Confining Adversary Actions via Measurement

Paul D. Rowe(✉)

The MITRE Corporation, Bedford, USA
prowe@mitre.org

Abstract. Systems designed with measurement and attestation in mind are often layered, with the lower layers measuring the layers above them. Attestations of such systems must report the results of a diverse set of application-specific measurements of various parts of the system. There is a pervasive intuition that measuring the system "bottom-up" (i.e. measuring lower layers before the layers above them) is more robust than other orders of measurement. This is the core idea behind trusted boot processes. In this paper we justify this intuition by characterizing the adversary actions required to escape detection by bottom-up measurement. In support of that goal, we introduce a formal framework with a natural and intuitive graphical representation for reasoning about layered measurement systems.

1 Introduction

Security decisions often rely on trust. Many computing architectures have been designed to help establish the trustworthiness of a system through remote attestation. They gather evidence of the integrity of a target system and report it to a remote party who appraises the evidence as part of a security decision. A simple example is a network gateway that requests evidence that a target system has recently run antivirus software before granting it access to a network. If the virus scan indicates a potential infection, or does not offer recent evidence, the gateway might decide to deny access, or perhaps divert the system to a remediation network. Of course the antivirus software itself is part of the target system, and the gateway may require integrity evidence for the antivirus software for its own security decision. This leads to the design of layered systems in which deeper layers are responsible for generating integrity evidence of the layers above them.

A simple example of a layered system is one that supports "trusted boot" in which a chain of boot-time integrity evidence is generated for a trusted computing base that supports the upper layers of the system. A more complex example might be a virtualized cloud architecture. The virtual machines (VMs) at the top are supported at a lower layer by a hypervisor or virtual machine monitor. Such an architecture may be augmented with additional VMs at an intermediate layer that are responsible for measuring the main VMs to generate integrity evidence. These designs offer exciting possibilities for remote attestation. They allow for specialization and diversity of the components involved, tailoring the capabilities of measurers to their targets of measurement, and composing them in novel ways.

B. Kordy et al. (Eds.): GraMSec 2016, LNCS 9987, pp. 150–166, 2016.
DOI: 10.1007/978-3-319-46263-9_10

However, the resulting layered attestations are typically more complex and challenging to analyze. Given a target system, what set of evidence should an appraiser request? What extra guarantees are provided if it receives integrity evidence of the measurers themselves? Does the order in which the measurements are taken matter?

This paper begins to tame the complexity surrounding attestations of these layered systems. We provide a formal model of layered measurement and attestation systems that abstracts away the underlying details of the measurements and focuses on the causal relationships among component corruption and measurement.

Limitations of Measurement. Our starting point for this paper is the recognition of the fact that measurement cannot *prevent* corruption; at best, measurement only *detects* corruption. In particular, the runtime corruption of a component can occur even if it is launched in a known good state. An appraiser must therefore always be wary of the gap between the time a component is measured and the time at which a trust decision is made. If the gap is large then so is the risk of a time-of-check-to-time-of-use (TOCTOU) attack in which an adversary corrupts a component during the critical time window to undermine the trust decision. A successful measurement strategy will limit the risk of TOCTOU attacks by ensuring the time between a measurement and a security decision is sufficiently small. The appraiser can then conclude that if the measured component is currently corrupted, it must be because the adversary performed a *recent* attack.

Shortening the time between measurement and security decision, however, is effective only if the measurement component can be trusted. By corrupting the measurer, an adversary can lie about the results of measurement making a corrupted target component appear to be in a good state. This affords the adversary a much larger window of opportunity to corrupt the target. The corruption no longer has to take place in the small window between measurement and security decision because the target can already be corrupted at the time of (purported) measurement. However, in a typical layered system design, deeper components such as a measurer have greater protections making it harder for an adversary to corrupt them. This suggests that to escape the burden of performing a *recent* corruption, an adversary should have to pay the price of corrupting a *deep* component.

Formal Model of Measurement and Attestation. With this in mind, our first main contribution is a formal model designed to aid in reasoning about what an adversary must do in order to defeat a measurement and attestation strategy. Rather than forbid the adversary from performing TOCTOU attacks in small windows or from corrupting deep components, we provide results that help to characterize and confine where such undesirable adversary actions must occur if the adversary is to corrupt a component without detection. Thus our model explicitly allows an adversary to corrupt (and repair) arbitrary system components at any time.

The model also features a true concurrency execution semantics which allows us to reason more directly about the causal effects of corruptions on the outcomes of measurement without having to reason about unnecessary interleavings of events. An important side benefit of this semantics is that it admits a natural, graphical representation that helps an analyst quickly understand the causal relationships between events of an execution. This pairs nicely with our analysis method based on characterizing executions consistent with some hypotheses, because it allows an analyst to quickly evaluate these executions without having to specify in advance a particular security goal.

Strategy for Measurement. We demonstrate the utility of this formal model by validating the effectiveness of an important strategy for measurement. An intuition manifest in much of the literature on measurement and attestation is that trust in a system should be based on a bottom-up chain of measurements starting with a hardware root of trust for measurement. This is the core idea behind trusted boot processes, in which one component in the boot sequence measures the next component before launching it. Theorem 1, which we refer to as the "recent or deep" theorem, validates this common intuition and characterizes exactly what an adversary must do to defeat such bottom-up measurement strategies. It roughly says the following:

> If a system has measured deeper components before more shallow ones, then the only way for the adversary to corrupt a component t without detection is either by *recently* corrupting one of t's dependencies, or else by corrupting a component even *deeper* in the system.

Paper Structure. The paper is structured as follows. We motivate our intuitions and informally introduce our model in Sect. 2. In Sect. 3 we formally define the systems of study and their executions. In Sect. 4 we prove some important facts about executions. We also define bottom-up measurement strategies and prove they confine adversary corruptions to be either recent or deep. Section 5 discusses some relevant related work. Finally, we conclude in Sect. 6.

