A New High-Performance Approach for Offline Replacement Attack Prevention in Trusted Clients

Hossein Rezaei Ghaleh
Department of Computer Engineering and Information Technology
Amirkabir University
Tehran, Iran

Siavash Khorsandi
Department of Computer Engineering and Information Technology
Amirkabir University
Tehran, Iran
khorsand@aut.ac.ir

***Abstract*— Trusted Computing has been a major research issue in recent years. Software integrity is a main part in a trusted computing environment. As a chain of invocations are involved in a computing system, it is imperative to build a trust relationship between various layers in the system. TLC is a novel approach proposed to build a trusted Linux system. However, it suffers from offline replacement problem. In this paper we propose a high-performance approach based on blacklist checking to countermeasure this problem. We have developed and presented an accelerated mechanism to maintain system performance during integrity checking phases. Tow main ideas are used for this purpose are synchronous cache consistency and blacklist partitioning with embedded blacklist identity. In addition, an analysis framework is developed for performance of the proposed approach that incorporates all important system and workload parameters.**

Keywords-trusted computing; operating system; replacement attack; cache consistency

I. INTRODUCTION

Trusted Computing considers many fields that one of this is software integrity. As these days the computer systems use layered and modular architecture, it is imperative to establish a trusted chain between the layers. For example, when we are not sure of the operating system's security, we can not trust the security of the application being executed by the system. The integrity of software means that we make sure of the integrity of the original code and all software modules involved in its execution. Consequently, when the module is loaded we would be certain that there exist no malicious code in it, or its data have not been tempered by unauthorized entities.

One leading approach to help in software integrity is developing the TLC project. But the TLC is vulnerable to replacement attack. We have proposed an improvement for TLC that defends it against this attack. Also we have used cache mechanism for better performance and we have proposed a new policy to improving it.

In reminder of paper, we have reviewed related works that have been attempted to support trusted client platform. In next section we have introduced Trusted Platform Module (TPM) that has been designed by TCG. After that we consider TLC closely and we have introduced its modules. Then we have explained replacement attack that is feasible on TLC system. In next section we have proposed our method that uses the blacklist for solving the problem. In continue we have considered performance problems and we have proposed a new policy for caching mechanism for improving it. In last section we have analyzed system performance.

978-1-4244-5113-5/09/$25.00 ©2009 IEEE

II. TRUSTED CLIENT PLATFORM

A. Related Works

Many projects have been down on software integrity, secure booting, application isolating and other technique for securing system with different approaches such as home and network applications. In this section most of the efforts made in this area are examined.

First method is AEGIS [8, 9]. It makes some modification on the standard startup of the computer and by adding a chip to the motherboard, uses this device as a root of trust. In this method, the digital signature has been used and each layer before launching of the next layer verifies its digital signature. By doing this, upon loading of each layer, a chain of trust is formed. In case of any error during this process, the system automatically connects to a special server and recovers the lost module. The chip used in this method contains codes for cryptography, required network protocols for recovery of the lost module and the digital signature of the one or a number of lower layers of the system [13]. sAEGIS is another method [7] that has added some new capabilities to AEGIS and some improvements have been made. One of they is using of smart card for access to the system. This card contains the digital signature of higher layers such as operating system and applications and is protected by a PIN code.

BitLocker [1, 2 and 5] is a method that is a part of Windows Vista provided by Microsoft and uses TPM. This software in order to provide the highest level of security requires the TPM hardware to be installed on the motherboard and support special BIOS. The unique advantage of this software is simplicity that it provides appropriate protection for the user's data through encryption before storing the data on the hard disk and during the execution. It also provides separate partitions for the operating system and its modules and keeps them encrypted.

