Two-Person Control Administration: Preventing Administration Faults through Duplication

Shaya Potter, Steven M. Bellovin and Jason Nieh
Department of Computer Science
Columbia University
{spotter, smb, nieh}@cs.columbia.edu

Abstract

Modern computing systems are complex and difficult to administer, making them more prone to system administration faults. Faults can occur simply due to mistakes in the process of administering a complex system. These mistakes can make the system insecure or unavailable. Faults can also occur due to a malicious act of the system administrator. Systems provide little protection against system administrators who install a backdoor or otherwise hide their actions. To prevent these types of system administration faults, we created ISE-T (I See Everything Twice), a system that applies the two-person control model to system administration. ISE-T requires two separate system administrators to perform each administration task. ISE-T then compares the results of the two administrators' actions for equivalence. ISE-T only applies the results of the actions to the real system if they are equivalent. This provides a higher level of assurance that administration tasks are completed in a manner that will not introduce faults into the system. While the two-person control model is expensive, it is a natural fit for many financial, government, and military systems that require higher levels of assurance. We implemented a prototype ISE-T system for Linux using virtual machines and a unioning file system. Using this system, we conducted a real user study to test its ability to capture changes performed by seperate system administrators and compare them for equivalence. Our results show that ISE-T is effective at determining equivalence for many common administration tasks, even when administrators perform those tasks in different ways.

1 Introduction

As computing systems become more complex, they have also become harder to administer. From a security perspective, these complex systems create an environment that is easier for rogue users, be they inside or outside attackers, to make changes to the system that hide their malicious attacks. For instance, Robert Hanssen, an FBI agent who was a Soviet spy, was able to evade detection because he was the system administrator for some of the FBI's counterintelligence computer systems [26]. This allowed him to determine if the FBI had identified his drop sites and if he was the subject of investigation [5].

Insider attacks have long been known to be very difficult to address. Most approaches involve intrusion detection or role separation. However, both are ineffective against rogue system administrators who can replace the system module that enforces the separation or performs the intrusion detection. This attack vector was described over thirty years ago by Karger and Schell [13] and still remains a serious problem.

Even if administrators can be trusted not to be malicious, they must deal with software that is very complicated. Mistakes can be easy to make and hard to identify before they cause problems. These mistakes can affect both the stability of the system and its security. A mistake that takes down an important service can prevent the machine from being usable or further administered, and can even let malicious attackers access the machine with impunity.

There are several approaches for preventing and recovering from faults that creep into a system, including partitioning, restore points, and peer review. One of the most effective approaches is two-person control [1]. This can be provided by having two pilots in an airplane, requiring two keys for a safe deposit box, or running two or more computations in parallel and comparing the results for a fault-tolerant computer system. We believe this concept can be extended to address problems in system administration by leveraging virtualization to create duplicate environments.

Toward this end, we created the "I See Everything Twice" [10] (ISE-T, pronounced "ice tea") architecture. ISE-T provides a general mechanism to clone execution environments, independently execute computations that

modify the clones, and compare how the resulting modified clones have diverged. The system can be used in a number of ways, such as performing the same task in two initially identical clones, or executing the same computation in the same way in clones with some differences. By providing clones, ISE-T creates a system where computation actions can be "seen twice", applying the concept used for fault-tolerant computing to other forms of two-person control systems. There is a crucial difference though between our approach for using replicas and replicas as used in fault-tolerant computing. Our goal is to compare for equivalence between two replicas that may not be completely identical, rather than simply run two identical replicas in lock step and ensure they remain identical.

We apply ISE-T's principle to change the way we administer machines to provide two-person control administration. As ISE-T allows a system to be easily cloned into multiple distinct execution domains, we can create separate clone environments for multiple administrators. ISE-T can then compare the separate set of changes produced by each administrator for equivalence to determine if the same changes were made. By comparing the sets of changes for equivalence, ISE-T improves management by allowing it to be done in both a fail-safe and auditable manner.

In ISE-T, we force administrative acts to be performed multiple times before it is considered correct. Current systems give full access to the machine to individual administrators. This means that one person can accidently or maliciously break the system. ISE-T's ability to clone an execution environment creates a new way to administer machines to avoid this problems. ISE-T does not allow any administrator to modify the underlying system directly, but instead creates individual clones for two administrators to work on independently. ISE-T is then able to compare the changes each administrator performs. If the changes are equivalent, ISE-T has a high assurance that the changes are correct and will commit them to the base system. Otherwise, if it detects discrepancies between the two sets of changes, it will notify the administrators about the differences so that they can resolve the problem. This enables fail safe administration by enabling a single administrator's accidental errors to be caught, while also preventing a single administrator from maliciously damaging the system.

