Trusted Virtual Containers on Demand*

Katelin A. Bailey†
University of Washington
kabailey@u.washington.edu

Sean W. Smith
Dartmouth College
sws@cs.dartmouth.edu

ABSTRACT

TPM-based trusted computing aspires to use hardware and cryptography to provide a remote relying party with assurances about the trustworthiness of a computing environment. However, standard approaches to trusted computing are hampered in the areas of scalability, expressiveness, and flexibility. This paper reports on our research project to address these limitations by using TPMs inside OpenSolaris: our kernel creates lightweight containers on demand, and uses DTrace and other tools to extend attestation to more nuanced runtime properties. We illustrate this work with prototype application scenarios from cyber infrastructure operating the U.S. power grid.

Categories and Subject Descriptors

D.4.6 [**Software**]: Security and Protection

General Terms

Security

1. INTRODUCTION

TPM-based trusted computing aspires to use hardware and cryptography to provide a remote relying party with assurances about the trustworthiness of a computing environment. However, standard approaches to trusted computing are hampered in three areas:

- *Scalability.* How quickly can a new trusted environment be establish? How many trusted environments can a single machine host?
- *Expressiveness.* The basic TCG architecture supports attestation of basic software and hardware configuration. What about more dynamic properties of runtime behavior?
- *Flexibility.* Is the relying party constrained by the attributed a server provides a priori for a trusted environment—or can she negotiate for the suite of attributes important to her?

This paper reports on our research project to address these limitations by using TPMs inside OpenSolaris.

- *Scalability.* We use the construct of Solaris *containers* as a basis for lightweight virtualized userland environments; our trusted kernel creates these containers and generates certified keypairs for them, rooted in the TPM.
- *Expressiveness.* We use the ability that our trusted kernel lives within the same OS as the containers to extend attestation to more general runtime properties; in particular, our kernel can use the Solaris *DTrace* tool to monitor more nuanced aspects of behavior, and to revoke a container's certificate if it departs from the appropriate specifications.
- *Flexibility.* On demand from remote clients, our kernel creates containers according to a suite of trust attributes the client and server find mutually acceptable.

We illustrate this work with prototype application scenarios from cyber infrastructure operating the U.S. power grid.

Section 2 discusses the standard approach to trusted computing, some potential shortcomings, and work to address these. Section 3 discusses the building blocks for our project. Section 4 discusses example application scenarios. Section 5 presents our solution. Section 6 presents more detail about our enforcement mechanisms. Section 7 evaluates scalability. Section 8 concludes.

This submission is an adaption of the first author's thesis [3]. A concurrent (and largely orthogonal) project on Solaris and trusted computing also appears in this conference [11].

*This work supported in part via ISTS by DHS, under grant 2006-CS-001-000001, by Sun, by the NSF under grant CNS-0524695, and by DOE under award DE-OE0000097. Views are of the authors alone.

†This paper reports work done while the author was at Dartmouth.

Clipart in Figures 2-6 courtesy Florida Center for Instructional Technology, `http://etc.usf.edu/clipart`

STC'10, October 4, 2010, Chicago, Illinois, USA.

2. TRADITIONAL TRUSTED COMPUTING, AND SHORTCOMINGS

2.1 Basic Approach

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

Trusted hardware, being (perhaps) the lowest level of base assumption one can make, is often the go-to method for establishing a root of trust for the machine and subsequent operations. In the standard TCG approach, a *Trusted Platform Module* participates in measuring (via cryptographic hashes) the software and hardware configuration of a machine, and using protected keypairs to make statements about current configuration and to bind secrets to it.

2.2 Addressing the Shortcomings

Thus, the initial approach to attestation-based trusted computing leads to the problems of *what* and *how much* and *how* to evaluate the system—and how to reconcile the trusted system a party provides with the requirements of a remote relying party. Subsequent research explored ways to solve this problem.

Towards Expressiveness.

Several researchers proposed ways to improve the expressiveness of basic attestation, beyond raw configuration information.

