Remote Attestation with Domain-based Integrity Model and Policy Analysis

Wenjuan Xu, Xinwen Zhang, *Member, IEEE,* Hongxin Hu, *Student Member, IEEE,* Gail-Joon Ahn, *Senior Member, IEEE,* and Jean-Pierre Seifert, *Member, IEEE*

Abstract—We propose and implement an innovative remote attestation framework called DR@FT for efficiently measuring a target system based on an information flow-based integrity model. With this model, the high integrity processes of a system are first measured and verified, and these processes are then protected from accesses initiated by low integrity processes. Towards dynamic systems with frequently changed system states, our framework verifies the latest state changes of a target system instead of considering the entire system information. Our attestation evaluation adopts a graph-based method to represent integrity violations, and the graph-based policy analysis is further augmented with a ranked violation graph to support high semantic reasoning of attestation results. As a result, DR@FT provides efficient and effective attestation of a system's integrity status, and offers intuitive reasoning of attestation results for security administrators. Our experimental results demonstrate the feasibility and practicality of DR@FT.

Index Terms—remote attestation, platform integrity, security policy, policy analysis.

1 Introduction

In distributed computing environments, it is crucial to measure whether a remote party runs buggy, malicious application codes or is improperly configured. Remote attestation techniques have been proposed for this purpose by analyzing the integrity of a remote system to determine its trustworthiness. Typical attestation mechanisms are designed based on the following steps. First, an attestation requester (*attester*) sends a challenge to a target system (*attestee*), which responds with the evidence of the integrity of its hardware and software components. Second, the attester derives runtime properties of the attestee and determines the trustworthiness of the attestee. Finally and optionally, the attester returns an attestation result, such as integrity status, to the attestee.

Various attestation approaches and techniques have been proposed. Trusted Computing Group (TCG) [1] introduces trusted platform module (TPM) which enables integrity measurements of a remote system. In addition, integrity measurement mechanisms facilitating the capabilities of TPM at application level have been proposed. For instance, Integrity Measurement Architecture (IMA) [2] is an implementation of TCG approach to provide verifiable evidence with respect to the current runtime state of a measured system. Moreover, several attestation methods have been introduced to address privacy properties [3], system behaviors [4], and information flow model [5]. However, existing approaches have several limitations that obstruct their wide deployment in real systems. First of all, TCG-based mechanisms is inefficient, since IMA need to verify all software components of a running system. Furthermore, without a build-in runtime integrity model, an attestation verification procedure cannot evaluate the integrity status with respect to the trust level of the system. The problems become worse when the attestation target is a dynamic system with continuously changing states due to some system-centric events, such as security policy updates and software package installations. Last but not least, existing attestation mechanisms can not provide an effective and intuitive way to represent attestation results and reflect such results in integrity violation resolution.

Towards a systematic attestation solution, we propose an efficient remote attestation framework, called dynamic remote attestation framework and tactics (DR@FT), to address aforementioned issues. Our framework is based on a *domain-based integrity model* to describe the integrity status of a system with information flow control. With this property, the high integrity processes of a system are first measured and verified, and these processes are then protected from accesses initiated by low integrity processes during runtime. In other words, the protection of high integrity process is verified by analyzing security policies and ensuring that the policies are correctly enforced. Having this principle in place, DR@FT enables us to verify whether certain applications (domains) in the attestee satisfy integrity requirements without verifying all components of the system. To accommodate the dynamic nature of a system, DR@FT only verifies the latest changes in a system state, instead of considering the entire system information for each attestation inquiry. Through these two tactics, our framework is able to to achieve an efficient attestation to the target system. Also, DR@FT adopts a

- *W. Xu is with Frostburg State University, Frostburg, MD, USA. E-mail: wxu@frostburg.edu*
- *X. Zhang is with Huawei Research Center, Santa Clara, CA, USA. E-mail: xinwen.zhang@huawei.com*
- *H. Hu and G-J. Ahn are with Arizona State University, Tempe, AZ, USA. E-mail: {hxhu, gahn}@asu.edu*
- *J-P. Seifert is with Deutsche Telekom Laboratories and Technical University of Berlin. Email: jean-pierre.seifert@telekom.de*

graph-based information flow analysis mechanism to examine security policy violations based on our integrity model, which helps cognitively identify suspicious information flows in the attestee. To further improve the efficiency of security violation resolution, we propose a ranking scheme for prioritizing policy violations, which provides a method for describing the *trustworthiness* of different system states with *risk levels*.

Overall, the contribution of this paper is three-fold:

- First, we propose a domain-based integrity model to describe the integrity and thus trustworthiness of a general computing system. Built on information flow control, the model is leveraged to identify the trusted computing base (TCB) of a system (system TCB) and trusted subjects and objects of an application domain (domain TCB). For integrity protection purpose, we define information flow control rules to protect the integrity of system TCB and domain TCBs. Our integrity model and information flow control rules serve as a theoretical foundation for attesting the trustworthiness of a system.
- We then introduce DR@FT along with its architecture, protocols and algorithms for measuring TCB subjects and objects, reporting measurements to attester, and evaluating integrity status of the attestee based on our proposed integrity model. We present the tactics to achieve efficient attestation processing with the measurements of TCB subjects at attestee side and accelerated verification processing based on policy update at attester side.
- To assist integrity verification, we also develop a graph-based policy analysis tool, which automatically checks possible integrity flow violations based on our integrity model. Based on a set of information flow control rules, our graph-based analysis tool further provides intuitiveness on how to resolve violations, which can be sent as the attestation result to the attestee. To measure the risks of variant information flow violations, we further propose a ranked policy violation graph, which can be used to guide the system administrator for effectively resolving identified violations.

This paper is organized as follows. Section 2 briefly overviews existing system integrity models, policy analysis, and attestation mechanisms. Section 3 describes our domain-based integrity model which provides a theoretical foundation of DR@FT. Section 4 presents design requirements and attestation procedures of DR@FT. We then illustrate our graph-based policy analysis tool in Section 5, followed by an introduction of our ranked policy violation graph and its application. We elaborate the implementation details and evaluation results in Section 7. Section 8 concludes this paper.

2 Background and Related Work

2.1 Integrity Models

To describe and maintain the integrity status of a system, there exist various information flow-based integrity models such as Biba [6], LOMAC [7], Clark-Wilson [8], and CW-Lite [9]. Biba integrity property is fulfilled if a high integrity process cannot read/execute a lower integrity object, nor obtain lower integrity data in any other manner. LOMAC allows high integrity processes to read lower integrity data, while downgrades the process's integrity level to the lowest integrity level that has ever been activated. Clark-Wilson provides a different view of dependencies, which states information flow from low integrity objects to high integrity objects through a specific program called transaction procedures (TP). Later, the concept of TP is evolved as a filter in the CW-Lite model. The filter can be a firewall, an authentication process, or a program interface for downgrading or upgrading the privileges of a process.

With existing integrity models, there is a gap between concrete measurements of a system's components and the verification of its integrity status. We believe an application-oriented and domain-centric approach accommodates the requirements of attestation evaluation better than directly adopoting an abstract model. For example, in a Linux system, a subject in traditional integrity models corresponds to a set of processes, which belong to a single application domain. A more concrete instance is that an Apache domain may include various process types such as `httpd_t`, `http_sysadm_devpts_t`, and `httpd_prewikka_script_t`, and all of these types may have information flows among them, which should be regarded as a single integrity level. Also, sensitive objects in a domain should share the same integrity protection of its subjects. In this paper, we propose a domain-based isolation model to capture the essence of application- or domain-centric integrity requirements.

2.2 Policy Analysis

Jaeger et al. [10] and Shankar et al. [11] use a tool called Gokyo for checking the integrity of a proposed TCB for SELinux [12]. Also, they attempt to implement their ideas in an automatic manner. Gokyo mainly identifies a common TCB in SELinux but a typical system may have multiple applications and services with variant trust relationships. Achieving the integrity assurance for these applications and services was not addressed in Gokyo. Policy analysis on SELinux is further proposed in [13] towards MLS security properties.

Several query-based policy analysis tools have been developed. APOL [14] is a tool developed by Tresys Technology to analyze SELinux policies by computing an information flow relation and the transitive closure of the relation. SLAT (Security Enhanced Linux Analysis Tool) [15] defines an information flow model and policies are analyzed based on this model. PAL (Policy Analysis using Logic Programming) [16] uses SLAT information flow model to implement a framework for analyzing SELinux

policies. All these tools try to provide a way for querying policies. However, they only display policies and policy query results in a text-based fashion, which is difficult to understand for policy developers or security administrators.

To overcome these limitations, we have developed a graph-based methodology for identifying and expressing interested information flows in SELinux policies [17]. We have also proposed a policy analysis mechanism using Petri Nets presented in [18]. However, these tools and mechanisms are limited in identifying system TCB and domain TCBs, but do not have the capability of identifying and ranking integrity violations thus providing evidential support for attestation results.

