6-4-2010

Virtual Container Attestation: Customized trusted containers for on-demand computing.

Katelin A. Bailey
Dartmouth College

Dartmouth Computer Science Technical Report TR2010-674

Virtual Container Attestation: Customized trusted containers for on-demand computing.

Katelin Bailey

Advisor Sean W. Smith

Department of Computer Science

Dartmouth College

Hanover, NH

katelin.a.bailey.10@alum.dartmouth.org

June 4, 2010

Abstract

In today's computing environment, data is moving to central locations and most computers are merely used to access the data. Today is the era of *cloud computing* and *distributed computing*, where users have control over neither data nor computation. As this trend continues there is an increasing frequency of mutually distrustful parties being forced to interact and share resources with each other in potentially dangerous situations.

Therefore, there is an urgent need for a means of creating trust between two entities, or at the very least providing some means of determining the trust level of a given machine.

Current approaches to the trust problem focus on various forms of isolation and attestation, but most have high overheads or are overly rigid in their requirements to users. I propose and implement an alternative solution which provides flexible, on-demand containers for untrusted applications, and enforcement of requested security properties. Together these provide assurance to the remote parties that the machines behave as required or are quickly shut down.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem

In the world of computing today, much of the vital interaction is done over the network, with machines we can neither see nor fully interact with. Sometimes we own and have root access over those machines, and sometimes we have no privileged access whatsoever. Regardless of the ownership we have, there are a number of occasions when knowing the state of the other machine is not only a positive but a necessity, as a means of determining whether we can trust the interactions performed remotely.

The level of knowledge required varies between users: one may wish only to interact with non-Windows machines, or machines with the latest security and anti-virus software, thus hopefully providing some assurance that they will not infect the client with a virus or other malware. At a more complicated level, however, the requirements can become quite subtle. "I want X amount of run time" or "I need access to files in this order" or even "These files should never be open at the same time". Even more complicated is when these requirements do not mesh well with the security concerns of the host: "This client needs to be able to interact over the internet, but I want to know what is going out". Given the flexibility of hardware, operating systems, and application-level software running on machines today, there are an almost infinite number of permutations and levels of inspection and trust within the machine.

Users, therefore, are left with a problem: they need to trust the computation of these machines, and in order to do so they must trust the computers themselves. In order to trust a computer, they must have some idea of state, but the possible states are too many to plausibly keep track of.

Trusted Computing is an oft-used term with many varied meanings. The context in which we employ it here is that of the Trusted Computing Group (TCG) [2]. Trusted Computing, as put forward by the TCG, is one method of approaching this trust problem, which starts from a small core of trusted hardware/software and expands that core via any of a number of methods. Doing so has a number of benefits: chief among them the concept that such an expansion provides a level of trust about the operation of the machine. Having either mutual trust or a means of evaluation allows untrusted parties to make informed decisions about if and to what degree they wish to interact with each other.

The common thread of Trusted Computing research leads people to use the *trusted core* to make measurements (or *audits*) of the computer state and attach various items to that state—whether those actions are secrets such as cryptographic keys or trust evaluations connecting to human ideas of safety. These connections provide additional assurance to a remote user. However, creating a useful audit of a system is difficult, because it must be accessible, meaningful, and useful to remote machines. Open machines currently have such flexibility in both software and hardware that determining what to audit is difficult, and the alternative of a closed and locked-down system is overly restrictive. (In this context, closed systems are those which are restricted to a fixed set of easily auditable and trustable hardware and software.)

There are several potential solutions to this problem. *Compartmented attestation* has been proposed, but the complexity of auditing such an approach is only slightly better than that of an open system [16, 18]. Compartmented attestation creates a sandbox for the program and controls the access using *Mandatory Access Control* (MAC) from SELinux. The MAC policies used to isolate each compartment, however, are complicated to create and proving one compartment on a system requires proof of the entire system: not a huge improvement over previous methods.

An alternative, *property-based attestation* allows the host to create a profile that describes the behavior of the system, rather than the system configuration [24, 12]. However, the vagueness of such an approach does not decrease the potential complexity to any appreciable degree, and may in fact increase it via the translation between system configuration and human-readable properties.

Virtualization has been explored as a means of limiting the necessary software to audit, however, the current solutions are neither flexible nor scalable, requiring a pre-fabricated virtual machine with its own OS and specific software [26, 10, 9, 12]. Alternatively, they may make some repeated attempts at creating an acceptable security setting through trial and error. While the former approach seems reasonable, the load of creating a full OS on top of the virtual machine is difficult to do and hard to verify. The latter approach is simply frustrating.

We discuss these and other approaches in more detail in section 1.2.

This thesis aims to create a combination of flexible trusted computing with OS-level virtualization as a means of creating compartmented attestation on demand. Virtualization not only provides a means of isolation, but it limits the software necessary to audit, and creates a limited set of routes into the compartment. The OpenSolaris zones which I use in this project not only provide a more lightweight structure for on-demand computing (by sharing the operating system), but they also provide tools such as DTrace and other run-time instrumentation which allow for virtual machine introspection and dynamic property enforcement. Together this allows the unification of several branches of Trusted Computing.

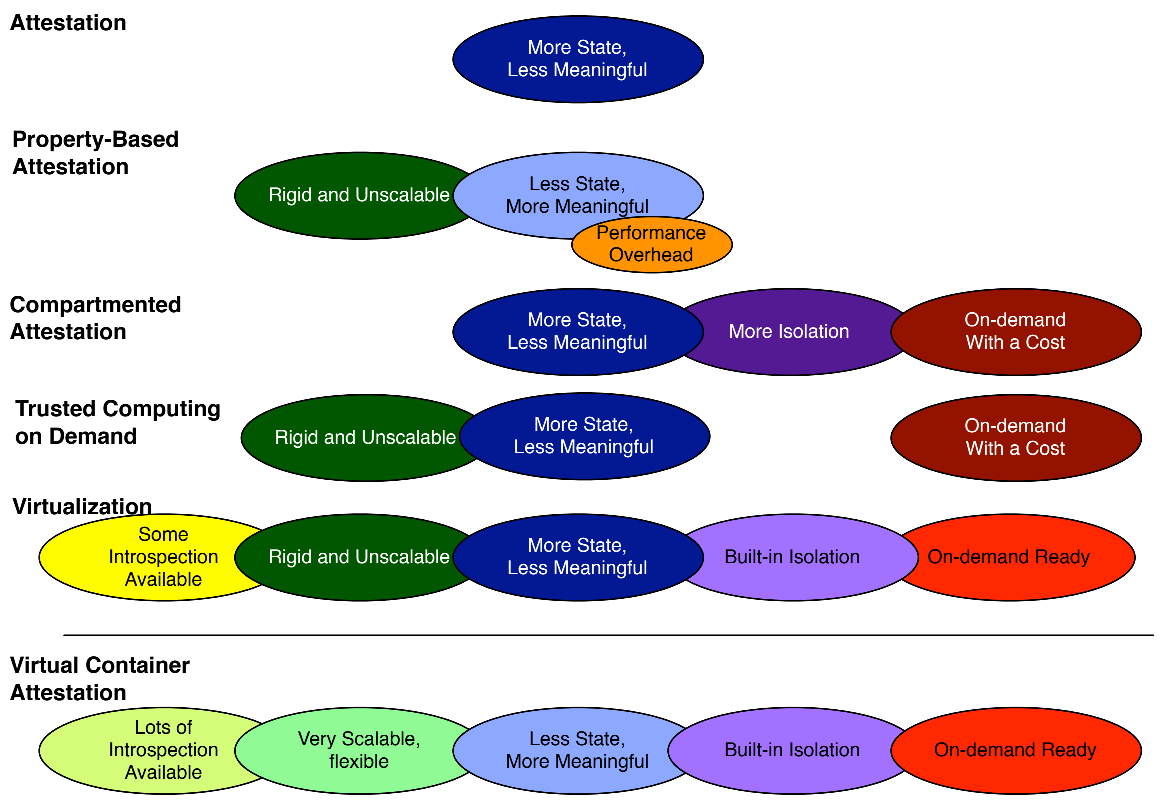

Attestation
More State, Less Meaningful
Property-Based Attestation
Rigid and Unscalable
Less State, More Meaningful
Performance Overhead
Compartmented Attestation
More State, Less Meaningful
More Isolation
On-demand With a Cost
Trusted Computing on Demand
Rigid and Unscalable
More State, Less Meaningful
On-demand With a Cost
Virtualization
Some Introspection Available
Rigid and Unscalable
More State, Less Meaningful
Built-in Isolation
On-demand Ready
Virtual Container Attestation
Lots of Introspection Available
Very Scalable, flexible
Less State, More Meaningful
Built-in Isolation
On-demand Ready

Figure 1.1: Summary of pros and cons of previous solutions.

1.2 Background on Trusted Computing

Trusted Computing is a vague concept which has historically been used in a variety of ways. For the purposes of this project, however, a computer can be "trusted" if there is some way to verify that it acts in the way it should, either currently or for some period of time. This verification must, to some degree, know the state of both hardware and software, for neither alone can provide assurance of the operation of the machine. In this vein, I have relied heavily upon the TCG's specifications for trusted computing in both hardware and software levels.

The difficulty behind trusted computing traditionally lies within a handful of areas:

1. What can we trust to evaluate the machine for us and determine trustworthiness?

2. What should we evaluate within the machine?

3. How we can we reconcile the difference between the automated and machine-accessible evaluation and a human-readable or intuitive trust judgements?

Although there are many approaches to trusted computing and numerous projects which address the questions outlined above, none of them unifies the answers to these problems in any optimal way.

Below, we take a moment to consider each approach.

1.2.1 Trusted Hardware

Trusted Hardware, being the lowest level of base assumption you can make, is often the go-to method for establishing a root of trust for the machine and subsequent operations. I also make use of trusted hardware—a *Trusted Platform Module*, to be specific—which is further discussed in section 3

1.2.2 Attestation

Attestation was one of the first attempts at answering the problems of trusted computing. It is the process by which a computer, workstation, or user attempts to verify the state of a machine. State may include hardware capabilities, operating system, installed software, or more complicated run-time statistics. The main way this is done is via some measurement of relevant hardware, software, and operating systems. However, determining what is actually necessary to evaluate is complicated, and frequently debated. Deciding what the evaluation will be applied to determines what level of detail we must attend to.

To provide a fully accurate description of the state of a machine requires an audit of the entire computer. However, that full audit may not be entirely necessary in every situation, and makes interpretation much more complex. Different applications use different measurements but distilling the measurements down is not a simple task: an attestation mechanism must be general enough to satisfy a wide range of security and measurement demands while still remaining fast and lightweight enough to useful in real-time computation. The line is a fine one.

Getting a meaningful measurement out of the machine is also quite complicated, due to the open nature of contemporary computers—in which there countless combinations of hardware, operating systems, and applications which may or may not interact during any given operations. The variability creates both numerous holes which must be checked, and a wide variety of redundant information generated in any software measurement. Thus, to evaluate the full audit of a machine provides both too much information, and information which does not easily relate to an intuitive concept of *safe* or *appropriate* use for humans.

On the other hand, to mandate a specific system is to overly restrict the user to a very specific and limited arrangement which may not fulfill the requirements for functionality on the machine. The demand for legacy application is a single stunning example of the human inability to switch their methodology or pattern of behavior, even in the face of motivating evidence: security solutions are most successful when they ease users into changes.

Additionally, the straight cost of moving to such a restricted system would be enormous in software—and potentially hardware—development costs, considering the level of custom software inherent in many fields.

Thus, the current methods of attestation provide no good solution to the problems of *what* and *how much* and *how* to evaluate the system, although the idea has significant potential.

1.2.3 Isolation: Compartmented Attestation

Some researchers have proposed the idea of *compartmented attestation* to limit the amount of data and software stack that needs to be evaluated for security reasons [16, 18]. By using Security Enhanced Linux (SE Linux) and mandatory access control (MAC) systems, they propose to isolate specific compartments of the platform. MAC uses a security monitor to control access to specific entities. By using this property to create sandboxes for secure programs they limit access to the given compartment, as well as the material that needs to be attested to in order to verify the state of the machine.

However, the policy creation needed to appropriately isolate various compartments is highly complex, and revealing the state of the MAC system in relation to a single compartment entails revealing the state of a large portion of the platform. While sandboxing is an idea I would like to bring forward and MAC is a good preliminary means of doing so, I believe that there are additional approaches which produce less overhead for the system and the associated attestation.