NGSCB [18] and Terra [15] are different methods that attempt to make the system secured by same idea. Both NGSCB and Terra explore a virtual machine monitor (VMM) to partition a tamper-resistant hardware platform into multiple isolated virtual machines. This mechanism is useful in new approaches [6, 10]. In NGSCB, a system is partitioned into two parts: trusted and untrusted, and only the trusted part is attested. Therefore, to ensure service trustworthiness, the service provider platform has to treat the service and all its code as trusted, which may not be true all the time. Terra partitions the system into virtual machines, each of which may be dedicated to a single application (e.g., a service). As such, the trustworthiness of a service can be evaluated by attesting its virtual machine. This attestation, however, is done at memory block level, which incurs high CPU and memory overhead. Terra achieves higher assurance of attestation because of strong process isolation provided by VMM, but lacks the capability of ensuring simple and efficient trusted execution across transactions.

Trusted Linux Client (TLC) is another method [12, 17]. The goal of this project is to protect desktop and mobile Linux clients from on-line and off-line integrity attacks, while remaining transparent to the end user. This is accomplished with a combination of a TPM security chip, verification of extensible trust characteristics, including digital signatures, for all files, authenticated extended attributes for trusted storage of the resultant file security metadata, and a simple integrity oriented Mandatory Access Control enforcement module. The resultant system defends against a wide range of attacks, with low performance overhead, and with high transparency to the end user. In continue we focus on TLC and TPM with more details.

B. Trusted Platform Module

The Trusted Computing Group has defined an open specification for TPM [19], which has been implemented by multiple chip vendors, and incorporated into desktop and mobile systems from the major manufacturers.

While the full TPM specification is quite long and difficult to understand, the chip's basic functionality is simple. From a programmer's perspective, a TPM looks like the following logical diagram.

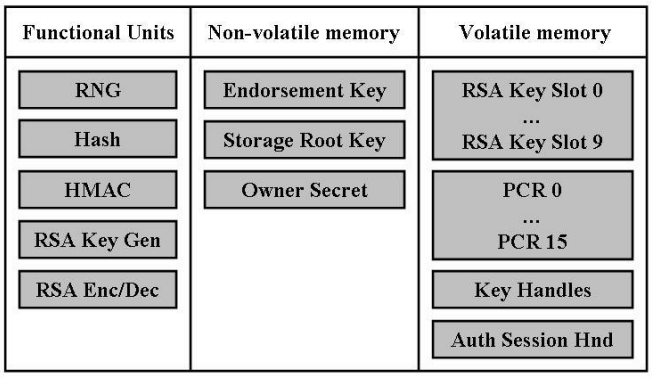

Figure 1. TPM logical diagram

The chip has a hardware random number generator (RNG) and RSA engine for on-chip key pair generation. When a key pair is generated, the private part is encrypted by the Storage Root Key (SRK) or a descendant, and the resultant pair exported out of the chip for storage. The chip has 10 volatile slots into which the key pairs can be loaded, decrypted, and then used for signature, encryption, or decryption. (Signature verification is not done on-chip, as it is not a sensitive operation.)

The TPM chip also has 16 Platform Configuration Registers (PCR), which are used to securely store 160 bit hashes. These hash registers are used to store hashes of the software boot chain (BIOS, master boot record, grub bootstrap loader, Linux kernel, and initial ramdisk image). Then the usage of keys for encrypting or decrypting can be tied to specific values of these PCR registers, so that if any part of the measured software is altered, the decryption is blocked. In TPM terminology, encryption tied to a specific PCR value is called “sealing”, and the corresponding decryption called “unsealing”. Malicious alterations to the master boot record, grub, kernel, or initrd cannot escape detection through the PCR values, as the measurements are always done on the next boot stage, before execution is transferred to it. Since the TPM hashes all presented data into a given PCR, it is computationally infeasible for malicious code to calculate and submit a measurement which would result in a target “correct” value after this hashing.

C. Trusted Linux Client

Many projects have been done for trusted computing in Linux environment such as Trusted Grub, IMA [11] and TLC [17]. In this paper we focus on TLC that has introduced a new method that combine code-signing with TPM. In this method tow major module has been developed; Extended Verification Module (EVM) and Simple Linux Integrity Module (SLIM).

