Practical assessment of Biba integrity for TCG-enabled platforms

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

Practical Assessment of Biba Integrity for TCG-enabled Platforms

Roberto Sassu, Gianluca Ramunno, Antonio Lioy
Dip. di Automatica e Informatica, Politecnico di Torino, Italy
Email: first.last@polito.it

***Abstract*—Checking the integrity of an application is necessary to determine if the latter will behave as expected. The method defined by the Trusted Computing Group consists in evaluating the fingerprints of the hardware and software components of a platform required for the proper functioning of the application to be assessed. However, this only ensures that a process was working correctly at load-time but not for the whole life-cycle. Policy-Reduced Integrity Measurement Architecture (PRIMA) addresses this problem by enforcing a security policy that denies information flows from potentially malicious processes to an application target of the evaluation and its dependencies (required by CW-Lite, an evolution of the Biba integrity model). Given the difficulty of deploying PRIMA, as platform administrators have to tune their security policies to satisfy the CW-Lite requirements, we propose Enhanced IMA, an extended version of the Integrity Measurement Architecture (IMA) that, unlike PRIMA, works almost out of the box and just reports information flows instead of enforcing them. In addition, we introduce a model to evaluate the information reported by Enhanced IMA with existing techniques.**

***Keywords*-Remote attestation; information flow; Biba integrity**

I. Introduction

The increasing number of cyber-attacks is pushing software developers, vendors and governments to fight this threat with greater effort than before [1].

A significant contribution in this direction comes from the Trusted Computing Group consortium (TCG). The TCG defined the specifications of a cryptographic chip, the Trusted Platform Module or TPM [2] which provides remote parties (via the *remote attestation* protocol) with the evidence that a platform is behaving as expected. If a platform is capable of demonstrating its *trustworthiness* from measurements of hardware and software components (e.g. a fingerprint of a firmware blob or file content) then a remote party (verifier) may *trust* that platform to perform the requested tasks, after comparing reported fingerprints against reference values known to be good.

The Integrity Measurement Architecture (IMA) [3] is one of the widely accepted TCG-compliant solutions (215 citations[1]). It provides measurements of a system up to application level (code executed and data read by processes) but with IMA a verifier cannot fully determine if an application will behave as expected in every situation or not. Indeed, while a software should work correctly (as intended by its developer) just after it has been loaded, it could start behaving unexpectedly (e.g. modifies user data in an unauthorized way) if it receives a malformed input from a corrupted module acting on the same system. The ability of an application to work correctly after loading is known as *load-time integrity* while *run-time integrity* refers to the whole process life-cycle.

Traditional integrity models (Biba [4] and Clark-Wilson [5]) ensure the run-time integrity of an application by preventing, with system-wide access control, the latter from reading low integrity data (malformed inputs) or by requiring (in the second model) the filtering of such data. Recently, these models were followed by CW-Lite [6], which remedies the scarce applicability of Clark-Wilson on conventional operating systems by lowering the requirements of formal code verification and the need of filtering for all application interfaces.

The CW-Lite model was the basis for a new proposal built upon IMA, Policy-Reduced Integrity Measurement Architecture (PRIMA) [6]. PRIMA overcomes IMA issues by isolating, with the SELinux software [7], the portion of the platform that is of interest to remote verifiers (a *target application* providing a service and its dependencies) from the rest of the system. By enforcing a SELinux policy where the rules do not violate the requirements of CW-Lite, PRIMA ensures that the integrity of the isolated portion is preserved for the entire process life-cycle and drastically reduces the likelihood of unknown measurements (frequent with IMA due to the necessity of inspecting all system processes), as only the isolated portion has to be measured.

Even if PRIMA's authors demonstrated through a formal proof the effectiveness of their solution with respect to IMA, it is not as user-friendly as the latter. Indeed, while IMA mostly works out of the box, configuring PRIMA means performing an expensive analysis of the reference SELinux policy to detect and eliminate CW-Lite violations. Although Jaeger et al. in a previous work [8] identified a nearly minimal portion of the Linux operating system that must be trusted, their result may not apply due to the specific configuration of the platform where PRIMA will run. If very skilled administrators may be able to accomplish this analysis, regular users are not likely to be included among the beneficiaries, as they are often not able to make security decisions [9] and "[...] expect a device with the maintenance factor of a toaster" [10].

Our work starts from Sailer et al.'s [3] idea of making IMA extensible, so that integrity relevant events not covered in the initial proposal can be captured in the future with enhancements to the base software. As one of the missing pieces is

[1] http://dl.acm.org/citation.cfm?id=1251391 (mid July 2014 access)

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

evaluating, in the absence of an isolation mechanism, if an application was compromised by another software running on the same platform, IMA could be extended to capture such type of events; this way, it would be possible to analyse the integrity of a system with the same degree of detail as PRIMA with a software that is accessible even for non-skilled users. We expect that, even if such modification could introduce some overhead both for capturing events at a higher rate and for managing a large amount of data, these issues are manageable: the delays experienced (in terms of pure system performance) will not be noticeable and managing measurements should be comparable to collecting the logs of a web server.

In this paper, we present an extended version of IMA (*Enhanced IMA*) that records all interactions occurring between processes through regular files. Developing the necessary extensions did not consist only in tweaking IMA to provide additional measurements, but required understanding of how to properly detect interactions between processes. Furthermore, we introduce a tool (*RA Verifier*) to represent the measurements of a platform with a graph. Through this graph, we are able to evaluate: (1) the load-time integrity of all processes executed with a known technique [11]; (2) the run-time integrity of a target application and its dependences (provided as input for the analysis), by checking whether they read a file written by the remaining (possibly malicious) processes (violation of the Biba model). The PRIMA verification of CW-Lite requirements will be considered in a future work.

The paper is articulated as follows. Firstly, in Section II, we provide a background to introduce the reader to the problems of PRIMA we are going to mitigate. The problems found, our proposed solution together with its attack model are presented in Section III. Related works are reported in Section IV. Secondly, we introduce in Section V a system model to compare our proposed solution to IMA and PRIMA, and we describe how to perform with that model the load-time and run-time integrity analyses. The implementation of Enhanced IMA and RA Verifier is illustrated in Section VI. Lastly, in Section VII, we evaluate the effectiveness of our solution and we conclude in Section VIII.

