Hiroshi Maruyama, Taiga Nakamura, Seiji Munetoh, Yoshiaki Funaki, Yuhji Yamashita

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Linux with TCPA Integrity Measurement

Hiroshi Maruyama
IBM Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa-ken, Japan
maruyama@jp.ibm.com

Taiga Nakamura
IBM Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa-ken, Japan
taiga@jp.ibm.com

Seiji Munetoh
IBM Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa-ken, Japan
munetoh@jp.ibm.com

Yoshiaki Funaki
IBM Yamato Software Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa-ken, Japan
yfunaki@jp.ibm.com

Yuhji Yamashita
IBM Yamato Software Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa-ken, Japan
yamasy@jp.ibm.com

ABSTRACT

This paper describes an end-to-end implementation of TCPA (Trusted Computing Platform Alliance) integrity measurement for Linux kernel. Integrity measurement of the kernel is done through a chaining of PCR (Platform Configuration Register) updates during the bootstrap process of the GRUB kernel loader. The measured integrity values can be reported to remote servers by a Java application that uses a digital signature containing the PCR values. As the hardware platform, we used a stock laptop computer (ThinkPad T30) that is TCPA-enabled.

1. INTRODUCTION

Today many computer applications have a distributed structure. Typical examples are client-server systems such as email and Web applications where the client component plays a vital role for the overall system to function properly. In such distributed systems, the remote components need to be trusted. By "trusted" here we mean that the behavior of the remote component is predictable according to the intended system design. If the network environment is potentially hostile, or subject to unexpected system changes, the distributed system cannot trust the remote component. For example, the distributed system may rely on the remote host to maintain a particular set of state. One way to ensure that the remote component can be trusted is to let the component prove the integrity of the remote component, i.e., that the component is a known implementation and is running with a known configuration of the environment. It is important to understand that the integrity of the platform (the hardware, the operating system, the shared libraries, and all the related configuration files) is essential for the integrity of the application component because ultimately the component's behavior relies on the platform. Therefore, platform integrity is an essential part of trustworthy distributed computing. In this paper, we consider the operating system integrity and describe an implementation that enables a distributed application to securely report the integrity of the operating system on which its remote component is running.

There are several proposed approaches to software integrity. Signed code, such as Active X Control with Authenticode [10] and Java signed applet [12], is one example. Before a code is installed or executed, a digital signature on the code is verified by the platform. Only codes with a valid signature according to the trusted root certificate stored in the platform can be installed and / or executed. Another approach to software integrity is to use a periodic scan of installed software for integrity. Integrity tools such as Tripwire [11] and virus scan software are in this category. Both approaches are effective to a certain extent in maintaining the platform integrity, but neither approach provides a direct means for a distributed system to test the integrity of a remote host.

TCPA (Trusted Computing Platform Alliance) [1][2] is unique in defining a security subsystem for network client platforms that provides the capability of reporting the platform integrity to remote hosts, called *attestation*. The TCPA 1.1b specification [1] was published in 2002 and the first products supporting this specification were shipped in 2002. In this paper, we describe a working prototype of a Linux-based system that has the attestation capability. The platform integrity is measured during the OS bootstrap process and the measured values are stored in the TCPA compliant chip (Trusted Platform Module, or TPM). The measured values are embedded in a signature value so that a distributed application can test the values.

This paper is organized as follows. In the next section, we give a brief overview of TCPA attestation. Section 3 shows the detailed implementation, focusing on the modification on the GRUB OS loader. Our sample application, including a proposed extension to XML Signature [5] and Web Services Security [6], is shown in Section 4. We briefly describe an application prototype that demonstrates an end-to-end integrity measurement in Section 5, followed by a discussion in Section 5.

2. Integrity Measurement in TCPA

TCPA (Trusted Computing Platform Alliance) is an industry forum to define open specifications of security subsystems for client platforms. It was formed by HP, Intel, Microsoft, Compaq, and IBM in 1999. Its version 1.1b core specification, published in Feb. 2002, has a detailed description of a security chip called TPM (Trusted Platform Module).

2.1 Trusted Platform Module (TPM)

TPM has two important functions. One is to securely store the most important security information, that is, cryptographic keys. Once a public key / private key pair is generated the private key never leaves the TPM. In this regard, TPM is similar to a smartcard embedded in a platform.