2 Motivating Examples of Measurement

Consider an enterprise that would like to ensure that systems connecting to its network provide a fresh system scan by the most up-to-date virus checker. The network gateway should ask systems to perform a system scan on demand when they attempt to connect. We may suppose the systems all have some component A_1 that is capable of accurately reporting the running version of the virus checker. Because this enterprise values high assurance, the systems also come equipped with another component A_2 capable of measuring the runtime state of the kernel. This is designed to detect any rootkits that might try to undermine the virus checker's system scan. We may assume that A_1 and A_2 are both measured by a hardware root of trust for measurement (rtm) as part of a secure boot process. Thus, the architecture for systems in this enterprise might look

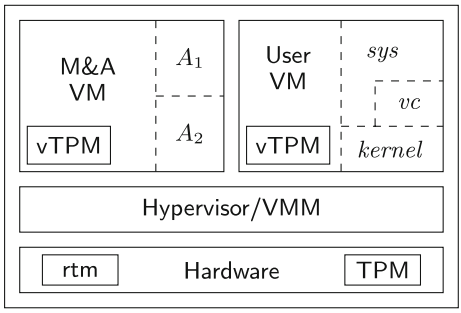


Fig. 1. Example measurement system.

something like Fig. 1 in which the (virtual) trusted platform modules ((v)TPMs) serve to store and report the measurement values to a remote appraiser.

We are thus interested in a system consisting of the following components: $\{sys, vc, ker, A_1, A_2, \mathsf{rtm}\}$, where *sys* represents the collective parts of the system scanned by the virus checker *vc*, and *ker* represents the kernel. Based on the scenario described above, we may be interested in the following set of measurement events

$$\{\mathsf{ms}(\mathsf{rtm}, A_1), \mathsf{ms}(\mathsf{rtm}, A_2), \mathsf{ms}(A_1, vc), \mathsf{ms}(A_2, ker), \mathsf{ms}_{ker}(vc, sys)\}$$

where $\mathsf{ms}_C(o_1, o_2)$ represents the measurement of o_2 by o_1 while C provides the runtime context. These measurement events generate the raw evidence that the network gateway can use to make a determination as to whether or not to admit the system to the network.

If any of the measurements indicate a problem, such as a failed system scan, then the gateway has good reason to believe it should deny the system access to the network. But what if all the evidence it receives looks good? How confident can the gateway be that the version and signature files are indeed up to date? The answer will depend on the order in which the evidence was gathered. The problem of determining the order in which measurements were taken given a set of signed quotes from (v)TPMs is addressed in [12]. In what follows, we assume the appraiser has some way of accurately determining the order in which measurements are taken. To get some intuition for why the order of measurement matters, consider the three different specifications pictured in Fig. 2 (in which time flows from top to bottom) for how to order the measurements. (The bullet after the first three events does not represent a separate event. It is inserted only for visible legibility, to avoid crossing arrows, so that each of the events on the top row occurs before both events on the next row down.)

Specification S_1 ensures that both *vc* and *ker* are measured before *vc* runs its system scan. Specifications S_2 and S_3 each relax one of those ordering requirements. Let's now consider some executions that respect the order of measurements in each of these specifications in which the adversary manages to avoid detection.

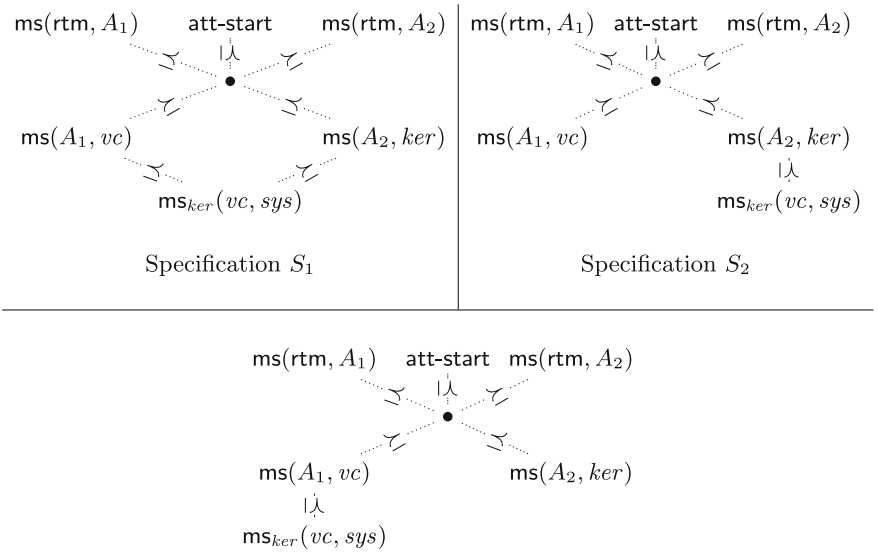


Fig. 2. Three orders for measurement

Execution E_1 of Fig. 3 is compatible with Specification S_1. The adversary manages to corrupt the system by installing some user-space malware sometime in the past. If we assume the up-to-date virus checker is capable of detecting this malware, then the adversary must corrupt either *vc* or *ker* before the virus scan represented by $\mathsf{ms}_{ker}(vc, sys)$. That is, either a corrupted *vc* will lie about the results of measurement, or else a corrupted *ker* can undermine the integrity of the system scan, for example, by hiding the directory containing the malware from *vc*. In the case of E_1, the adversary corrupts *vc* in order to lie about the results of the system scan, but it does so after $\mathsf{ms}(A_1, vc)$ in order to avoid detection by this measurement event.