1.2.4 Property-based Attestation

Property-based attestation is a variant of the attestation approach that focuses on categorizing system behavior, rather than system setup [24, 12]. Rather than evaluating dependent on certain operating systems or applications, this approach seeks to evaluate certain properties of the system itself. This has the advantage of coming closer to attestation that makes sense to users and *their* trust evaluations. However, it makes a number of assumptions leading to an inconclusive solution. Firstly, they rely on a Trusted Third Party (TTP) to take a system setup and make it into a list of properties. Secondly, they simplify their design by assuming that the OS is policy neutral. The design of this approach suffers from both an overwhelming level of knowledge about the system, which someone must convert to attestation, and from an assumption that an external computing source can make the majority of these evaluations.

1.2.5 Virtualization

Virtualization has been used to implement alternate forms of compartmented attestation in specific cases. Virtualization often works by having a pre-established image of a virtual machine that is loaded and starts running only when necessary [26, 10, 9, 12]. It is an ideal approach, since virtualization often has isolation built in to the design so that each virtual machine is (ideally) unaware it does not have dedicated hardware. This isolation quality nicely replicates the idea of compartmented attestation in a more well-defined way. Additionally, the built-in features of virtualization often provide some means of monitoring the virtual machines and a layer of restrictions between the platform and the virtual machines. For example, it is often trivial to shut off network access to a virtual machine, or control the mounted file systems in the machine.

However, the prior work in this area has a few specific issues. Firstly; the systems do not scale well to instances where more than one dedicated application or OS is used on the system. Because of the way the virtual machine images are created, they do not account for the increased application stack when operating systems are used, one on top of another, in the virtual machines, nor do they generalize their checks to an environment where multiple applications or types of containers may be required. Secondly, they require significant negotiation for security specifications in immediate circumstances (where the virtual machine is not prearranged). In the case where an image does not already exist, they provide no methodology for dealing with negotiation for a new security VM, or proving security assurance about that

VM, which leads to trial and error being the default method of choice. Thirdly, they do not account for the still-monstrous attestation needed when each virtual machine has its own software stack needing verification. Because most virtualization methods lead to an increase in the amount of software on the stack between operating system, virtual machine monitor, and the operating system within the VM, even before the attestation gets to application software itself, the efficiency of requiring these machines to run in virtual environments is not clear: the benefits of running in a virtual machine provide isolation and a limit to the breadth of the attestation needs, but add significantly to the overhead and the depth of the attestation needs.

1.2.6 Computing on Demand

Several researchers have considered the concept of *Trusted Computing on Demand*, which operates under the philosophy that we should only provide the attestation for those tasks/environments which call for it [4, 20, 18]. This optimization allows the user to operate with both greater freedom and greater efficiency when they are willing to also accept greater risk, or forgo participating in activities requiring a high level of trust. They suggested selecting at boot-time the need for a trusted machine, and carried out the auditing at that point in the boot process [26]. This creates an average increase in the efficiency of operations of everyday actions, as well as allowing users the choice of what risks they wish to accept. However, due to the need to reboot every time a change in status occurs, and the still-inconvenient task of auditing an entire system, the option is less-than ideal and provides a minimal set of benefits.

1.3 Proposed Solution

The solution I propose is a variant on the approaches of *virtualization* and *trusted computing on demand*, although it pulls ideas from most of the approaches to trusted computing. The tools run on OpenSolaris and provide a networked interface for lightweight virtual machines

Approach	*Benefits*	*Drawbacks*
Attestation	1. Provides a measurement of the machine state	1. Too difficult to distill to a meaningful value while maintaining generality
Property Based Attestation	1. Provides subtler distinctions to the attestation. 2. Provides more coherence between the human-desired traits and the computer-provided system configuration	1. Difficult to quantify. 2. Introduces cost of translation and enforcement.
Compartmented Attestation	1. Provides a rigid set of controls into and out of the compartment. 2. Reduces to smaller subset of materials to measure.	1. difficult to maintain the MAC policies. 2. Does not significantly improve upon the audit complexity
Virtualization	1. Provides isolation to the secure applications 2. Provides built-in mechanisms to inspect the compartments	1. Fails to scale well to multiple applications or systems 2. Adds bulkiness to the attestation 3. Lacking in flexibility
Computing on Demand	1. Allows users to take the cost of attestation only when necessary.	1. Provides a large overhead for switching. 2. Does not significantly improve upon the audit complexity
Virtual Container Attestation	1. Provides little overhead for both isolation and flexibility 2. Provides mechanisms for runtime analysis and inspection of the compartments	

Table 1.1: Summary of approaches to the trusted computing problem

to be created and destroyed on command, eliminating much of the overhead of the platform-wide trusted computing on demand, as well as preserving the isolation of virtualization and adding a flexibility in setup that allows a larger variety of applications to be run on the virtual machines.

The virtual machines embedded in that operating system (called *containers*, or *zones*) are ideal for our purposes: by sharing the underlying operating system, in addition to much of the common files and utilities, they are (theoretically) both more lightweight to start up and more easily attested to.

1.3.1 Goals of Virtual Container Attestation

The solution we implemented provides a central virtual machine monitor (VMM) which doubles as a security monitor. This VMM has five main parts.

1. It intercepts commands to the zone (or controlling the zone) from within the operating system or users logged into the machine itself and redirects them to the VMM-specific versions, which have more checks and logging capabilities, particularly for container creation.

2. It accepts zone commands over an SSL-based service, and redirects those commands to the VMM-specific versions. Additionally, this SSL server provides the means for external clients to interact with non-networked zones.

3. It monitors specific attributes of the system and containers and halts zones which do not comply with the requested attributes. Such attributes are requested at creation-time.

4. It employs PKI as a means of providing attestation via property-attributed certificates that may be revoked to negate the attestation: asserting that the user can no longer trust the zone to have that property and comply to the requested policy.

5. It utilizes secure hardware—the *Trusted Platform Module*—to ensure ensure that zones whose certs have been revoked cannot continue to attest or to operate within the system.

The more specific design features of the tools are discussed in section 4.

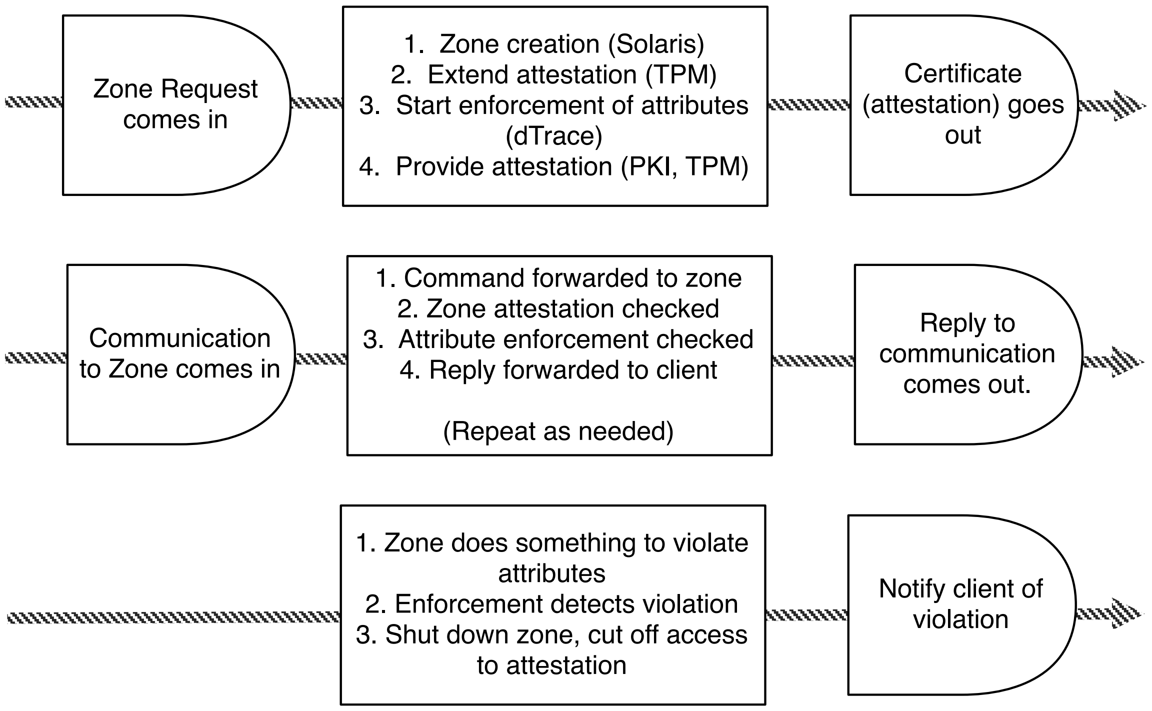

Zone Request comes in
1. Zone creation (Solaris)
2. Extend attestation (TPM)
3. Start enforcement of attributes (dTrace)
4. Provide attestation (PKI, TPM)
Certificate (attestation) goes out
Communication to Zone comes in
1. Command forwarded to zone
2. Zone attestation checked
3. Attribute enforcement checked
4. Reply forwarded to client
(Repeat as needed)
Reply to communication comes out.
1. Zone does something to violate attributes
2. Enforcement detects violation
3. Shut down zone, cut off access to attestation
Notify client of violation

Figure 1.2: Client functionality provided by our solution.

1.3.2 Previous Work in this Direction, and Contributions of this Thesis

Members of the lab community at Dartmouth have begun work in this direction previously. John Baek and Evan Tice brought the project to the point where there was a fairly solid SSL service to create zones over the network, as well as an interface with which to interact with the zones, once created, and send them commands.

This thesis modifies the services provided and takes the project further by creating property enforcement for the zones in a number of areas, tying that to the TPM, and utilizing a Certificate Authority to bring the remote zone creation to the formalized attestation problem, making it a more realistic and reliable solution to the trust problem, rather then merely a utility for remote zone creation.

Additionally, the thesis builds functioning prototypes for a handful of power grid applications, used for testing the VCA framework, but useful in their own right.

Chapter 2

Motivation

As discussed in the introduction, this project has applications throughout computing, and an increase in distributed or cloud computing leads to increasing demands for good solutions to the trust problem. Any situation where the protection of data and computation is vital to both parties is one in which our solution, *Virtual Container Attestation* would be appropriate.

This project could be useful in almost any area of computing, and certainly within distributed computing. However, the applications I focus on in this project are those of the power grid entities, who must, by nature of the grid, have interactions with other entities of whom they are inherently distrustful.

2.1 Power Grid Background

The power grid, as it is currently run in North America, is administrated by a number of large corporate entities, each of whom may run a variety of different facilities. These facilities include generators, substations, control centers, etc. The largest control entities are Regional Transmission Organizations (RTOs), with varying control over other entities in their region. The RTOs were created by a federal committee hoping to regulate the flow of power supply throughout a variety of independent companies.

Below the RTOs there is no uniform form of organization, but there may be varying levels of corporate participation between companies who have control over generation, transmission, distribution, or all three in a particular area.

This amorphous organizational structure means that rather than having explicitly regulated interactions, these corporate entities must cooperate to get their jobs done and provide the best service. This cooperation inherently conflicts with the secretive and mistrustful corporate environment in which each entity must inherently be suspicious of their competitors. Currently, the communication between these entities—and even between centers within the same company—is vulnerable and the companies are quite interested in securing the various interactions.

For the purposes of the thesis, I selected two connected protocols to be the basis for my applications. However, there are a variety of possibilities out there.

2.1.1 Inter-Control Center Communication Protocol

There is an existing international standard, *Inter-Control Center Communication Protocol (ICCP)* for communicating between power grid units, which is implemented in a variety of different ways. Although details are of these implementations are difficult to come by, it is common for the ICCP application to keep a table of permissions for a specific set of interactions. There are officially nine different forms of interaction, which can be narrowed to the following four categories: device control, general messaging, control of programs at a remote control center, and exchanging time-critical data between control centers [29]. The ICCP often runs as an overlay network on several control centers, so data and transactions may pass through several node machines before coming to rest at the final destination.

There is clear support here for the ICCP to disallow requests of an unprivileged entity, but there is no enforcement of ICCP itself to be running on a workstation. Thus, there is little support for trust of the machines or of the data and computation that must pass through them. All trust relationship are caught up in the structure of the network, even when such relationships may not truly exist: there is no structural reason to trust a neighbor who says that they run ICCP. This is the level of security that Virtual Container Attestation attempts to resolve, by providing assurance to these mutually distrustful entities that their data and program operations are happening in a manner which enforces the properties they desire.

In summary: the problem with the current implementation is such that a request can leave a machine (the client), go to another machine (the host), and data can travel the other direction with no assurance that the host is actually providing accurate data. While ICCP seems to do well with access matrices at ensuring that requests come from validated people, they do not do much validation of the data coming back, or the operation at either end.