ISE-T leverages both virtualization and unioning file systems to provide the administration clones for each administrator. ISE-T is able to leverage both operating system virtualization techniques, such as Solaris Zones [18] and Linux VServer [20], as well as hardware virtualization such as VMware [24], to provide each administrator with an isolated environment in which to perform the changes. ISE-T builds upon DejaView [14], leveraging union file systems to provide a layered file system that is able to provide the same initial file system namespace in one layer, while capturing all the system administrator's file system changes into a separate layer. This enables easy isolation of changes, simplifying equivalence testing.

ISE-T's approach of requiring everything to be installed twice blocks many real attacks. A single malicious system administrator can no longer install modules that create an intentional back-door to allow future access into the system. Similarly, they cannot unilaterally weaken firewall rules, nor create unauthorized accounts to allow others into the system.

ISE-T is admittedly an expensive solution, too expensive for many commercial sites. For high-risk situations, such as in the financial, government, and military sectors, the added cost can be acceptable if the risk is reduced. In fact, the two-person controls are already routine in those environments, ranging from checks that require two signatures to the well known requirement of requiring two people for any work involving nuclear weapons. However, we also demonstrate how ISE-T can be used in a less expensive manner by introducing a form of auditable system administration. Instead of requiring two system administrators at all times, auditable ISE-T captures all the changes performed by the system administrator in the same manner it uses for equivalence testing, but instead immediately saves it to an audit log while committing it to the underlying system. An audit can then be performed on the log to provided a higher level of assurance that the administrator was only performing the changes they claimed they were performing.

In a similar manner, ISE-T can be extended to train less experienced system administrators. First, ISE-T allows a junior system administrator to perform tasks in parallel with a more senior system administrator. While only the senior system administrator's solution will be committed to the underlying system, the junior system administrator can learn from how his solution differs from the senior system administrator. Second, ISE-T can help train junior system administrators by being extended to provide an approver mode. In this mode, a junior system administrator will be provided administration tasks to complete. However, instead of the changes being committed directly, they will be presented to the senior system administrator who can approve or disapprove of the changes, without being required to do the same actions in parallel.

We have implemented an ISE-T Linux prototype without requiring any source code changes to the underlying kernel or system applications. To evaluate its ability to do equivalence testing, we conducted a user study to determine ISE-T's ability to efficiently capture administration changes through its layered file system, as well as to

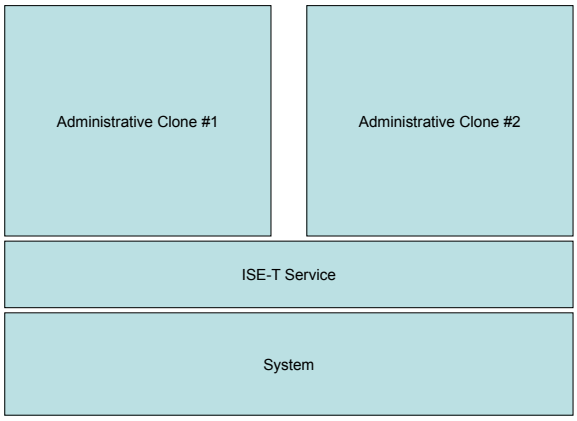

Figure 1: ISE-T Usage Model

compare the environments of the multiple administrators for equivalence. Our results demonstrate that ISE-T is effective at determining equivalence for many common administration tasks even when administrators perform those tasks in different ways. Furthermore, we demonstrate that ISE-T is able to easily show the differences that occur when the actions are not performed equivalently, in what situations the actions cannot be performed equivalently, as well as ISE-T's ability to detect malicious administration changes.

2 Usage Model

Systems managed by ISE-T are used by two classes of users, regular unprivileged users and the privileged system administrators who manage the machines. ISE-T does not change how regular users interact with the machine. They are able to install any program into their personal space, as well as run any program on the system, including regular programs and programs with special privileges, such as `setuid` UNIX programs that raise the privileges of the process on execution. This allows regular users to execute programs such as `passwd` to change their passwords.