In Execution E_2, which is consistent with Specification S_2, the adversary is capable of avoiding detection while corrupting *vc* much earlier. The system scan $\mathsf{ms}_{ker}(vc, sys)$ is again undermined by the corrupted *vc*. Since *vc* will also be measured by A_1, the adversary has to restore *vc* to an acceptable state before $\mathsf{ms}(A_1, vc)$. Execution E_3 is analagous to E_2, but the adversary corrupts *ker* instead of *vc*, allowing it to convince the uncorrupted *vc* that the system has no malware. Since Specification S_3 allows $\mathsf{ms}(A_2, ker)$ to occur after the system scan, the adversary can leverage the corrupted *ker* to lie about the scan results, but must restore *ker* to a good state before it is measured.

Execution E_1 is ostensibly harder to achieve for the adversary than either E_2 or E_3, because the adversary has to work quickly to corrupt *vc during* the attestation. In E_2 and E_3, the adversary can corrupt *vc* and *ker* respectively at any time in the past. He still must perform a quick restoration of the corrupted component during the attestation, but there are reasons to believe this may be

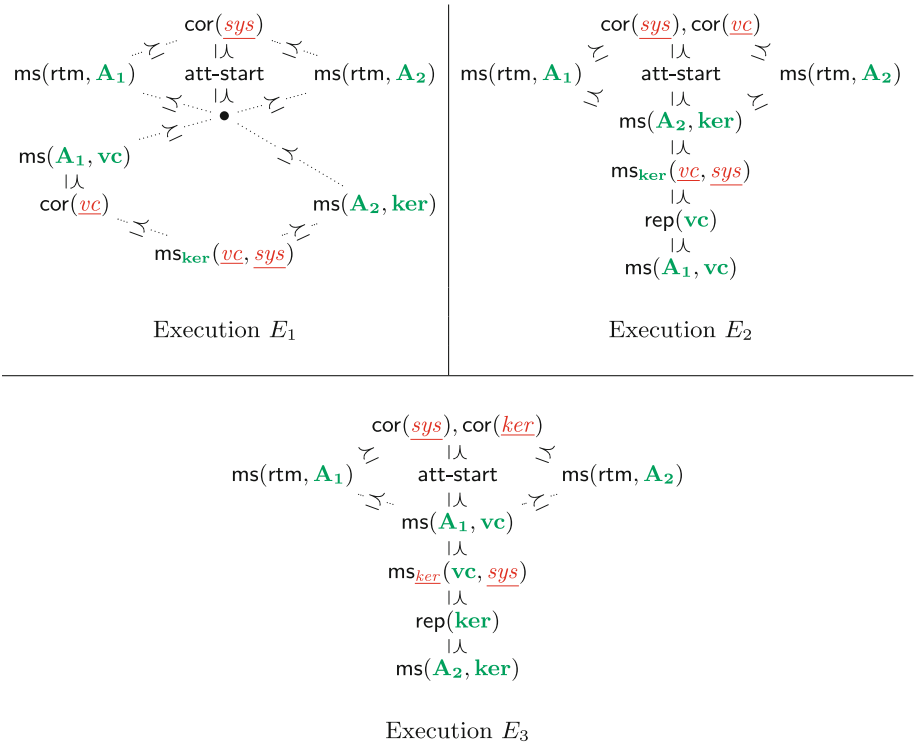


Fig. 3. Three system executions

easier than corrupting the component to begin with. The results of this paper provide a way of characterizing where and when adversary actions must occur in order to avoid detection by measurement. This leads to a result that any execution consistent with S_1 in which the adversary corrupts sys without detection forces the adversary to perform either a recent or a deep corruption.

3 Measurement Systems

In this section we formalize the intuitions we used for the examples in the previous section.

System Architecture. We start by describing the core types of dependencies that make a system layered.

Definition 1 (Measurement Systems). *We define a* measurement system *to be a tuple* $\mathcal{MS} = (O, M, C)$*, where* O *is a set of objects (e.g. software components) with a distinguished element* rtm*.* M *and* C *are binary relations on* O*. We call*

M the measures *relation, and*
C the context *relation.*

We say M is rooted *when for every $o \in O \backslash \{\mathsf{rtm}\}$, $M^+(\mathsf{rtm}, o)$, where M^+ is the transitive closure of M.*

M represents who can measure whom, so that $M(o_1, o_2)$ iff o_1 can measure o_2. rtm is the root of trust for measurement. For this reason we henceforth always assume M is rooted and M^+ is acyclic (i.e. $\neg M^+(o, o)$ for any $o \in O$). This guarantees that every object can potentially trace its measurements back to the root of trust, and there are no measurement cycles. As a consequence, rtm cannot be the target of measurement, i.e. for rooted, acyclic M, $\neg M(o, \mathsf{rtm})$ for any $o \in O$. The relation C represents the kind of dependency between *ker* and *vc* in the example above in which one object provides a clean runtime context for another. Thus, $C(o_1, o_2)$ iff o_1 contributes to maintaining a clean runtime context for o_2. (C stands for context.) We henceforth always assume C is transitive (i.e. if $C(o_1, o_2)$ and $C(o_2, o_3)$ then $C(o_1, o_3)$) and acyclic. This means that no object (transitively) relies on itself for its own clean runtime context.

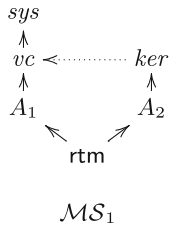
Given an object $o \in O$ we define the measurers of o to be $M^{-1}(o) = \{o' \mid M(o', o)\}$. We similarly define the context for o to be $C^{-1}(o)$. We extend these definitions to sets in the natural way.

We additionally assume $M \cup C$ is acyclic. This ensures that the combination of the two dependency types does not allow an object to depend on itself. Such systems are stratified, in the sense that we can define an increasing set of dependencies as follows.

$$D^1(o) = M^{-1}(o) \cup C^{-1}(M^{-1}(o))$$
$$D^{i+1}(o) = D^1(D^i(o))$$

So $D^1(o)$ consists of the measurers of o and their context. As we will see later, $D^1(o)$ represents the set of components that must be uncompromised in order to trust the measurement of o.