The other function of TPM is to measure the integrity of the platform. This is a unique feature of TCPA.

The simplified structure of a trusted platform is shown in Figure 1. The platform has an embedded TPM, which has a factory generated key pair called the endorsement key. The endorsement key is a unique to the platform and will never be changed. The platform manufacture issues a certificate for the endorsement key, certifying that the platform is a genuine TCPA platform.

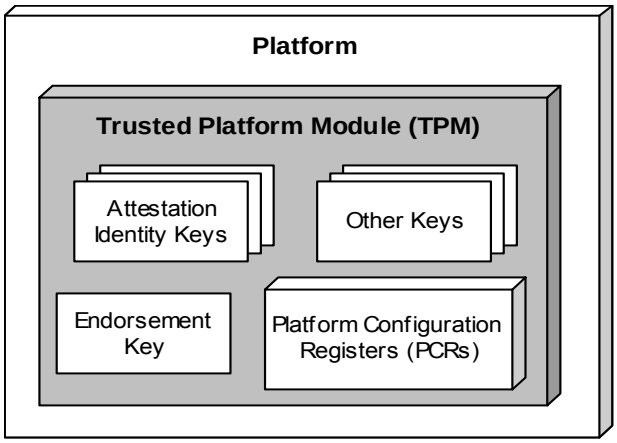

Figure 1. Trusted Platform in TCPA

The endorsement key is only used for taking the ownership of the platform. Application specific keys can also be stored in the TPM. One special type of key, called *attestation identity key*, is used exclusively for attestation. TPM also has a set of special volatile registers called *platform configuration register*s (PCRs). These 160 bit-long registers are used for keeping track of the integrity information during a bootstrap process.

2.2 Measured Bootstrap

Figure 2 illustrates the process of measured bootstrap in TCPA. When a TCPA platform is powered on or reset, all PCRs are cleared to zero. Then a trusted portion of the BIOS ROM is executed. This portion, called the *core root of trust of measurement* (CRTM), is responsible for measuring the integrity of the BIOS. The measured value is used for updating the PCRs. This operation, called PCR_Extend, is defined as follows:

$$PCR\_New = PCR\_Extend(v) = hash(PCR\_Old + v)$$

where "+" denotes bit-string concatenation. Then the BIOS measures the integrity of the next component, which is the operating system in Figure 2, and extends the PCRs. Likewise, the OS can extend the PCRs by the integrity values of libraries, executables, and / or configuration files. At the end of this bootstrapping process, the PCRs will have values that are unique to this particular bootstrap sequence of these particular versions of the BIOS, the operating system, and any other resources that have been measured. In other words, the PCR values represent the integrity of the platform. Note that even though any program can extend the PCRs, it is impossible for a malicious program to set the PCRs to arbitrary values because the only way to change the PCR values is through the PCR_Extend operation and the initial PCR values are already determined by the bootstrap sequence from the unmodifiable portion of the BIOS.

The recent models of IBM ThinkPad (models T30, R32, and X30) have TCPA 1.1b compliant chip on the system board. In addition, the unmodifiable part of the BIOS in these models measure the integrity of the BIOS and then the BIOS measures the Master Boot Record of the bootstrap device. We use this capability for implementing Linux with integrity measurement.