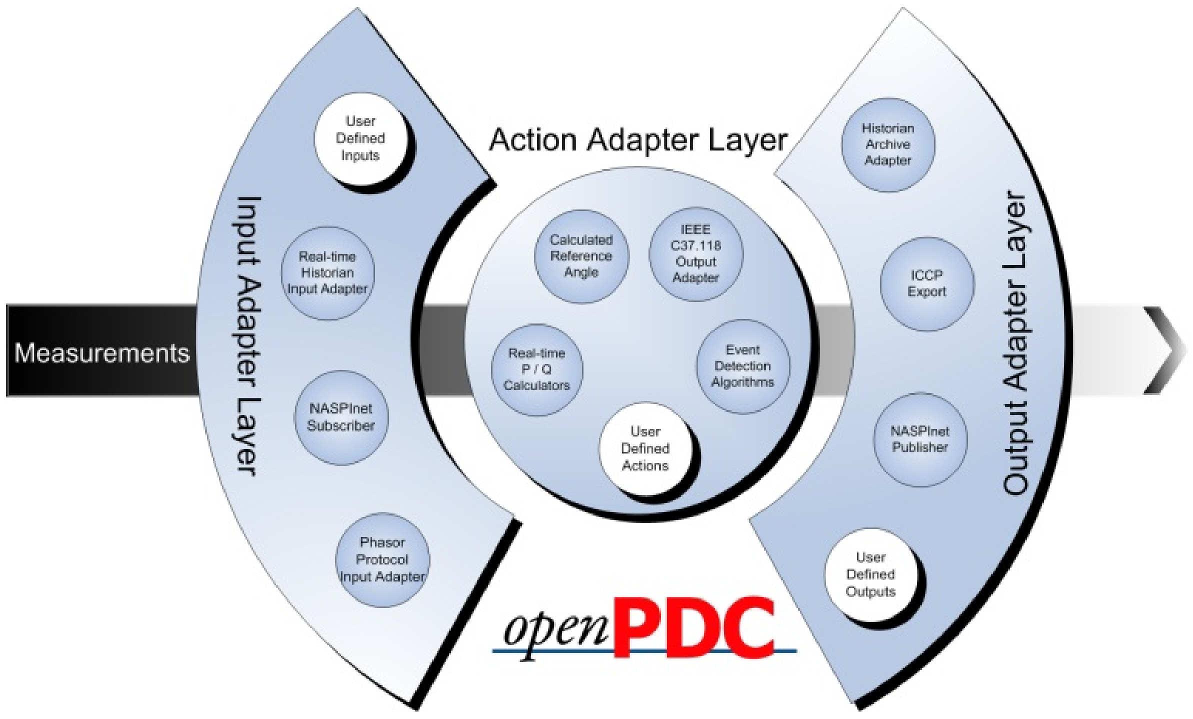
2.1.2 OpenPDC: the open-source solution

OpenPDC—open source phasor data concentrator—is power grid software released by the Tennessee Valley Authority (TVA) with the intent to make it easy to manage, process, and respond to changes in fast streams of data from the devices used in the power grid. This includes all devices, not just phasors. Time-stamped data includes temperature, voltage, location, etc, and gets merged with other input data as it goes through the application [1].

Although OpenPDC is written in such a way that I am unable to adapt it for direct use in OpenSolaris, I was able to use it as the basis for the handful of testing applications created for this project. Its adaptors are modular pieces of code which allow for mass processing after the fact or data streaming through protocols such as the ICCP. See Figure 2.1 for the full outline of PDC functionality.

2.1.3 Emerging Smart Grid Technology

One of the promising up-and-coming technologies in the power industry is that of the Smart Grid. The Smart Grid focuses on bringing better service through a variety of new technology. Theoretically, this technology provides two-way support through sensors and smart appliances which provide information to the power suppliers and users [13]. From the view



Measurements
Input Adapter Layer
User Defined Inputs
Real-time Historian Input Adapter
NASPInet Subscriber
Phasor Protocol Input Adapter
Action Adapter Layer
Calculated Reference Angle
IEEE C37.118 Output Adapter
Real-time P / Q Calculators
Event Detection Algorithms
User Defined Actions
Output Adapter Layer
Historian Archive Adapter
ICCP Export
NASPInet Publisher
User Defined Outputs
openPDC

Figure 2.1: Modular OpenPDC design, from http://openpdc.codeplex.com/

of the suppliers, they receive information that allows them to make more informed choices to keep the grid running and more efficient. From the point of the users, the information allows them to be more in touch with where their power is going within their home (for instance, which appliances use disproportionate amounts of power) and where their power is coming from: the grid hopes to provide support for alternative energy sources to the users.

Industry leaders hope this technology will lead to better management of power and technology by providing users with utilities which report information to the control center and react to changes in the power grid, to more efficiently utilize its resources. One proposed service provides an itemized power bill to users, which compares their usage to those of their neighbors, for context. Other propositions have been to create appliances that, for example, only run the dishwasher when it is most efficient and power is at low demand. The power companies have shown a willingness to create price incentives to encourage this behavior, in which case consumers would get some idea of how their utilities and consumption patterns affect their monthly bill.

However, the amount of information gathered just to provide a single one of those applications is enormous, and without appropriate controls, the security and privacy issues of this emerging technology are glaringly obvious. If I get an itemized bill comparing my usage to Mr. Smith's usage next-door, then that information has probably traveled through a number of different hands, and the information may be accessible to a wide number of people: allowing utter strangers (or Mr. Smith) to know my habits, including when and how much I watch TV, microwave popcorn, or run the lights in my bathroom. Such information can easily lead to other, more compromising vulnerabilities.

Such a network of Smart Grid utilities would involve an enormous amount of data collection, aggregation, and redistribution. The path of this data has not been specified, but it is commonly believed that the the utilities will report to a local network, which communicates with data-collection servers, who then distribute the information to other entities both inside and outside of the Smart Grid. I believe that my technology can provide sterile environments where such collection can occur—allowing the Smart Grid technology to overhaul our national power usage—but in a manner which allows each consumer to maintain their privacy and security.

2.2 Motivating Scenarios

I have selected four real-world scenarios within the context of the power grid technology. These scenarios represent the motivating scenarios for the thesis and demonstrate several different uses for the Virtual Container Attestation technology.

Each scenario is an instance of the case where a user (client, Alice) is making use of a virtual machine on another machine (host, Bob). Alice would like to be sure that Bob's machine is enforcing certain properties, and so she asks for attestation in return before the operation commences.

In the following scenarios, Alice represents the client requesting a container from the host, *Bob. When necessary, Carlos represents client2, who also interacts with the host container. Mallory is, as usual, a malevolent force somewhere out in the internet. In one particular application, Dave is an external authority, an auditor*

2.2.1 Data Transfer

Alice requests data digests from Bob, but wishes to do the data processing on his machine. The zone should have an external network connection so that it can send encrypted data digests to Alice (and only Alice). Bob allows Alice to run her (provided) executable on his machine, but watches for data corruption over the data he has let Alice see.

In the context of the power grid, this is the sort of thing that would run persistently between two partner companies, or a generation and transmission company, sending Alice periodic digests of Bob's data. It requires little interaction with either party and is thus a fairly simple application to monitor.

2.2.2 Program Control

Alice and Bob are companies at the same level of power organization (ie. transmission). For whatever reason, they need to exchange control of a device. Alice needs access to some device X. The physical connection to device X is attached to and managed by Bob's computer. He grants her a container on his server, with some restrictions.

When creating the container, he guarantees her container some percentage of his cpu cycles. He allows her access to a variety of programs, and to the network, but he watches to ensure that data corruption does not change the status of the device to a state which he believes to be dangerous.

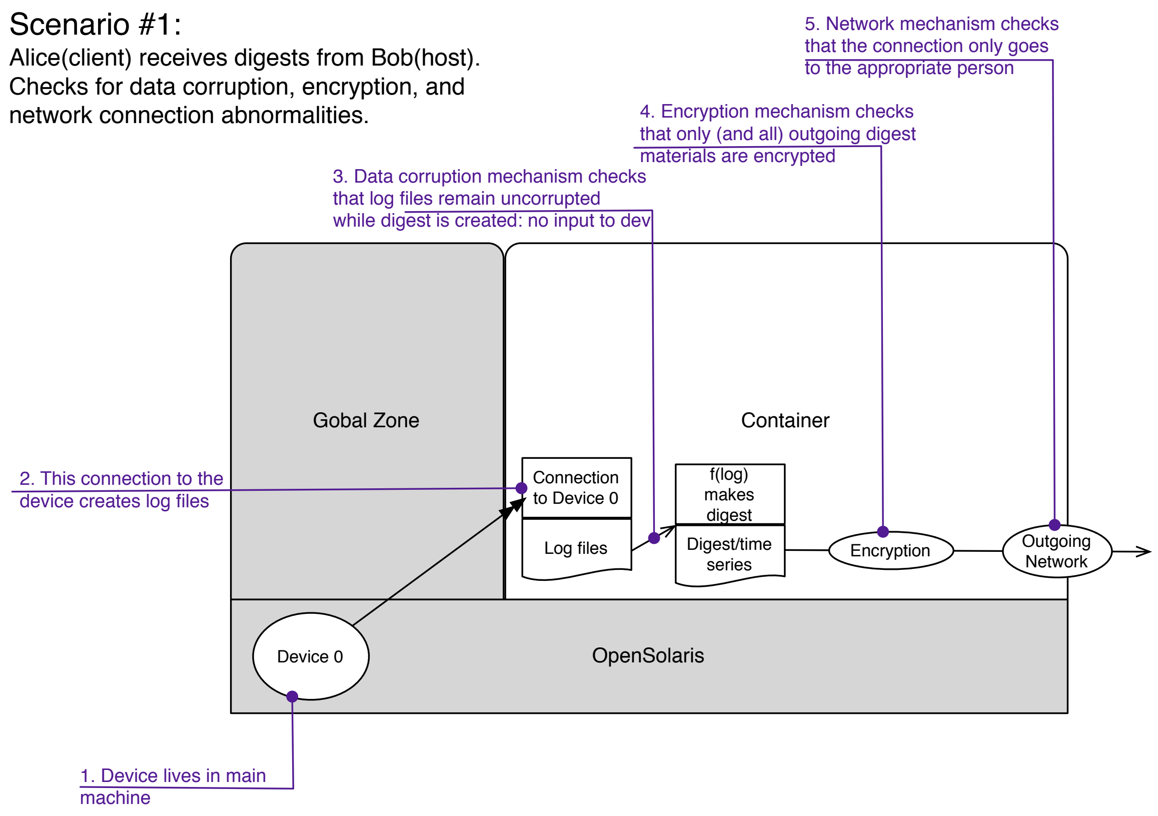

Scenario #1:
Alice(client) receives digests from Bob(host). Checks for data corruption, encryption, and network connection abnormalities.
5. Network mechanism checks that the connection only goes to the appropriate person
4. Encryption mechanism checks that only (and all) outgoing digest materials are encrypted
3. Data corruption mechanism checks that log files remain uncorrupted while digest is created: no input to dev
Gobal Zone
Container
2. This connection to the device creates log files
Connection to Device 0
Log files
f(log) makes digest
Digest/time series
Encryption
Outgoing Network
Device 0
OpenSolaris
1. Device lives in main machine