Sadeghi and Stüble [18] and others [16] proposed *property-based attestation*. 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.

In contrast, Haldar et al. [8] proposed basing attestation on programming language semantics. Nauman et al. propose basing it on the UCON usage control model [15]. Projects such as *SecVisor* [19] use the newer LaGrande/Presidio-style CPUs and resettable PCRs to provide periodic configuration measurement. England [4] speculates on approaches such as certifying OS policy or creating a new "birth certificate" at each boot—or looking at virtual machines.

In our own lab, our initial work on integrating TPMs and Linux [12] tried to bind attestation to a manifest of currently trusted configuration and update policy. Our later *compartmented attestation* work [2, 1] tied attestation to an SELinux security policy—a tool which proves too complex to be usable.

Towards Scalability.

The traditional approach to attestation uses hardware to testify the configuration of a single machine. Researchers have explored getting more trusted platforms per machine by gluing trusted computing to *virtualization* (e.g., [6, 5]). 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. *Trusted virtual domains* extends the trusted environment across multiple machines [7].

However,these approaches 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. They also do not account for the still-monstrous attestation needed when each virtual machine has its own software stack needing verification.

(The recently announced work of Löhr et al [11], noted earlier, pursues Solaris Containers/Zones as a solution to scalability, as do we.)

Towards Flexibility.

Virtualization alone still leaves open the question of how to configure the virtual machine. *Trusted Computing on Demand* operates under the philosophy that we should only provide the attestation for those tasks/environments which call for it [13]. 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. 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.

3. OUR BUILDING BLOCKS

Our project [1, 23, 3] seeks to address these shortcomings by using Solaris Containers/Zones [17] as a lightweight vehicle (addressing scalability), using monitoring tools such as DTrace [14, 22] to measure more nuanced aspects of trustworthiness (addressing expressiveness), and creating these on demand by remote clients (addressing flexibility). (The literature uses the terms "Containers" and "Zones" almost synonymously, although we've been told that "container" is the more appropriate term, encompassing "zone" and related machinery.)

3.1 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

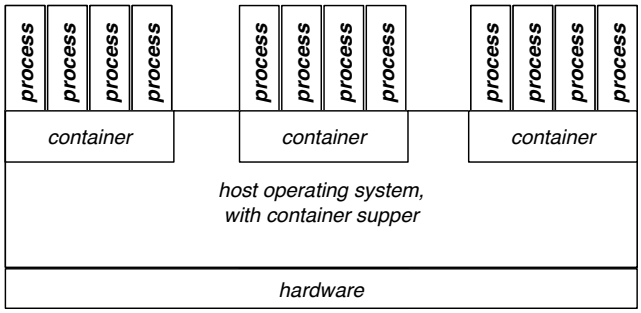

Figure 1: Paenevirtualization

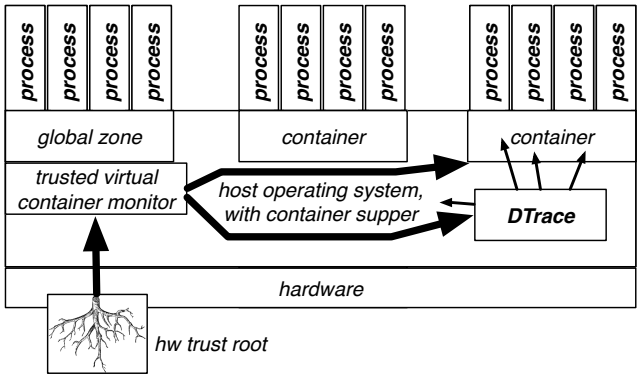

Figure 2: Our idea: the hw root of trust measures the monitor, which in turn measures containers and (via tools such as DTrace) container behavior.

OpenSolaris, meaning that we can delve into the operating system if required. For this project, we 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 us 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 Containers/Zones

In traditional virtualization, guest OS instances live on top of a a virtual machine monitor, which itself lives either directly on top of the hardware (*Type I*) or on top of a host OS (*Type II*). These approaches permit each guest OS to be different from each other and different from the host OS (if one exists), but at the cost of significant software and memory usage for each instance–and increased opacity of each instance.