II. Background

A. Integrity Fundamentals

In the Biba model [4], the first relevant work in the integrity field, users are given with the right to access (security clearance) a certain amount of data (access class) according to two fundamental properties. Considering security clearances and data classes as numeric levels and the dominance as the $\leq$ comparison operator between levels, the *simple integrity property* states that the level of a subject must be dominated by that of the object the former wants to read; the **-property* requires, on the contrary, that the level of the subject dominates that of the object it is attempting to write.

These two properties guarantee that a malicious program can not corrupt critical system processes, if the level of the former is lower than that of the latter entities. Thus, to ensure that a target application behaves as expected, the system can be partitioned in two parts: the portion which the application depends on for the execution of its tasks (with high integrity level) and the rest of the system (with low integrity level). The first part, together with the system components required to enforce the chosen policy is defined as a Trusted Computing Base (TCB) in the Orange Book [12], a superseded specification of system evaluation criteria, whose definitions still apply.

B. Access Control

According to [13], a system can provide integrity (or confidentiality) with high assurance only if it is built around a security kernel, as the latter mediates every operation requested by active system entities (subjects) to passive entities (objects), and enforces a mandatory policy. While in the past mandatory enforcement was almost confined to military and government installations, the recent addition into the Linux kernel of the Linux Security Module (LSM) framework [14] favoured the diffusion of security kernels also in commodity operating systems. This framework consists of a set of security hooks, inserted in critical points of the kernel code (e.g. inside the code of a system call), which a Mandatory Access Control software can implement to allow or deny requested operations.

In this paper, we focus especially on SELinux [7], as it is very flexible (supports policy models based on user identity, roles, types and levels [15]) and because it is shipped with the so called *SELinux Example Policy*, to achieve comprehensive system security. In particular this policy, which is mainly based on the Type-Enforcement (TE) model [16], assigns a label to all system objects (e.g. processes, inodes) and contains rules that allow subjects to access objects depending on the types of those entities, the class of the object (e.g. regular file, directory) and the operation requested (e.g. read, write).

C. Integrity Analysis

Although SELinux is suitable for various security goals, like process confinement and isolation, it has been demonstrated [8] that the TE model is not appropriate for integrity.

While the satisfaction of the Biba model requirements can be directly verified from the policy rules (by comparing the levels of the subject and the object of each rule), this is undecidable for TE and access matrix models in general, as pointed out by Harrison et al. [17]. Indeed, since TE does not impose restrictions on the operations that a subject can do with respect to an object, a rule may have been granted for the proper functioning of an application regardless of the fact that it violates an integrity requirement.

In order to ensure whether a given policy is *safe* with respect to the desired security goals, a method commonly used is to define a set of *constraints*, i.e. the specifications of what a subject must not do to violate a goal, and to check whether policy rules violate or not specified constraints. If the previous condition is true, it is said that a violating rule generates a *conflict*. To check for conflicts, a number of tools (e.g. Gokyo [18]) represent a policy with a graphical model, which

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

highlights the flows of information among subjects, and detect if flows are licit or not.

More formally, given an information flow defined as:

Information flow: *given an object* o*, there is an information flow from the subject* s1 *to* s2 *iff* s1 *is able to write* o *and* s2 *can read* o.

To satisfy the Biba model, a policy must not allow information flows from low integrity subjects to high integrity ones.

D. IMA and PRIMA

IMA [3] is a Linux kernel software that records in a list (stored in the kernel memory) measurements upon the events (binary/library execution, file access, kernel module loading) specified in a policy. When a measurement is added to the list, a register of the TPM accumulates the digest of captured event data to protect their integrity. Performing the measurements just after the LSM hooks makes IMA suitable for reporting the same information used for making security decisions; this aspect is crucial for moving from policy enforcement to reporting operations performed by system entities.

According to IMA's authors, a verifier can evaluate two of the three data types that affect the integrity of an application: executable code and structured data (e.g. configuration files). Indeed, while such data can be identified with a reference database, unstructured data (the third type, e.g. temporary application data) do not have a predictable value. PRIMA solves this issue by preventing that dynamic data are modified by possibly malicious processes through the enforcement of a security policy meeting the requirements of the CW-Lite model. Supposing that a trusted process checks dynamic data at boot, before the use by applications, and that the result of the validation is given to verifiers, dynamic data do not need to be measured as they are necessarily of high integrity and, thus, will not affect applications behaviour.

III. Problem Statement

A. Motivations

The SELinux policy analysis [8] used as the basis for PRIMA consists in proposing an initial TCB of subjects, depending on their early appearance during the boot process, in deriving constraints from the Biba model and in resolving found conflicts semi-automatically with Gokyo. The authors believe that their approach, based on the use of access control spaces (permissible, precluded and unknown permissions), makes conflict resolution feasible for administrators. Indeed, once they find a resolution strategy for a group (subspace) of semantically similar permissions (from the intersection of the above sets), they can apply the chosen strategy to all permissions within the same group.

However, the main problems are that an administrator must have a deep knowledge of the system to identify subspaces and must find an appropriate way to resolve conflicts (excluding a subject or an object type, requiring the sanitization of program inputs or modifying the policy). In particular, an administrator must know, from the features he intends to support on a managed platform, which part of the policy can be excluded from the analysis[2] and must be able to determine if a program is really capable to sanitize its inputs, if he chose this strategy. To further complicate things, an administrator must face with privileged programs which usually have a broad set of permissions assigned and generate many conflicts.

Furthermore, another problem of PRIMA, derived from reducing the system portion to be measured, is that some useful information are not reported. In particular, the distinction between the three data types defined by IMA's authors cannot be done, as only the first two types are reported. Measuring also dynamic data would be preferable for the following reasons. Firstly, to determine from the digest if a file was malformed at the first access by a TCB subject (not known with PRIMA). Secondly, if the measurements list contains an unknown digest, a verifier may want to check if a measured file is unknown due to a previous write by another TCB subject (which may be doing something bad) or, instead, is a structured data whose digest is not present in the reference database (e.g. a customized configuration file or a SSH key).

B. Goal

The above issues motivated us to find an alternative solution easier to deploy and that, at the same time, provides comprehensive information (the latter is one of the five requirements identified by Coker et al. [19] for a remote attestation solution). If we can move the complexity of the integrity assessment from platforms administrators to verifiers, this would facilitate, in our view, the adoption of the remote attestation as a concrete solution to mitigate cyber-attacks. Given its extensibility by design and the very low effort needed for the deployment, we decided to base our work on IMA.