Figure 2.2: Scenario 1

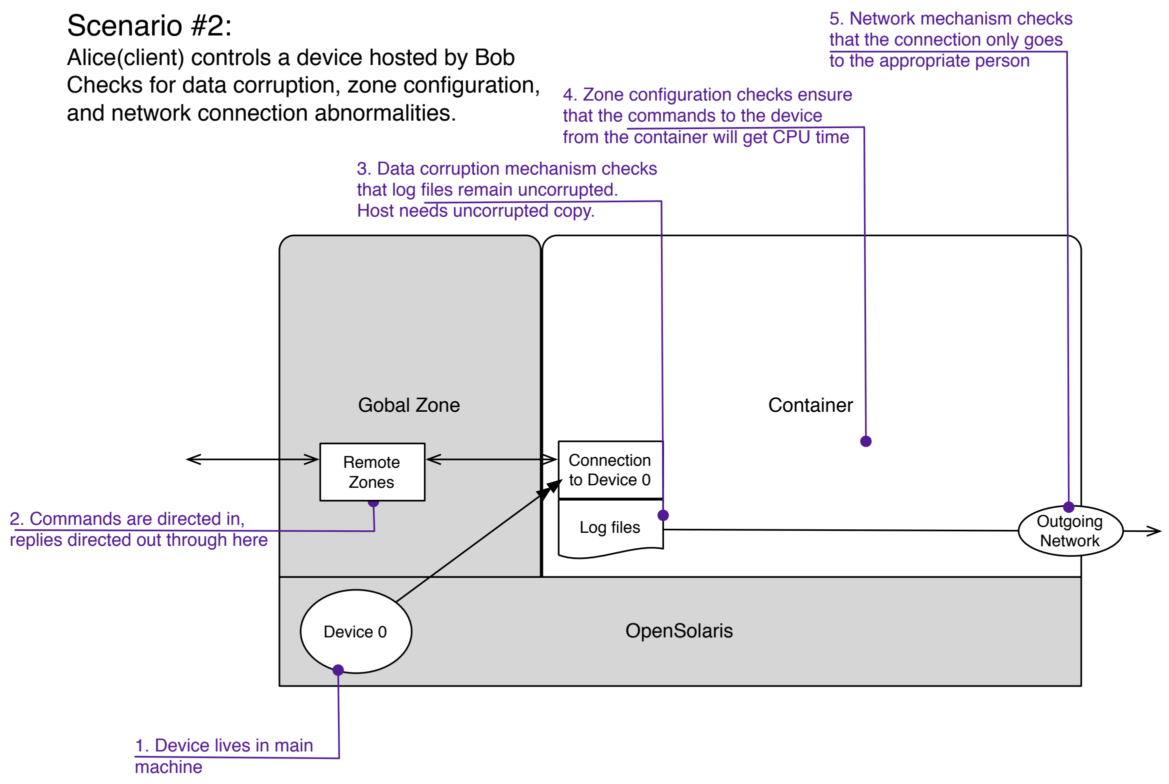

Scenario #2:
Alice(client) controls a device hosted by Bob
Checks for data corruption, zone configuration, and network connection abnormalities.
5. Network mechanism checks that the connection only goes to the appropriate person
4. Zone configuration checks ensure that the commands to the device from the container will get CPU time
3. Data corruption mechanism checks that log files remain uncorrupted. Host needs uncorrupted copy.
Gobal Zone
Container
Remote Zones
Connection to Device 0
Log files
Outgoing Network
2. Commands are directed in, replies directed out through here
Device 0
OpenSolaris
1. Device lives in main machine

Figure 2.3: Scenario 2

2.2.3 External Audit

Dave is an auditor, either externally or from a higher-level power grid organization that needs access to a variety of data and device information, as well as the ability to investigate the state of various devices. He is currently auditing Bob's system and needs to be able to run both his own code and some of Bob's executables, and needs them to happen in a specific order such that Bob cannot alter the outcome of the audit, but can provide information as necessary. Additionally, Bob would like to ensure that Dave is not inserting anything malicious into his executables, and that he does not change the device state.

Because Bob's audit data is sensitive, particularly to his business, he would like Dave to encrypt anything that leaves the server. However, Bob would also like to make sure that Dave does not have the capability to encrypt any of Bob's files, and hold them hostage until Bob does something for him. Thus, encryption is also a vital enforcement technique here.

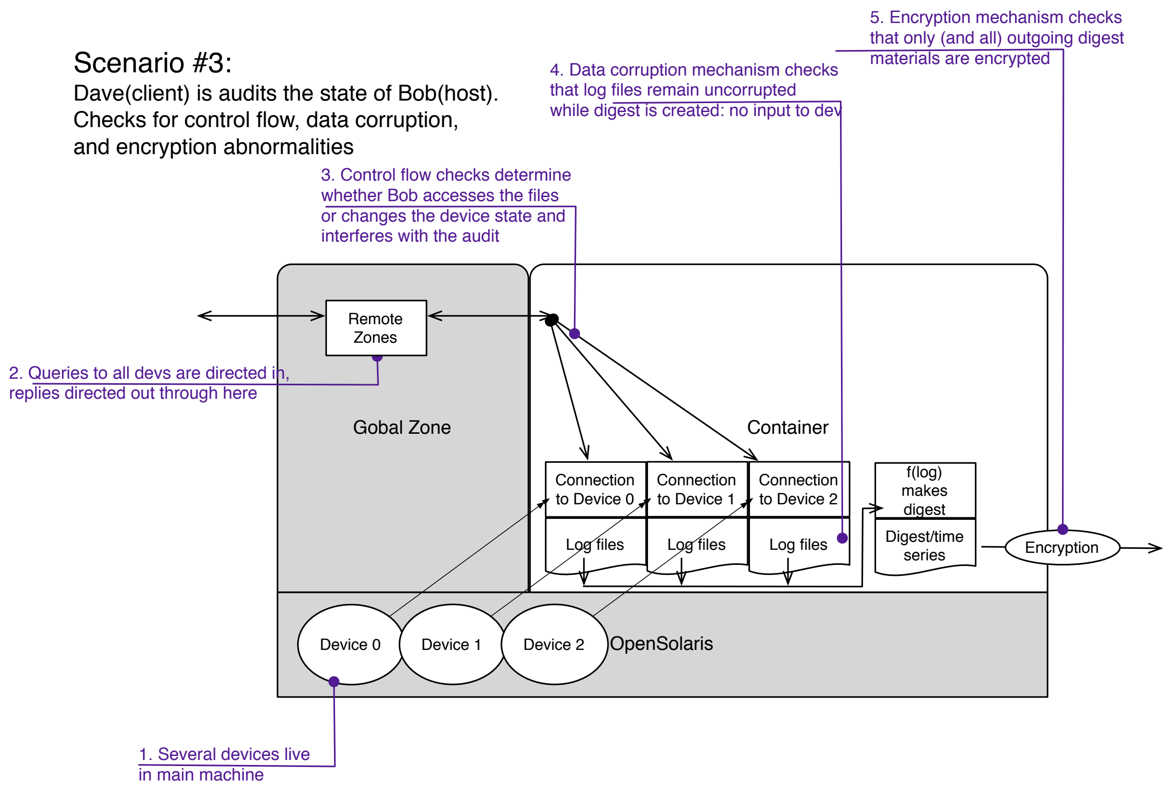

Scenario #3:
Dave(client) is audits the state of Bob(host).
Checks for control flow, data corruption,
and encryption abnormalities
5. Encryption mechanism checks that only (and all) outgoing digest materials are encrypted
4. Data corruption mechanism checks that log files remain uncorrupted while digest is created: no input to dev
3. Control flow checks determine whether Bob accesses the files or changes the device state and interferes with the audit
2. Queries to all devs are directed in, replies directed out through here
1. Several devices live in main machine
Remote Zones
Gobal Zone
Container
Connection to Device 0
Connection to Device 1
Connection to Device 2
Log files
Log files
Log files
f(log) makes digest
Digest/time series
Encryption
Device 0
Device 1
Device 2
OpenSolaris

Figure 2.4: Scenario 3

2.2.4 Smart Grid Data Aggregation

Bob is the data collection center for a section of Hanover, NH. Bob collects and aggregates data from the main campus of Dartmouth College, as well as a number of residences, churches, and business that lie within a short radius of the campus. All of the data will be taken and used for power regulation purposes by Carlos, who has no need for identifying information to be attached.

Alice, the owner of a residence just outside the college border, is concerned about who gets her data. She fears that if Dartmouth gets the information, they can use it to somehow get her to sell her land to them. She is also a bit concerned that Bob will give her information to some advertising agents who see her habits and harass her with spam. But she likes the idea of being able to see an itemized bill at the end of the month.

Bob does the data aggregation within a container on his server. Alice's machines need access to report in and get information back, and Carlos needs the data to provide better service to his customers.

In this case, Bob requests the zone. He specifies a control flow that he expects the data to take, such that anonymization happens after Alice's operations have finished and before Carlos jumps in. He isolates data structures that need to remain uncorrupted for valid business or accounting purposes. In addition, he asks that the data going in and out must be encrypted for Alice's privacy.

2.2.5 Provenance of these applications

Each of these applications is based on actual examples of code or power grid interactions. Most are based on openPDC code.

The first, data requests, is common in both the ICCP realm and openPDC. Data requests between providers and within companies allow the load to remain balanced and internal checks to be made. The enforcement in this scenarios prevents the host from modifying the data which the client needs, and prevents the client from modifying the state of the device

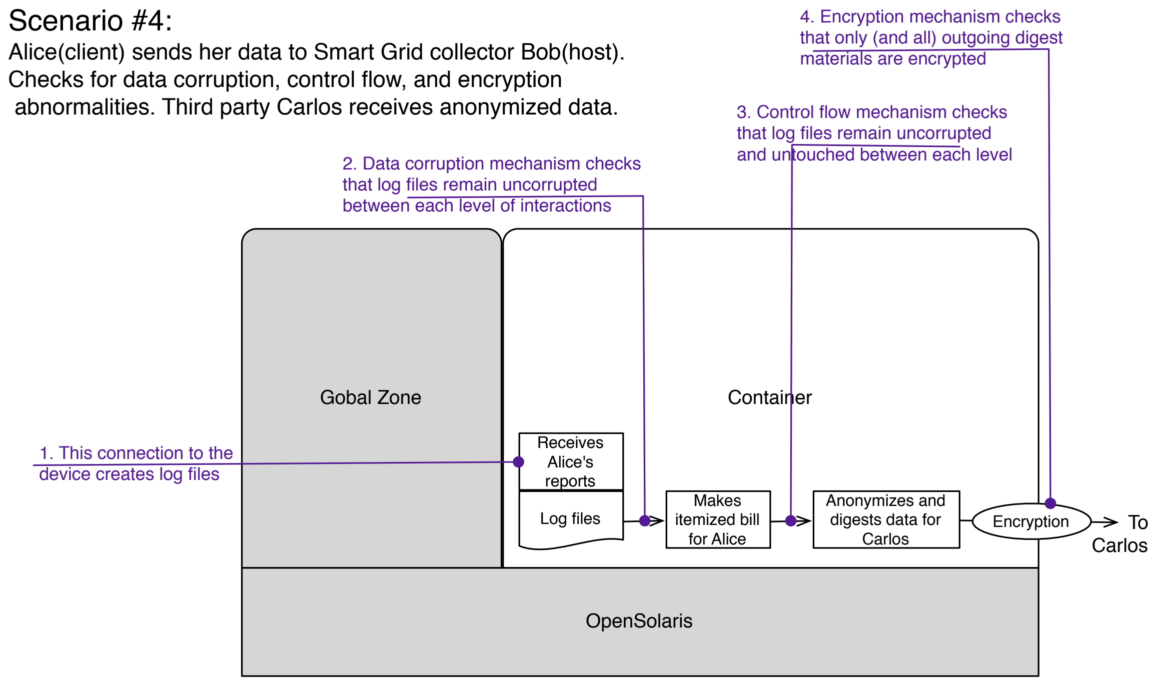

Scenario #4:
Alice(client) sends her data to Smart Grid collector Bob(host). Checks for data corruption, control flow, and encryption abnormalities. Third party Carlos receives anonymized data.
4. Encryption mechanism checks that only (and all) outgoing digest materials are encrypted
3. Control flow mechanism checks that log files remain uncorrupted and untouched between each level
2. Data corruption mechanism checks that log files remain uncorrupted between each level of interactions
1. This connection to the device creates log files
Gobal Zone
Container
Receives Alice's reports
Log files
Makes itemized bill for Alice
Anonymizes and digests data for Carlos
Encryption
To Carlos
OpenSolaris

Figure 2.5: Scenario 4

she is getting data from. It explicitly protects against the attack of client-provided code running amuck, and restricts it to a specific set of input processing.

The second, device control, is explicitly laid out in the openPDC code, and has some minimal specifications in the ICCP. Again, it is common practice for device control to be shared, particularly in border areas. The enforcement done in this scenario explicitly protects against malicious use of the device, hiding the use of the device from the host, and delving into data outside of the device.

The third, device auditing, receives peripheral attention in both the openPDC and the ICCP and is essentially a more complicated form of the first scenario, data requests and aggregation. The enforcement done in this scenario protects the accuracy of the information: attacks of the host on the outcome of the audit. Additionally, it prevents the auditor from effecting a change in state through sending active commands or executables to the devices it queries.

Scenario	*Attrib 1*	*Attrib 2*	*Attrib 3*	*Attrib 4*	*Attrib 5*
Scenario 1: Data Transfer		X		X	X
Scenario 2: Program Control	X	X		X	
Scenario 3: External Audit			X	X	X
Scenario 4: Smart Grid Data Aggregation			X	X	X

Table 2.1: Summary of application scenarios, as discussed in section 2.2.

Attrib 1	Basic cfg file restrictions. No network connection, for example, or run-time specifications, specific mount points, etc.
Attrib 2	Network connection. Regulates "abnormal" network connections, or connections to people not in a pre-specified set.
Attrib 3	Control Flow. Regulates the control flow of a container to ensure that the jobs originate from trusted or "normal" sources and that, if a control flow is specified, it follows that flow.
Attrib 4	Data Corruption. Regulates specific userland data structures to verify that they are modified only by the correct jobs and are not morphing to something strange.
Attrib 5	Encryption. Verifies that encryption is being used in appropriate places (for external communication) but that it is not being used to send or capture information for nefarious purposes.