We can represent measurement systems pictorially as a graph whose vertices are the objects of $\mathcal{MS}$ and whose edges encode the M and C relations. We use the convention that $M(o_1, o_2)$ is represented by a solid arrow from o_1 to o_2, while $C(o_1, o_2)$ is represented by a dotted arrow from o_1 to o_2. The representation of the system described in Sect. 2 is shown in Fig. 4.



$\mathcal{MS}_1$

Fig. 4. Graphical representation of an example measurement system.

Events, Executions, and Outputs. The components $o \in O$ and the adversary on this system perform actions. In particular, objects can measure each other and the adversary can corrupt and repair components in an attempt to influence the outcome of future measurement actions. Additionally, an appraiser has the ability to inject a random nonce $n \in \mathcal{N}$ into an attestation in order to control the recency of events.

Definition 2 (Events). *Let $\mathcal{MS}$ be a target system. An event for $\mathcal{MS}$ is a node e labeled by one of the following.*

a. A measurement event *is labeled by* $\mathsf{ms}_{C^{-1}(o_2)}(o_2, o_1)$ *such that* $M(o_2, o_1)$*. We say such an event* measures o_1*, and we call* o_1 *the* target *of* e*. When* $C^{-1}(o_2)$ *is empty we omit the subscript and write* $\mathsf{ms}(o_2, o_1)$*.*
b. An adversary event *is labeled by either* $\mathsf{cor}(o)$ *or* $\mathsf{rep}(o)$ *for* $o \in O \setminus \{\mathsf{rtm}\}$*.*
c. The attestation start *event is labeled by* $\mathsf{att\text{-}start}$*.*

When an event e is labeled by ℓ we will write $e = \ell$. We will often refer to the label ℓ as an event when no confusion will arise.

An event e touches *o iff o is an argument to the label of e.*

The $\mathsf{att\text{-}start}$ event serves to bound events in time. It represents the choice by the appraiser of a random nonce. Typically the measurements will be cryptographically bound to this nonce before sending them back to the appraiser. In this way, the appraiser will know that anything occurring after this event can reasonably be said to occur "recently". Regarding the measurement events, the rtm is typically responsible for measuring components at boot-time. All other measurements are load-time or runtime measurements of one component in O by another. Adversary events represent the corruption ($\mathsf{cor}(\cdot)$) and repair ($\mathsf{rep}(\cdot)$) of components. Notice that we have excluded rtm from corruption and repair events. This is not because we assume the rtm to be immune from corruption, but rather because all the trust in the system relies on the rtm: Since it roots all measurements, if it is corrupted, none of the measurements of other components can be trusted.

As we saw in the motivational examples, an execution can be described as a partially ordered set (poset) of these events. We choose a partially ordered set rather than a totally ordered set because the latter unnecessarily obscures the difference between *causal* orderings and *coincidental* orderings. However, due to the causal relationships between components, we must slightly restrict our partially ordered sets in order to make sense of the effect that corruption and repair events have on measurement events. To that end, we next introduce a sensible restriction to these partial orders.

A poset is a pair $(E, \prec)$, where E is any set and $\prec$ is a transitive, acyclic relation on E. When no confusion arises, we often refer to $(E, \prec)$ by its underlying set E and use $\prec_E$ for its order relation. Given a poset $(E, \prec)$, let $e{\downarrow} = \{e' \mid e' \prec e\}$, and $e{\uparrow} = \{e' \mid e \prec e'\}$. Given a set of events E, we denote the set of adversary events of E by $adv(E)$ and the set of measurement events by $meas(E)$.

Let $(E, \prec)$ be a partially ordered set of events for $\mathcal{MS} = (O, M, C)$ and let $(E_o, \prec_o)$ be the substructure consisting of all and only events that touch o. We

say $(E, \prec)$ is *adversary-ordered* iff for every $o \in O$, $(E_o, \prec_o)$ has the property that if e and e' are incomparable events, then neither e nor e' are adversary events.

Lemma 1. *Let $(E, \prec)$ be a finite, adversary-ordered poset for $\mathcal{MS}$, and let $(E_o, \prec_o)$ be its restriction to some $o \in O$. Then for any non-adversarial event $e \in E_o$, the set $adv(e{\downarrow}) \cap E_o$ is either empty or has a unique maximal element.*

Proof. Since $(E, \prec)$ is adversary-ordered, $adv(E_o)$ is partitioned by $adv(e{\downarrow})$ and $adv(e{\uparrow})$. Suppose $e{\downarrow}$ is not empty. Then since E_o is finite, it has at least one maximal element. Suppose e' and e'' are distinct maximal elements. Thus they must be $\prec_o$-incomparable. However, since $(E, \prec)$ is adversary-ordered, either $e' \prec_o e''$ or $e'' \prec_o e'$, yielding a contradiction. □

Definition 3 (Corruption State). *Let $(E, \prec)$ be a finite, adversary-ordered poset for $\mathcal{MS}$. For each event $e \in E$ and each object o the* corruption state *of o at e, written $cs(e, o)$, is an element of $\{\bot, \mathsf{r}, \mathsf{c}\}$ and is defined as follows. $cs(e, o) = \bot$ iff $e \notin E_o$. Otherwise, we define $cs(e, o)$ inductively:*

$$cs(e,o) = \begin{cases} \mathsf{c} & : e = \mathsf{cor}(o) \\ \mathsf{r} & : e = \mathsf{rep}(o) \\ \mathsf{r} & : e \in meas(E) \wedge adv(e{\downarrow}) \cap E_o = \emptyset \\ cs(e', o) & : e \in meas(E) \wedge e' \text{ maximal in } adv(e{\downarrow}) \cap E_o \end{cases}$$

When $cs(e, o)$ takes the value c we say o is corrupt *at e; when it takes the value r we say o is* uncorrupt *or* regular *at e; and when it takes the value $\bot$ we say the corruption state is* undefined.