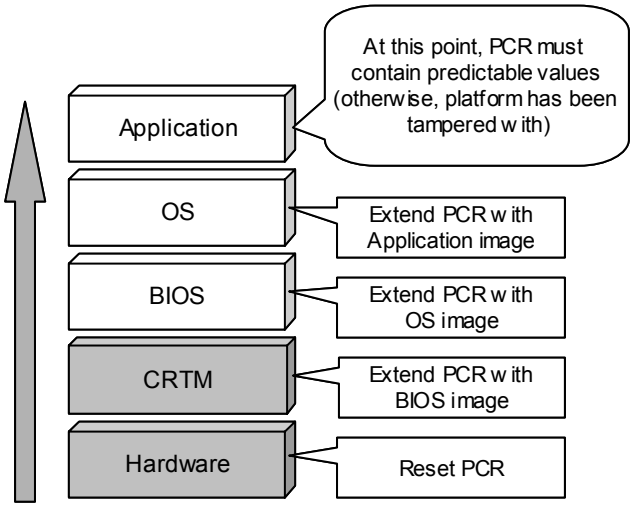

Figure 2.Measured bootstrap process in TCPA

2.3 Reporting Integrity Measurement Value

For a remote system to trust a platform, the platform's PCR values, which represent the platform configuration, need to be reported reliably to the remote system. The TCPA specification defines a set of functions for reporting PCR values to the application. Among them, the "quote" operation provides cryptographic reporting of PCR values. When a remote server sends a TPM_Quote request with a 160 bit challenge, the TPM embedded in the platform digitally signs the current PCR values together with the given challenge and returns the signature to the server:

$$Quote\_value = TPM\_Quote(PCR\_composite, challenge)$$

$$= sign\ (version, ordinal, hash(PCR\_composite), challenge)$$

where *version* is the value of 4 bytes long that contains the version of supported TCPA, *ordinal* is a constant 4-byte string that corresponds to the ASCII "QUOT" string, and *PCR_composite* is a structure that contains the indices and values of the PCR values to be reported. Only attestation keys can be used for this operation. By verifying the signature and the certificate associated with the attestation key, the server will know that the PCR values originate from a certified platform and the PCR values have not been tampered with.

If the reported PCR values are known to the server that they represent a trusted configuration of the platform, the server can trust the platform. If the client platform may have many different configurations, the server needs to maintain a large database of possible trusted client configurations. Instead of that each distributed application maintains its own list of trusted client configurations, a trusted third party can provide a service to maintain such a database, to which the server can ask whether a particular set of PCR values is trusted.

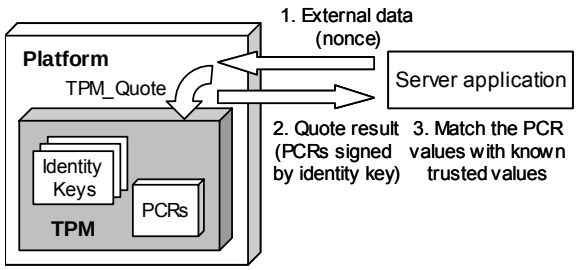

Figure 3.Reporting PCR values reliably to remote hosts

Note that TCPA identity is essentially the pseudonym generated inside the TPM chip. The private part of the identity key never gets out of the chip in plaintext. The public key is exported and signed by a Certificate Authority the user has chosen. This CA is called privacy CA, because only this CA can relate the identity key to individual TPM chip.

3. Linux Kernel Measurement in Boot Loader

IBM's TCPA-enabled ThinkPad currently supports measured bootstrap in the sense that its unmodifiable part of the BIOS measures the remainder of the BIOS and the BIOS also measures the MBR (Master Boot Record) of the bootstrap device (usually the hard drive). However, no further measurement is done for any commercial operating systems today to our knowledge.

For measuring the integrity of an operating system, an OS boot loader should take the responsibility according to the *TCPA PC Specific Implementation* specification [2], because the BIOS does not usually directly load the operating system. Instead, the BIOS loads a portion of a boot loader (which resides in the MBR) into the memory and transfers the control to the loaded code. In order for the boot loader to properly measure the integrity of the operating system, the boot loader needs to satisfy the following requirements:

1) If the boot loader has multiple stages, an executing stage must be able to measure the next stage and extend the PCR value prior to transferring the control to it.
2) The boot loader should be able to measure the O/S image and all security-critical configuration files before transferring the control to the O/S.. This implies that the boot loader must have at least a partial implementation of the file system on which these files are stored. .

For Unix-like operating systems for PC, several boot loaders are available now under the GPL license, such as LILO (LInux LOader) [7], GRUB (GRand Unified Bootloader) [8] and XOSL (Extended Operating System Loader) [9]. We adopt GRUB for our prototyping of a trusted boot loader because it is widely used and it looked feasible to modify it so that the modified boot loader satisfies the above requirements.

3.1 Boot Sequence in GRUB

GRUB is divided into multiple stages, that is, *Stage 1*, *Stage1.5*, and *Stage 2*. This is due to the restricted size of disk space in the current PC Architecture. *Stage 1* code was designed to fit in the MBR space that is 446 bytes long. *Stage 1.5* contains the file system implementation, so it is significantly larger (approximately 10KB) and placed in the sectors right after the MBR which are normally unused space. Since Stage 1.5 can now access the file system, *Stage 2* code and the GRUB configuration file can be placed anywhere in the filesystem.