Table 2.2: Summary of attributes enforced in this thesis as discussed in section 2.2.

Finally, the fourth scenario has been explicitly laid out by the smart grid planning, and has be historically pointed to as a significant privacy hole. [13] The enforcement protects the accuracy of information for each stage (itemized billing, flow control, etc). It also ensures that privacy is kept as high as possible.

Chapter 3

Building Blocks

3.1 Trusted Platform Module

The Trusted Platform Module (TPM) is a piece of hardware shipped with the machine. It is designed specifically for secure operations and therefore has a number of special capabilities, only some of which will be taken advantage of in this project.

The TPM is equipped to generate RSA keys, SHA-1 hashes, random numbers, and to do limited encryption and decryption. The TPM supports storage of these keys as well as a number of permanent keys granted at production and initialization of the TPM. Additionally, the TPM supports the option to restrict use of such keys, explained below. It is important to remember that the computational power of the TPM is limited and its role in encryption and decryption should be somewhat limited.

The TPM has a number of Platform Configuration Registers (PCRs) which provide extendable data storage for the TPM. They cannot be written to, but only extended by concatenating the new and old values together and storing a SHA-1 hash of value. They are only cleared at reboot of the machine. Thus, the PCRs not only provide storage, but they provide a record of the previous values.

TPM keys can easily be tied to a specific configuration of the machine by *sealing* them to a specific PCR value. This capability restricts usage of the key to the states where the PCR values match the stored value for the key. Since the keys are stored in the TPM they are not accessible outside and the TPM check against the PCR cannot be bypassed. This capability also allows data to be sealed to a specific state or PCR value. The number of uses for this sealing property are numerous.

The TPM is often used for trusted computing, and is in fact designed for it, since it is an effective means of providing a ”trusted root”. The TPM can be wet up to perform a *trusted boot*, wherein it evaluates each level of boot, from hardware to firmware, BIOS, and bootloader, evaluations of which are stored in the PCRs. The TPM can continue to be used for trusted computing by extending state into PCRS and wrapping keys or data to the stored state. Thus, it becomes possible to only access that data or keypair when the state matches the stored state: in other words, when the state is approved. In such a situation, having access to the keys implies approved state for everything stored in the TPM.

I take advantage of these features for the Virtual Container Attestation code.

3.2 Solaris

Solaris is the operating system released by Sun (now Oracle) following SunOS, with the capability to run on either SPARC or x86 processors. Although similar to Linux in form, it has a number of differences, which make it quirky to code with. We chose this operating system for a few reasons: its OS-level virtualization, runtime analysis tools, and its open-source code.

Recently, much of the code for Solaris 10 was released as OpenSolaris, meaning that we can delve into the operating system if required. For this project, I run in the current development repository for OpenSolaris (dev-133 or 0.5.11-0.133 from http://pkg.opensolaris.org/dev). While this allows me to take advantage of a number of features of the trusted computing software stacks and TPM drivers, it unfortunately comes with the risk that such packages are not fully ready for release.

3.2.1 DTrace

The other main reason for using Solaris is DTrace. As a dynamic tracing method, DTrace is invaluable. Its ability to dig into both user-and kernel-land code, as well as monitoring syscalls and other attributes makes it ultimately one of the best tools we have on hand, particularly for the enforcement of various attributes, which will be discussed in more detail in 4.

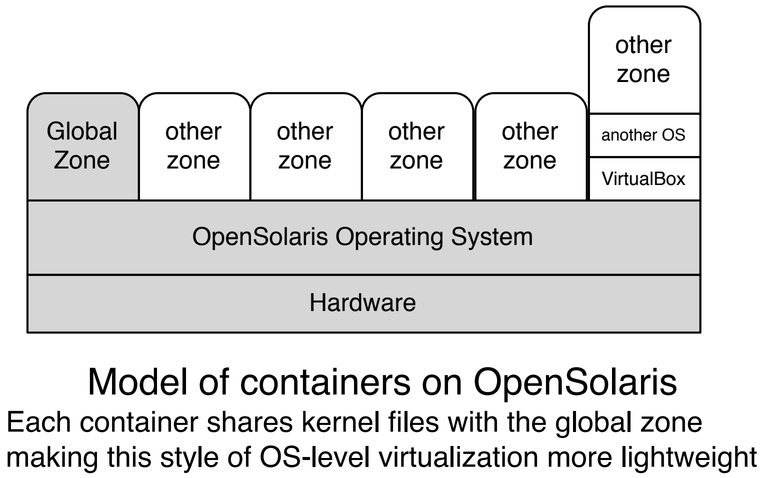

Global Zone
other zone
other zone
other zone
other zone
other zone
another OS
VirtualBox
OpenSolaris Operating System
Hardware
Model of containers on OpenSolaris
Each container shares kernel files with the global zone making this style of OS-level virtualization more lightweight

Figure 3.1: Zones, as represented in the OpenSolaris system.

3.2.2 Zones

The Solaris zones are an example of operating system-level virtualization, and as such, a single kernel is shared among all of the zones. Applications thus see a guest container as a standalone system, but only copy of Solaris is present on the machine. A single global zone is persistent, always defined, and is the only zone which may access other zones. The global zone encompasses all jobs present on the computer, regardless of whether they run in a non-global zone or not, and may observe any process on the computer. Thus, our tools run in the global zone and have access to the non-global zones.

Theoretically, the Solaris Zones are more lightweight and have a lower start-up time, overhead, etc, than do normal forms of virtualization. Additionally, they do not run a fully unique form of the operation system, thus reducing the software stack down to a more manageable size. Sparse zones can be created which share most of the files with the global zone, or whole-root zones, which contain copies of a subset of the files. Additionally, the theoretical isolation provided between zones is ideal for our purposes: the only zone that can observe other zones is the global (initial) zone, and therefore no client running in a container (zone) will be able to peek into other system usage or containers, including the global zone.

The zones are created from configuration files which are easily parse-able and provide several useful options for our project, including options to limit install options, network activity, mount points and run-time resources. These are all properties that would be important to monitor in creating sterile containers.

Although options such as Xen run at the hypervisor level, and therefore give up the advantages of both observability and quick install from kernel-sharing, BSD jails are more comparable to the Solaris containers. However, Solaris maintains an advantage over the BSD alternative through heightened observability (through the global zone and DTrace), as well as the easy configuration options and a more convenient method of zone creation, startup, and deletion. Where BSD jails focus more on process isolation, Solaris creates a more full-formed isolation.

Chapter 4

Design

To reiterate from section 1.2.6, the following are the capabilities of the Virtual Container Attestation system created in this project.

1. It intercepts zone commands from within the operating system or users logged into the machine itself and redirects them to the VMM-specific versions, which have more checks and logging capabilities, particularly for container creation.

2. It accepts zone commands over an SSL-based service, and redirects those commands to the VMM-specific versions. Additionally, this SSL server provides the means for external clients to interact with non-networked zones.

3. It monitors specific attributes of the system and containers and halts zones which do not comply with the the requested attributes. Such attributes are requested at creation-time.

4. It employs PKI as a means of providing attestation via property-attributed certificates that may be revoked to negate the attestation.

5. It utilizes secure hardware—the *Trusted Platform Module*—to ensure ensure that zones whose certs have been revoked cannot continue to attest or to operate within the system.

The end goal of the project is twofold. On the one hand: we wish to create a means of attesting to the state of a given remote zone, specifically to secure attributes of that zone. Additionally, the machine should have some means of ensuring those attributes hold true, and that the attestation matches the enforcement. We do the former through a chain of certificates, and the latter with a dedicated zone manager.

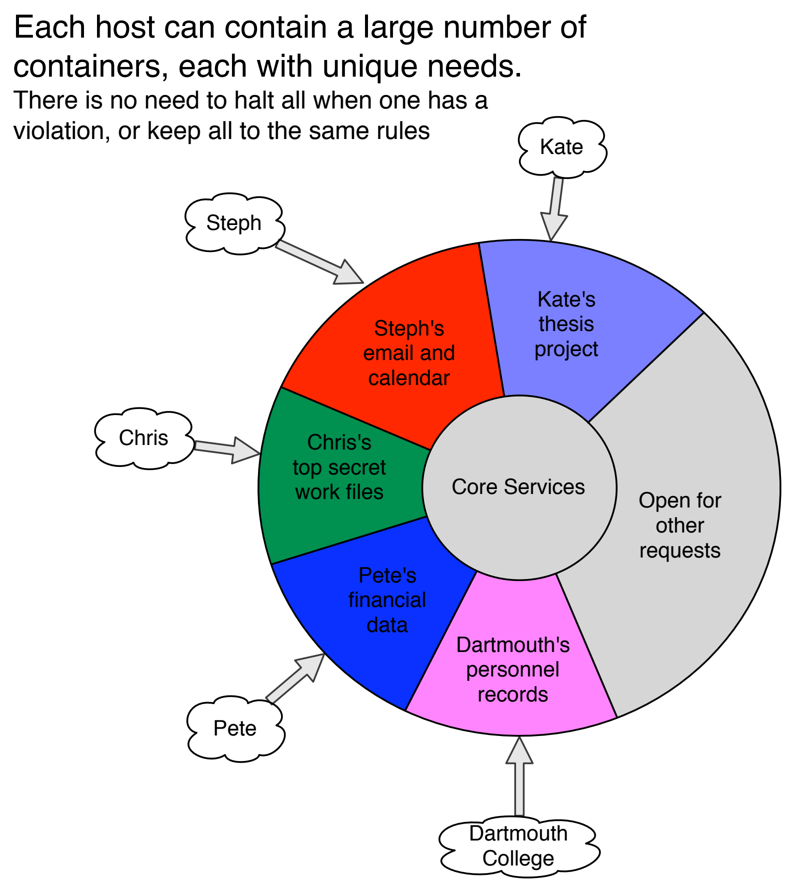

Each host can contain a large number of containers, each with unique needs.
There is no need to halt all when one has a violation, or keep all to the same rules
Kate
Steph
Kate's thesis project
Steph's email and calendar
Chris
Chris's top secret work files
Core Services
Open for other requests
Pete's financial data
Dartmouth's personnel records
Pete
Dartmouth College

Figure 4.1: Variety of containers and purposes available with OpenSolaris.

4.1 Certificate Chain

The TPM is granted a certificate during manufacturing, as well as the ability to create other keys, but *not* the ability to act as a certificate authority. However, the certificate granted during manufacture is the basis of our certificate chain. Our certificate chain builds from here. An external certificate authority, run by a trusted third party, verifies the state of the TPM, OS, and zone manager, allowing the zone manager to act as a certificate authority if it passes inspection.

This zone manager then has its private key wrapped to the verified machine configuration stored in the PCR values, and the private key is stored in the TPM volatile key storage. Thus, if the machine changes state, the private key cannot be accessed. The zone manager may then issue certificates to the individual zones, based on their attributes. The certificate is issued for a relatively short period of time and is revoked if a zone is halted, and re-issued upon start-up. For the purposes of the project, issuing certificates on the period of a couple days was reasonable; for longer-running projects, the certificates could be issued for longer time periods. The keys are stored in TPM volatile key storage, under the keys granted to the TPM and the zone manager. The key is wrapped to the same value as the zone manager, and between its binding and its subordination to the previous keys, it restricts the subordinate keys to the system setups where the base application is working properly.

When a zone wishes to attest to its secure properties, it must only produce the certificate granted by the zone manager, which contains a list of attributes, and prove knowledge of the private key by signing the certificate. If the process fails because the key is inaccessable or the certificate has been revoked, the client knows either the machine, the zone service, or the zone itself has been compromised at some point (or at least has violated one of the property attributes).

There are a number of actions that can happen if there is a failure higher up the chain. The design decision of this thesis was to revoke all the certificates dependent on the parent cert: a failure of the zone manager leads to an invalid setting for all zones. While this is perhaps overly harsh, in the case where we only implement one revocation policy, we decided that erring on the side of caution is more important.

The policy followed when a violation of a security property happens is discussed below.