In contrast, Figure 1 shows *OS-level virtualization* (termed by some researchers as *paenevirtualization* [21]). Solaris Containers/Zones are an example. A single kernel is shared among all of the zones. Applications thus see a guest container as a standalone system, but only one 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.

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 good 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.

3.3 Solaris DTrace

Another main reason for using Solaris is DTrace, [14, 22] lauded on *Slashdot* as "the one true tool" As a dynamic tracing method, DTrace is invaluable; it provides a lightweight way for appropriately privileged code to monitor execution of other processes (both at user-level and kernel-level). Starting with Solaris 10, kernel code comes pre-instrumented with *providers* that publish information to the DTrace subsystem, but are structured to have minimal performance impact in general—and particularly when no consumers are interested. The programmer can provide DTrace *scripts* that specify predicates for interesting providers (including their execution context, such as which process triggered them), and code to be executed when these predicates are satisfied. Nearly every function in kernel-land has a DTrace provider at entrance and exit; triggered operations include things such as examining both the user stack and kernel stack of the relevant process.

Thus, DTrace gives us an easy way for our monitor code (living as user-level code in the global zone) to measure fairly arbitrary dynamic behavior, even at the kernel level, of our created zones. In our system, the TPM statically measures our core monitor code, which then creates zones and measures their behavior continually (via DTrace providers).

Figure 2 shows our general approach.

4. APPLICATION SCENARIOS

For motivating application scenarios, we looked to the power grid cyberinfrastructure, where numerous somewhat distrustful entities need to quickly create trustworthy environments on each other's machines.

Our idea here was to identify situations where parties might require rapid and numerous creation of environments (hence the need for scalability) whose trustworthiness requirements may vary by relying party (hence flexibility) and which cannot necessarily be characterized solely by measurement of code (hence expressiveness).

4.1 Power Grid Cyberinfrastructure

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

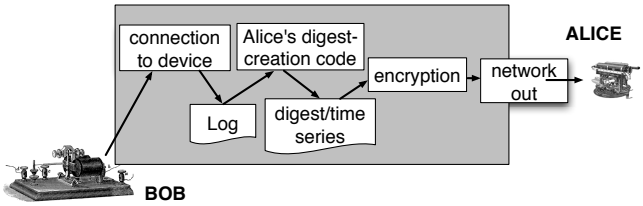

Figure 3: Scenario 1

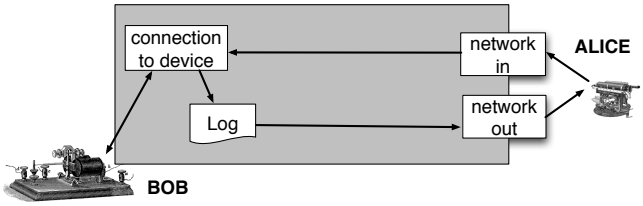

Figure 4: Scenario 2

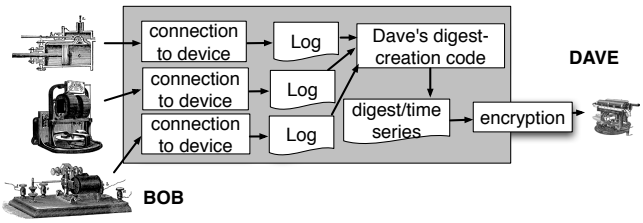

Figure 5: Scenario 3

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.

We drew on the *Inter-Control Center Communication Protocol (ICCP)* [9] and *openPDC* `http://openpdc.codeplex.com/` for Scenario 1 through Scenario 4 below, each based on actual examples of code or power grid interactions.

4.2 Scenario 1: Digests of Remote Device Logs

In Scenario 1 (Figure 3), 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.

4.3 Scenario 2: Control of Remote Device

In Scenario 2 (Figure 4), Alice and Bob are companies at the same level of power organization (e.g., 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.