We now define what it means to be an execution of a measurement system.

Definition 4 (Executions). *An* exectuion *of a measurement system $\mathcal{MS}$ is any finite, adversary-ordered poset E for $\mathcal{MS}$.*

Since executions are finite and adversary-ordered, for every o, we can always determine the corruption state of o at every event e touching o. We can therefore use these corruption states to determine the outputs of measurements. Abstractly, we assume that for each target of measurement o_t, every measurer of o_t outputs values within some set $\mathcal{MV}(o)$.

The question of measurement accuracy is complicated because there are two primary sources of inaccuracy. First, the measurer may not produce values that strongly correlate to the corruption state of the target. For example, an asset inventory tool may only output the version numbers of software installed, and this cannot detect undiscovered (and thus unpatched) vulnerabilities. Second, the appraiser is ultimately the one to *interpret* the output. That is, the appraiser partitions $\mathcal{MV}(o)$ into $\mathcal{G}(o)$ and $\mathcal{B}(o)$. The first set $\mathcal{G}(o)$ represents measurement values the appraiser believes represent an uncompromised component, while $\mathcal{B}(o)$ are those values the appraiser believes represent a corrupted component. Thus

the composition of measurement output with appraiser interpretation forms a classifier for the corruption state of the target whose false positive and negative rates depend on both the measurer and the appraiser.

In this work, to simplify the analysis, we assume there are no false positives or negatives as long as the measurer and its context are uncorrupted. However, we assume a corrupted measurer (or its context) can always convince the appraiser that the target of measurement is uncorrupted.

Assumption 1 (Measurement Accuracy). Let $\mathcal{G}(o)$ and $\mathcal{B}(o)$ be a partition for $\mathcal{MV}(o)$. Let $e = \mathsf{ms}(o_2, o_1)$. The *output* of e, written $out(e)$, is defined as follows.

$$out(e) = \begin{cases} v \in \mathcal{B}(o_1) & cs(e, o_1) = \mathsf{c} \text{ and } \forall o \in \{o_2\} \cup C^{-1}(o_2) \,.\, cs(e, o) = \mathsf{r} \\ v \in \mathcal{G}(o_1) & \text{otherwise} \end{cases}$$

If $out(e) \in \mathcal{B}(o_1)$ we say e *detects a corruption.* If $out(e) \in \mathcal{G}(o_1)$ but $cs(e, o_1) = \mathsf{c}$, we say the adversary *avoids detection at* e.

Given an execution E, Assumption 1 says we can always determine the appraiser's classification. However, it can also be used to infer the corruption states of some components given the corruption states of others and the classification. That is, suppose we know the adversary avoids detection at $e = \mathsf{ms}_{C^{-1}(o)}(o, o_t)$. Then we can conclude that at least one member of $\{o\} \cup C^{-1}(o)$ is corrupt at e. This is an important inference for our main result.

One can imagine weakening Assumption 1 to account for imperfect classification. For example, it would be interesting to perform a probabilistic analysis accounting for false positive and negative rates. However, we leave such investigations for future work.

Although executions always allow us to infer the corruption state of components at events and the outputs of measurements, this only holds if we have accounted for all the adversary actions. The main goal of our framework is to allow an appraiser to infer what adversary events must have occurred and when, assuming some basic facts about an execution. To that end we introduce specifications, which formalize the partial knowledge an appraiser has about the execution of the system.

Definition 5 (Specifications). *A* specification *for measurement system $\mathcal{MS}$, is a finite adversary-ordered poset S with some (possibly empty) set of assumptions about measurement events regarding*

1. *the corruption states of some of their arguments, or*
2. *the output classification ($\mathcal{G}(o)$ or $\mathcal{B}(o)$).*

A specification S admits *execution E iff there is an injective, label-preserving map of partial orders $\alpha : S \to E$ preserving assumptions on corruption states and output classifications. The set of all executions admitted by S is denoted $\mathcal{E}(S)$.*

We can annotate our diagrams in order to convey the assumptions about measurement events. In particular, we underline the corrupted components and display them in red, and we use bold typeface for the uncorrupted components and display them in green. We can also annotate measurements with a ✓ (resp. ✗) to indicate the output is in $\mathcal{G}(o)$ (resp. $\mathcal{B}(o)$). This allows for a quite compact graphical representation of the relevant information as seen, for example, in Figs. 3 and 5. We omit these visual annotations from executions when the diagram is too cluttered because they can be inferred.

4 Confining Adversary Behavior

In this section we explore what an appraiser can infer about $\mathcal{E}(S)$, given a specification S. In particular, we are interested in characterizing the ways in which an adversary can corrupt a component without the execution giving any indication of corruption. We thus start with some simple general results about executions, before presenting our main theorem.

Lemma 2. *Let E be an execution of $\mathcal{MS}$, and let e be an event of E touching o. If $cs(e, o) = \mathsf{c}$, then there is a most recent corruption event $e' \preceq e$.*

Proof. This follows immediately from Lemma 1 and Definition 3. □

This lemma is useful for inferring the existence of a corruption event *before* some given event e. However, since we are also interested in the recency of corruption, we would like to infer the existence of a corruption event *after* a given event. The following lemma allows us to do just that.