2.3 Remote Attestation

TCG specifications [1] define mechanisms for a TPM-enabled platform to report its current hardware and software configuration status to a remote challenger. A TCG attestation process is composed of two steps: (i) an attestee platform measures hardware and software components starting from BIOS block and generates a hash value. The hash value is then stored into a TPM Platform Configuration Register (PCR). Recursively, it measures BIOS loader and operating system (OS) in the same way and stores them into TPM PCRs; (ii) an attester obtains the attestee's digital certificate with an attestation identity key (AIK), AIK-signed PCR values, and a measurement log file from the attestee which is used to reconstruct the attestee platform configuration, and verifies whether this configuration is acceptable. From these steps we notice that TCG measurement process is composed of a set of sequential steps including the bootstrap loader. Thus, TCG does not provide effective mechanisms for measuring a system's integrity beyond the system boot, especially considering the randomness of executable contents loaded by a running OS.

IBM IMA [2] extends TCG's measurement scope to application level. A measurement list *M* is stored in OS kernel and composed of m_0 ... m_i corresponding to loaded executable application codes. For each loaded m_i, an aggregated hash H_i is generated and loaded into TPM PCR, where H_0=H(m_0) and H_i=H($H_{i-1}||H(m_i)$). Upon receiving the measurements and TPM-signed hash value, the attester proves the authentication of measurements by verifying the hash value, which helps determine the integrity level of the platform. However, IMA requires to verify the entire components of the attestee platform while the attestee may only demand the verification of certain applications. Also, the integrity status of a system is validated by testing each measurement entry independently, and it is impractical to disregard newly installed untrusted applications or data from the untrusted network.

PRIMA [5] is an attestation mechanism based on IMA and CW-Lite integrity model [9], and attempts to improve the efficiency of attestation by verifying only codes, data, and information flows related to trusted subjects. On one hand, PRIMA needs to be extended to capture the dynamic nature of system states such as software and policy updates, which could be an obstacle for maintaining the efficiency. On the other hand, PRIMA only represents an attestation result with binary decisions (either trust or distrust) and does not give semantic information about how much the attestee platform can be trusted.

Property-based attestation [3] is an effort to protect the privacy of a platform by collectively mapping related system configurations to attestation properties. For example, "SELinux-enabled" is a property which can be mapped to a system configuration indicating that the system is protected with an SELinux policy. That is, this approach prevents the configurations of a platform from being disclosed to a challenger. However, due to the immense configurations of the hardware and software of the platform, mapping all system configurations to properties is infeasible and impractical.

Semantic remote attestation [4] is proposed with a trusted virtual machine running on a platform. Through monitoring the policy attached to a running software, its behavior is verified. However, this approach does not define the correct behavior of a security policy with respect to integrity, which leads the attestation to be intractable. Behavior-based attestation [19] attempts to attest system behaviors based on an application level policy model. Such a model-based approach may not comprehensively realize the complex behaviors of dynamic systems.

3 Domain-based Integrity Model

According to TCG and IMA, the trustworthiness of a system is described with the measured integrity values (hash values) of loaded software components. However, the measurement efficiency and attestation effectiveness are major problems of these approaches since (i) too many components have to be measured and tracked, (ii) too many known-good hash values are required from different software vendors or authorities, and (iii) runtime integrity cannot be guaranteed and verified. Fundamentally, in order to trust a single application of a system, every software component in the system has to be trusted; otherwise the attestation result should be negative. In our work, we believe that the trustworthiness of a system is tightly related to the integrity status, which is, in turn, described by a set of integrity rules that are enforced by the system. If any of the rules is violated, it should be detected. Hence, so as to trust a single application domain, we just need to ensure the TCB – including reference monitor and integrity rules protecting the target application domain – is not altered.

Based on this anatomy, we introduce domain-based isolation principles for integrity protection, which are the criteria to describe the integrity status of a system as well as its trustworthiness. We first propose general methodologies to identify high integrity processes, which include *system TCB* and *domain TCB*. We then specify security rules for protecting these high integrity processes.

3.1 System TCB

The concept of system TCB (TCB_s) is similar to that of traditional TCB [20], which can be identified along

with the subjects functioning as the reference monitor of a system [21]. Applying this concept to SELinux [22], for example, subjects functioning as the reference monitor such as `checkpolicy` and `loading policy` belong to system TCB. Also, subjects used to support the reference monitor such as `kernel` and `initial` should also be included into system TCB. With this approach, an initial TCB_s can be identified, and other subjects such as `lvm` and `restorecon` can be added into TCB_s based on their relationships with the initial TCB_s. Other optional methods for identifying TCB_s are proposed in [10]. Considering the similarity of operating systems and configurations, we expect that the results would be similar. Furthermore, for the attestation purpose, TCB_s also includes integrity measurement and reporting components, such as kernel level integrity measurement agent [23] and attestation request handling agent.

3.2 Domain TCB

In practice, other than TCB_s, an application or user-space service can also affect the integrity and the behavior of a system. An existing argument [20] clearly states the situation: "A network server process under a UNIX-like operating system might fall victim to a security breach and compromise an important part of the system's security, yet is not part of the operating system's TCB." Accordingly, a comprehensive mechanism of policy analysis for TCB identification and integrity violation detection is desired. Hence, we introduce a concept called information domain TCB (or simply *domain TCB,* TCB_d). Let *d* be an information domain functioning as a certain application or service through a set of related subjects and objects. A domain *d*'s TCB or TCB_d is composed of a set of subjects and objects in information domain *d* which have the same level of integrity. By the same level of integrity, we mean that, if information can flow to some subjects or objects of a domain, it can flow to all others in the same domain with legitimate operations of the application. That is, they need the same level of integrity protection. Prior to the identification of TCB_d, we first identify the information domain *d* based on its main functions and relevant information flow associated with these functions. For example, a running web server domain consists of many subjects–such as `httpd` process, plugins, and tools, and other objects–such as data files, config files, and logs. [1]

The integrity of an object is determined by the integrity of subjects that have operations on this object. All objects dependent on TCB_d subjects are classified as TCB protected objects or resources. Thus we need to identify all TCB_d subjects from an information domain and verify the assurance of their integrity. To ease this task, a minimum TCB_d is first discovered. In the situation that the minimum TCB_d subjects have dependency relationships with other subjects, these subjects should be added to domain TCB, or the dependencies should be removed. Based on these principles, we first identify initial TCB_d subjects which are predominant subjects for the information domain *d*. We further discover other TCB_d subjects considering subject dependency relationships with the initial TCB_d through *information flow transitions*, which means subjects that can only flow to and from initial TCB_d subjects should be included into TCB_d. For instance, for a web server domain, `httpd` is the subject that initiates other web server related processes. Hence, `httpd` is the predominant subject and belongs to TCB_d. Based on all possible information flows to `httpd`, we then identify other subjects such as `httpd-suexec` in TCB_d.

1. Section 7 shows the Apache domain TCB in our evaluation system.

3.3 Integrity Protection with Domain Isolation

To protect the identified TCB_s and TCB_d, we develop principles similar to those in Clark-Wilson [8]. Clark-Wilson leverages TP to allow information flow from low integrity processes to high integrity processes. To support TP, we adopt the concept of filters. Filters can be processes or interfaces [24] that normally are distinct input information channels and are created by a particular operation such as `open()`, `accept()`, or other calls that enable data input. For example, `su` process allows a low integrity process (e.g., staff) being changed to be a high integrity process (e.g., root) by executing `passwd` process, thus `passwd` can be regarded as a filter for processes run by root privilege. Also, high integrity process (e.g., `httpd` administration) can accept low integrity information (e.g, network data) through the secure channel such as `sshd`. Consequently, `sshd` can be treated as a filter for higher privilege processes. In general, filters include any available sanitization process and TP in traditional integrity model. Since it is usually very difficult to build and completely verify a sanitization process, we believe it is more reasonable to assume that the system is in higher trusted state when any low-to-high information flow is explicitly authorized by a user.

With the identifications of TCB_s, TCB_d, and filters for information domain `d`, all the other subjects in a system are categorized as NON-TCB. Our domain-based isolation is defined as follows:

Definition 1: Domain-based isolation is satisfied for an information domain d if

- information flows are from TCB_d; or
- information flows are from TCB_s to either TCB_d or TCB_d protected resources; or
- information flows are from NON-TCB to either TCB_d or TCB_d protected resources via filter(s).

Deductively, we articulate the rule for integrity evaluation based on domain-based isolation.

Rule 1: If there is information flow from NON-TCB to TCB_d without passing through any filter, there is an integrity violation related to TCB_d protected resources.