4.4 Scenario 3: Audit Remote Machine

In Scenario 3 (Figure 5), 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.

4.5 Scenario 4: Smart Grid Aggregation

In Scenario 4 (Figure 6), 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.

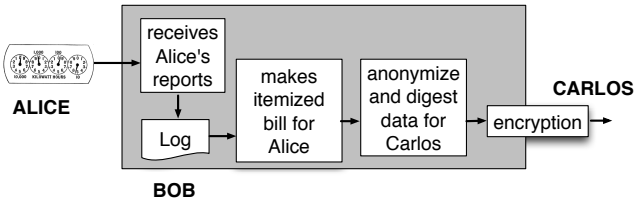

Figure 6: Scenario 4

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.

4.6 Trust Attributes

For these scenarios, we identified a number of attributes which the relying parties might require for trustworthiness—and which would demonstrate a range of properties more general than mere software configuration.

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 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.

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. [10] 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.

Table 1 summarizes the attributes and scenarios. (Section 6 discusses how we measure/enforce these attributes.)

	Digest	Control	Audit	Smart Grid
A. Network connection: no "abnormal" connections, no destinations outside whitelist.	X	X		
B. Data integrity: userland structures modified only by correct jobs, maintain invariants	X	X	X	X
C. Safe encryption: used in appropriate places, not used in inappropriate ones	X		X	X
D. Config restrictions: run-time specs, mount points, cpu time, no network out, etc.		X		
E. Control flow: jobs originate from normal sources, control flow follows spec			X	X

Table 1: Summary of trust attributes we considered for these scenarios

5. OUR SOLUTION

Our solution (Figure 7) incorporates *virtualization* and *trusted computing on demand*; it runs on OpenSolaris and provides a networked interface for lightweight virtual machines 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.

Our trusted virtual container monitor has five main functions:

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 our TVCon-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 TVCOn-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.

Certificates.

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 monitor, allowing the zone monitor to act as a certificate authority if it passes inspection.