Lemma 3. *Let E be an execution of $\mathcal{MS}$, and let $e_1 \prec e_2$ be measurement events touching o such that $cs(e_1, o) = \mathsf{r}$ (resp. c) and $cs(e_2, o) = \mathsf{c}$ (resp. r). Then there exists a corruption (resp. repair) event e' such that $e_1 \prec e' \prec e_2$.*

Proof. Let $A_1 = adv(e_1{\downarrow}) \cap E_o$ and $A_2 = adv(e_2{\downarrow}) \cap E_o$. Since $e_1 \prec e_2$, $A_1 \subseteq A_2$. By Lemma 1, A_2 is either empty or has a unique maximum. However, it can't be empty because then A_1 would also be empty and Definition 3 would imply that $cs(e_1, o) = cs(e_2, o) = \mathsf{r}$, contrary to the hypothesis. So let e' be the unique maximum of A_2. Since $(E, \prec)$ is adversary ordered, either $e' \prec e_1$ or $e_1 \prec e'$. In the first case, e' would also be a maximum of A_1 since $A_1 \subseteq A_2$. But this would imply $cs(e_1, o) = cs(e', o) = cs(e_2, o)$ violating our assumption. Thus $e_1 \prec e' \prec e_2$, and since the corruption state of o is different at e_1 and e_2, e' must change the corruption state. □

We now turn to a formalization of the rule of thumb at the end of Sect. 2. In particular, we characterize what a bottom-up measurement strategy guarantees. That is, if whenever o_1 depends on o_2 we measure o_2 before measuring o_1, then we seek to understand the constraints this puts on the adversary actions in order to avoid detection. For this discussion we fix a target system $\mathcal{MS}$. Recall that $D^1(o)$ represents the measurers of o and their runtime context.

Definition 6. *A measurement event* $e = \mathsf{ms}(o_2, o_1)$ *in execution* E *is* well-supported *iff either*

i. $o_2 = \mathsf{rtm}$, *or*
ii. for every $o \in D^1(o_1)$, *there is a measurement event* $e' \prec_E e$ *such that* o *is the target of* e'.

When e *is well-supported, we call the set of* e' *from Condition ii above the* support *of* e. *An execution* E measures bottom-up *iff each measurement event* $e \in E$ *is well-supported.*

Theorem 1 (Recent or Deep). *Let* E *be an execution with well-supported measurement event* $e = \mathsf{ms}(o_1, o_t)$ *where* $o_1 \neq \mathsf{rtm}$. *Suppose that* E *detects no corruptions. If the adversary avoids detection at* e, *then either*

1. *there exist* $o \in D^1(o_t)$ *and* $o' \in M^{-1}(o)$ *such that* $\mathsf{ms}(o', o) \prec_E \mathsf{cor}(o) \prec_E e$
2. *there exists* $o \in D^2(o_t)$ *such that* $\mathsf{cor}(o) \prec_E e$.

Proof. Since the adversary avoids detection at e, o_t is corrupt at e, and by Assumption 1, there is some $o \in \{o_1\} \cup C^{-1}(o_1) \subseteq D^1(o_t)$ that is also corrupt at e. Also, since e is well-supported, and $o_1 \neq \mathsf{rtm}$, we know there exists $e' = \mathsf{ms}(o', o)$ with $e' \prec_E e$. We now take cases on $cs(e', o)$.

If $cs(e', o) = \mathsf{r}$ then we apply Lemma 3 to conclude there must be a corruption $\mathsf{cor}(o)$ between e' and e satisfying Clause 1.

If $cs(e', o) = \mathsf{c}$, then since E detects no corruptions, then by Assumption 1, there must be some $o^* \in \{o'\} \cup C^{-1}(o') \subseteq D^2(o_t)$ such that $cs(e', o^*) = \mathsf{c}$. We then apply Lemma 2 to infer there must be a previous corruption $\mathsf{cor}(o^*) \prec_E e' \prec_E e$ satisfying Clause 2. □

This theorem says, roughly, that if measurements indicate things are good when they are not, then there must either be a recent corruption or a deep corruption. This tag line of "recent or deep" is particularly apt if (1) the system dependencies also reflect the relative difficulty for an adversary to corrupt them, and (2) the higher level measurements occur after the att-start event. By ordering the measurements so that more robust ones are measured first, it means that for an adversary to avoid detection for an easy compromise, he must have compromised a measurer recently (i.e. since it itself was measured and typically after the att-start event), or else, he must have previously (though not necessarily recently) compromised a more robust component. In this way, the measurement of a component can raise the bar for the adversary. If, for example, a measurer sits in a privileged location outside of some VM containing a target, it means that the adversary would also have to break out of the target VM and compromise the measurer to avoid detection. The skills and time necessary to perform such an attack are much greater than simply compromising the end target.

Let's illustrate this result in the context of the example of Sect. 2. Consider the specification S' in Fig. 5, which is like specification S_1 from Fig. 2, except that it is annotated with more assumptions in order to satisfy all the hypotheses of Theorem 1. Since these facts are (by definition) preserved by homomorphisms

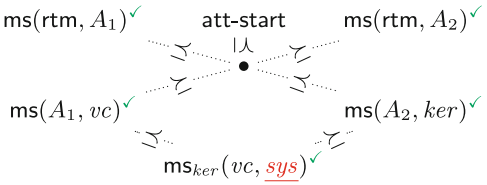


Fig. 5. Specification S'.