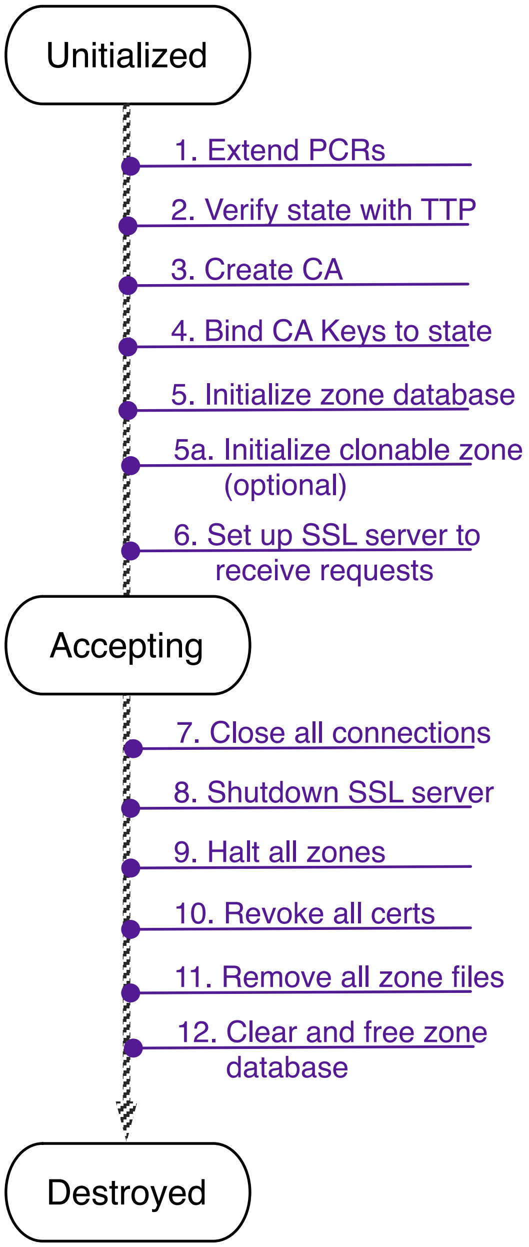


Figure 4.2: The long-term creation and destruction of global state.

4.2 External Certificate Authority and OS Repository

A third party external certificate authority is established prior to any use of the TPM for Virtual Container Attestation usage. This certificate authority also has access to a repository of accepted hardware/firmware/operating system evaluations. It has the authority to grant the zone manager Certificate Authority capabilities, and provides some level of standardization across all instances of the Virtual Container Attestation hosts.

The operations at the trusted third party are relatively simple, accepting a connection, comparing the PCR values to known values, and making a trust decision based on that, returning both answer and key if needed.

4.3 TPM initialization

The TPM, aided by the driver and low-level software, evaluates itself, the other hardware, BIOS, boot-loader, etc at boot time, storing these evaluations in specific PCRs inside the TPM itself. However, these values alone have no meaning or ability to create secure usage. Something external to the TPM must do this for us, and in this case that is the trusted third party.

For this project, we will store additional evaluations of specific parts of the operating system into PCRs. Specifically, we store evaluations of a selection of zone utilities (generally prefaced by *zoneadm* or *zonecfg*) in addition to the evaluation of the code in our tools, to ensure that the zones are in complete isolation and that our enforcement of attributes is correct. As clarification, the evaluation we most often talk about here is a SHA-1 hash.

These evaluations form the base measurements evaluated by the External Certificate Authority: without having some assurance of these utilities, we cannot make any assumptions about the rest of this chain, and as discussed above, they are required for the creation and release of any other keys.

4.4 Virtual Container Attestation Zone Manager

At the core of the project lies the Virtual Container Attestation Zone Manager (VZM). At a basic utilitarian level, the VZM acts as a coordinator for the attested zones. It receives and processes requests for creation and removal of VCA zones, and it acts an intermediate between any zone interaction and the zone itself. It is, in short, a buffer between the zones and external influences. (The exception being a handful of cases where the zone will have network access to the outside world, in which case the traffic in that connection is closely monitored.) This VZM also has more a more complicated role as a means of ensuring the validity both of the state of the machine and the protections on any given zone.

For simplicity of implementation, and without loss of generality, we assumed that all non-global zones on the machine were attested zones (or could be run as such with a null set of security properties).

In the contrast between figures 4.3 and 4.4 the evolution between the zone lifecycles is clear.

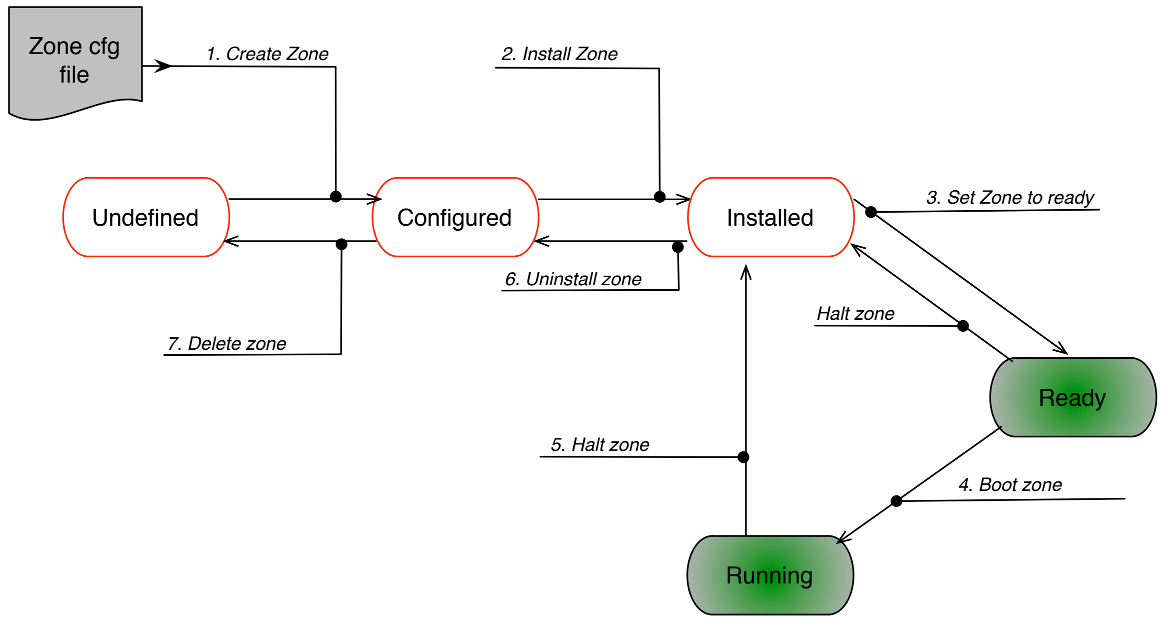


Figure 4.3: Lifecycle of a Solaris zone.

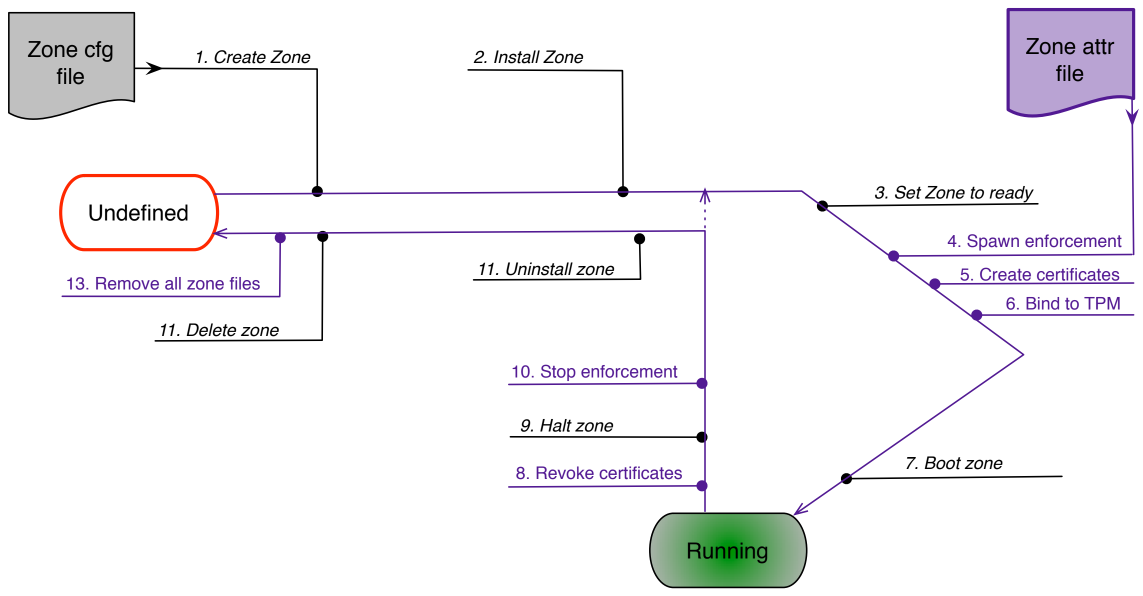


Figure 4.4: Lifecycle of a Virtual Container Attestation zone.

We go into more detail about the VZM below.

4.4.1 Certificate Authority

Once validated through the TPM and the external certificate authority, the VZM acts as a certificate authority itself, granting attribute certificates to the individual zones. The keys for the CA will remain sealed to state in the TPM, meaning that if the core code for the zone manager changes, the certificate chain will be broken.

This service is perhaps the simplest to implement, and yet lies at the core of the process, since the certificate chain is the means of attestation we have chosen to provide.

4.4.2 Enforcement Methods

A series of utilities run at the same time as the main zone manager code, maintaining the properties and taking appropriate measures should things be violated.

For simplicity of design, we implemented only one policy for dealing with zones whose attributes fail: we halt the zone and revoke the certificate. While other policies may easily be implemented, for the scope of this project, we felt this was appropriate.

Current zone enforcement mechanisms range from the simplest checking of the zone config files to the more complicated ones involving external checks.

Zone config file checks run prior to zone creation. They are just internal consistency checks, more for the host than the client. The client should specify a cfg file that satisfies her config-level requirements, and the host simply checks that it also matches his. Examples of such checks are network connectivity, cputime guarantees, mount points, etc.

Network Connectivity checks run concurrent with the active zone, to ensure that any abnormal network activity (either too much, of an abnormal sort, or encrypted when it should not be) is stopped in its tracks. A few examples of these checks are those that check the destination and frequency of the network traffic leaving a zone.

This enforcement is moot in some cases where the zone is not allowed to connect to the outside world, except through the original SSL remote zones connection.

Data Corruption periodically checks data vital to userland (zone) applications, to ensure that no data is morphing in unexpected ways. Sometimes these data corruption checks look at files. This ties in well with *Control Flow* checking, which monitors when and where specific programs get executed or called. For example, one of our scenarios mandates that executables get called in a specific order, and presumably by someone specific.

Both of these checks are reliant on the userland application provided. We have a number of applications provided that create specific instances to check these enforcements. They lie as close to the specifications as possible, and where possible we tried to use actual code from the power grid, or close descriptions of the interactions.

Encryption An interesting, but less common problem is the encryption of data to either hold it hostage or to do other sneaky things with sensitive data. By tracking the encryption capabilities and actions of the zone, we keep an eye on these situations.

In cases where sensitive data is being exchanged or being sent over the wire, the host —and often the client—would like encryption to be used. On the other hand, the host also wants to be sure that if the client is playing with sensitive data, it does not remove that data from the host's access. By tracking the encryption going on in a zone, we are able to minimize the cases of insufficient external encryption or restrictive internal encryption.

Design of Enforcement

Most of the enforcement mechanisms run monitoring via some form of DTrace and report to the security monitor based on the information gathered therein. These are persistent processes, most of which watch a single container and do no revocation themselves. We chose to have enforcement be enacted with a modular approach due to the potentially flexible set of security properties required. With this design, future modules may be installed, allowing a further extension to the security capabilities.

In addition to the modular format, there a handful of interceptions that take place, discussed immediately below.

4.4.3 Zone Commands

Zone command will remain mostly unchanged from those normally called in an OpenSolaris system. However, because the Zone manager acts as a buffer between the outside world and the zones, it may make slight changes and checks on the calls to zone commands. In particular, interception before the creation of a zone is important to guarantee the that the zone created is compliant with and registered in the database of VCA zones, for future checks and enforcement utilities, as discussed below.

4.4.4 Other Utilities

A number of other utilities run to check various things, mostly having to do with the setup of the zone. Currently unimplemented is a utility to check the basic configuration file of the zone for anything untoward. In the same vein, but implemented at this time is a utility to check the configuration file, extracting all pieces related to the attribute requests. This information is given to the zone manager as the basis for attribute certificates and as a check against enforcement mechanisms.

4.5 Outside Interaction

Connected with the VCA is a service running over SSL that accepts requests for the creation of VCA zones, and allows remote commands to be directed to them. As the network connections between VCA zones and the outside world may be limited, we assume that much of the interaction is done through this mechanism. Due to the SSL encryption, we know that those making requests must be trusted to some degree. This interface allows the remote user to create, destroy, and interact with zones from afar, and all commands have basic checks run on them before execution. Once a zone is created, its identifier allows the remote client to interact with the zone as through a shell, albeit with other modifications and checks inside.