This zone montior 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

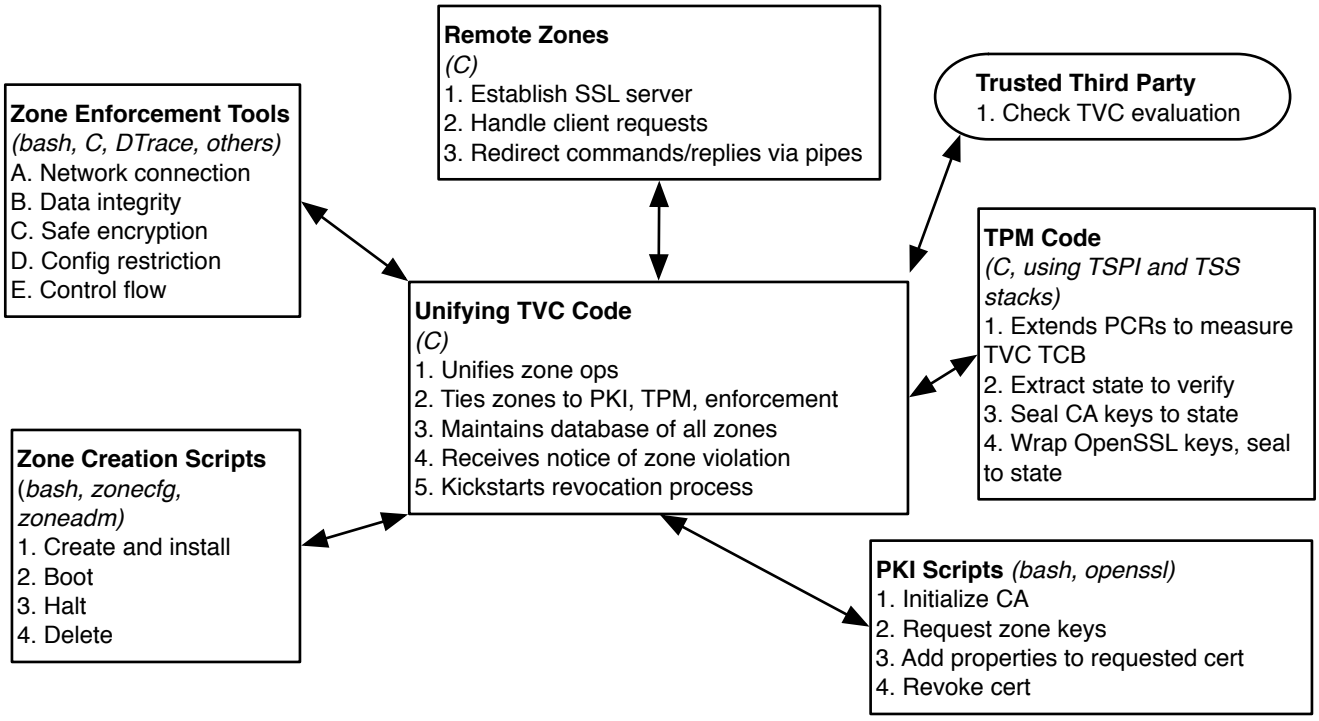

Figure 7: Our overall software structure.

be accessed. The zone monitor may then issue certificates to the individual zones, based on their attributes.

We give each protected zone its own certified keypair, so it can participate as a first-class citizen in broader cryptographic protocols. However, questions arise regarding how long such a keyholding entity should live (across reboots? across code updates) and how long the official key-entity binding should last (after zone death?). ([20] gives a more exhaustive analysis of such issues.) For this prototype, we chose the conceptually simplest approach: when the zone halts, the entity dies and the certification of key to entity is revoked.

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. The keys are stored in TPM volatile key storage, under the keys granted to the TPM and the zone monitor. The key is wrapped to the same value as the zone monitor, 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 monitor, which contains a list of attributes.

TVCon Monitor.

At the core of the project lies our TVCon Monitor (TVCM). At a basic utilitarian level, the TVCM acts as a coordinator for the trusted virtual containers. It receives and processes requests for creation and removal of containers, and it acts an intermediate between any container interaction and the container itself. It is, in short, a buffer between the containers and external influences. The monitor 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.

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 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

Operations.

Figure 8 shows the overall operations of our system. Figure 9 shows how it handles client requests. Figure 10 annotates the Solaris zone lifecycle with the additional operations we added.

6. MEASUREMENT TOOLS

A series of utilities run at the same time as the main TVCM 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.

Current zone enforcement mechanisms range from the sim-

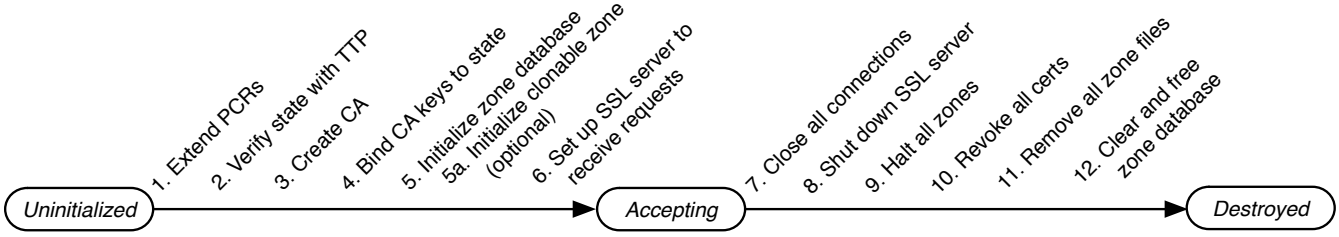

Figure 8: Establishment and teardown of global state in our system.