Our goal is to extend IMA to provide the measurements of the three data types affecting application behaviour (code, structured and unstructured data). With these measurements a verifier can, unlike with PRIMA, directly assess the load-time and run-time integrity of a target application and its dependencies. The main advantages of this approach are that: (1) platform administrators are not required (if the limitations listed in Section III-C are acceptable) to perform the expensive SELinux policy analysis; (2) the target of the analysis does not need to be pre-determined because all interactions through supported object types would be reported[3]; (3) a verifier can identify *exactly* the minimal TCB (whose code must be trusted) needed to support an application, since the information flow graph built from measurements would represent only interactions occurred and not all the possible ones.

C. Attack Model

Reporting processes interactions with IMA restricts the guarantees that could be inferred from the integrity analysis: the only cause, a verifier would be aware of, whereby an application can get compromised at run-time is the reading of a malformed datum previously written by a malicious

[2]Although SELinux labels are descriptive enough, finding the binding with a specific program configuration option is not always straightforward.

[3]Differently from PRIMA, we do not enforce integrity on a system portion.

process. Unlike PRIMA, where SELinux mediates accesses to every object, our solution does not report for performance and technical reasons (IMA deals only with inodes[4]) possible attacks through other Inter Process Communication (IPC) mechanisms, such as shared memory. Also, it cannot be used to detect information flows through covert channels[5] (e.g. creation/deletion of files).

Although this can be seen as a serious drawback, we believe that our approach is acceptable for the following reasons: (1) regular files represent with network sockets the majority of processes interfaces and thus, from the first source, a verifier could determine the integrity of a platform with a reasonable degree of confidence; (2) as a future work, we are planning to add support in IMA for other inode-related objects (e.g. fifos, pipes), thus reducing the gap with PRIMA; (3) whenever evaluating processes interactions through all channels is needed, PRIMA adopters could report, with our solution, information flows through supported channels and analyse only the portion of the SELinux policy not covered by IMA (about the 33% does not refer to inode-related objects); intuitively, doing the analysis on this portion would be simpler.

Nevertheless, to assert the validity of information reported by IMA, making the following assumptions is necessary. First, as other works that rely on a TPM, we exclude hardware-based attacks that could prevent a malicious action from being shown in a measurements list. Second, we give to verifiers the evidence that system events were properly recorded by Enhanced IMA from the measurements of the hardware and software components involved in the boot process. Third, we do not audit direct accesses to the disks and the memory; we will include such events when Enhanced IMA supports special inodes. Fourth, we assume that measurements are delivered correctly to verifiers with a proper remote attestation protocol. Lastly, we do not inspect dynamic data (possibly malformed) at boot time; as supposed by PRIMA, a trusted subject could check those data and write the result in the measurements list.

IV. Related Work

The problem of assessing the run-time integrity has been addressed in several ways. A class of solutions (PRIMA [6], DR@FT [20]) employs a Mandatory Access Control software to ensure information flow integrity. However, as said in [21], they do not consider how an application internally handles its inputs; a software may misuse assigned privileges if it operates on information at different security levels. Decentralized Information Flow Control (DIFC) solutions address this problem, allowing applications developers to specify flexible policies, but these solutions require in general modifications to the applications or the operating system [22].

Another class of solutions tries to solve the run-time integrity problem by running an application to be attested in an isolated environment, whose underlying mechanism can be attested e.g. through the Trusted Computing technology. For example, SecureBus [23] ensures strong isolation and flexible communication between processes at run-time in a way that is transparent for applications. Nexus [24] is a micro-kernel based architecture that uses a labelling mechanism to provide meaningful information about the run-time properties of an application to verifiers. Also, Gu et al. [25] proposed a mechanism to attest the correctness of the program execution through the separation of processes and generation of a dependency graph for the data required by the program to be evaluated. Finally, Haldar et al. [26] use a *trusted virtual machine* to derive high-level properties of programs in a platform-independent way.

The last category contains solutions that try to infer the run-time integrity of an application by collecting measurements of dynamic data. In particular, some solutions aim at evaluating the dynamic state of the Linux kernel, like LKIM [27] (further improved in [28] by employing a Copy-on-Write mechanism), and SBCFI [29]. Among the solutions at application level, we mention ReDAS [30], an architecture to attest two dynamic properties of applications (structural integrity and global data integrity) and DynIMA [31], which detects return-oriented programming attacks with a new module, the Process Integrity Manager (PIM), measured by IMA.

V. Design

The first step toward our goal of assessing the integrity of an application from measurements is to represent them in a graphical model. The idea is to follow the PRIMA verification procedure, i.e. building an information flow graph and detecting through that graph if a high integrity process is able to read a datum that can be written by an untrusted process (violation of the Biba model). Unlike PRIMA, our model is built from real interactions (although it supports also policy rules) and is suitable for the load-time integrity analysis.

Given that the current IMA version is not adequate for building a graph as accurately as PRIMA (we confirm this by evaluating with own model the impact of an unknown digest for both solutions), the second step consists in identifying how the format of IMA measurements should be extended to overcome this issue. The outcome will exceed our expectations; considering interactions through regular files, the information flow graph built with Enhanced IMA will be more accurate than that of PRIMA, as it will take into account the temporal and inode information (not available in a security policy).

A. Generic System Model

Our starting point to build a system model suitable for the integrity analyses is the definition of the reference monitor [13] as the component that mediates all accesses by active entities (subjects) to passive entities (objects). Since the integrity property of a secure system depends solely on the decisions that the reference monitor makes according to the configured security policy, we believe that our model would be appropriate for verifying integrity as long as it represents information provided during the remote attestation in terms of interactions between

[4]Data structure to represent a filesystem object (e.g. file, directory).
[5]System objects used in a manner that they are not originally intended.

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

subjects and objects. The integrity of a remote system would then be determined by analysing those interactions.

Thus, if we generically define the set of information provided by a platform for its assessment (IMA measurements and security policy) as *system activities*, and the tuple <subject, object, operation> as a *system activity*, we can model a system by representing in a graph the subject and the object of each system activity through two nodes[6] (a circle for a subject and a square for an object). These nodes will be connected with an edge directed from subjects to objects for write operations and vice-versa for read and execution actions.