Rule 2: Or if there is information flow from NON-TCB or TCB_d to TCB_s without passing through any filter, there is an integrity violation related to TCB_s protected resources.

4 DR@FT: DESIGN AND PROCEDURES

DR@FT consists of three main parties: an attestee (the target platform), an attester (attestation challenger), and a trusted authority as shown in Figure 1. The attestee is required to provide its system state information to the attester for verification. Here, we assume that the attestee is initially in a *trusted system state* and the system state is changed to a new state after certain system behaviors.

A reporting daemon on the attestee gets the measured new state information (step 1) with IMA and generates the policy updates (step 2). This daemon then gets AIK-signed PCR value(s) and sends to the attester. After the attester receives and authenticates the information, with the attestee's AIK public key certificate from the trusted authority, it verifies the attestee's integrity through codes and data verification (step 3), reporting process authentication (step 4) and policy analysis (step 5).

4.1 System State and Trust Requirement

For attestation purpose, a system state is captured as a snapshot of the attestee system at a particular moment, where the factors characterizing the state can influence the system integrity on any future moment of the attestee system. Based on our domain-based integrity model, the attestee system integrity can be represented via information flows, which are characterized by the trusted subject list, filters, policies, and the trustworthiness of TCB_s. Based on these properties, we define the system state of the attestee as follows.

Definition 2: A *system protection state* (or simply *system state*) at the time period i is a tuple T_i={ TSL_i, CD_i, $Policy_i$, $Filter_i$, $RProcess_i$ }, where

- TSL_i=$\{s_0, s_1....s_n\}$ represents a set of high integrity processes which corresponds to a set of subjects s_0, $s_1....s_n$ in TCB_s and TCB_d;
- CD_i=$\{$cd (s_0), cd (s_1)....cd (s_n)$\}$ is a set of codes and data for loading a subject $s_j \in TSL_i$;
- $Policy_i$ is the security policy currently configured on the attestee;
- $Filter_i$ is a set of processes defined to allow information flow from low integrity processes to high integrity processes; and
- $RProcess_i$ represents a list of processes that measure, monitor, and report the current TSL_i, CD_i, $Filter_i$ and $Policy_i$ information. IMA agent and the attestation reporting daemon are the examples of the $RProcess_i$.

According to this definition, a system state does not include a particular application's running state such as its memory page and CPU context (stacks and registers). It only represents the security configuration or policy of an attestee system. A system *state transition* indicates one or more changes in TSL_i, CD_i, $Policy_i$, $Filter_i$, or $RProcess_i$. A system state T_i *is trusted* if TSL_i belongs to $TCB_s \cup TCB_d$, CD_i does not contain untrusted codes and data, $Policy_i$ satisfies domain-based isolation, $Filter_i$ belongs to defined filters in domain-based isolation, and $RProcess_i$ codes and data do not contain malicious codes and data and these $RProcess_i$ processes are integrity protected from the untrusted processes via $Policy_i$.

As mentioned earlier, we assume that there exists an initial trusted system state T_0. Through changing the variables in T_0, the system transits to other states such as T_1, ..., T_i. The attestation in DR@FT is to verify whether or not T_i is trusted.

4.2 Attestation Procedures

Integrity Measurements The measurement at the attestee side has two different forms, depending on *how much* the attestee system changes. Specifically, in case any subject in TCB_s is updated, the attestee must be fully remeasured from the system reboot and the attester needs to attest it completely, since this subject may affect the integrity of subjects in $RProcess$ of the system such as the measurement agent and reporting daemon. After the reboot and all TCB_s subjects are remeasured, a trusted initial system state T_0 is built. To perform the remeasurement, the attestee measures a state T_i and generates the measurement list M_i which is added by trusted subject list (TSL_i) measurement, codes and data (CD_i) measurement, policy ($Policy_i$) measurement, filter ($Filter_i$) measurement and attestation process ($RProcess_i$) measurement, as explained as follows:

- Trusted subject list (TSL_i) measurement: With TPM and measurement mechanisms such as IMA, the trusted subject list TSL_i is measured with a result m_{tsl}, which is added to a measurement list by $M_i = M_i||m_{tsl}$. $\mathcal{H}(M_i)$ is extended to a particular PCR of the TPM, where $\mathcal{H}$ is a hash function such as SHA1.
- Codes and data (CD_i) measurement: For every subject s_j in TSL_i, its codes and data $cd(s_j)$ are measured. To specify the mapping relationship, the measurement m_{cdsj} consists of the information of $cd(s_j)$ and s_j. After m_{cdsj} is generated, it is added to the measurement list by M_i=$M_i||m_{cdsj}$ and its hash value is extended to PCR.
- Policy ($Policy_i$) measurement: Corresponding to $Policy_i$, the attestee generates the measurement m_p, which is added to the measurement list with M_i=$M_i||m_p$, and corresponding PCR is extended with its hash value.
- Filter ($Filter_i$) measurement: The codes and data of $Filter_i$ are measured as m_f, which is extended to the measurement list M_i=$M_i||m_f$, and the PCR is extended.
- Attestation Process ($RProcess_i$) measurement: The codes and data of $AProcess_i$ are measured as m_r, which is added to the measurement list M_i=$M_i||m_r$, and the PCR is also extended.

In another case, where there is no TCB_s subject updated and the TSL_i or $Filter_i$ subjects belonging to TCB_d are updated, the attestee only needs to measure the updated codes and data loaded by the changed TSL or filter subjects, and generates a measurement list M_i. The generation of this

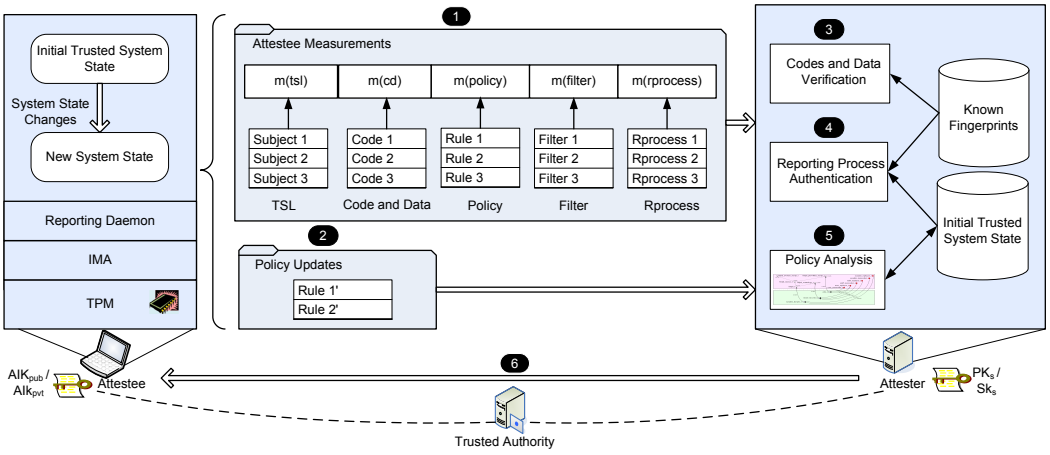


Fig. 1. Overview of DR@FT.

measurement list is realized through the runtime measurement supported by the underlying measurement agent.

To support both types of measurements, we develop an attestation reporting daemon which monitors the runtime measurements of the attestee. In case the runtime measurements for TCB_s are changed, the attestee is required to be rebooted and fully measured with IMA. The measurements values are then sent to the attester by the daemon. On the other side, the changed measurement value is measured by IMA and captured with the reporting daemon only if the measurement for TCB_d is changed. Obviously, this daemon should be trusted and is included in TCB_s. That is, its codes and data are required to be protected with integrity policy and corresponding hash values are required to be stored at the attester side.

Policy Updates To analyze if the current state of the attestee satisfies domain-based integrity property, the attester requires information about the current security policy loaded at the attestee side. Due to the large volume of policy rules in a security policy, sending all policy rules in each attestation and verifying all of them by the attester may cause the performance overhead. Hence, in DR@FT, the attestee only generates policy updates from the latest attested trusted state and sends them to the attester for the attestation of such updates.

To support this mechanism, we have the attestation reporting daemon to monitor any policy update on attestee system and generate a list of updated policy rules. The algorithm of this operation is shown in Algorithm 1. Upon the system policy $Policy_{i-1}$ is changed on state T_{i-1}, the daemon process reads the policy rules in the stored security policy $Policy_0$ of the trusted system state T_0 and the current policy $Policy_i$ separately, and compares these rules. Then, the daemon finds the added, changed and deleted rules through functions *added, changed, deleted*, respectively, and saves them into the policy update file. Note that the policy update comparison is performed between the current updated policy and the stored trusted security policy $Policy_0$ or previously attested and trusted $Policy_{i-1}$. The complexity of this policy update algorithm is *O(nr)*, where *nr* represents the number of the policy rules in the new policy file $Policy_i$.