For security reasons, it is desirable to check security characteristics, including the authenticity, integrity, revision level, and robustness of an application before its execution, to determine whether or not to run the executable, or under what level of privilege to run it. EVM presents a single comprehensive mechanism, Extended Verification, to cover security goals such as Message Authentication Codes, signed executables, anti-virus, and patch management systems, but implemented in a single, optimally fast mechanism, with a flexible policy based management system. This module proposed that executables be digitally signed, and that the kernel check the signature every time an executable is to be run, refusing to run it if the signature is not valid. Viruses or other malicious codes, lacking a valid signature would be unable to run. The digital signatures on the executables can be of two types: symmetric or asymmetric. A symmetric signature uses a secret key to key a Message Authentication Code (HMAC), taken across the entire content of the executable file. Symmetric signatures can be verified with relatively little overhead, but the key must remain secret, or the attacker can forge valid signatures. This makes symmetric signatures useful mainly in the local case. In addition, the key must be kept secret on the local machine, and this is very difficult to do. EVM provides a single, symmetric key based verification

function, with TPM protection of the key. Also EVM uses a policy based verification function based on storage of verification data in authenticated extended attributes.

In normal operation, when a new executable is installed, it is first checked by all of the verification methods listed in the EVM policy file, and the results inserted into the extended attribute list, along with a hash of the file, and HMAC of the attributes. At run time, the kernel then looks at the verification attributes and rapidly compares them to the current policy, and determines how to run it according to policy. Checking the header does require hashing the executable file, to verify that it hasn't been modified, but this hash and subsequent symmetric key HMAC is very fast compared to the original checking methods, and is cached until the next reboot. Thus the verification is done in optimal time, allowing checking on all accesses.

The EVM module verifies that all files are authentic, unmodified, current, and not known to be malicious. EVM does not (and cannot) determine if files are correct - that is that given any (possibly malicious) input data that they will operate properly. A data driven compromise of the operation of verified files can still lead to the compromise of a system, despite EVM checking. An integrity enforcing model is needed to block, or at least contain any such compromise. The Simple Linux Integrity Module (SLIM) classifies programs as trusted or untrusted based on the verifications done by the EVM, and then enforces the access policy.

Figure 2 has been shown keys relationships in TLC. In this method when system has turned on, the PIN of TPM was requested from user. After that TPM measures boot sequence software and store their hashes in PCRs.

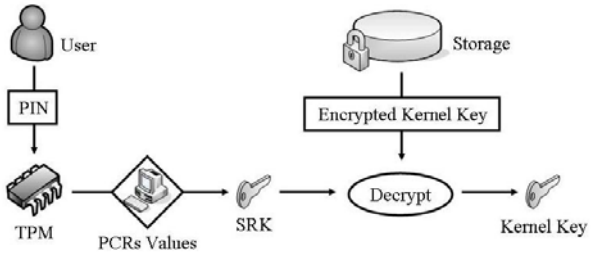

Figure 2. Keys relationships in EVM

If the PIN has verified and PCRs are correct then SRK become unsealed. After this process we can use SRK for decrypting kernel key. By kernel key, EVM can verify every file's digital signature. At boot time the kernel key is loaded to Linux kernel. After that in every open file process, EVM verify HMAC of file by kernel key.

D. Replacement Attack to TLC

In the TLC when we install new program on operating system, we sign all files of that program. In this situation the files that we have not installed have not valid digital signature and could not be run. This mechanism defenses system against malicious programs such as viruses, but the system is not secure against vulnerable program such as bugged software. For example, a trusted program such as a driver or application has some security bug and has a drawback for hackers. Then the developer of this program find this bug and fixes it. So, we get new version of the program, sign it and overwrite on old version. But the attacker can replace old version and new version with offline attack and our system can not detect it.

In TLC when a bugged program has been replaced with new version, EVM sign new file, but the attacker can replace old bugged file with new version and EVM can not detect it, because old file has valid signature. There is a simple solution for this problem and it is resigning all files by new kernel key, but it has much performance penalty and is not practical. In next part we explain a solution for this problem by using of blacklist.

III. REPLACEMENT ATTACK PREVENTION

In this part we explain some solutions for solving the replacement attack. At first, we review existing method for countermeasure the attack and we show they are not feasible. Then we explain our solution in detail. Our method is practical in many systems, but we design it with focus on the TLC.