However, while in an ideal model subjects and objects would coincide with real system entities (e.g. processes and inodes), as they are the ones which the reference monitor bases its decisions upon[7], in the real model built from system activities they can be just an approximation. Indeed, as shown in the middle of Figure 1, the subject and object of a policy rule both represent all entities having the same LSM label, while in a IMA measurement actually they can be seen respectively as the process that triggered that measurement and the snapshot of an inode at a given time (inode state). For the latter case, we remark that a snapshot does not unambiguously identify an inode; two snapshots may refer to two distinct inodes or to the same one measured at different times. In the graph, we represent inode states with the diamond shape.

The way system entities are represented from system activities has an impact on the analysis results. If the granularity of the representation is too coarse, we may get a false positive; for example, if two policy rules allow an information flow between two subjects, those subjects did not necessarily communicate between them, as they may have accessed two different inodes with the same label. On the contrary, if the granularity is too fine, we may detect a false negative; in this case, if we erroneously represent two inode states as distinct inodes when they refer to the same one, we are unable to detect an information flow from a writer that caused the state change and a process that read the inode while in the second state.

Additionally, handling system activities with different degrees of detail, depending on the source of information, makes it difficult to correlate them, as activities may represent different real entities. However, our model should be able to represent all system activities together because different sources of information may describe complementary aspects of a system; for example, measurements may describe the load-time integrity status while a policy the run-time aspect.

In order to address the above challenges, i.e. eliminating at least the possibility of false negatives and handling heterogeneous system activities, we build the system model in two steps: in the *inode states aggregation* step, we build an approximation of the ideal model from measurements; in the *label aggregation* step, we connect each entity with a known label to the node representing all the entities with that label.

In the first step, since two inode states do not necessarily belong to two distinct inodes, we can avoid false negatives by aggregating all unknown inode states into the same inode; this way, we can correctly handle the worst case where an information flow effectively occurred through a measured inode. In addition, if only reads are reported, the model contains an unknown writer (depicted in Figure 1 in grey) that could have possibly caused an inode state change. Lastly, in the second step, processes are connected to the group of entities having the same label, if this is reported in a measurement.

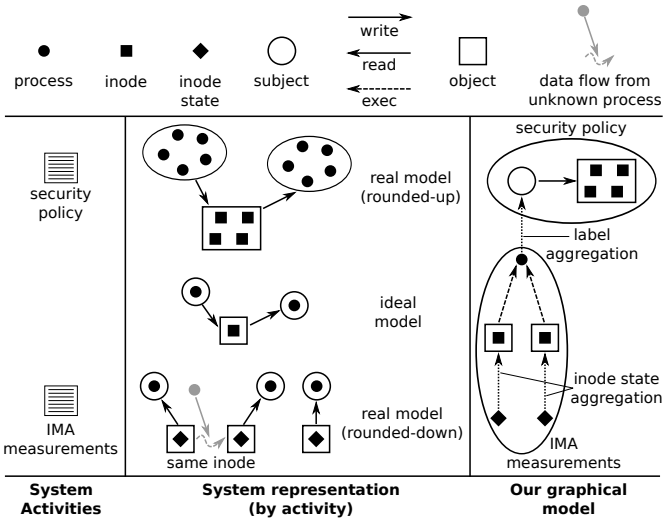

Fig. 1. Modelling system activities.

On the right side of Figure 1, we show a possible result of the graph building process where we represent together security policy rules and IMA measurements. From the bottom upwards, inode states are depicted with two distinct inodes[8], which were executed by a process. By retrieving from IMA measurements the LSM label of that process, we can connect the latter to the group of entities with that label; this group is in turn the subject of an operation allowed by a parsed security policy rule. The second aggregation is usually not displayed (measurements actions are associated directly to a LSM label) as a process identity is not reported as part of a measurement but inferred with other mechanisms (a process could have been created if a digest corresponds to an executable file).

In the following we evaluate with our model the accuracy of information provided by IMA, PRIMA and Enhanced IMA.

B. IMA-based System Model

With IMA, a verifier can perform the assessment of a platform by using, as system activities, only the measurements collected on that platform. The assessment consists in comparing measurement digests with values in a reference database.

In the scenario depicted in Figure 2, a sample platform provides an IMA policy (similar to the default IMA policy) indicating that the code executed and files read by all root processes have been measured. The platform also reports in a measurements list the digest of the `cat` binary, its required dependency (`libc.so`) and another file (`/tmp/bad`). To

[6]Subjects or objects present in multiple activities are drawn only once.

[7]This model would permit the most accurate analysis of a system.

[8]From the digest, it has been inferred that they are two binary files.

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

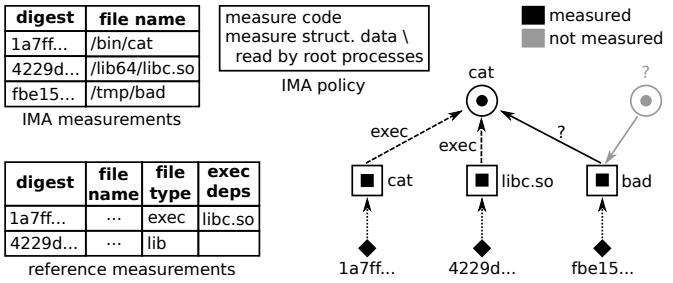

Fig. 2. IMA-based system model.

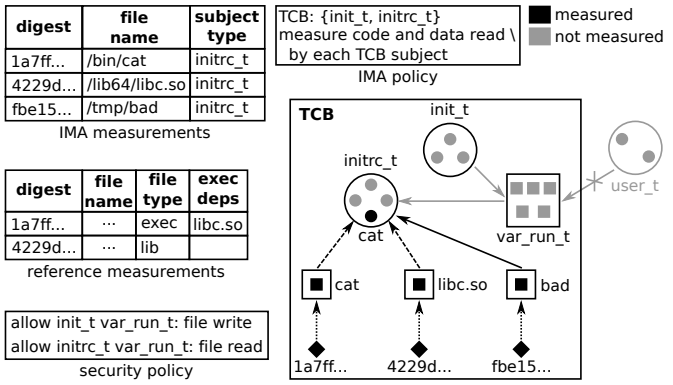

Fig. 3. PRIMA-based system model.

perform the analysis for this sample scenario, a verifier retrieves from a reference database the file type associated to each digest and, for executables, the list of their dependencies (shared libraries).