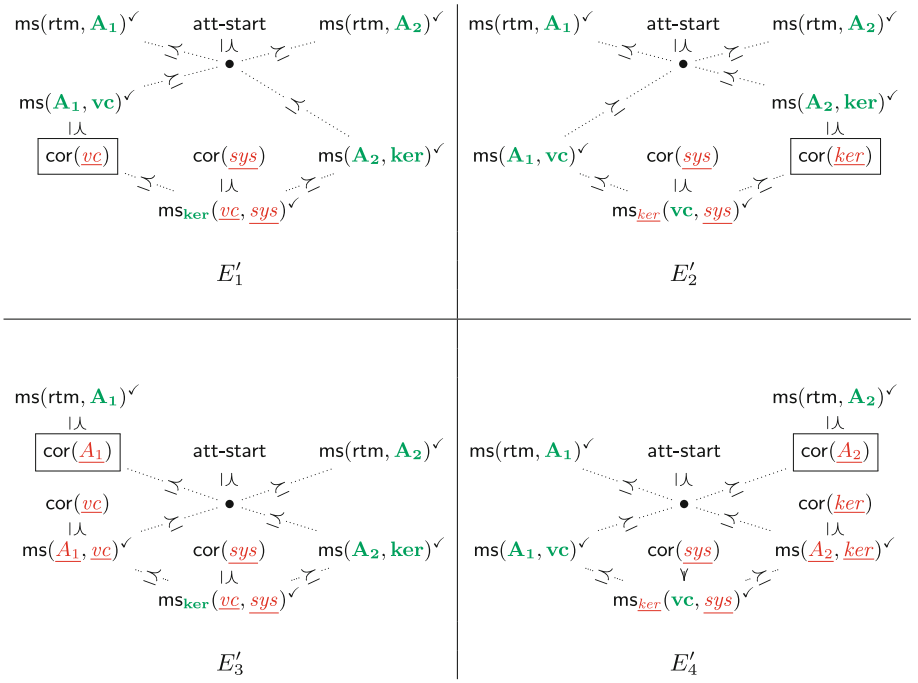


Fig. 6. Executions in $\mathcal{E}(S')$ that do not detect corruption of sys.

$\alpha : S \to \mathcal{E}(S)$, all executions in $\mathcal{E}(S)$ must satisfy them too. Execution E_1 illustrates an example of the first clause of the conclusion being satisfied. There is a "recent" corruption of vc in the sense that vc is corrupted after it is measured. Since the measurement of vc occurs after the start of the attestation, this is truly recent, in that the adversary has very little time to work. The appraiser can control this by ensuring that attestations time out after some fixed amount of time.

Theorem 1 also indicates other possible executions in which the adversary can undetectably corrupt sys. There could be a recent corruption of vc, or else there could be some previous corruption of either A_1 or A_2. All the various options are shown in Fig. 6 in which the corruption events guaranteed by the theorem

are boxed. Our theorem allows us to know that these executions essentially characterize all the cases in which a corrupted *sys* goes undetected.

Automation. The analysis above was performed by hand. It would be possible to automate the reasoning steps codified by Assumption 1 and Lemmas 2 and 3. An automated algorithm would have to implement the process of building an execution consistent with (a) the reasoning principles laid out above, and (b) the initial assumptions given by a specification. This is an instance of the more general problem of model finding. That is, given a logical theory and a set of assumptions about a structure, model finding techniques can produce a set of models consistent with the theory and the assumptions.

General purpose tools have been developed that can automate the model finding process [8,13]. We have not yet attempted to use these tools for the analysis of measurement systems. As such, it is unclear if the general algorithms they use will yield efficient analyses, or if they will suffer from combinatorial state space explosions. We have also not investigated the computational complexity of finding a minimal set of executions consistent with a given specification. This would be an interesting question for future work. It is worth noting, however, that Theorem 1 obviates the need to perform such case-by-case analyses when the specification in question is already bottom-up. The value of automated algorithms is greatest when an analyst is unable to apply Theorem 1, for example if the measurement system has cycles in $M \cup C$. Such cyclic dependencies are surprisingly common in production systems because the necessary isolation provided by hardware virtualization is still relatively rare. Thus we believe an automated tool implementing the reasoning principles presented in this paper would be a valuable asset for the analysis of layered attestations.

5 Related Work

There has been much research into measurement and attestation. While a complete survey is infeasible for this paper, we mention the most relevant highlights in order to describe how the present work fits into the larger context of research in this area.

Much of the early work on measurement and attestation was focused on techniques for measuring low-level components that make up a trusted computing base (TCB). These ideas have matured into implementations such as Trusted Boot [11]. Recognizing that many security failures cannot be traced back to the TCB, Sailer et al. [14] proposed an integrity measurement architecture (IMA) in which each application is measured (by hashing its code) before it is launched. More recently, there has been work trying to identify and measure dynamic properties of system components in order to create a more comprehensive picture of the runtime state of a system [5,9,10,15]. All these efforts try to establish what evidence is useful for inferring system state relevant to security decisions. The present work takes for granted that such special purpose measurements can be taken and that they will accurately reflect the system state. Rather, our focus is on developing principles for how to combine a variety of these measurers in

a layered attestation. We envision a system designer choosing the measurement capabilities that best suit her needs and using our work to ensure an appraiser can trust the integrity of the result.

In [4], Datta et al. introduce a formalism that accounts for actions local to the target machine as well as network events such as sending and receiving messages. Although they give a very careful treatment of the effect of a corrupted component on an attestation, their work differs in two key ways. First, the formalism represents many low-level details making their proof rather complex, sometimes obscuring the underlying principles. Second, their framework only accounts for static corruptions, while ours is specifically designed around the possibility of dynamic corruption and repair of system components.

Cabuk et al. [1] have proposed an architecture designed to support layered platforms with hierarchical dependencies. It introduces trusted software into the TCB as a software-based root of trust for measurement (SRTM). Although they explain how measurements by the SRTM integrate with the chain of measurements stored in a TPM, they do not study the effect corruptions of various components have on the outcome of attestations. In [2], Coker et al. identify five guiding principles for designing an architecture to support remote attestation. They also describe the design of a (layered) virtualized system based on these principles, although there does not appear to be a publicly available implementation at the time of writing. Of particular interest is a section that describes a component responsible for managing attestations. The emphasis is on the mechanics of selecting measurement agents by matching the evidence they can generate to the evidence requested by an appraiser. There is no discussion or advice regarding the relative order of measurements or the creation of an evidence bundle to reflect the order. More recently, modular attestation frameworks instantiating [2]'s principles have been implemented [3,6,7]. These are integrated frameworks that offer plug-and-play capabilities for measurement and attestation for specific usage scenarios. It is precisely these types of systems (in implementation or design) to which our analysis techniques would be most useful. We have not been able to find a discussion of the potential pitfalls of misconfiguring these complex systems. Our work should be able to help guide the configuration of such systems and analyze particular attestation scenarios for each architecture.