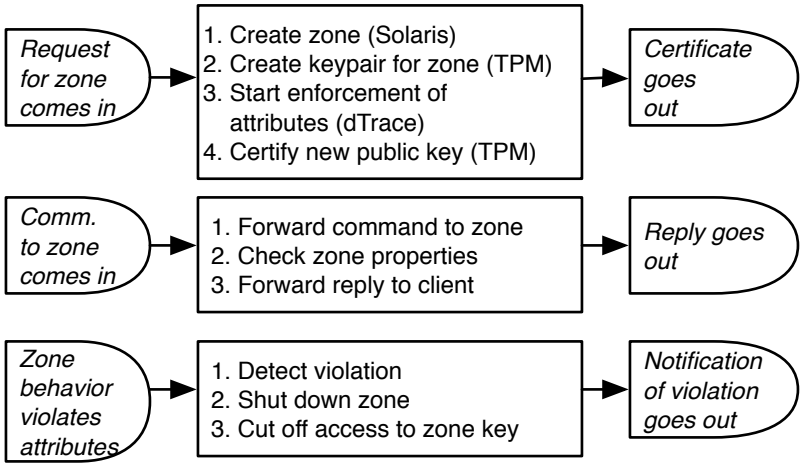

Figure 9: What our server does in response to runtime events

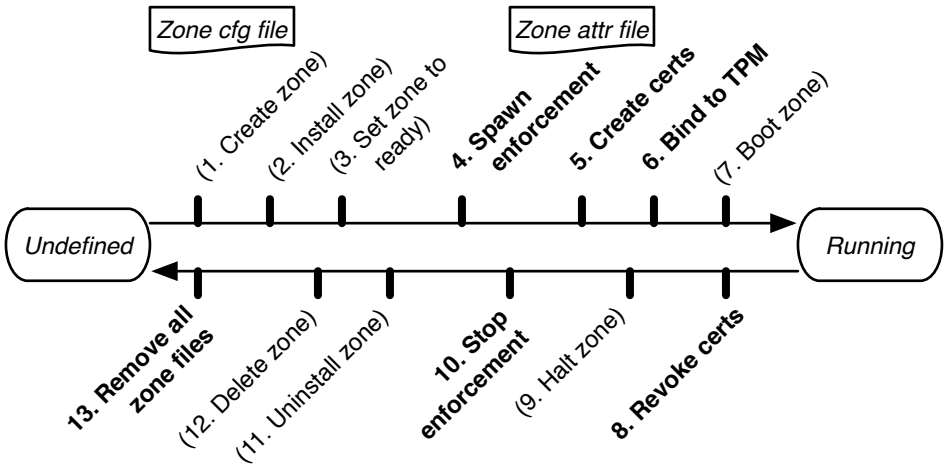

Figure 10: Our extensions (in boldface) to standard Solaris zone/container lifecycle.

plest checking of the zone config files to the more complicated ones involving external checks.

Zone config file.

We ran zone configuration checks run prior to zone creation. These 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. Our implementation consisted of a simple check of the configuration file, and did not require persistent enforcement checks—rather, we had a script run through and ensure that, for example, the zone lived in an appropriate part of the filesystem such that it could not grab additional data. Since configuration files are very simple, this is a simple matter of checking a handful of properties. (The content of such checks was intentionally left flexible, since specifying a percentage of CPU usage or max size of footprint and whatnot are also possible for zones, and this would be quite useful to take advantage of, especially on high-throughput servers.)

Network Connectivity.

We ran 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.

Control Flow.

Control Flow checking 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; this task was complicated by the fact that the industry does not like to share real code.

We implemented control flow checking via DTrace. It's a relatively simple methodology to check that program B gets called by program A (or even just "after" program A) with the system probes in DTrace. We used probes to track the syscalls for process execution: fork, exec, etc. As soon as we recognized program A, we'd store the fact that it'd been called, and continue checking for B. Once B was called, we check either the store to see if A has already run, or we check the caller—-depending on how strict we're begin with enforcement. The former can be stored in a single variable; the latter is included information in the DTrace probe publishing. This approach should be easily extensible to constraints of the form "don't let program D run during this period of execution."