Figure 4 shows the boot sequence of GRUB. BIOS loads *Stage 1* (MBR) data from the hard drive into memory and jumps to the starting point of *Stage 1*. *Stage* 1 loads the first sector of *Stage 1.5* and jumps to it. This *Stage 1.5* loads the rest of *Stage 1.5* sectors into memory and jumps to it. Now *Stage 1.5* has filesystem interface and finds and loads *Stage 2* into memory, and then jumps to the start address of *Stage 2*. At this point, GRUB checks the configuration file and displays an O/S selection menu to the users. After the user selects an O/S, GRUB loads the selected O/S image into memory and starts the O/S boot process.

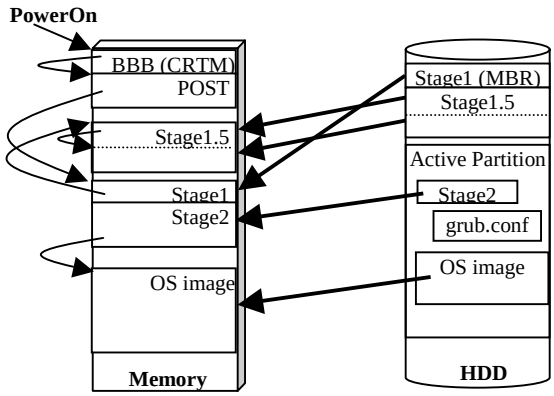

Figure 4. Boot sequence of GRUB

3.2 Measurement Steps in GRUB

The chain of measurement must also reflect this bootstrap sequence: The Stage 1 must measure the first sector of the Stage 1.5, and the first block of the Stage 1.5 must measure the rest of the Stage 1.5. Then the Stage 1.5 measure the stage 2, which finally measures the operating system. Because the Stage 1 and the first sector of the Stage 2 have a severe size limitation, inserting integrity measurement codes to these stages was our major challenge.

Stage 1 code, which is loaded from the MBR, has to measure the first sector of the Stage1.5. This means that we must compute the hash of the loaded sector and call the TPM_Extend function of the TPM. However, the original GRUB code already used up almost all of the 446 bytes allocated for the MBR. Fortunately, we could call a BIOS service that does exactly this. Still, we were forced to discard the support of an older drive format (the C.H.S. mode as opposed to the LBA mode).

Basically, each step of a trusted boot must 1) load the next stage data from the hard drive into memory, 2) measure the loaded memory image and extend the PCR value using the measured value, and 3) jump to the entry point of the next stage.

The next stage, the first sector of Stage 1.5, has the same space limitation as Stage 1 and was modified in the almost same way as Stage 1. The remaining part of Stage 1.5 has no such space limitation and it was easier to add the capability of measuring the next stage, Stage 2.

Stage 2 is responsible for measuring several things. First, Stage 2 measures the configuration file of GRUB (`grub.conf`). This file is used for specifying bootable O/S images that will be presented in the selection menu. Second, Stage 2 is required to measure the O/S image that is selected to boot. Third, Stage 2 is

also responsible for measuring any other security-critical files (configuration files, library files, etc.). To enable these measurements, we have extended `grub.conf` in the following two ways;. (1) all loading commands such as "kernel" measures the target image automatically, and (2) we added the "measure" command that measures any file specified in its argument. See Figure 6 for a sample configuration file.

```
title Measured SELinux
root (hd0,0)
measure (hd0,1)/etc/security/policy.12 8
measure (hd0,1)/opt/jdk/jre/lib/security/java.policy 8
measure (hd0,1)/opt/jdk/jre/lib/security/java.security 8
kernel –pcr=8 /vmlinuz-2.4.20-selinux ro \
root=/dev/dha1 enforcing=1
…
```

Figure 5. Example of `grub.conf`

During these steps, we used PCR#8 for measuring Stage 1.5, Stage 2, and the O/S files.

The size of the modification is summarized in the following table.