Algorithm 1: Generating Policy Updates

Input: Currently configured policy file P_i, stored trusted policy file P_0 , currently running updates thread U_s
Output: Policy updates file $P_{updates}$

```
state := threadMonitor(U_s, P_i)
if isChanged(state) then
    fileFlow := Initialize(P_updates)
    R_list := readPolicy(P_i)
    foreach r_i ∈ R_list do
        compareResult := (r_i, P_0)
        if compareResult == 1 then
            added(fileFlow, r_i)
        else
            if compareResult == 2 then
                changed(fileFlow, P_0, r_i)
                markPolicy(P_0, r_i)
            else
                markPolicy(P_0, r_i)
R_list := readPolicy(P_0)
foreach r_i ∈ R_list do
    if isnotMarked(P_0, r_i) then
        deleted(fileFlow, r_i)
```

Codes and Data Verification With received measurement list M_i and AIK-signed PCRs, the attester first verifies the measurement integrity by re-constructing the hash values and compares with PCR values. After this is passed, the attester performs the analyses. Specifically, it obtains the hash values of CD_i and checks if they corresponds to known-good fingerprints. Also, the attester needs to assure that the TSL_i belongs to TCB_s and TCB_d. In addition, the attester also gets the hash value of $Filter_i$ and ensures

that they belong to the filter list defined on the attester side. In case this step succeeds, the attester has the assurance that target processes on attestee side are proved without containing any untrusted code or data, and the attester can proceed to next steps. Otherwise, the attester sends a proper attestation result denoting this situation.

Reporting Process Authentication To prove that the received measurements and updated policy rules are from the attestee, the attester authenticates them by verifying that all the measurements, updates and integrity measurement agent processes in the attestee are protected. That is, the $RProcess_i$ does not contain any untrusted codes or data and its measurements correspond to PCRs in the attester. Also, there is no integrity violated information flow to these processes from subjects of TSL_i, according to the domain isolation rules. Note that these components can also be updated, but after any update of these components, the system should be fully remeasured and attested from the bootstrap to rebuild a trusted initial system state T_0.

Policy Analysis by Attester DR@FT analyzes policy using a graph-based analysis method. In this method, a policy file is first visualized with a graph. The policy graph is then analyzed against a pre-defined security model, such as our domain-based isolation model, and a policy violation graph is generated. The main goal of this intuitive approach is to give semantic information of attestation result to the attestee, such that system administrators can easily recognize any violated configuration.

Note that verifying all of the security policy rules in each attestation request decrease the efficiency, as loading policy graph and checking all of the policy rules are costly. Thus, we need to develop an efficient way for analyzing the attestee policy. In our method, the attester stores the policy of initial trusted system state T_0 or the latest trusted system state T_i, and loads its corresponding policy graph, which does not have any policy violation. Upon receiving the updated information from the attestee, the attester just needs to analyze these updates to examine if there is new information flow violating integrity requirements. The details of graph-based policy analysis are explained in Section 5.

Attestation Result Sending to Attester In case the attestation is successful, a new trusted system state is built and the corresponding information is stored at the attester side for subsequent attestations. On the other hand, if the attestation fails, there are several possible attestation results including any combination of the following cases: CD_i integrity success/fail, $RProcess_i$ un/authenticated, and $Policy_i$ success/fail. To assist the attestee reconfiguration, the attester also sends a representation of the policy violation graph to the attestee. Moreover, with this policy violation graph, the attester ranks the violation graph and measures the trustworthiness of the attestee, which is explained in Section 6.

5 GRAPH-BASED POLICY ANALYSIS

As we have discussed, existing attestation solutions such as TCG and IMA lack the expressiveness of the attestation result. Instead of a boolean value for an attestation result, DR@FT adopts a graph-based policy analysis mechanism, where a policy violation graph can be constructed for identifying all policy violations on the attestee side. We further introduce a risk model based on a ranking scheme, which implies how severe the discovered policy violations are and how efficiently they can be resolved. This section illustrates the graph-based integrity analysis and violation identification. We explain the risk model in next section.

5.1 Information Flow and Policy Graph

For information flow purpose, all operations between subjects and objects in a policy can be classified as $write\_like$ or $read\_like$ [15] and operations between subjects can be expressed as $calls$. Depending on the types of operations, information flow relationships can be identified. If a subject *x* can write an object *y*, there is information flow from *x* to *y*, which is denoted as $write(x, y)$. On the other hand, if a subject *x* can read an object *y*, there is information flow from *y* to *x* denoted as $read(y, x)$. Another situation is that if a subject *x* can call another subject *y*, there is information flow from *y* to *x*, which is denoted as $call(y, x)$. Moreover, the information flow relationships between subjects and objects can be further described through *flow transitions*. In a policy, if a subject s_1 can write an object o which can be read by another subject s_2, it implies that there is an *information flow transition* from subject s_1 to subject s_2, denoted as $flowtrans(s_1, s_2)$. Also, if a subject s_2 can call a subject s_1, there is a flow transition from s_1 to s_2. A sequence of flow transitions between two subjects represents an *information flow path*.

With information flow and flow transitions between subjects and objects, we define a policy graph as follows:

Definition 3: A *Policy Graph* of a system is a directed graph G=(V, E), where the set of vertices V represents all subjects and objects in the system, and the set of edges E=$V \times V$ represents all information flow relations between subjects and objects. That is,

- $V \subseteq V_o \bigcup V_s$, where V_o and V_s are the sets of nodes that represent objects and subjects, respectively;
- $E \subseteq E_r \bigcup E_w \bigcup E_c$. Given the vertices v_{s1},v_{s2} $\in V_s$ separately representing subjects s1 and s2, and vertices $v_o \in V_o$ representing object o, $(v_{s1}, v_o) \in E_w$ if and only if $write(s1, o)$, $(v_o, v_{s2}) \in E_r$ if and only if $read(o, s2)$, and $(v_{s1}, v_{s2}) \in E_c$ if and only if $call(s1, s2)$.

We use semantic substrates to display policies [25]. We divide a canvas into different areas based on the classification of entities (subjects and objects) and then layout nodes expressing the entities into corresponding areas. We also use non-spacial cues (e.g., color or shape) to emphasize certain nodes or a group of nodes. Figure 2 shows some example policy rules within a policy graph. The Y-axis is divided into regions, where each region

contains nodes representing entities such as subjects and objects. Furthermore, in each region, nodes representing entities of different classifications are placed in different spaces along with the X-axis. For subjects and objects in a policy, $S_{c1}...S_{cn}$ and $O_{c1}...O_{cm}$ separately represent certain classifications. Different colors and shapes are used to distinguish the identification of different nodes. Circles and rectangles are used to represent subjects and objects, respectively. Relationships between subjects and objects are expressed with lines in different colors or shapes. For instance, the write operation between subject s_2 and object o_2 is expressed with a red link.

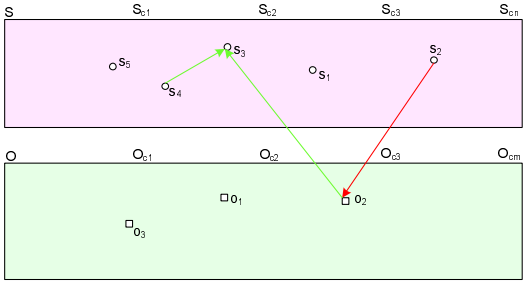


Fig. 2. Policy graph: The link between s_2 and o_2 represents write operation, s_3 and o_2 for read operation, and s_4 and s_3 for call operation.

5.2 Policy Violation Graph

An information flow represents how data flows among system subjects. A sequence of information flow transitions between two subjects shows an information flow path. Applying our domain-based isolation approach, policy violations can be detected to identify information flows from low integrity subjects to high integrity subjects. Corresponding information flow paths representing such policy violations are named *violation paths*.

Based on domain-based integrity model, we can discover two kinds of violation paths, *direct violation paths* and *indirect violation paths*. A direct violation path is a one-hop path through which an information flow can go from a low integrity subject to a high integrity subject. We observe that information flows are transitive in general. Therefore, there may exist information flows from a low integrity subject to a high integrity subject via several other subjects. This multi-hop path is called indirect violation path. All direct and indirect violation paths belonging to a domain can construct a policy violation graph for this domain.