4.6 Design Rationale: The Big Picture

Now that we have discussed the small pieces to the toolset, we discuss the larger picture and how these pieces fit together. At the point, we would also like to remind the reader that this system has been implemented and code is available upon request.

The first time the Virtual Container Attestation code is instantiated on a machine, or when the machine is rebooted, the TPM initializes itself, extends the PCR values with those of selected operating system evaluations, and measurements of the Virtual Container Attestation code itself. These PCR values are then sent to a trusted third party, which gives the virtual machine monitor the ability to grand certificates.

At this point, an SSL service is opened on the machine, and a database initialized to hold the information about the containers. When a container is requested via SSL its configuration and requests are checked, the requests are parsed and removed from the config file, registered in the database as a VCA zone, and its attributes are logged. If the machine currently meets the requirements, keys and certs are created, and the enforcement policies are spawned to watch the zone. If any of the enforcement processes report an error, or access to the key is lost, the certificate is revoked and the entry in the database is marked as invalid.

Depending on the setup of the zone, future interactions may occur through the SSL connection, or may happen through a networked connection provided in the zone. Either way, a close eye is kept on the commands and monitor is persistent through the life of the zone.

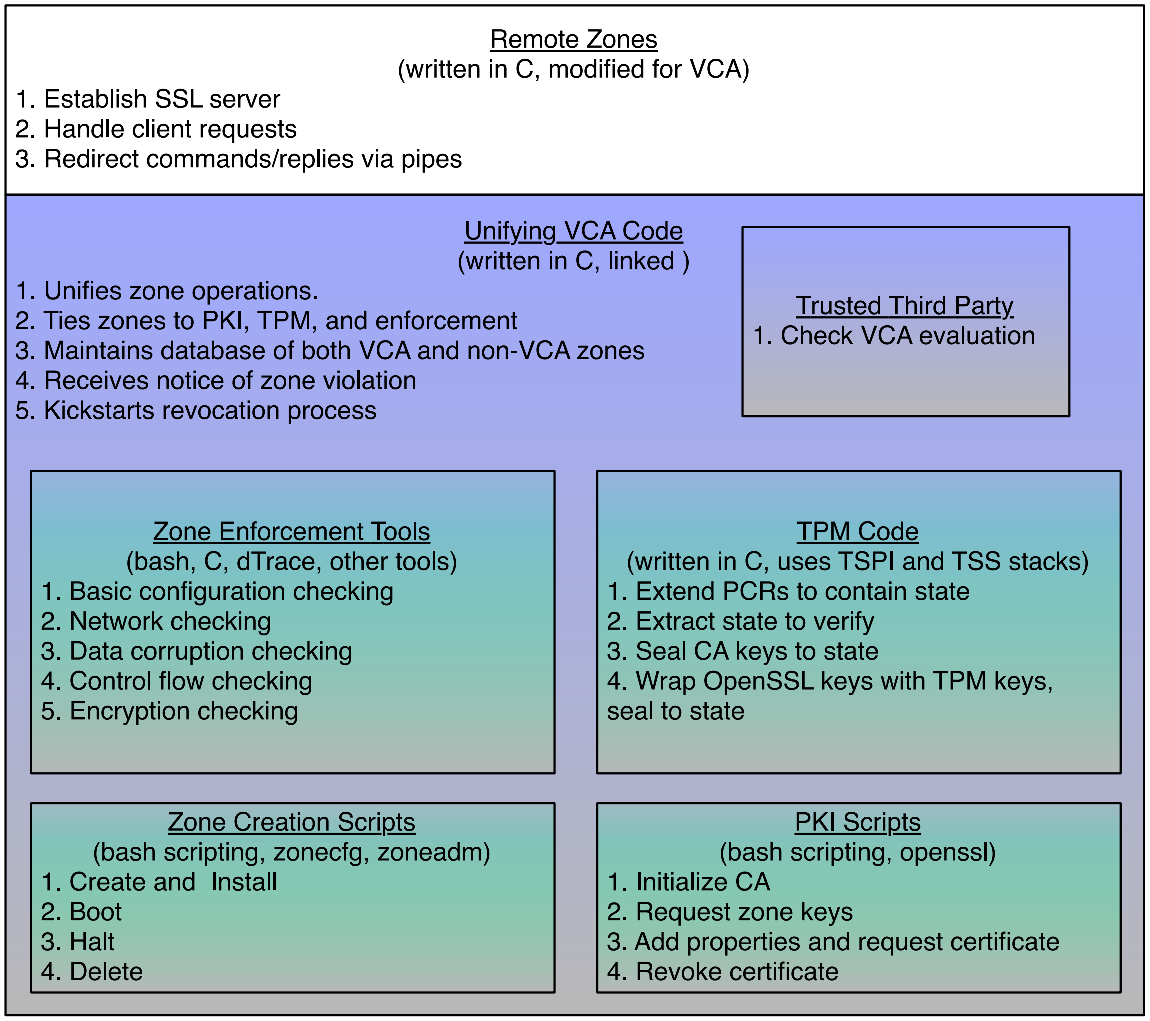

Remote Zones
(written in C, modified for VCA)
1. Establish SSL server
2. Handle client requests
3. Redirect commands/replies via pipes
Unifying VCA Code
(written in C, linked)
1. Unifies zone operations.
2. Ties zones to PKI, TPM, and enforcement
3. Maintains database of both VCA and non-VCA zones
4. Receives notice of zone violation
5. Kickstarts revocation process
Trusted Third Party
1. Check VCA evaluation
Zone Enforcement Tools
(bash, C, dTrace, other tools)
1. Basic configuration checking
2. Network checking
3. Data corruption checking
4. Control flow checking
5. Encryption checking
TPM Code
(written in C, uses TSPI and TSS stacks)
1. Extend PCRs to contain state
2. Extract state to verify
3. Seal CA keys to state
4. Wrap OpenSSL keys with TPM keys, seal to state
Zone Creation Scripts
(bash scripting, zonecfg, zoneadm)
1. Create and Install
2. Boot
3. Halt
4. Delete
PKI Scripts
(bash scripting, openssl)
1. Initialize CA
2. Request zone keys
3. Add properties and request certificate
4. Revoke certificate

Figure 4.5: The structure of our solution. The white sections are those which were available at the beginning of the project from Evan Tice. The rest of the project was written explicitly for this project.

Chapter 5

Testing

For the purposes of providing a reasonable thesis basis, we restricted ourselves to four main properties to enforce, and came up with a number of scenarios which mandated we check those properties. However, the results are extensible to more properties.

A short description of the testing and evaluation of our implementation follows.

5.1 Testing Applications

For this thesis, we have a handful of applications run on top of the Virtual Container Attestation tools, both to test the concept and to provide some performance measure. Each of the scenarios from section 2.2 became a test application for the evaluation of the project. Each of these applications tests at least two of the enforcement mechanisms that we have established.

Because of the closed nature of the power-grid, and the experimental nature of the smart grid, these applications were created specifically for the thesis. We were able to get our hands on one application currently in use, but it could not be adapted to the OpenSolaris platform. However, being able to see inside the application allowed us to tie our applications to real-world situations and code design, thus using that for more rigorous testing of our tools.

5.2 Evaluation of Virtual Container Attestation

To test the implementation of our Virtual Container Attestation tools, we considered the scenarios as described in 2.2, but also ran a number of other tests to ensure that the system behaved as expected.

1. We qualitatively evaluated the behavior of the system under a variety of creation requests (some rather far-fetched) and basic interactions within those zones.

2. We also tested whether the system would appropriately enforce the qualities we gave it, by creating zones and intentionally causing revocation scenarios.

3. In addition, we gave those same zones working input to established Virtual Container Attestation zones to ensure that under ideal conditions the zones would run well for an extended period of time.

The results of these tests are generally discussed in the next section,

Chapter 6

Results

6.1 Challenges to the Implementation

6.1.1 Solaris Containers

At the beginning of the project we believed that the Solaris Containers would provide a very lightweight form of virtual machine that would be ideal for the dynamic and flexible creation we had in mind for Virtual Container Attestation. Unfortunately, this turned out to not be the case. The Solaris containers go through a number of stages before they run, including *created*, *installed*, and *ready*. The installation phase, during which necessary operating system files are copied over, is far too long for our purposes here. On the order of about three to five minutes is a common installation time.

Despite our best efforts at looking into why this would be the case and what we could change to provide a more speedy interface, it turns out that there is little that can currently be done. We were able to achieve a marginal speedup using a clone option for the zones, which works well for closely-related zones. However, cloning zones is not ideal.

Over the course of some discussion with Sun, it became clear that Solaris containers have a faster startup than their OpenSolaris alternatives, and the capability for this to occur may move over to OpenSolaris at some point in the near future.

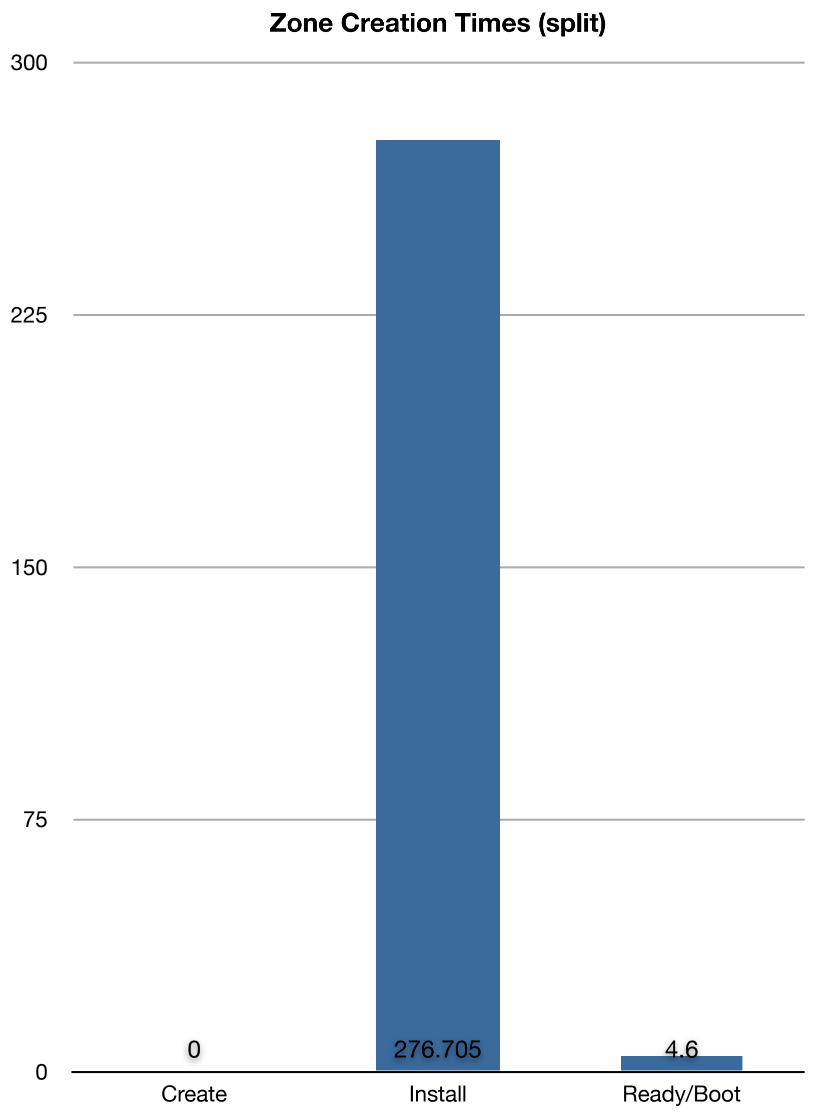


Figure 6.1: Time split for the creation times of various stages. Typical figures shown.

6.1.2 Communication with Zones

The communication with zones is still somewhat clunky, as we have to deal with the dual concerns of isolating the non-global zones from each other, controlling every movement and information exchange, against the concern of being able to communicate with the containers, specifically those for whom a network connection was disallowed. Currently, this is done via the built-in methods for doing so, but there are significant downsides to doing so, as it limits the style of command that can go through to the restricted zones.

6.1.3 Driver and Software Stack Difficulties

Because the TPM and other trusted computing notions are relatively recent to the OpenSolaris platform, there are a number of difficulties within the drivers and software stack relating to these issues.

Mainly, some exits are handled poorly, leading to residual bad state in the system, and the incapacitation of future operations. While these bugs have been reported to Sun, in the meantime I have done my best to avoid such instances. Unfortunately for the Virtual Container Attestation system, not all instances are avoidable, leading to unpredictable issues within the software.