Component	Language	LOC changed
Stage 1 (stage1.S, stage1.h)	Assembler	150
Stage 1.5, first sector	Assember	150
Stage 1.5, the rest	C	50
Stage 2,	C	650

Measuring OS Kernel and Configurations

In our implementation, the chain of the measurement process covers OS kernel, security-critical files such as OS configuration files, security policy files, and, a few executables such as Java virtual machine. However, if a large number of files were measured, the number of possible PCR values of trusted configuration would be combinatorial and therefore the server-side database of the trusted PCR values would be unmanageable. Instead, we let the OS kernel be responsible for maintaining the integrity of less security-critical files and resources. We apply the Linux Security Module (LSM) [13] and the Security-Enhanced Linux (SELinux) [14] to the kernel.We apply LSM and SELinux to the kernel. If an appropriate SELinux policy file is used, the combination of LSM and SELinux should provide a better protection against unwanted modifications on any parts of the system, such as executables, libraries, and other data files (see Figure 6)

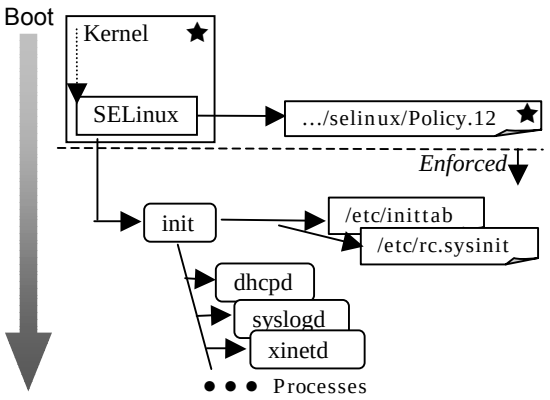

Figure 6. Boot sequence of SELinux

Note that the integrity measurement done at the bootstrap time does not guarantee the integrity of the measured code *after* bootstrap. If the O/S does allow some program to modify the already-measured portion of the system, the PCR values stored in the TPM no longer reflect the correct integrity status. Therefore, a strong protection against illegal modification at the OS level is very important.

4. Reporting Integrity Metrics to Remote Applications

After the operating system (Linux) is successfully booted through the process of integrity measurement, any application can be executed on this platform as long as it is authorized by the access control enforcement mechanism within the OS. In a networked environment, the platform is accessed by an application running on the remote server. In both cases, the user of the device may want to provide the server with the value of the integrity metrics as a proof that the application component is running in a platform with the expected configuration. This is one of the most important parts of TCPA functionalities because it conveys hardware-shielded security information to the application, even beyond network, thus contributes to serve an end-to-end security mechanism.

In this section, we describe how to make use of the integrity reporting functions from the application layer perspective.

4.1 TCPA Supporting Software

Figure 7 shows the supporting software of TCPA. An application accesses TCPA functions via these software components, which wrap communication with TPM, manage resource objects, and provide standardized interfaces. Note that each component of the Service Providers provides high level APIs for upper layers, while the Core Services serve as a common interface to TPM with advanced features such as multi-threaded access, memory management, etc. Each component software stack is measured and/or protected by the OS kernel and its access control policy, which are also measured during the measured boot.