Definition 4: A policy violation graph for a domain d is a directed graph $G^v = (V^v, E^v)$:

- $V^v \subseteq V^v_{NTCB} \cup V^v_{TCBd} \cup V^v_{TCBs}$ where V^v_{NTCB}, V^v_{TCBd} and V^v_{TCBs} are subject vertices containing in direct or indirect violation paths of domain d and belong to NON-TCB, TCB_d, and TCB_s, respectively.
- $E^v \subseteq E^v_{Nd} \cup E^v_{dT} \cup E^v_{NT} \cup E^v_{NTCB} \cup E^v_{TCBd} \cup E^v_{TCBs}$ where $E^v_{Nd} \subseteq (V^v_{NTCB} \times V^v_{TCBd})$, $E^v_{dT} \subseteq (V^v_{TCBd} \times V^v_{TCBs})$, $E^v_{NT} \subseteq (V^v_{NTCB} \times V^v_{TCBs})$, $E^v_{NTCB} \subseteq (V^v_{NTCB} \times V^v_{NTCB})$, $E^v_{TCBd} \subseteq (V^v_{TCBd} \times V^v_{TCBd})$, and $E^v_{TCBs} \subseteq (V^v_{TCBs} \times V^v_{TCBs})$, and all edges in E^v are contained in direct or indirect violation paths of domain d.

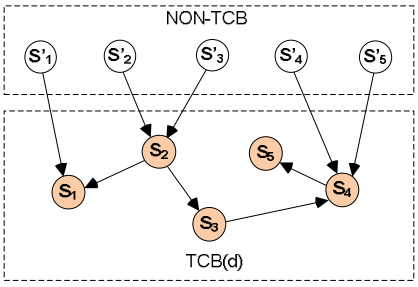
Figure 3 (a) shows an example of policy violation graph which examines information flows between NON-TCB and TCB_d. [2] Five direct violation paths are identified in this graph: $<S'_1, S_1>$, $<S'_2, S_2>$, $<S'_3, S_2>$, $<S'_4, S_4>$, and $<S'_5, S_4>$ across all the boundaries between NON-TCB and TCB_d. Also, eight indirect violation paths exist. For example, $<S'_2, S_5>$ is a four-hop violation path passing through other three TCB_d subjects S_2, S_3, and S_4.

Algorithm 2 realizes the processing to identify policy violation graph with inputs of a policy graph and updated policy file. First, a function *policyParse* parses the updated policy information into subjects, objects, and their relationships with permission mapping. Second, we load this information onto the previously generated $Policy_0$ graph with functions *changeNodes* and *changeLinks*, respectively. Then, it determines if there is a new information flow generated on this graph with a function *getnewFlow*. Third, we need to check if this new information flow violates our domain-based isolation rules using a function *identifyViolation*. Through this approach, rather than analyzing all the policy rules and all information flows for each attestation, we verify the new policy through only checking the updated policy rules and the newly identified information flows. The complexity of this policy analysis algorithm is *O(nn +nl +nt)*, where *nn* represents the number of changed subjects and objects, *nl* is the number of changed subject and object relationships in the policy update file; and *nt* represents the number of changed TCB in the TSL file.

Algorithm 2: Finding Integrity Violations with Policy Graph and Updates

Input: Initial trusted policy graph G_0, policy updating file P_u, TSL file T_{listi}, policy explanation file F_e, permission mapping File F_p, subject classification file F_s, object classification file F_o
Output: Policy violation graph G_{flow}
$Policy_u := policyParse(P_{u,Fe}, F_p, F_s, F_o)$
$N_{list} := getUpdateSO(Policy_u)$
foreach $r_i \in R_{list}$ **do**
  $changeNodes(n_a, G_0)$
$L_{list} := getUpdateLink(Policy_u)$
foreach $l_a \in L_{list}$ **do**
  $changeLinks(l_a, G_0)$
  $getnewFlow(l_a, G_0)$
foreach $t_a \in T_{list}$ **do**
  $changeTCB(t_a, G_0)$
  $G_{flow} := identifyViolation(t_a, G_0)$

2. Similarly, the information flows between NON-TCB and TCB_s, and between TCB_d and TCB_s can be examined accordingly.



(a) A policy violation graph

Subject	SubjectRank
S_1	0.2799
S_2	0.3999
S_3	0.0799
S_4	0.4319
S_5	0.3456

(b) SubjectRank in TCB(d)

Path	PathRank
$\langle S'_1, S_1 \rangle$	0.2799
$\langle S'_2, S_2 \rangle$	0.8104
$\langle S'_3, S_2 \rangle$	0.8104
$\langle S'_4, S_4 \rangle$	0.6048
$\langle S'_5, S_4 \rangle$	0.6048

(c) PathRank between NON-TCB and TCB(d)

Fig. 3. Example policy violation graph and rank. SubjectRank and PathRank indicate severity of violating paths.

6 Ranking Policy Violation Graph

In order to explore more features of policy violation graphs and facilitate efficient policy violation detection and resolution, we introduce a scheme for ranking policy violation graphs. There are two steps to rank a policy violation graph. First, TCB_d subjects in the policy violation graph are ranked based on dependency relationships among them. The rank of a TCB_d subject shows reachable probability of low integrity information flows from NON-TCB subjects to the TCB_d subject. In addition, direct violation paths in the policy violation graph are evaluated based on the ranks of TCB_d subjects to indicate the severity of these paths which allow low integrity information to reach TCB_d subjects. The ranked policy violation graphs are valuable for system administrators as they need to estimate the *risk level* of a system and provide a guide for choosing appropriate strategies for resolving policy violations efficiently.

6.1 Ranking Subjects in TCB$_d$

Our notation of *SubjectRank* (SR) in policy violation graphs is a criterion that indicates the likelihood of low integrity information flows coming to a TCB_d subject from NON-TCB subjects through direct or indirect violation paths. The ranking scheme we adopt is a similar process of rank analysis applied in hyper-text link analysis system, such as Google's PageRank [26] that utilizes a link structure provided by hyperlinks between web pages to gauge their importance. Comparing with PageRank which focuses on analyzing a web graph where the entries are *any* web pages contained in the web graph, the entries of low integrity information flows to TCB_d subjects in a policy violation graph are only identified NON-TCB subjects.

Consider a policy violation graph with *N* NON-TCB subjects, and s_i is a TCB_d subject. Let $N(s_i)$ be the number of NON-TCB subjects from which low integrity information flows could come to s_i, $N^{'}(s_i)$ the number of NON-TCB subjects from which low integrity information flows could *directly* reach to s_i, $In(s_i)$ a set of TCB_d subjects pointing to s_i, and $Out(s_j)$ a set of TCB_d subjects pointed from s_j. The probability of low integrity information flows reaching a subject s_i is given by

$$SR(s_i) = \frac{N(s_i)}{N}\left(\frac{N^{'}(s_i)}{N(s_i)} + \left(1 - \frac{N^{'}(s_i)}{N(s_i)}\right) \sum_{s_j \in In(s_i)} \frac{SR(s_j)}{|Out(s_j)|}\right) \quad (1)$$

SubjectRank can be interpreted as a *Markov Process*, where the states are TCB_d subjects, and the transitions are the links between TCB_d subjects which are all evenly probable. While a low integrity information flow attempts to reach a high integrity subject, it should select an entrance (a NON-TCB subject) which has the path(s) to this subject. Thus, the possibility of selecting correct entries to a target subject is $\frac{N(s_i)}{N}$. After selecting correct entries, there still exist two ways, through direct violation or indirect violation paths, to reach a target subject. Therefore, the probability of flow transition from a subject is divided into two parts: $\frac{N^{'}(s_i)}{N(s_i)}$ for direct violation paths and $1 - \frac{N^{'}(s_i)}{N(s_i)}$ for indirect violation paths. The $1 - \frac{N^{'}(s_i)}{N(s_i)}$ mass is divided equally among the subject's successors s_j, and $\frac{SR(s_j)}{|Out(s_j)|}$ is the rank value derived from s_j.

Figure 3 (b) displays a result of applying Equation (1) to the policy violation graph shown in Figure 3 (a). Note that even though a subject s_4 has two direct paths from NON-TCB subjects like a subject s_2, the rank value of s_4 is higher than the rank value of s_2, because there is another indirect flow path to s_4 (via s_3).

6.2 Ranking Direct Violation Path

We further define *PathRank* (PR) as the rank of a direct violation path,[3] which reflects the severity of the violation path through which low integrity information flows may come to TCB_d subjects. Direct violation paths are regarded as the entries of low integrity data to TCB_d in policy violation graph. Therefore, the ranks of direct violation paths give a guide for system administrators to adopt suitable defense countermeasures for solving identified violations. To calculate *PathRank* accurately, three conditions are needed to be taken into account: (1) the number of TCB_d that low integrity flows can reach through this direct violation path; (2) SubjectRank of reached TCB_d subjects;

3. It is possible that a system administrator may also want to evaluate indirect violation paths for violation resolution. In that case, our ranking scheme could be adopted to rank indirect violation paths as well.

and (3) the number of hops to reach a TCB_d subject via this direct violation path.