A. Issues and Requirements

There are some solutions for solving replacement attack. First method is access control. We can use a strong physical access control mechanism to prevent the attacker from copy his files to our system. In this situation, the only way to access the system is network access that is through our secure operating system. So our system check the version of files before install it and can prevent form replace new version by old version.

But this method is not feasible in many situations. In normal conditions, many people have access to system and can boot it with another operating system such as Live-OS. Then the attacker can copy old version bugged program and use it when original operating system running.

Another solution for solving replacement attack problem is maintaining a versions list. In this method the operating system maintains a list that has an entry for each file that is updated. So, when operating system loads the file, check this list that the file is last version or not. The hash of versions list can be stored in the TPM for preventing unauthorized changes in the list.

This method is not suitable because the versions list grows up quickly and the opening and loading the file consumes long time.

Another solution is generating new kernel key and revoke old key and signatures. In this method, the TPM replace old key with new key and operating system signs all files with new key. It is clear that not feasible because has mush performance penalty.

B. Proposed blacklist approach

The blacklist is a list that has the name or identifier of entities that are revoked or malicious. This list is very useful in secure systems such as PKI-enabled systems. In these systems, Certificate Authority (CA) that issues certificates, publishes Certificate Revoked List (CRL) that lists revoked and

invalidate certificates. By using this mechanism, system can detects entities that are valid previously, but are not valid now.

In our solution, we have used the blacklist for defense system against replacement attacks. We have changed the algorithm of loading and opening files in the TLC. We have added a new step to it that check blacklist for the file identifier. Figure 3 has shown this algorithm.

```
BEGIN
IF HASH(file.data)<>file.attrib.hash
      ERROR
ELSE IF HMAC(kernelkey, file.attrib)
      <>file.attrib.hmac
      ERROR
ELSE IF file.attrib.id IS IN
      FILE(file.attrib.blacklist)
      ERROR
ELSE
      VERIFIED
END
```

Figure 3. Enhanced file opening and verification algorithm

In this method according to TLC, the file has some additional security attributes and we have added some new attributes to it. When a file is requested for opening, first the hash of its content is calculated. Then this value is compared with correct hash value that is stored in file's security attributes. If these tow values are equal then HMAC of file's security attributes except "hmac" attribute, is calculated by kernel key and is compared with hmac value stored in file's security attributes. If these tow values are equal then the blacklist file address and file's identifier that is a numeric value, are extracted from file's security attributes. Then the blacklist file has been searched for this file's identifier. If this identifier is not in the blacklist then it has security conditions for executing. If at least one of the conditions on above algorithm is not passed then the file can not be executed or loaded.

In TLC the last step is not exist and therefore it is malicious against replacement attack, because old signed bugged executable file can be verified. But in our solution old file is registered in the blacklist and can not pass last step.

A problem exists yet, because the attacker can modify the blacklist file and can delete the entry of his bugged file. We have used the TPM for solving this problem. We have added blacklist files hashes to TPM's PCRs and we have sealed kernel key with this value and then if any file of the blacklists has changed, then the kernel key can not be extracted.

Another problem is performance penalty in our solution for big blacklist file. In the bingeing, this file is very small and therefore loading and searching in it is quick. But when bugged and invalid files have been increased, the blacklist file become larger and larger. For solving this problem we have used a mechanism similar to Distribution Point in CRLs. In this mechanism, the address of blacklist has been stored in the file. Therefore we have more than one blacklist and each data or executable file has a reference to it's correspond blacklist. By breaking the one blacklist file to more files, searching will be quicker.

Also we need an identification item to detect each file uniquely entire the system. We have defined an ID for each file that is sequential and incremental. This ID is unique in the system and assigned when we sign the file. Also we save last assigned ID in the TPM securely.