Since a measurement alone does not provide all the elements characterising a system activity (the subject and the operation are missing), a verifier can combine it with information extracted from a reference database. In particular, from a database it could be inferred that: (1) if the file type of a digest is `exec`, the subject could be a process created during the loading of that binary and the operation could be an execution; (2) if the file type is `lib`, a subject could be a process needing that dependency and the operation still an execution; (3) otherwise, the subject could be a process of the TCB (all root processes) and the operation a read. These assumptions do not imply that the above activities really occurred; for example, an executable could have been measured because it was simply read. However, we think that our model is appropriate for the analysis as we assumed that the worst case happened (only false positives are possible).

The proposed model could lead to a positive analysis result only if all digests in a measurements list are present in the reference database, because otherwise we would not be able to fully determine the operations presumably occurred on a platform. Furthermore, as stated in Section V-A, an unknown digest could mean that a TCB process reads from an inode maliciously modified by another process (with low integrity). Without the enforcement of a mandatory security policy, which isolates the TCB from the rest of the system, the presence of an unknown measurement is sufficient to consider a platform as compromised.

Indeed, in our example, `/tmp/bad` could be a modified version of libc created by a malicious process to corrupt all other system processes. Thus, even if the correct version of libc is present in the list, we cannot conclude that the `cat` process loaded the good version instead of `/tmp/bad` because the assumption we made to infer a complete a system activity is that a binary *requires* a shared library in order to be executed; however, this requirement could be satisfied also by `/tmp/bad` (if it exports the functions needed by `cat`).

As already mentioned by IMA's authors, and also confirmed by the analysis done by Cesena et al. [11] on the measurements obtained from a Fedora Linux distribution, the presence of unknown digests in the list is very likely either because of dynamic data (e.g. a temporary file used by an application to persistently store its state) but also because there may be customized configuration files, SSH keys and log files whose content is not predictable. This was the main motivation that led PRIMA's authors to propose an alternative solution.

C. PRIMA-based System Model

With respect to IMA, PRIMA provides more information which verifiers can use to perform the assessment of a platform. First, the IMA policy contains the list of TCB SELinux subjects for which the code and structured data were measured. Then, the IMA measurements list has an additional field to associate code executed and data read to the SELinux label of the process that performed a recorded operation. Finally, from a security policy[9] a verifier can build the information flow graph and determine whether the supplied TCB is integrity protected from the rest of the system.

Figure 3 depicts a sample system model built with PRIMA information. Here, the TCB is composed by two subjects (`init_t` and `initrc_t`) and the measurements list reports the same files as the previous example. Additionally, a simple security policy allows the two TCB subjects to interact through the `var_run_t` object: all other interactions must be considered denied.

It appears clear from the graph that PRIMA information sensibly increases the accuracy of the system representation. Since measurements now include the new subject field, execution and read actions can be associated to a precise entity even in the presence of unknown digests. The file type from a reference database could be still used to distinguish between actions, if they are evaluated differently. Furthermore, unlike IMA, PRIMA reports different aspects of a system: from measurements, it gives information about the load-time integrity of subjects (relevant events refer generically to the group of the processes with the same SELinux label); from a policy, it reports the system activities affecting the run-time integrity of TCB subjects.

The accuracy of the system representation guaranteed by PRIMA makes it very easy to determine the impact of an unknown digest. Indeed, since only the code and structured data accessed by TCB subjects are measured, `/tmp/bad` can be one of these two file types. Also, since this file was loaded

[9]The hash is included in the measurements list.

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

as part of the `initrc_t` subject, the load-time integrity of the whole TCB must be considered as low.

Despite there could be the possibility of unknown digests due for example to the presence of customized configuration files, the real advantage of PRIMA is that dynamic data (the root cause of unknown digests) accessed by TCB subjects do not need to be measured, as the security policy prevents processes outside the TCB from writing them.

However, the problem is indeed tuning the policy in a way that undesired modifications of dynamic data are not permitted. Although [8] provides an estimation, the composition of a TCB on a real platform is application-specific (applications may rely on different software) and configuration-specific (applications dependencies may vary depending on settings used). Thus, in most cases administrators have to perform the security analysis specifically for their platforms. In the next paragraph, we propose a remote attestation solution comparable with PRIMA, requiring a low effort for the deployment.

D. Enhanced IMA System Model

In order to match the accuracy of PRIMA, IMA needs to be modified in order to report the missing information previously identified: the LSM label of the process that triggered a measurement, to determine exactly the load-time integrity of each subject (group of processes with the same label); all accesses to dynamic data, to infer the run-time integrity of a subject from information flows by other subjects.

To perform the run-time integrity analysis, a better way to correlate reported operations is necessary; previously, in the model definition, we argued that digest values are inappropriate because, to avoid false negatives, all unknown inode states must be aggregated into the same inode. Instead, IMA could provide as part of a measurement also the LSM label of accessed inodes, so that a graph can be built from information similar to those provided by a policy (permissions allowed).

Although feasible, this choice does not ensure the best model accuracy. Indeed, IMA can record system activities at a granularity of inodes and, thus, can avoid the detection of false positives due to the use of a policy. Additionally, knowing the sequence of accesses to an inode further helps in reducing the number of interactions between processes: if, for example, an inode was read and then written after the first operation terminated, the processes accessing that inode did not communicate between them because no data was exchanged.

The approach chosen to build an optimized model was to add to each measurement a new field[10] (the index of the record reporting the previous inode write) that allows measurements referring to the same inode to be reliably identified. With this information, we are able not only to represent system objects with the same granularity of the ideal model in the middle of Figure 1 (one object for inode), but also to further increase the degree of detail by decomposing an inode into multiple objects, in order to exploit the temporal information (as described above). Indeed, referring to the previous example, the detection of an information flow cannot be prevented if the reader and the writer are connected to the same object.

[10]In addition to the LSM label of an inode for symmetry with the format of a policy rule.

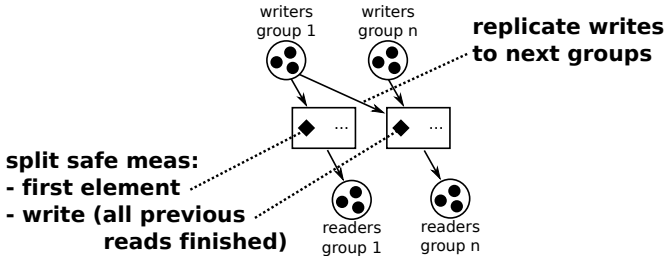

Fig. 4. Revised inode states aggregation step.