Suppose $< s_i^{'}, s_j >$ is a direct violation path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_j in a policy violation graph. Let $Reach(< s_i^{'}, s_j >)$ be a function returning a set of TCB_d subjects to which low integrity information flows may go through a direct violation path $< s_i^{'}, s_j >$, $SR(s_l)$ the rank of a TCB_d subject s_l, and $H_s(s_i^{'}, s_l)$ a function returning the hops of the shortest path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_l. The following equation is utilized to compute a rank value of the direct violation path $< s_i^{'}, s_j >$.

$$PR(< s_i^{'}, s_j >) = \sum_{s_l \in Reach(< s_i^{'}, s_j >)} \frac{SR(s_l)}{H_s(s_i^{'}, s_l)} \quad (2)$$

Figure 3 (c) shows the *PathRank* of the example policy violation graph, which is calculated by the above-defined equation. For example, $< s_2^{'}, s_2 >$ has a higher rank than $< s_1^{'}, s_1 >$, because $< s_2^{'}, s_2 >$ may result in low integrity information flows to reach more or important TCB_d subjects than $< s_1^{'}, s_1 >$.

6.3 Evaluating Trustworthiness

Let P_d be a set of all direct violation paths in a policy violation graph. The entire rank, which can be considered as a risk level of the system, can be computed as follows:

$$RiskLevel = \sum_{< s_i^{'}, s_j > \in P_d} PR(< s_i^{'}, s_j >) \quad (3)$$

The calculated risk level could reflect the trustworthiness of an attestee. Generally, the lower risk level indicates the higher trustworthiness of a system. When an attestation is successful and there is no violation path being identified, the risk level of the attested system is *zero*, which means an attestee has the highest trustworthiness. On the other hand, when an attestation is failed, corresponding risk level of a target system is computed. A *selective service* could be achieved based on this fine-grained attestation result. That is, the number of services provided by a service provider to the target system may be decided with respect to the trust level of the target system. Hence, a system administrator could refer to this attestation result as the evaluation of her/his system as well as resolution guidelines since this quantitive response would give her a proper justification to adopt countermeasures for improving the trustworthiness of a target system.

6.4 Attestee Reconfiguration

When the current system state of an attestee does not satisfy integrity requirements, a system administrator may need to change the system configuration based on the attestation result to enhance the system integrity and improve its trust level. The total rank of all violation paths provides a measurement of risk level of a system and thus reflects the system's trustworthiness. A system administrator is able to adopt different countermeasures to resolve identified violations with respect to the different rank values of violation paths. We summarize several options as follows:

- Changing CD_i and TSL_i: In case an attestation is failed because of loading malicious or unknown codes or data into the system, removing these codes and data can change the system state to be trusted, while there is no change to the policy.
- Identifying filter: If a filter is identified along with the information flow path that causes a violation, the filter can be added to CD_i and TSL_i. In this case, the violation is treated as a false alarm and there is no change to the policy graph.
- Changing $Policy_i$: we are able to modify policy for violation resolution, either by excluding subjects or objects from the violated information flow paths, or by replacing subjects or objects with more restricted privileges. Based on the violated information flow paths and corresponding ranks, a system administrator can first eliminate highly ranked violation paths since they generally involve more violations and more TCB_d subjects. Through modifying policy rules, $Pflow_i$ is changed to satisfy our domain isolation requirements. In other words, an attestee can maintain integrity requirements by modifying system policies properly for a newly installed software which interacts with TSL subjects.
- Adding filter: We can also introduce a new filter subject that acts as a gateway along a violation flow path between NON-TCB and TCB subjects. By including such a filter in CD_i and TSL_i, the violation can be resolved in subsequent attestations.

7 Implementation and Evaluation

We have implemented DR@FT to evaluate its effectiveness and measure the performance. This section first describes our experimentation setup, followed by implementation details, effectiveness evaluation and, performance study.

Our attestee platform is a Lenovo ThinkPad X61 with Intel Core 2Duo Processor L7500 1.6GHz, 2 GB RAM, and Atmel TPM. We enable SELinux with the default policy based on the current distribution of SELinux [22]. To measure the attestee system with TPM, we update the Linux kernel to 2.6.26.rc8 with the latest IMA implementation [23], where SELinux hooks and IMA functions are enabled.

Having IMA enabled, we configure the measurement of the attestee information. After the attestee system kernel is booted, we mount the `sysfs` file system and inspect the measurement list values in `ascii_runtime_measurements` and `ascii_bios_measurements`. Figure 5(a) shows a sample of the measurement list.

7.1 Attestation Implementation

We start from a legitimate attestee and make measurements of the attestee system for the later verification. To invoke a new attestation request from the attester, the attestation

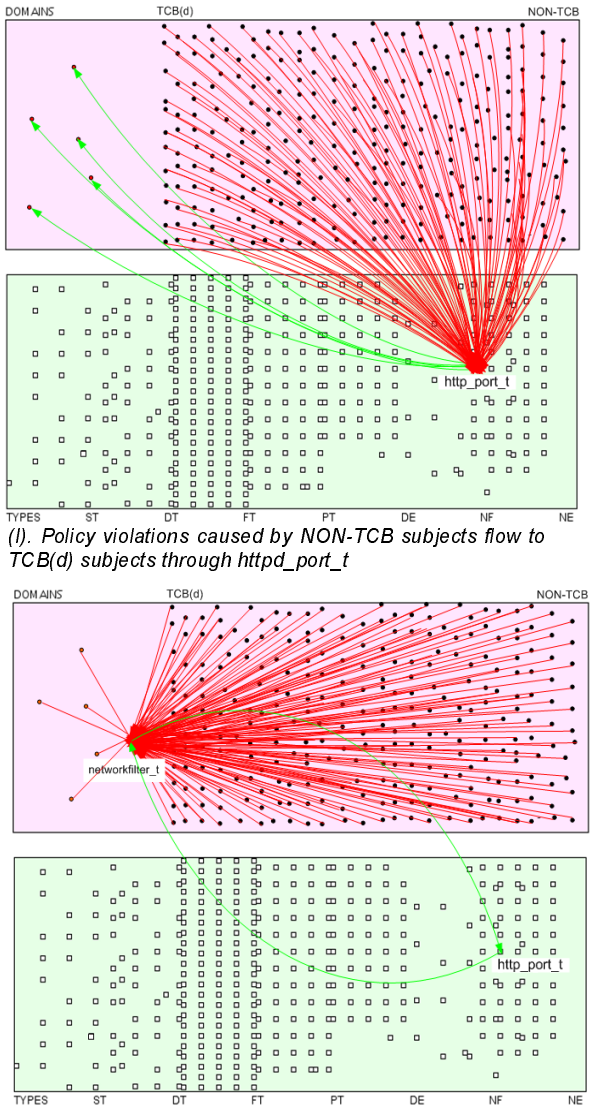
reporting daemon runs in the attestee and monitors the attestee system. This daemon is composed of two main threads: One monitors and obtains new system state measurements, and the other monitors and obtains the policy updates of the attestee. The daemon is also measured and the result can be obtained through the legitimate attestee. Thus the integrity of the daemon can be verified later by the attester. In case the attestee system state is updated due to new software installation, changing policy, and so on, the corresponding thread of the daemon automatically obtains the new measurement values as discussed in Section 4. The daemon then securely transfers the attestation information to the attester based on available security mechanisms supported by the trusted authority.

After receiving the updated system information from the attestee, the measurement module of the attester checks the received measurements against the stored PCR to prove its integrity. To analyze the revised attestee policy, the policy analysis module is developed as a daemon, which is derived from a policy analysis engine [27]. We extend the engine to support information flow query functions, so as to identify violated information flows from the updated policy rules based on domain-based isolation rules. We use JDK1.6 and other necessary Java libraries to develop the main analysis components. We implemented graph drawing with graph package Piccolo [28]. We also accommodate the attestation procedures presented in Section 4.2, as well as our rank scheme to evaluate the trustworthiness of the attestee.

7.2 Identifying TCBs and Resolving Violations

To evaluate the proposed attestation framework, we attest our testbed platform with Apache web server installed. To configure the trusted subject list of the Apache domain, we first identify TCB_s based on the reference monitor-based TCB identification [21], including the integrity measurement, monitoring agents, and daemon. For TCB_d of the Apache, we identify the Apache information domain, Apache TCB_d, including `httpd_t` and `httpd_suexec_t`, and the initial filters `sshd_t`, `passwd_t`, `su_t`, through the domain-based isolation principles. Both TCB_s and TCB_d are initially identified with a graphical policy analysis tool [27], and then all possible policy violations are detected as well according to our domain-based integrity model. Figure 4(I) shows an example with automatically identified policy violations, which indicates write_like operations from NON-TCB subjects to objects that can be read by TCB_d subjects. We then apply the system reconfiguration strategies discussed in Section 6.4 to resolve these violations.