According to above discussion, we have added some new security attributes to file header. These attributes have been added to security.evm and have been shown in figure 4.

```
security.evm.id
security.evm.blacklist
```

Figure 4. Added security attributes to security.evm

For example *security.evm.id* can be '0807191727552' and *security.evm.blacklist* can be '/etc/blacklist/1.blk'. Also we have added some new lines to *grub.conf* file that force GRUB to calculates hash (measure) of blacklist files and extends a PCR that seal kernel key. Figure 5 shows an example.

```
Measure (hd0,1)/etc/blacklist/1.blk 9
Measure (hd0,1)/etc/blacklist/2.blk 9
Measure (hd0,1)/etc/blacklist/3.blk 9
```

Figure 5. Example of adding measure of blacklist to grub.conf file

C. Consistent Cache Acceleration

There is one major problem when we add some security and verification mechanism to opening and loading files. This problem is performance, because we calculate hash of file's content and verify digital signature and check the blacklist and therefore we have tow or more access to the disk instead of one access for read a file.

There is same problem in TLC [17] and other methods [21] and they use some solutions such as caching of verification results. In this method, some changes are accomplished on OS kernel. For example when a file has being opened, hash of content has being checked and digital signature has being verified. Then the result that can be true or false has being cached and until this file has not opened with write access, cached result is valid. For implementation of this mechanism some of system calls such as open or execute must be changed. In our method that uses the blacklist, caching is suitable and we have used it.

For improving performance and more efficiency, we propose that verification cache (VC) and normal operating system file cache (FC) be consistent. It means that every cached file in FC has corresponded entry in VC. We can guarantee it by a policy that is discussed in continues. We force that if a file is in FC, the corresponded entry in VC maintained and is not overwrote. Also if a file that is in FC is opened with write access, after finish this operation, the OS confirms its modification by user, calculates its hash and HMAC and saves they in its security attributes. Therefore if a file is in FC, its corresponded entry exists in VC certainly.

IV. PERFORMANCE ANALYSIS

In this section we analysis the performance of a system that has our verification mechanism on opening and execution. We analysis it with mathematic formulas that shows efficiency is not depend to present file systems and file caching mechanisms. We have considered efficiency in read and write operations.

In this analysis, we have assumed that all required lists in caches or files are sorted and we can search in it by Binary Search algorithm with logarithmic order. Also this system has tow caches includes File Cache (FC) and Verification Cache (VC) with h_{VC} and h_{FC} hit rates. The file system can caches most recently used files by FC and can improves disk performance. VC is another cache that stores security verification results. If a file has been loaded and verified successfully then it's Id and verification result has added to VC. Therefore in the next read operation, verification mechanism has not down and VC has been referenced. We show the number of entry in VC and FC by n_{VC} and n_{FC}. Another assumption is that we have a blacklist that has n_{BL} entries.

We consider read efficiency by calculating T_{read} to $T_{verifiedread}$. T_{read} is the time of read operation in usual present file system and $T_{verifiedread}$ is the time in our system that has verification mechanism. Figure 6 shows the equations in read efficiency.

$$E_{read} = \frac{T_{read}}{T_{verifiedread}}$$

$$T_{read} = h_{FC}(T_{FCread}) + (1 - h_{FC})(T_{filediskread} + T_{FCwrite})$$

$$T_{verifiedread} = h_{FC}(T_{FCread}) +$$

$$(1 - h_{FC})(T_{filediskread} + T_{FCwrite} + T_{verify})$$

$$T_{verify} = h_{VC}(T_{VCread}) + (1 - h_{VC})(T_{measure} + T_{VCwrite})$$

$$T_{measure} = T_{filecontenthash} + T_{fileattribsHMAC} + T_{BLsearch}$$

Figure 6. Calculating of read operation efficiency (1)

In this analysis, we assumed some primitive time equations that are normal orders in traditional operating systems and hardware. Also, we chose a usual PC and calculated its performance parameters by a performance test tool and checked its test results with our theory equations that have been shown in figure 7. In these equations we assumed all lists are maintained sorted and HAMC has two hash operations according to [19]. Also in these lists T_x shows the time of operation 'x' in second, S_x shows the size of 'x' in byte and V_x shows the velocity of operation 'x' in byte per second.