However, determining how the states of an inode could be aggregated into different objects cannot be simply done by retrieving the action from a measurement (each time a read is followed by a write, the next inode states would be assigned to a new object). Indeed, a measurement indicates only when an operation started, while the time the latter ended is unknown. Only if we are sure that a read ended before a write begun, we can connect the processes accessing the same inode to different objects; under some circumstances, this information can be inferred from IMA violations.

Violations allow integrity verifiers to detect (from measurements with a recognizable digest value) that the fingerprint of an inode in the next measurement does not reflect the content accessed by a process, because that inode was accessed concurrently by other processes: the *Time-of-Measurement Time-of-Use (ToMToU)* violation is triggered when a writer accesses a measured inode while it is still being used by readers; the *open_writers* violation when a reader accesses an inode to be measured while it is still opened by writers. Although we are not able to determine, when an inode is opened by several processes, which data a reader obtained from a writer, we can safely conclude that a read is unrelated to the next write if the latter operation did not trigger a ToMToU violation (the number of readers at that time was zero).

The above statement clarifies how an inode can be decomposed into different objects without introducing the possibility of false negatives. The revised inode states aggregation step for building a model is depicted in Figure 4. Considering an ordered sequence of measurements of the same inode, the first measurement (we call it as *split safe measurement*) of each group of inode states can be the first inode measurement or a measurement reporting a write that does not follow a ToMToU violation. Furthermore, since readers obtain data modified by past writers, to correctly detect information flows write edges directed to a group are replicated for all the next groups.

After defining the new format of measurements produced by Enhanced IMA and how to process those data, we show the degree of accuracy that can be obtained with our model in the scenario depicted in Figure 5. Similarly to the previous examples, the IMA measurements list reports, other than the execution of `cat` and its dependency `libc.so` by the subject type `initrc_t`, the file `/tmp/bad` accessed three times by

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

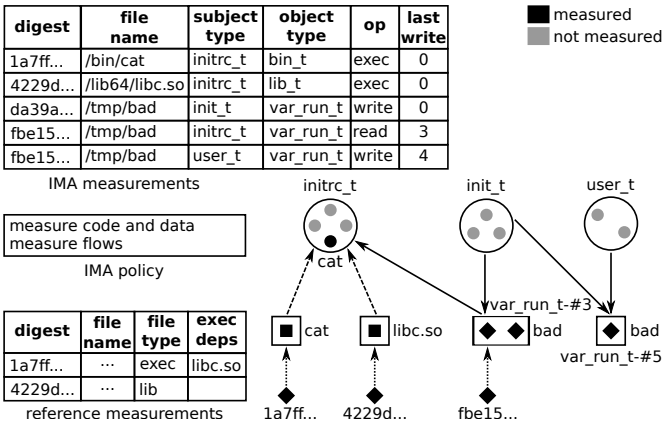

Fig. 5. Enhanced IMA-based system model.

the subject types `init_t`, `initrc_t` and `user_t`. With respect to PRIMA, the IMA policy does not contain a predetermined set of TCB subjects and the security policy is not provided (in this example, we assume that measuring only regular files is acceptable for a verifier).

Even if `initrc_t` reads an unknown content of `/tmp/bad` during the second access, we can determine differently from IMA if `cat` is of high integrity or not[11] by analysing the processes that wrote this file. By using the temporal information we found that the only writer that communicated with `initrc_t` was `init_t`, as the last write by `user_t` did not cause a ToMToU violation. Thus, if we add the subject type `init_t` to the TCB and we identify the content of `/tmp/bad` at the time of the first measurement (from the digest it can be inferred that this file was empty), we can conclude that the integrity of `initrc_t` is high.

Regarding the integrity analysis, there are two main differences with PRIMA. First, if the processes interactions of the previous scenario are inferred from a security policy, also the `user_t` subject type should be added to the TCB because the temporal information is not given. Second, with our approach, the integrity analysis is more flexible. With PRIMA, the detection of a low integrity subject always leads to a negative analysis result because allowed rules have to be considered as actions effectively occurred. With Enhanced IMA, verifiers can truly determine if the same subject generated Biba violations and, if not, can exclude it from the TCB.

In Section VII-B, we motivate our choice of reporting information flows by showing the size of the `init_t` TCBs obtained from the SELinux policy of Fedora 19 and from measurements collected with Enhanced IMA.

E. Integrity Analyses

The ultimate purpose of the model previously defined is to determine if a represented system is of sufficient integrity to perform the tasks it is expected to do. Once a verifier identifies the portion of the system (the target application and its TCB) required to perform the desired tasks, the integrity assessment of a platform consists, similarly to PRIMA[12], in: checking

[11]We assume that the digests of the main executable and libc are known.

[12]PRIMA evaluates whether the requirements of CW-Lite are satisfied.

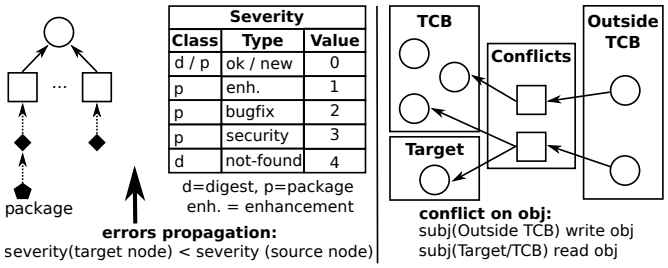

Fig. 6. Load-time and run-time integrity analyses.

with the load-time analysis that the integrity of the processes responsible for those tasks was high at the time of execution; verifying with the run-time analysis that the integrity of such processes did not become low due to an information flow from an untrusted process (Biba requirement). The verification of CW-Lite requirements and the initial state of dynamic data is out of the scope of this paper.

To perform both the integrity analyses, we mainly reuse existing methods that deal with graphs. In Figure 6, we illustrate the analysis process: on the left side, we perform the load-time analysis by using the method developed by Cesena et al. [11], which consists in propagating errors found on packages (not up to date) or digests (value not recognized) to subjects; for the run-time analysis, similarly to [8], we group subjects in three sets (`TCB`, `Target` of evaluation, `Outside TCB`) and we draw both the operations causing a Biba violation and the objects involved (`Conflicts` set).

VI. Implementation

a) Enhanced IMA: One part of our implementation work consisted in modifying IMA to correctly report interactions between processes, as this software was not designed for such objective.

We first introduced a flexible mechanism for defining the format of measurements. This mechanism makes it very easy supporting a new data type and including it in a custom measurement template format. Its code has been recently merged in the mainline Linux kernel since version 3.13. Consequently, we defined the new fields to create the custom template format shown in Figure 5[13]: the LSM label of the process and the inode measured, the type of operation performed on the latter and a list index to correlate measurements of the same inode.