Update TCB We observe that some violations are generated due to the reason that all information flows to some subjects in turn flows to TCB subjects. We then include these subjects in the TCB. For example, in our evaluation policy, `httpd_awstats_script_t` can flow to TCB_d subjects through `httpd_awstats_script_rw_t`. At the same time, it is flown in by many NON-TCB subjects through some common types such as `devtty_t`. Hence, we ignore the violations caused by this *awstats_script* and include it into TCB_d. Similar situation occurs for `httpd_apcupsd_cgi_script_t` and `httpd_prewikka_script_t`. However, `httpd_staff_script_t` cannot be included into TCB_d since it would lead unlimited file access for the staff services such as `staff_t`, `staff_mozilla_t`, and `staff_mplayer_t`.



(I). Policy violations caused by NON-TCB subjects flow to TCB(d) subjects through httpd_port_t

(II). Policy violation caused by http_port_t is solved through adding networkfilter into the related policies.

Fig. 4. Policy violations and corresponding resolution for Apache domain.

Remove Policy Rules Another way for resolving policy violations is to remove related policy statements. For example, `webalizer_t` is to label a tool for analyzing the log files of web server and is not necessary to modify web server information. To resolve the policy violations caused by `webalizer_t` due to the write access to `httpd_sys_content_t`, we remove the policy rule

stating write_like operation between webalizer_t and httpd_sys_content_t.

Modify Policy Rules Many policy violations are caused because related subjects or objects are given too much privileges. Hence, rather than just removing related policy statements, we also need to replace these subjects or objects with more restricted rights. For example, for policy violations caused by read and write accesses to initrc_devpts_t, our solution is to re-define initrc_devpts_t by introducing initrc_devpts_t, system_initrc_devpts_t, and *_daemon_initrc_devpts_t. [4] Corresponding policy rules are modified as follows:

```
allow httpd_t
initrc_devpts_t:chr_file {ioctl read
getattr lock write append};
```
is changed to
```
allow httpd_t
httpd_daemon_initrc_devpts_t:chr_file
{ioctl read getattr lock write
append};
```

Add Filter Based on our domain-based integrity model, a filter can be introduced into policies to resolve policy violations. For example, to remove the violations caused by http_port_t, we introduce a network filter subject as follows:

```
allow user_xserver_t
networkfilter_t:process transition;
allow networkfilter_t
http_port_t:tcp_socket {recv_msg
send_msg};
```

After the modification is applied, the original policy violations are eliminated. In general, to validate the result of a policy modification, we need to recheck the relationships between the policy violation related domains and types. Comparing Figure 4 (I) with Figure 4 (II), we can observe that all read operations between TCB_d and type http_port_t are removed. Also, the write operations between NON-TCB and http_port_t are also removed. Instead, a new domain networkfilter_t is added, which has write and read operations on http_port_t. Also, all TCB_d and NON-TCB subjects can transit to this new domain type.

The initial TCB_d is then adjusted during the process of resolving violations. Table 1 shows the final system TCB and Apache TCB identified with our approach. We then install unverified codes and data to evaluate the effectiveness of our attestation framework.

7.3 Installing Malicious Code

We first install a Linux rootkit, which gains administrative control without being detected. Here, we assign the rootkit with the domain unconfined_t, which enables information flows to domain initrc_t which labels initrc process in the TCB_s of the attestee.

Following the framework proposed in Section 4, the attestee system is measured from the bootstrap with the configured IMA. After getting the new measurement values, the reporting daemon sends these measurements to the attester. Note that there is no policy update in this experiment. Different from IMA, we only measure the TCB_s and TCB_d subjects. After getting the measurements from the attestee, the attester verifies them by analyzing the measured hash values. Figure 5 shows partial measurements of a trusted initial system state and the changed state because of the installed rootkit. The difference between these two measurements indicates that the original initrc is altered, thus the attester confirms that the attestee is not in a trusted state.

4. * representing the corresponding service name.

TABLE 1
Identified System TCB and Apache TCB

System TCB		
kernel_t	load_policy_t	initrc_t
mount_t	ipsec_mgmt_t	useradd_t
hwclock_t	admin_passwd_exec_t	cardmgr_t
kudzu_t	sshd_login_t	restorecon_t
syslogd_t	sysadm_t	getty_t
dpkg_t	logrotate_t	snmpd_t
bootloader_t	quota_t	init_t
ldconfig_t	apt_t	sshd_t
passwd_t	newrole_t	klogd_t
automount_t	checkpolicy_t	fsadm_t
lvm_t	local_login_t	setfiles_t
ima_t	rpdaemon_t	
Apache domain TCB		
httpd_suexec_t	httpd_awstats_script_t	httpd_t
httpd_helper_t	httpd_php_t	httpd_rotatelogs_t
httpd_sysadm_script_t	httpd_prewikka_script_t	httpd_apcupsd_cgi_script_t

7.4 Identifying Vulnerable Software

In this experimentation, we install a vulnerable software called Mplayer on the attestee side. Mplayer is a media player and encoder software which is susceptible to several integer overflows in the real video stream dumuxing code. These flaws allow an attacker to cause a denial of service or potentially execution of the arbitrary code by supplying a deliberately crafted video file. During the installation of a Mplayer, a Mplayer policy module is also loaded into the attestee policy. In this policy module, there are several different subjects such as usr_mplayer_t, staff_mplayer_t and sysadm_mplayer_t. Also, some objects are defined in security policies such as user_mplayer_home_t, staff_mplayer_home_t and sysadm_mplayer_home_t. After the Mplayer is installed, the attestation daemon finds that the new measurement of Mplayer is generated and the security policy of the system is changed. As the Mplayer does not belong to TCB_s and Apache TCB_d, the attestation daemon does not need to send the measurements to the attester. Consequently, the daemon only computes the security policy updates and sends the information to the attester.

Upon receiving the updated policies, we analyze these updates and obtain a policy violation graph as shown in Figure 6. Through objects such as cifs_t, sysadm_devtps_t, and ncsd_var_run_t, information flows from Mplayer can reach Apache domain. In addition, rank values are calculated and shown in the policy

```
....
10 033e4f559247b256b5a163568275d7901ebe3734  GBK.so
10 041789c808648eac2d7a20c5105f7996f55a6c56  GEORGIAN-PS.so
10 4d019ac9fd103d42f2f7f7d081becc3dd5beb806  IBM850.so
10 73b86a44681c8d778bb143bdd233a6477dc19884  IBM852.so
10 9348eeafbbe7fe8627330b543c3f88d028c7e16f  ECMA-CYRILLIC.so
....
10 38f30a0a967fcf2bfeele3b2971de540115048c8  initrc
....
```

(a) Measurement example

```
....
10 033e4f559247b256b5a163568275d7901ebe3734  GBK.so
10 041789c808648eac2d7a20c5105f7996f55a6c56  GEORGIAN-PS.so
10 4d019ac9fd103d42f2f7f7d081becc3dd5beb806  IBM850.so
10 73b86a44681c8d778bb143bdd233a6477dc19884  IBM852.so
10 9348eeafbbe7fe8627330b543c3f88d028c7e16f  ECMA-CYRILLIC.so
....
10 d63d12ced978aca120bfe6ee7683e394c2ffaef0  initrc
....
```

(b) Measurement after installing rootkit

Fig. 5. Measurement list example consisting a PCR location, file SHA-1 hash, and file name.

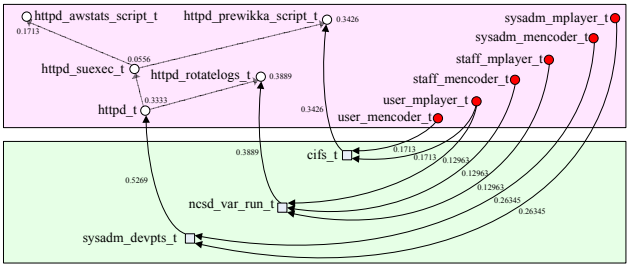


Fig. 6. Information flow verification for Mplayer. The links show the information flow from Mplayer (filled circle nodes) to Apache (unfilled nodes). The rank values on the paths indicate the severity of violations.

violation graph, which guides effective violation resolutions. For example, there are three higher ranked paths including paths from `sysadm_devpts_t` to `httpd_t`, from `ncsd_var_run_t` to `httpd_rotatelogs_t`, and from `cifs_t` to `httpd_prewikka_script_t`. Meanwhile, a risk level value (1.2584) reflecting the trustworthiness of the attestee system is computed based on the ranked policy violation graph.