With more complex flows, we need to specify constraints such as program C runs after program B which runs after program A. This increases the complexity rapidly, but is relatively easy to specify for DTrace by running an additional trace for each level of complexity. The entire set-up can exit as soon as one program mis-steps and goes out of order.

Data Corruption.

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.

Data corruption checks ran in two different methods. One existed much like control flow, and checked which processes accessed which files when by probing into the filesystem calls and checking the callers against the list provided. The other method ran actual diffs on the data during times when execution should be happening but data-changing should not be happening; this latter approach incurred higher overhead but offered more assurance. Imagine an interleaving such as this: program A modifies file M, then no one touches file M or file N until program B reads in both together. The accesses can be checked by the same methodology as control flow (above) and doing a diff at point A and point B: the frequency of checks would be mandated by the frequency of accesses. (However, we found that we did not need to use diffs all that much, since DTrace did the job.)

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.

For external encryption, we inspected network traffic, and for internal encryption, we used DTrace to track the usage of known encryption methods on the computer: we set a hook onto openssl for example, and tracked which processes inside a zone employed the openssl tools. The main vulnerability here is that if there is an unknown implementation of encryption used, it would be difficult for DTrace or other runtime tools to catch it.

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.

There is a performance hit associated with doing such enforcement, although it is rela- tively minor, due to the nature of DTrace (intended to have "negligible" impact [22]). We did not specifically measure DTrace performance impact for this project—and the potential certainly exists for pathologically bad impact.

Certifying Properties.

For each of these properties, we added a property field (via "extensions") into the openSSL X.509v3 indentity certificates created by the monitor. The entry into the field specified what enforcement was provided for the property—for example "NETWORK: no traffic" or "ENCRYPTION: all outgoing." These phrases should match the phrases used to specify the enforcement requested, and are a small set of specified words. For the purposes of this project, the certificates did not mandate code to exist on the other end to interpret the properties (so we could tell which were succeeding), although this could ostensibly be changed in future iterations of the project and make the properties more heavily enforced on the other end, rather than rely on humans noticing.

The Global Zone.

In our system, the global zone became part of the TCB; we did not run enforcement processes in the global zone itself. (There are few times when the global zone, being all-powerful, cannot do exactly what it wants.) However, wed did run hooks into the zone commands (such as start, stop, login, etc) to make sure no one interfered with the zones we had running. The last is actually quite important, as it's the only convenient method of giving commands to the non-global zones.

7. SCALABILITY EVALUATION

Figure 11 shows how we did with respect to scalability.

In terms of trusted platforms per machine,e paenevirtualization scalability far surpasses that of Xen. Sun reports over 8000 Solaris zones per machine; Figure 11 gives the more conservative paenevirtualization measurement reported by Soltesz et al [21] running Apache 2.0.46 as the workload, on Xen 2.0.7 on a patched 2.6.12 XenoLinux versus Vserver 2.0.1 on Linux 2.6.12 on a 3.06Ghz Leon with 4GB of RAM. 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.

However, in terms of time of creation, zone *installation* time was unacceptably long. (Figure 11 reports measurements we took on an Intel Core 2 Duo CPU T9600, 2.80GHz, with 4096MB of RAM.) It appears that most of this time is due to checking software packages for update (unnecessary, in our scenarios)—and it turns out that there is little that can currently be done without more substantive changes to the OS. 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.

In situations where clients merely wish to boot an instance of a zone that has already been installed, the status quo will be fine.

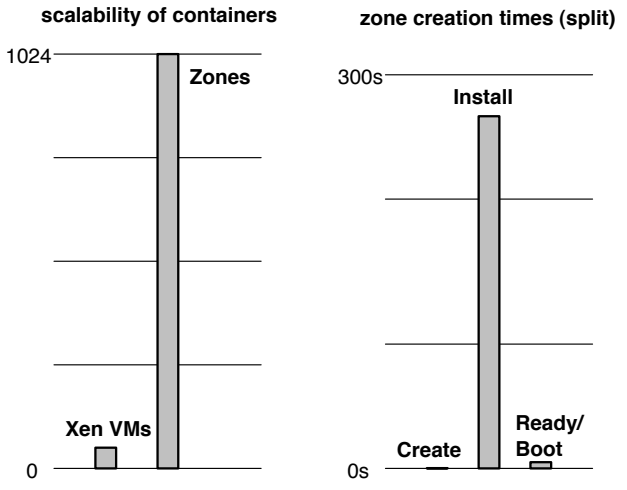

Figure 11: Scalability of zones: the instances per machine is good (left), but current installation times are disappointing (right).