Furthermore, we developed the following IMA extensions for properly detecting all information flows: a new IMA hook, called `ima_bprm_committing_creds`, to capture file descriptors inherited by a child process from its parent[14]; the new violation types `flow_ToMToU` and `flow_open_writers` to properly handle concurrent accesses to inodes opened for both reading and writing; a missing call to the LSM hook `security_bprm_check`, added to the ELF interpreter.

We also defined new IMA policy actions (`measure_all` and `measure_flows`) for capturing respectively the loading

[13]For sake of simplicity, we omitted a new field displaying the future credentials of a process after the `execve` system call.

[14]The child is able to access files opened by the parent without executing the `open` system call.

This is the author's version of an article published in the TRUSTCOM'14 proceedings. Changes were made to this version by the publisher prior to publication.
The published version of this paper is available at http://dx.doi.org/10.1109/TrustCom.2014.63

TABLE I
SYSTEM PERFORMANCES.

	system config (policy / template)			
	none ima-ng	default ima-ng	PRIMA ima-ng+subj	flows custom
# meas.	1	1350	1216	4604
meas. file size (KB)	0.166	187	211	1138
memory usage (KB)				
- kmalloc	21027	21273	21304	23098
- iint cache	0	102	101	265
- total	21027	21375	21405	23363
boot time (secs)				
- no TPM	15.71	18.73	19.38	19.34
- with TPM	15.64	61.56	50.69	180.79

of code and structured data for each subject type (otherwise only one measurement is taken for efficiency reasons) and information flows through dynamic data. In addition, to avoid that IMA does not run out the kernel space memory, we developed a new IMA interface that removes measurements from the kernel list (thus, freeing allocated resources) after they are transmitted to a requesting user-space tool.

b) RA Verifier: The remaining part of the work consisted in the implementation of our verification tool. Basically, it parses measurements, queries a reference database to verify digests contained in measurement entries, builds a graph with the NetworkX library (http://networkx.github.io/), according to the procedure described in Section V-A, and finally performs the integrity analyses described in Section V-E.

VII. EVALUATION

A. Performances

In order to demonstrate the goodness of our approach, we measured the performances of our prototypes in a testing environment composed by a TCG-enabled platform (a HP 6730b notebook), the target of the integrity evaluation, and a conventional workstation (an assembled PC with an Intel i7-4770 CPU and 8 GB of RAM) acting as a verifier. Both platforms run the Fedora 19 Linux distribution, the former with Enhanced IMA and the latter with RA Verifier.

During the first test, we quantified the overhead derived from producing frequent and large measurements. In particular, we analysed four different configurations: Enhanced IMA running without a policy and with the three IMA policies illustrated in Figures 2, 3 and 5 (also including kernel modules). We installed Fedora on the target platform by selecting the minimal software configuration. Table I shows the memory occupied by kernel objects created with `kmalloc` and the IMA cache (`iint`), and the boot duration either with the TPM disabled (for pure system performance) and enabled.

As expected, we see an almost linear increase of the memory occupied with the number of measurements. The exception between the second and the third case is due to the different measurement templates. The number of PRIMA measurements could be further reduced by recording only the actions performed by TCB subjects; however, the TCB proposed in [8] cannot be used, as it refers to an old SELinux policy. Boot time values obtained with the TPM disabled are good; differences are negligible when Enhanced IMA is initialized with a policy. Instead, the boot time obtained with the TPM enabled reveals that, to have an usable system, the number of measurements should be lowered. To overcome this issue, we could synchronously record with the TPM system critical events (code and kernel modules loading) and, asynchronously, other operations.

TABLE II
ANALYSIS STATISTICS.

	Minimal	Mediawiki	Tomcat
meas. stats (unknown)			
- code + kernel mods	258 (1)	356 (0)	327 (1)
- struct. data	2399 (187)	3214 (270)	2892 (225)
- unstruct. data	1947	2413	2201
- total (no boot_aggregate)	4604	5983	5420
RA Verifier perf. (secs)			
- parse meas.	0.06434	0.08109	0.05651
- build graph	0.28544	0.36624	0.31200
- query db	1.56593	2.38566	1.92309
- load-time analysis	0.30594	0.34446	0.39180
- run-time analysis	0.04230	0.06285	0.05643
- total	2.26395	3.24030	2.73983

In Table II, we report the statistics and performances obtained with RA Verifier for three different software configurations of the platform to be attested: a minimal installation, an Apache web server with Mediawiki, and Tomcat. Statistics show that, while all the code is recognized (except for our test script), many digests of structured data (SSH keys, customized configuration files) are unknown. Furthermore, the numbers for unstructured data demonstrate that the effort to manage them is almost the same, compared to the other two data types. Regarding the performances, the most important result is that except for the *query db* step, both the load-time and run-time analyses are done very fast.

With the last test, we demonstrate the scalability of our approach by showing the benefits of the new IMA flushing interface. During the experiment with Tomcat, for three times we observed the memory usage, collected measurements through the new interface and recorded the number of measurements obtained and the size of the file containing those measurements. Table III reports that the memory usage decreased with a noticeable variation after the first flush (observed at 2nd attest), with a lower variation after the second iteration (seen at 3rd attest). The table also shows that the number of new measurements did not grow rapidly during the experiment.

B. Discussion

The main advantage of Enhanced IMA is that it simplifies the integrity analysis due to the low number of interactions represented in the information flow graph. We quantified the improvement with respect to PRIMA by comparing the size of the TCB for the *init* process (one of the most critical part of a system) with both measurements and the Fedora 19 policy. We found that, in the Mediawiki scenario, the measurements and the policy TCBs contain respectively 8 and 38 subjects (among 40 subjects recorded in the measurements list), confirming our expectation that in practice the real number of an application

TABLE III
SCALABILITY STATISTICS.

	1st attest	2nd attest	3rd attest
cur memory usage (KB)	24428	23847 (-581)	23719 (-128)
# flushed measurements	5421	52	49
file size (KB)	1339	14	13

dependencies is lower than that derived from a policy (that must support all the possible configurations).