We have done some performance tests for evaluating actual records for our PC [2, 3]. We employed PassMark™ Performance Test which is software with many tools for performance tests like CPU, memory, and disk test suits. The general system properties of our used test system are: Intel Pentium IV CPU with 3 GHz, 2048 KB cache size, 400 MB available RAM, and FAT32 file system. We have tested three main performance parts include CPU, memory and disk. In the CPU test we have done different 8 tests like floating point math, finding prime number, and image rotation. The memory test includes some parameters like allocating small block, reading cached and uncached data, and writing in memory. Last test suite is disk test that is very important because usually it is bottleneck for total performance. This test includes both sequential reading and writing, and random searching for reading and writing operations. Results of these tests showed that our system has following orders that have been shown in figure 8. In these equations the size of file cache (FC) calculated from Windows XP and 2003 server. Also we assumed that usual files are about 1 KB to 1 GB and its header has an order of 1000 bytes.

$$T_{FCread} = \log_2^{n_{FC}} \times t_{memread}$$

$$T_{FCwrite} = \log_2^{n_{FC}} \times t_{memread} + t_{memwrite}$$

$$T_{filediskread} = \frac{S_{file}}{V_{diskread}}$$

$$T_{filediskwrite} = \frac{S_{file}}{V_{diskwrite}}$$

$$T_{BLsearch} = \log_2^{n_{BL}} \times t_{memread}$$

$$T_{VCread} = \log_2^{n_{VC}} \times t_{memread}$$

$$T_{VCwrite} = \log_2^{n_{VC}} \times t_{memread} + t_{memwrite}$$

$$T_{filecontenthash} = \frac{S_{file}}{V_{SHA1}}$$

$$T_{fileattribsHMAC} = 2 \times \frac{S_{fileheader}}{V_{SHA1}}$$

Figure 7. Primitive time equations

$$t_{memread} = t_{memwrite} \sim O[10^{-7}]$$

$$V_{diskread} = V_{diskwrite} \sim O[10^{7}]$$

$$V_{SHA1} \sim O[10^{7}]$$

$$n_{FC} \sim O[10^{3}]$$

$$n_{VC} \sim O[10^{4}]$$

$$n_{BL} \sim O[10^{4}]$$

$$S_{fileheader} \sim O[10^{3}]$$

$$S_{file} \sim O[10^{x}] \rightarrow 3 \le x \le 9$$

Figure 8. Calculated and assumed primitive orders

Therefore we can calculate the order of read operation efficiency. It has been shown in figure 9. We can see that it depends to h_{VC} (hit rate on verification cache) only and if h_{VC} is 90% then E_{read} is about 91% and it is acceptable.

$$T_{read} \sim (1-h_{FC})O[10^{x-7}]$$
$$T_{verifiedread} \sim (1-h_{FC})(2-h_{VC})O[10^{x-7}]$$
$$\Rightarrow E_{read} = \frac{1}{2-h_{VC}}$$

Figure 9. Calculating of read operation efficiency (2)

Calculating of write operation efficiency is similar to read operation and has been shown in figure 10. In verified write operation, we calculate the hash and HMAC of the file and then update VC. We can see that efficiency of write operation is about 1 and our verification mechanism doesn't affect it.

$$E_{write} = \frac{T_{write}}{T_{verifiedwrite}}$$
$$T_{write} = T_{FCwrite} + T_{filediskwrite} \Rightarrow T_{write} \sim O[10^{x-7}]$$
$$T_{verifiedwrite} = T_{FCwrite} + T_{filediskwrite} + T_{measure}$$
$$\Rightarrow T_{verifiedwrite} \sim O[10^{x-7}]$$
$$\Rightarrow E_{write} = 1$$

Figure 10. Calculating of write operation efficiency

V. CONCLUSION

In this paper we reviewed trusted client platform and focused on the systems that use TPM such as TLC. In continue we proposed an improvement to its verification mechanism. This improvement was blacklist checking in file opening and loading operations. Also we considered performance penalty and we proposed a cache consistency policy to improve hit rate. Our analysis showed that the difference between performance of normal system and secure system in our method is very little and depends to Verification Cache hit rate only that is approximately 90 percent.