Once receiving the attestation result shown in Figure 6, the system administrator can resolve the violation that has the higher rank than others. Thus, the administrator can first resolve the violation related to `httpd_t` through introducing `httpd_sysadm_devpts_t`.

```
allow httpd_t
httpd_sysadm_devtps_t:chr_file {ioctl
read write getattr lock append};
```

After the policy violation resolution, the risk level of the attestee system is lowered to *0.7315*. Continuously, after the attestee resolves all the identified policy violations and the risk level is decreased to be *zero*, the attestation daemon gets a new policy update and sends it to the attester. Upon receiving this update, the attester verifies whether the identified information flows violate domain-based isolation integrity rules since these flows are within the NON-TCB– even though there are new information flows compared to the stored $Policy_0$. Thus, an attestation result is generated which indicates the risk level (in this case, *zero*) of the current attestee system. Consequently, a new trusted system state is built for the attestee. In addition, the information of this new trusted system state is stored in the attester side for the later attestation.

7.5 Performance

To examine the scalability and efficiency of DR@FT, we investigate how well the attestee measurement agent, attestation daemon, and the attester policy analysis module scale along with increased complexity, and how efficiently DR@FT performs by comparing it with traditional approaches.

In DR@FT, the important factors influencing the attestation performance include system updates and policy changes. Hence, we evaluate the performance of DR@FT by changing codes and data to be measured and modifying the security policies. Based on our study, we observe that normal policy increased or decreased no more than 40KB when installing or uninstalling software. Also, a system administrator does not make the enormous changes over the policy. Therefore the performance is measured with the range from zero to around 40KB in terms of policy size.

Performance on the attestee side Based on DR@FT, the attestee has three main factors influencing the attestation performance. (1) Time spent for the measurement: Based on our experimentation, the measurement time increases roughly linearly with the size of the target files. For example, measuring policy files with 17.2MB and 20.3MB requires 14.63 seconds and 18.26 seconds, respectively. Measuring codes around 27MB requires 25.3sec. (2) Time spent for identifying policy updates $T_{Pupdate}$: Based on the specification in Section 4, policy updates are required to be identified and sent to the attester. As shown in Table 2, for a system policy which is with the size of 17.2MB at its precedent state, the increase of the policy size requires more time for updating the policy and vice versa. (3) Time spent for sending policy updates T_{Psend}: Basically, the more policy updates, the higher overhead was observed.

Performance on the attester side In DR@FT, the measurement verification is relatively straightforward. At the attester side, the time spent for policy analysis $T_{Panalysis}$ mainly influences its performance. As shown in Table 2, the analysis time roughly increases when the policy change rate increases.

Comparison of dynamic and static attestation To further evaluate the efficiency of DR@FT, we compare the overhead of DR@FT with a static attestation. In the static approach, the attestee sends all system state information to an attester, and the attester verifies the entire information step by step. As shown in Table 2, the time spent for the static attestation is composed of T_{Psend} and $T_{Panalysis}$, which represent the time for sending policy module and analyzing them, respectively. Obviously, the dynamic approach can dramatically reduce the overhead compared to the static approach. It shows that DR@FT is an efficient way when policies on an attestee are updated frequently.

TABLE 2
Attestation Performance Analysis (in seconds)

Policy Change	Dynamic				Static		
Size	$T_{Pupdate}$	T_{send}	$T_{Panalysis}$	Overhead	T_{Psend}	$T_{Panalysis}$	Overhead
0	0.23	0	0	0.23	14.76	90.13	104.89
-0.002MB (Reduction)	0.12	0.002	0.02	0.14	14.76	90.11	104.87
-0.019MB (Reduction)	0.08	0.01	0.03	0.12	14.74	89.97	104.34
-0.024MB (Reduction)	0.04	0.02	0.03	0.09	14.74	89.89	104.23
0.012MB (Addition)	0.37	0.01	0.03	0.41	14.77	90.19	104.96
0.026MB (Addition)	0.58	0.02	0.03	0.63	14.78	90.33	105.11
0.038MB (Addition)	0.67	0.03	0.04	0.74	14.79	90.46	105.25

8 Conclusion

We have presented a dynamic remote attestation framework called DR@FT for efficiently verifying if a system satisfies integrity protection property and indicates integrity violations which determine its trustworthiness level. The integrity property of our work is based on an information flow-based domain isolation model, which is utilized to describe the integrity requirements and identify integrity violations of a system. To achieve the efficiency and effectiveness of remote attestation, DR@FT focuses on system changes on the attestee side. We have extended our intuitive policy analysis engine to represent integrity violations with a rank scheme. In addition, our results showe that our dynamic approach can dramatically reduce the overhead compared to the static approach. We believe such a comprehensive method would help system administrators reconfigure the system with more efficient and strategic manner.

There are several directions that our attestation framework can be extended to. First, DR@FT attests system configurations according to security policies, while it does not attest the trustworthiness of dynamic contents of a system such as runtime state of applications. Second, our risk evaluation does not explain the tolerable risk level and relevant system properties. In addition, all verification tasks are performed at the attester side. The attester may need to delegate some attestation tasks to trusted components at the attestee or other trusted side. Our future work would seek a more flexible and systematic way to address these issues.

Acknowledgments

The work of G-J. Ahn and H. Hu was partially supported by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360) and Department of Energy (DE-SC0004308). The work of G-J. Ahn and W. Xu was also partially supported by the grants from National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References

[1] "Trusted computing group, https://www.trustedcomputinggroup.org."
[2] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *USENIX Security, 2004*.
[3] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble, "A protocol for property-based attestation," in *ACM STC, 2006*.
[4] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation: a virtual machine directed approach to trusted computing," in *USENIX conference on Virtual Machine Research And Technology Symposium, 2004*.
[5] T. Jaeger, R. Sailer, and U. Shankar, "Prima: policy-reduced integrity measurement architecture," in *ACM SACMAT, 2006*.
[6] K. J. Biba, "Integrity consideration for secure compuer system," Mitre Corp. Report TR-3153, Bedford, Mass., Tech. Rep., 1977.
[7] T.Fraser, "Lomac: Low water-mark integrity protection for cots environment," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000.
[8] R. S. Sandhu, "Lattice-based access control models," *IEEE Computer*, vol. 26, no. 11, pp. 9–19, 1993.
[9] U. Shankar, T. Jaeger, and R. Sailer, "Toward automated information-flow integrity verification for security-critical applications." in *NDSS, 2006*.
[10] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the selinux example policy," in *USENIX Security, 2003*.
[11] U. Shankar, T. Jaeger, and R. Sailer, "Toward automated information-flow integrity verification for security-critical applications." in *NDSS*. The Internet Society, 2006. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2006.html#ShankarJS06
[12] S. Smalley, "Configuring the selinux policy," http://www.nsa.gov/SELinux/docs.html, 2003.
[13] B. Hicks, S. Rueda, L. S. Clair, T. Jaeger, and P. McDaniel, "A logical specification and analysis for selinux mls policy," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, 2010.
[14] "Tresys Technology Apol." http://www.tresys.com/selinux/.
[15] J. Guttman, A. Herzog, and J. Ramsdell, "Information flow in operating systems: Eager formal methods," in *Workshop on Issues in the Theory of Security (WITS)*, 2003.
[16] B. Sarna-Starosta and S. D. Stoller, "Policy analysis for security-enhanced linux," in *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS)*, April 2004, pp. 1–12.
[17] W. Xu, M. Shehab, and G. Ahn, "Visualization based policy analysis: case study in selinux," in *Proc. of ACM Symposium of Access Control Models and Technologies*, 2008.
[18] G. Ahn, W. Xu, and X. Zhang, "Systematic policy analysis for high-assurance services in selinux," in *Proc. of IEEE Workshop on Policies for Distributed Systems and Networks*, 2008, pp. 3–10.
[19] M. Alam, X. Zhang, M. Nauman, T. Ali, and J.-P. Seifert, "Model-based behavioral attestation," in *ACM SACMAT, 2008*.
[20] *Trusted Computer System Evaluation Criteria*. United States Government Department of Defense (DOD), Profile Books, 1985.
[21] A. P. Anderson, "Computer security technology planning study," *ESD-TR-73-51*, vol. II, 1972.
[22] S. Smalley, "Configuring the selinux policy," http://www.nsa.gov/SELinux/docs.html, 2003.
[23] "LIM Patch," http://lkml.org/lkml/2008/6/27.
[24] N. Provos, M. Friedl, and P. Honeyman, "Preventing privilege escalation," *12th USENIX Security Symposium*, p. 11, August 2003.
[25] M. Green, "Toward a perceptual science of multidimensional data visualization: Bertin and beyond," *Available from http://www.ergogero.com/dataviz/dviz2.html*, 1998.
[26] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.
[27] W. Xu, X. Zhang, and G.-J. Ahn, "Towards system integrity protection with graph-based policy analysis," in *Proc. of the IFIP WG 11.3 Working Conference on Data and Applications Security*, 2009.
[28] "Piccolo ToolKit," http://www.cs.umd.edu/hcil/jazz/.