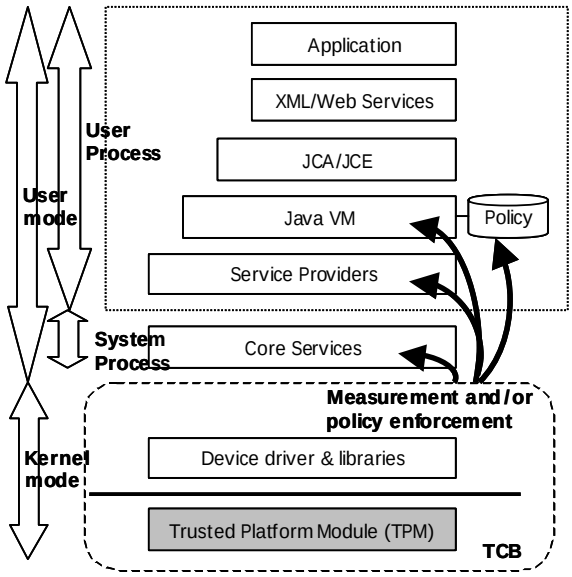

Figure 7. TCPA Supporting Software

4.2 Signature with integrity measurement values

As described in Section 2.3, the results of integrity measurements are stored in the PCRs inside the TPM and are reported by the quote function. Actually, this usage of the quote function is not limited to the response to the server challenge. One of the ways to use the platform integrity metric is to include the metric in a digital signature. To realize this idea, we defined a signature algorithm for XML Signature [5] that includes the PCR values. The PCR values represent the software configuration of the platform at the time of the signing. If there is any question about the signature (such as a virus might have been active when the signature was done), a signature verifier is able to examine the PCR values associated with the signature whether there have been known vulnerabilities in the specific platform configuration (BIOS and OS revisions, configuration files, antivirus definition files etc.).

4.2.1 Signing algorithm

We have defined our new signature algorithm as a concatenation of a structure representing the current PCR values (quoteInfo) and the signature value on the structure as follows.

```
SignatureValue =
 quoteInfo | SignatureOnQuoteInfo
```

The first part of the signature value (quoteInfo) is a 48 byte data object as defined by TCPA_QUOTE_INFO as follows:

```
typedef struct tdTCPA_QUOTE_INFO{
   TCPA_VERSION version;
   BYTE fixed[4];
   TCPA_COMPOSITE_HASH digestValue;
   TCPA_NONCE externalData,
} TCPA_QUOTE_INFO;
```

Where

- `version` is TCPA version as defined in Section 4.5 of TCPA 1.1b. Specifically, its first two octets MUST be 0x01 and 0x01.
- `fixed` is always the ASCII string “QUOT”.
- `digestValue` is the result of the composite hash algorithm using the current values of the requested PCR indices.
- `externalData` is the SHA-1 hash of the octet stream of the canonicalization of <SignedInfo>

The second part of the signature value is the actual RSASSA-PKCS1-v1.5 signature value on the TCPA_QUOTE_INFO data structure. This part is at least 256 byte long because the length of an attestation key is at least 2048 bits.

4.2.2 Verification algorithm

A verifying application of XML Signature with TCPA PCR Values needs to split the base64-decoded <SingatureValue> into two parts. The first part consists of the first 48 bytes of the octet string and the second part is the rest. The verifying application verifies the following things.

1. The first part contains a valid TCPA_QUOTE_INFO structure. This means the first two octets of the version field must be 0x01 and 0x01, and the fixed field must be ASCII “QUOT”.
2. The externalData field of the first part is the SHA-1 hash value of the canonicalized <SignedInfo>.
3. The second part MUST be the RSASSA-PKCS1-v1.5 signature of the first part according to the given public key.

The verifying application can then verify the PCR values (the digestValue field) in the TCPA_QUOTE_INFO structure against known trusted values. If the value is known to be trusted, the server can conclude that it is communicating with a trusted platform. Otherwise, the server may reject the request according to its policy.

4.2.3 Implementing PCR-enabled signature in JCE

We implemented this as a new signature algorithm (called “SHA1withRSATcpa”) in a Java Crypto Environment (JCE) provider. This way, the application does not need to be modified to use the signature with PCR values.

5. Demonstration Scenario

In order to show the end-to-end integrity value reporting capability, we implemented a demonstration prototype using Web service. **Figure 8** shows the structure of this demo system. The client, which is our Linux-based system with TCPA attestation, sends a request to a Web service. The Web service needs to authenticate the client based on a digital signature attached to the SOAP message, as defined in the OASIS WS-Security specification [6]. In this signature, which is defined as XML Signature, we use the Signature algorithm described in Section 4.2. Therefore, all the messages originated from this client will have the platform measurement values.