Finally, we mention a companion paper to the present work [12]. While the present work helps us characterize how a given measurement order can confine the actions of an adversary, it does not address the question of how a remote appraiser learns the order and results of measurements. This is typically done by storing the evidence in a trusted platform module (TPM) and quoting the results. In [12] it is shown that some methods of using a TPM allow an adversary to bypass Theorem 1 by convincing the appraiser that measurements were taken bottom-up when in fact they were not. The main result is a proposed method for storing and reporting evidence using TPMs ensuring that if an adversary successfully avoids the hypothesis of Theorem 1, then he must nonetheless submit himself to its conclusions.

6 Conclusion

In this paper we have developed a formalism for reasoning about measurement in layered systems. Within this framework we have demonstrated some reusable principles for inferring properties of adversary actions in executions, and we have applied those principles to justify the intuition (pervasive in the literature on measurement and attestation) that it is important to measure a layered system from the bottom up (Theorem 1). Our model admits natural graphical representations of measurement systems, specifications and executions. We believe this graphical representation makes the formalism more intuitive to use, as it allows an analyst to apply her intuitions more immediately to the diagrams.

We believe the model is also relatively extensible in that further types of components and events could be added without disrupting the current results. Indeed, in our companion paper [12], we add events for interacting with a Trusted Platform Module (TPM) in order to perform a more complete analysis of how not just the outcomes of measurements but also their order can be conveyed to a remote appraiser. This is a crucial part of an end-to-end analysis as the appraiser cannot directly observe the order of measurements.

In future work, we would like to consider relaxing Assumption 1 to allow for some probabilistic errors in the classification of measurement targets. Disentangling how much of those errors is due to inaccurate measurement and how much to inaccurate interpretation of the measurement could yield more fine grained results that allow a more nuanced risk decision on the part of the appraiser. We also believe the model could benefit from tool support. The basic problem of discovering adversary actions given assumptions on an execution can be viewed as an instance of model finding. As such, tools such as [8,13] that have been developed for that purpose could be applicable here.

Acknowledgments. I would like to thank Pete Loscocco for suggesting and guiding the direction of this research. Many thanks also to Perry Alexander and Joshua Guttman. Their valuable feedback on during the formation of these ideas was invaluable. Thanks also to Sarah Helble and Aaron Pendergrass for lively discussions about implementations of measurement and attestation systems. Finally, I would like to thank the anonymous reviewers as well as the GraMSec participants for their insightful comments and suggestions for improving the paper.

References

1. Cabuk, S., Chen, L., Plaquin, D., Ryan, M.: Trusted integrity measurement and reporting for virtualized platforms. In: Chen, L., Yung, M. (eds.) INTRUST 2009. LNCS, vol. 6163, pp. 180–196. Springer, Heidelberg (2010)
2. Coker, G., Guttman, J.D., Loscocco, P., Herzog, A.L., Millen, J.K., O'Hanlon, B., Ramsdell, J.D., Segall, A., Sheehy, J., Sniffen, B.T.: Principles of remote attestation. Int. J. Inf. Secur. **10**(2), 63–81 (2011)
3. Intel Corporation: Open attestation. Accessed 16 Dec 2015

4. Datta, A., Franklin, J., Garg, D., Kaynar, D.K.: A logic of secure systems and its application to trusted computing. In: 30th IEEE Symposium on Security and Privacy (S&P 2009), Oakland, California, USA, 17–20 May 2009, pp. 221–236 (2009)
5. Davi, L., Sadeghi, A.-R., Winandy, M.: Dynamic integrity measurement, attestation: towards defense against return-oriented programming attacks. In: Proceedings of the 4th ACM Workshop on Scalable Trusted Computing, STC 2009, Chicago, Illinois, USA, 13 November 2009, pp. 49–54 (2009)
6. Fisher, C., Bukovick, D., Bourquin, R., Dobry, R.: SAMSON - Secure Authentication Modules. Accessed 16 Dec 2015
7. Trusted Computing Group. TCG Trusted Network Connect Architecture for Interoperability version 1.5 (2012)
8. Jackson, D.: Software Abstractions: Logic Language and Analysis, 2nd edn. MIT Press, Cambridge (2012)
9. Kil, C., Sezer, E.C., Azab, A.M., Ning, P., Zhang, X.: Remote attestation to dynamic system properties: towards providing complete system integrity evidence. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009, Estoril, Lisbon, Portugal, 29 June–2 July 2009, pp. 115–124 (2009)
10. Loscocco, P., Wilson, P.W., Pendergrass, J.A., McDonell, C.D.: Linux kernel integrity measurement using contextual inspection. In: Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, 2 November 2007, pp. 21–29 (2007)
11. Maliszewski, R., Sun, N., Wang, S., Wei, J., Qiaowei, R.: Trusted boot (tboot). Accessed 16 Dec 2015
12. Rowe, P.D.: Bundling evidence for layered attestation. In: Franz, M., Papadimitratos, P. (eds.) TRUST 2016. LNCS, vol. 9824, pp. 119–139. Springer, Heidelberg (2016). doi:10.1007/978-3-319-45572-3_7
13. Saghafi, S., Dougherty, D.J.: Razor: provenance and exploration in model-finding. In: 4th Workshop on Practical Aspects of Automated Reasoning (PAAR) (2014)
14. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004, pp. 223–238 (2004)
15. Wei, J., Calton, P., Rozas, C.V., Rajan, A., Zhu, F.: Modeling the runtime integrity of cloud servers: a scoped invariant perspective. In: Cloud Computing, Second International Conference, CloudCom 2010, Indianapolis, Indiana, USA, Proceedings, 30 November–3 December 2010, pp. 651–658 (2010)