6.2 Results of Evaluation

6.2.1 Creation Requests and Basic Event Handling

The overall implementation of Virtual Container Attestation was a success, leading to a successful and useful system, with a few caveats noted above. Alteration of the Remote Zones code given to us, as well as the addition of TPM and PKI capabilities led to smooth creation and command handling within the VCA framework.

Specifically, the capability to start the system using trusted boot, extend PCRs, initialize state, and accept clients happens without a hitch. Clients are successfully accepted, their requests are parsed and created, with appropriate properties. We have options to work from either installed zones or cloned zones, and both integrate well with the system.

Commands and replies also transpire without any problem: from the client's viewpoint, it appears as if one is interacting with a slightly different terminal, but with essentially the same capabilities.

Property-based identity certificates are provided with clear properties and elucidation about the specific requests included in the certificate itself. This simplified attestation improves vastly over previous solutions: a few lines of reading in the certificate make it clear what the status of the zone is. According to the policy implemented, revocation of those certificates is simultaneous with the shutdown of the zone.

See Figure 6.2 for a comparison of scalability within OpenSolaris and Xen. Clearly, OpenSolaris scalability far surpasses that of Xen. This is a great benefit in situations such as the power grid, wherein a single provider many want to talk with hundreds of clients at a given time, and have a reasonable response time. Note that depending on workload, numbers for both products may be somewhat lower.

6.2.2 Enforcement of Properties and Revocation

Enforcement being the main target of this approach, it received the most testing during the evaluation of the system. Property enforcement indeed works with this system, such that the enforcement continues for an indeterminate period of time, and halts the zone within a reasonable period of time of a violation, cascading into certificate revocation.

There is a performance hit associated with doing such enforcement, although it is relatively minor, due to the nature of DTrace. We did not specifically measure performance for this project.

The modular design of the enforcement means that additional properties can be evaluated relatively quickly. Unfortunately, this is not quite as flexible or dynamic as it could be—the

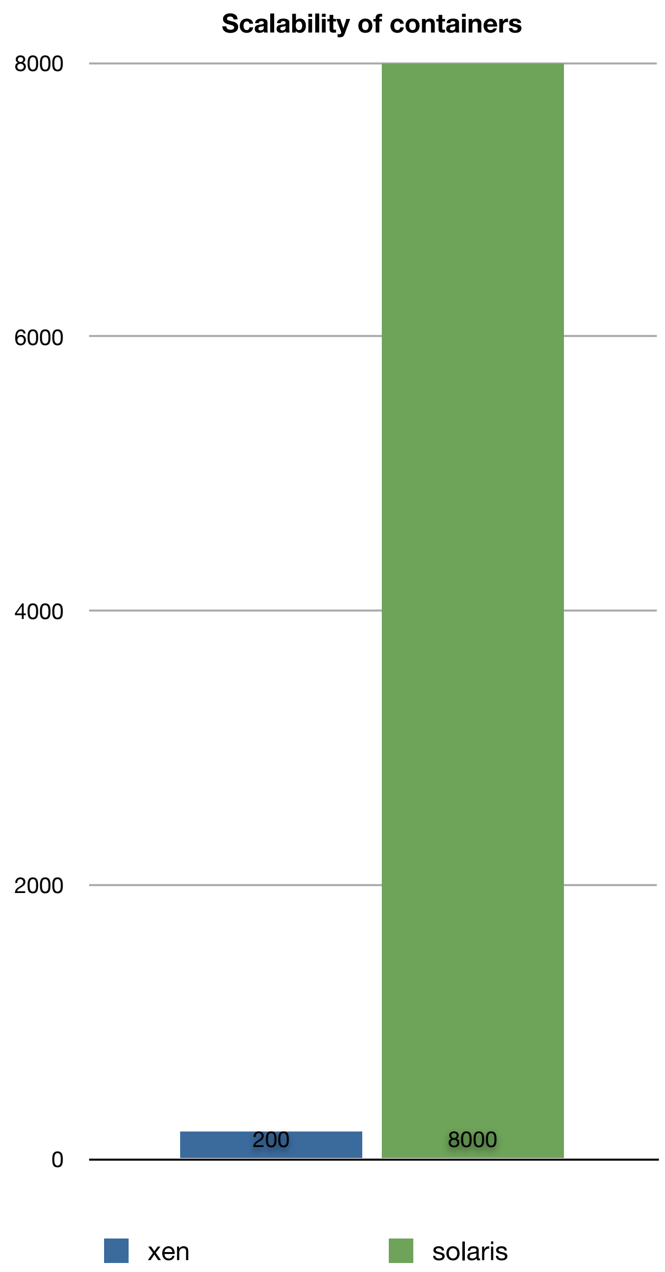


Figure 6.2: Scalability of OpenSolaris zones as opposed to Xen.

installation of modules is simple, but not brainless, as a handful of parameters must change.

6.2.3 Potential Security Weaknesses

Unfortunately, the Solaris isolation is not complete. While the global zone's window into non-global zone operations is useful for this project, it does provide a vulnerability if code running into the zones can gain access to the zone, through tricking the filesystem or some other method.

Additionally, DTrace does have some time of check vs time of use (TOCTOU) issues which makes it possible for each vulnerability to have a momentary window of freedom before the monitor detects the misuse of the zone.

6.2.4 Code Availability

Code and documentation are available upon request to katelin.a.bailey.10@alum.dartmouth.org

6.3 Future Work

As always, with research, there are areas not within the scope of this project which would be interesting, necessary, or otherwise good to investigate further or take the research in a different direction. We discuss these areas below.

6.3.1 Varied Revocation Scheme

For the purpose of this thesis, I restricted the revocation scheme to a particularly harsh one which revokes the certificate, denies access to the private key, and halts the compartment at fault. It would be interesting work to allow a more flexible revocation scheme, or allow communication with the client that attempts to resolve issues.

6.3.2 Additional Security Checks

There are a number of security checks that went unimplemented in this version of the tools that would be useful or interesting to have in future iterations, among them the ability to translate between human specifications and configuration files, and the ability to have a simple check of a configuration file to see if it matches with the host's security policy. While the latter is partially implemented, it is a very rough attempt, and a further expansion would be warranted.

6.3.3 Negotiation of Security

Even with the property-based specification of the security desired, there is likely to be some mismatch of desires between host and client. It would be wonderful to get a negotiation mechanism built into the initial SSL handshake to automate some of the back-and-forth necessary to come to an agreement.

6.3.4 Communication with Zones

As discussed previously, the communication with the zones is a bit clunky. Further research along these venues should address that issue within the kernel itself.

6.4 Conclusions

This thesis concludes with a working prototype of the design specified. It is, however, merely a prototype. Limitations in the technology provided, as well as areas of future work make it clear that such an implementation is not practical in use today. Nonetheless, if work continues in this direction, there is significant potential for the solution to become practical in the near future. At that point, we expect this solution to be more usable and secure than the alternatives currently available.

Acknowledgements

I would like to thank Sean W. Smith for serving as my adviser and answering my many questions, as well as Evan Tice and John Baek for beginning this work. Thanks are also due to Anna Shubina, Wyllys Ingersol, and the rest of the team at SUN who provided support with OpenSolaris and TPM drivers throughout the lifetime of the project.

Additionally, I would like to acknowledge the grad students, faculty, and staff in the PKI/Trust lab who have been both extremely supportive and excellent examples for me as I proceeded from a naive young freshman to a future graduate student doing independent work. The community has shaped me both academically and personally.

This project was supported by Sun, the National Science Foundation under Grant No. CNS-0524695 and by the Department of Energy under Award No. DE-OE0000097. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of Sun, the National Science Foundation or the Department of Energy.

Bibliography

[1] http://openpdc.codeplex.com/.

[2] http://www.trustedcomputinggroup.org/.

[3] John Baek. Trusted Container on Demand. 2007.

[4] John Baek and Sean W. Smith. Preventing theft of quality of service on open platforms. In *First IEEE/CREATE-NET Workshop on Security and QoS in Communication Networks (IEEE/CREATE-NET SexQoS 2005)*, Athens, Greece, September 2005.

[5] Fabrizio Baiardi, Diego Cilea, Daniele Sgandurra, and Francesco Ceccarelli. Measuring Semantic Integrity for Remote Attestation. In *Second International Conference on Trusted Computing and Trust in Information Technologies, TRUST 2009*, pages 240–263, Oxford, UK, April 2009.

[6] David J. Clark, editor. *Dynamics of a Trusted Platform: A building block approach.* Intel Press, 2008.

[7] Paul England. Practical Techniques for Operating System Attestation. In *First International Conference on Trusted Computing and Trust in Information Technologies, TRUST 2008*, pages 1–13, Villach, Austria, March 2008.

[8] Eimear Gallery, Aarthi Nagarajan, and Vijay Varadharajan. A Property-Dependent Agent Transfer Protocol. In *Second International Conference on Trusted Computing*

and Trust in Information Technologies, TRUST 2009, pages 240–263, Oxford, UK, April 2009.

[9] T. Garfinkel, B. Pfaff, J. Chow, M Rosenblum, and D Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of Symposium on Operating System Principles*, October 2003.

[10] T. Garfinkel, M Rosenblum, and D Boneh. Flexible OS Support and Applications for Trusted Computing. In *9th Hot Topics in Operating Systems (HOTOS-IX)*, 2003.

[11] M Gasser, A Goldstein, C Kaufman, and B Lampson. The digital distributed system security architecture. In *Proceedings of the 12th NIST-NCSC National Computer Security Conference*, pages 305–319, 1989.

[12] JohnL Griffen, Trent Jaeger, Ronald Perex, Reiner Sailer, Leedert van Doorn, and Ramon Caceres. Trusted virtual domains: Toward secure distributed services. In *1st IEEE Workshop on Hot Topics in System Dependability*, Yokohama, Japan, June 2005.

[13] Information and Ontario Canada Privacy Commissioner. *SmartPrivacy for the Smart Grid: Embedding Privacy into the Design of Electricity Conservation*, November 2009.

[14] S. Ioannidis, S. M. Bellovin, J. Ioannidis, A. D. Keromytis, and J. M. Smith. Design and implementations of Virtual Private Services. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 269–274. IEEE Computer Society, June 2004.

[15] B Lampson, M Abadi, M Burrows, and E Wobber. Authentication in distributed systems: Theory and practice. 10(4):265–310, 1992.

[16] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The inevitability of failure: The flawed assumption of security in mod-

ern computing environments. In *Proceedings of National Information Systems Security Conference*, pages 303–314, October 1998.

[17] John Marchesini, Sean Smith, Omen Wild, and Rich MacDonald. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love the Bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, December 2003.

[18] John Marchesini, Sean Smith, Omen Wild, Josh Stabiner, and Alex Barsamian. Open-Source Applications of TCPA Hardware. In *20th Annual Computer Security Applications Conference,*, December 2004.

[19] John Marchesini and Sean W. Smith. Virtual Hierarchies- An Architecture for Building and Maintaining Efficient and Resilient Trust Chains. In *NORDSEC2002 - 7th Nordic Workshop on Secure IT Systems*, November 2002.

[20] Hiroshi Maruyama, Frank Seliger, Nataraj Nagaratnam, Tim Ebringer, Seiji Munetoh, Sachiko Yoshihama, and Nakamura Taiga. Trusted platform on demand. IBM Technical Report RT0564, IBM, February 2004.

[21] Chris Mitchell, editor. *Trusted Computing*, volume 2 of *IEE Professional Applications of Computing Series 6*. Institution of Electrical Engineers (IEE), 2005.

[22] Mohammad Nauman, Masoom Alam, Xinwen Zhang, and Tamleek Ali. Remote Attestation of Attribute Updates and Information Flows in a UCON System. In *Second International Conference on Trusted Computing and Trust in Information Technologies, TRUST 2009*, pages 240–263, Oxford, UK, April 2009.

[23] J Poritz, M Schunter, E. V. Herreweghen, and M. Waidner. Property Attestation-scalable and privacy-friendly security assessment of peer computers. Research Report RZ3548, IBM, May 2004.

[24] Ahmad-Reza Sadeghi and Christian. Stuble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms,*, pages 66–77, New York, NY, USA, 2005. ACM Press.

[25] Reiner Sailer, X Zhang, T Jaeger, and Leedert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th Usenix Security Symposium*, pages 223–238, San Diego, CA, Aug 2004. USENIX.

[26] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *SOSP '07: Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pages 335–350, New York, NY, USA, 2007. ACM.

[27] Sean W. Smith. Outbound authentication for programmable secure coprocessors. *International Journal of Information Security*, 3(1):28–41, October 2004.

[28] Evan Tice. Remote Zones. June 2009.

[29] Utility Communications Specification Working Group. *IEC 870-6-503 TASE.2 Services and Protocol.*