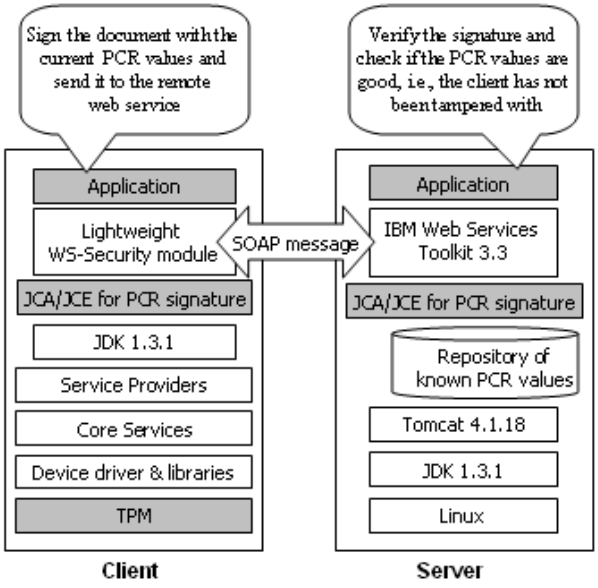

Figure 8. Demo system configuration

When receiving a request from this client, the server does the normal verification of the signature. Our signature algorithm first verifies the signature and then compares the measurement value with the known trusted values in the local database.

Figure 9 shows the client GUI. This client is implemented on top of a cellular phone emulator. The right hand side window shows the SOAP request/response messages. We envision that the platform integrity measurement will become more important for mobile devices such as ThinkPad, PDAs, and even mobile phones, because these mobile devices are easier to be lost or stolen.

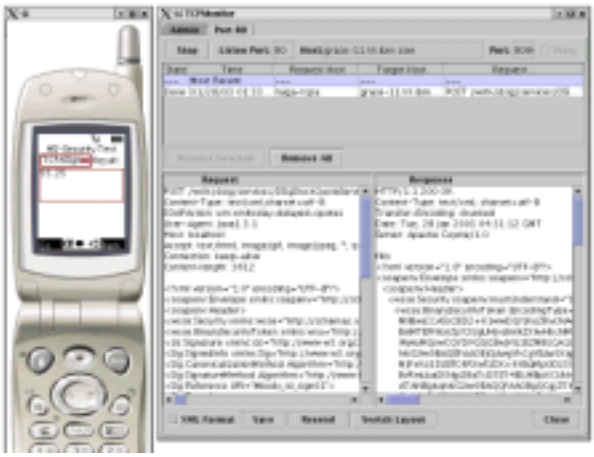

Figure 9. Screen capture of demo client GUI

6. CONCLUSION

We showed a Linux-based system that measures its integrity during a bootstrap process and reports the integrity metric to remote servers. . An application can include the measured values in a digital signature so that the integrity of the system software can be verified at the signature verification time. The modification on the O/S boot loader was minimum and there is no change on the O/S itself except for the TPM device driver and related API that is implemented.

The implementation is a proof that the TCPA integrity measurement really works. Although we need to do thorough investigations what pieces of the O/S should be measured to ensure the overall integrity of the platform, this is certainly a first concrete step towards trusted platforms.

7. Acknowledgement

We thank R. Catherman for his implementation of TCPA support software for Linux and also for his comments. We also thank Frank Seliger for his valuable discussions.

8. REFERENCES

[1] Trusted Computing Platform Alliance, *TCPA Main Specification v.1.1b*, February, 2002. http://www.trustedcomputing.org/docs/main%20v1_1b.pdf

[2] Trusted Computing Platform Alliance, TCPA PC Specific Implementation Specification v1.00, September 2001. http://www.trustedcomputing.org/docs/TCPA_PCSpecificSpecification_v100.pdf

[3] Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, Greg Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," USENIX Security Symposium, 2002.

[4] Stephen Smalley, Chris Vance, and Wayne Salamon, "Implementing SELinux as a Linux Security Module," http://www.nsa.gov/selinux/module-abs.html, Dec., 2001.

[5] W3C Recommendation, *XML Signature*, http://www.w3.org/Signature/, Feb. 2002.

[6] OASIS TC draft, *Web Services Security Core*, http://www.oasis-open.org/committees/wss/, Dec. 2002.

[7] LILO, http://brun.dyndns.org/pub/linux/lilo/.

[8] GNU GRUB, http://www.gnu.org/software/grub/.

[9] XOSL, http://www.xosl.org/.

[10] Microsoft, Authenticode, http://www.microsoft.com/security/tech/authenticode/default.asp.

[11] Tripwire Open Source Project, http://www.tripwire.org/.

[12] Li Gong, *Java (TM) 2 Platform Security* Module, http://lsm.immunix.org/

[13] Security-Enhanced Linux *Architecture*, http://www.nsa.gov/selinux/indexjava.sun.com/j2se/1.4/docs/guide/security/spec/security-spec.doc.html .