8. CONCLUSIONS AND FUTURE WORK

Our overall implementation 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 TVCon 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.

Several areas suggest themselves for future work. The primary one is reducing the time necessary for container installation. Others include exploration of less drastic approaches about what to do when a container misbehaves (e.g., merely suspend its access to its key, rather than revoking its certificate), providing for inter-container communication (our goal had been isolation), and exploring more thoroughly the effectiveness of our enforcement measurements against actively malicious code. (In particular, it is rumored that DTRace may have a TOCTOU issue.) Another area would be to incorporate negotiation of trusted container properties with the ciphersuite negotiation already happening with SSL. We also look forward to integrating this work with the concurrent Solaris zone project from our colleagues at Bochum [11]. Another natural area of future work would be to carry out penetration tests to evaluate the effectiveness of the enforcement mechanisms.

Of course, besides considering future work pertaining to the base technology itself, we also look forward to integrat-

ing this work into actual power grid cyberinfrastructure—provided we can find industry players to partner with us on such pilots.

Acknowledgments

We thank John Baek, Anna Shubina, Evan Tice and the anonymous referees for their help with this project.

9. REFERENCES

[1] John Baek. Trusted Container on Demand: Flexible Confinement for Compartmented Attestation. Ph.D. proposal, Dartmouth College, November 2006.

[2] 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.

[3] Katelin Bailey. Virtual Container Attestation: Customized Trusted Containers for On-Demand Computing. Technical Report Computer Science TR2010-674, Dartmouth College, June 2010. Senior high honors thesis.

[4] Paul England. Practical Techniques for Operating System Attestation. In *Second International Conference on Trusted Computing and Trust in Information Technologies, TRUST 2009*, Oxford, UK, April 2009.

[5] 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.

[6] 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.

[7] John 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.

[8] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic Remote Attestation—A Virtual Machine Directed Approach to Trusted Computing. In *3rd Virtual Machine Research and Technology Symposium (USENIX VM)*, pages 29–41, May 2004.

[9] IEC Utility Communications Specification Working Group. *Telecontrol Equipement and Systems: Telecontrol Protocols Compatible with ISO Standards and ITU-T Recommendations- TASE.2 Services and Protocol (2nd Edition)*. International Electrotechnical Commission, 2002.

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

[11] Hans Löhr, Thomas Pöppelmann, Johannes Rave, Martin Steegmanns, and Marcel Winandy. Trusted Virtual Domains on OpenSolaris: Usable Secure Desktop Environments. In *Scalable Trusted Computing*, 2010.

[12] 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.

[13] 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.

[14] R. McDougall, J. Mauro, and B. Gregg. *Solaris Performance and Tools*. Prentice Hall, 2007.

[15] 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.

[16] 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.

[17] Daniel Price and Andrew Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *18th Large Installation System Administration Conference*, November 2004.

[18] 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.

[19] 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.

[20] Sean W. Smith. Outbound Authentication for Programmable Secure Coprocessors. *International Journal of Information Security*, 3(1):28–41, October 2004.

[21] Stephen Soltesz, Marc E. Fiuczynski, Larry Peterson, Michael McCabe, and Jeanna Matthews. Virtual Doppelgänger: On the Performance, Isolation, and Scalability of Para- and Paene- Virtualized Systems. March 2006.

[22] Sun Microsystems. *Solaris Dynamic Tracing Guide*, January 2005.

[23] Evan Tice. Remote Zones. Functional Tutorial, Dartmouth College, June 2009.