ACKNOWLEDGMENT

I would like to thank Dr. Khorsandi for his advice and for his suggestions to improve the language of paper. Also I thank Dr. Doostari at Shahed University for his contributions.

REFERENCES

[1] H. Rezaei Ghaleh, M.A. Doustari, *A new approach for secure and portable OS*, The Second International Conference on Emerging Security Information (Securware 2008), Systems and Technologies, IEEE Computer Society, August 2008.

[2] H. Rezaei Ghaleh, S. Norouzi, *A new approach to protect the OS from off-line attacks using the smart card*, The Third International Conference on Emerging Security Information, Systems and Technologies (Securware 2009), IEEE Computer Society, June 2009.

[3] H. Rezaei Ghaleh, M.A. Doustari, *Improving the Client Security using the Smart Card and Trusted Server*, International Conference on Security and Management (SAM 2009), WROLDCOMP'09, July 2009.

[4] Peng Shaunghe, Han Zhen, *Enhancing PC Security with a U-Key*, IEEE Security and Privacy, Volume 4, Issue 5, September 2006.

[5] *BitLocker Drive Encryption Technical Overview*, Microsoft TechNet, 2008.

[6] Peter M. Chen, Brian D. Noble, *When Virtual Is Better Than Real*, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, IEEE Computer Society , 2001.

[7] Naomaru Itoi, William A. Arbaugh, Samuela J. Pollack, Daniel M. Reeves, *Personal Secure Booting*, Proceedings of the 6th Australasian Conference on Information Security and Privacy, Springer-Verlag, 2001.

[8] William Albert Arbaugh, *Chaining layered integrity checks*, University of Pennsylvania, Philadelphia, PA, USA, 1999.

[9] Arbaugh, Farber, Smith, *A secure and reliable bootstrap architecture*, IEEE Symposium on Security and Privacy, 1997.

[10] Hermann Hartig, *Security architectures revisited*, Proceedings of the 10th workshop on ACM SIGOPS European workshop, 2002.

[11] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, Leendert van Doorn, *Design and Implementation of a TCG-based Integrity Measurement Architecture*, 13th Usenix Security Symposium, San Diego, California, August, 2004.

[12] Hiroshi Maruyama and others, *Trusted Platform on demand (TPod)*, IBM, February 1, 2004.

[13] William A. Arbaugh, Angelos D. Keromytis, David J. Farber, and Jonathan M. Smith, *Automated Recovery in a Secure Bootstrap Process*, Network and Distributed System Security Symposium, Internet Society, March 1998.

[14] James Hendricks, Leendert van Doorn, *Secure bootstrap is not enough: shoring up the trusted computing base*, Proceedings of the 11th workshop on ACM SIGOPS European workshop, 2004.

[15] Tal Garfinkel, Ben Pfaff, and others, *Terra: a virtual machine-based platform for trusted computing*, Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003.

[16] M. Nakamura, Seiji Munetoh, *Designing a trust chain for a thin client on a live Linux CD*, Proceedings of the 2007 ACM symposium on Applied computing, 2007.

[17] D. Safford and M. Zohar, *A Trusted Linux Client (TLC)*, Technical Paper, IBM Research, 2005.

[18] P. Englund, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, *A trusted open platform*, IEEE Spectrum, pages 55–62, 2003.

[19] *TCG TPM Specification, Version 1.2*, Trusted Computing Group, 2005, https://www.trustedcomputinggroup.org/.

[20] *TCG Specification Architecture Overview, Revision 1.2*, Trusted Computing Group, April 2004, https://www.trustedcomputinggroup.org/.

[21] L. van Doorn, G. Ballintign, and W. A. Arbaugh, *Signed executables for linux*, Technical Report CS-TR-4259, University of Maryland, 2001.

[22] Siani Pearson, *Trusted Computing Platforms: TCPA Technology in Context*, Prentice Hall PTR, 2002.

[23] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn, *A Practical Guide to Trusted Computing*, IBM Press, 1st edition, January 2008.