However, as said in the introduction, the main concern is managing large amounts of data to be sent to verifiers. Although nowadays broadband connections are common and transferring files of 1.4 MB (TCG integrity reports are a bit larger due to the use of the XML format) should not be a problem, our solution becomes scalable by adopting the attestation delegation proposal described in [19]: an attestation proxy trusted by a target machine and verifiers could be adopted so that it maintains a fresh list of measurements and determines locally if the TCB supplied by a verifier is of high integrity or not. OpenAttestation v1.7 (https://01.org/openattestation/), which we extended to support IMA and periodic attestations, is a perfect candidate for taking the role of a proxy.

Lastly, currently Enhanced IMA does not permit to make deep integrity assessments, at least until all inode types will be supported. Nonetheless, we foresee an interesting application of our solution: it may be used as an intrusion detection software to detect anomalies in a monitored system. If a verifier previously found a Biba compliant TCB, the occurrence of a new conflict could mean either that a TCB subject may have legitimately executed an action not previously captured, or may reveal the intrusion of an attacker trying to exploit a vulnerability in the target machine software.

VIII. Conclusions

In this paper, we have reported the results of our experiment of capturing with IMA all the events relevant for determining an application behaviour. We have shown that our attempt is worthwhile as, although the guarantees that can be inferred from the reported information are lower than those provided by PRIMA, the ease of deployment of Enhanced IMA makes it attractive for those that want to use the remote attestation to detect cyber-attacks but are not skilled enough to perform the SELinux policy analysis. As a future work, we are planning to capture events for all inode types and to verify on reported information the requirements of the CW-Lite model. We will also investigate if we can overcome the IMA limitation of measuring only inodes by implementing the LSM hooks for other IPC communications. By achieving these goals, Enhanced IMA would report processes interactions through reads and writes with the same completeness of PRIMA.

Acknowledgement

The research described in this paper is part of the SECURED project, co-funded by the European Commission under the ICT theme of FP7 (grant agreement no. 611458).

References

[1] European Commission, "Cybersecurity Strategy of the European Union: An Open, Safe and Secure Cyberspace," http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=1667.

[2] Trusted Computing Group, "TPM Main Specification, Version 1.2, Revision 103," https://www.trustedcomputinggroup.org/, 2007.

[3] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proc. SSYM'04*, 2004, pp. 223–238.

[4] K. J. Biba, "Integrity considerations for secure computer systems," MITRE Corp., Tech. Rep., 1977.

[5] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *IEEE Symposium on Security and Privacy*, 1987, pp. 184–195.

[6] T. Jaeger, R. Sailer, and U. Shankar, "PRIMA: Policy-Reduced integrity measurement architecture," in *Proc. SACMAT'06*, 2006, pp. 19–28.

[7] National Security Agency, "Security-Enhanced Linux SELinux," http://www.nsa.gov/research/selinux/index.shtml, 2009.

[8] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the SELinux example policy," in *Proc. 11th USENIX Security Symposium*, 2003, pp. 59–74.

[9] K. Yee, "Aligning security and usability," *Security Privacy, IEEE*, vol. 2, no. 5, pp. 48–55, 2004.

[10] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor, "Subdomain: Parsimonious server security," in *Proc. LISA'00*, 2000, pp. 355–368.

[11] E. Cesena, G. Ramunno, R. Sassu, D. Vernizzi, and A. Lioy, "On scalability of remote attestation," in *Proc. STC'11*, 2011, pp. 25–30.

[12] "Trusted Computer System Evaluation Criteria (Orange Book)," United States Department of Defense, Tech. Rep., 1985.

[13] M. D. Abrams, S. G. Jajodia, and H. J. Podell, Eds., *Information Security: An Integrated Collection of Essays*. IEEE Computer Society Press, 1995.

[14] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General security support for the Linux kernel," in *Proc. 11th USENIX Security Symposium*, 2002, pp. 17–31.

[15] S. Smalley, "Configuring the SELinux Policy," http://www.nsa.gov/research/_files/selinux/papers/policy2-abs.shtml, NSA, Tech. Rep., 2005.

[16] W. E. Boebert and R. Y. Kain, "A practical alternative to hierarchical integrity policies," in *Proc. 8th National Computer Security Conference*, 1985, pp. 18–27.

[17] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[18] T. Jaeger, X. Zhang, and A. Edwards, "Policy management using access control spaces," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 327–364, 2003.

[19] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *Int. J. Inf. Secur.*, vol. 10, no. 2, pp. 63–81, 2011.

[20] W. Xu, G.-J. Ahn, H. Hu, X. Zhang, and J.-P. Seifert, "Dr@ft: Efficient remote attestation framework for dynamic systems," in *Proc. ESORICS'10*, 2010, pp. 182–198.

[21] B. Hicks, S. Rueda, T. Jaeger, and P. McDaniel, "From trusted to secure: building and executing applications that enforce system security," in *Proc. ATC'07*, 2007, pp. 205–218.

[22] W.-K. Sze and R. Sekar, "A portable user-level approach for system-wide integrity protection," in *Proc. ACSAC'13*, 2013, pp. 219–228.

[23] X. Zhang, M. J. Covington, S. Chen, and R. Sandhu, "SecureBus: towards application-transparent trusted computing with mandatory access control," in *Proc. ASIACCS'07*, 2007, pp. 117–126.

[24] A. Shieh, D. Williams, E. G. Sirer, and F. B. Schneider, "Nexus: a new operating system for trustworthy computing," in *Proc. SOSP'05*, 2005, pp. 1–9.

[25] L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei, "Remote attestation on program execution," in *Proc. STC'08*, 2008, pp. 11–20.

[26] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation: a virtual machine directed approach to Trusted Computing," in *Proc. VM'04*, vol. 3, 2004, pp. 29–41.

[27] P. A. Loscocco, P. W. Wilson, J. A. Pendergrass, and C. D. McDonell, "Linux kernel integrity measurement using contextual inspection," in *Proc. STC'07*, 2007, pp. 21–29.

[28] M. Thober, J. A. Pendergrass, and C. D. McDonell, "Improving coherency of runtime integrity measurement," in *Proc. STC'08*, 2008, pp. 51–60.

[29] N. L. Petroni, Jr. and M. Hicks, "Automated detection of persistent kernel control-flow attacks," in *Proc. CCS'07*, 2007, pp. 103–115.

[30] C. Kil, E. Sezer, A. Azab, P. Ning, and X. Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence," in *Proc. DSN'09*, 2009, pp. 115–124.

[31] L. Davi, A.-R. Sadeghi, and M. Winandy, "Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks," in *Proc. STC'09*, 2009, pp. 49–54.