However, ISE-T fundamentally changes the way system administrators interact with the machine. In a regular system, when administrators want to perform maintenance on the machine, they will leverage their ability to execute arbitrary programs with administrative privileges. This can be accomplished by executing a shell with the privilege so that they can execute arbitrary commands with ease, or by leveraging a program such as `sudo` that will just execute the arbitrary programs itself that way. In these systems, administrators are able to modify the system in a direct manner, change files, and execute programs and have those changes occur directly.

As ISE-T prevents system administrators from executing arbitrary programs with administrative privileges, this model cannot be directly used in a system managed by ISE-T. Instead, ISE-T provides a new model as shown in Figure 1. Instead of administering a system directly, ISE-T creates administration clones. Each administration clone is fully isolated from each other and the base system. ISE-T instantiates an administration clone for each administrator to perform the administrative acts within. Once both administrators are finished, ISE-T compares the clones for equivalence and commits the changes if the clones pass the equivalence test. As opposed to a regular system, where the administrator can interleave file system changes with program execution, in ISE-T only the file system changes get committed to the underlying system. Therefore ISE-T requires administrators to use other methods if they require file system changes and program execution to be interleaved on the actual system, such as for rotating log files or to do exploratory changes in order to diagnose a subtle system malfunction.

To allow this, ISE-T provides a new `ise-t` command that is used in a manner similar to `su`. Instead of spawning a shell on the existing system, `ise-t` spawns a new isolated container for that administrator. This container contains a clone of the underlying file system. Within this clone, the administrators can perform generic administrative actions, as on a regular system, but the changes will be isolated to this new container. When the administrators are finished with the desired administration changes, they exit their new container's shell, much as they would exit a root shell; the container itself is terminated, while its file system remains around.

ISE-T then compares the changes each administrator performed for equivalence. ISE-T performs this task automatically after the second administrator exits his administration session and notifies both of the administrators of the results. If the changes are equivalent, ISE-T automatically commits the changes to the underlying base system. Otherwise, ISE-T notifies the administrators of the file system discrepancies that exist between the two administration environments, allowing the administrators to recreate their administration environments and correct the discrepancies.

Command	Description
`ise-t new`	Create an administration environment
`ise-t enter`	Enter administration environment
`ise-t done`	Ready for equivalence testing
`ise-t diff`	Results of a failed equivalence test

Table 1: ISE-T Commands

As ISE-T only looks at file system changes, this can prevent it from performing administrative actions that just affect the runtime of the system. In order to han-

dle this, ISE-T provides a raw control mechanism via the file system, as well as enabling itself to be integrated with configuration management systems. First, ISE-T's raw control mechanism is implemented via a specialized file system namespace where an administrator can write commands. For instance, if the administrators want to kill a process, stop a service or reboot the machine, those actions performed directly within their administration container will have no affect on the base system. Some actions can be directly inferred from the file system. For instance, if the system's set of startup programs is changed, by having a file added, removed or replaced, ISE-T can infer that the service should be started, stopped or restarted when the changes are committed to the underlying system. However, this only helps when one is changing the file system. There are times when administrators will want the services stopped or restarted without modifying the file system of the system. Therefore, ISE-T provides a defined method for killing processes, stopping and starting services and rebooting the machine using files the administrator can store on the local file system. ISE-T provides each administrator with a special `/admin` directory for performing these predefined administrative actions.

For example, if the administrator wants to reboot the machine, they create an empty `reboot` file within the `/admin` directory. If both administrators create the file, after the the rest of their changes are committed to the system, it will reboot itself. Similarly, the administrators can create a `halt` file to halt the machine. In addition, the `/admin` directory has `kill` and `services` subdirectories. To kill a process, administrators create individual files with the names of the process identifiers of processes running on the base system that they desire to kill. Similarly, if a user desires to stop, start, or restart a `init.d` service, they can create a file named by that service prefixed with `stop`, `start` or `restart`, such as `stop.apache` or `restart.apache` within the `services` directory to have ISE-T perform the appropriate actions when the changes are committed to the base system. The files created within the `/admin` directory are not committed to the base system; they are only used for performing runtime changes to the system.

However, many systems already exist to manage systems and perform these types of tasks, namely configuration management systems, such as `lcfg` [2]. At a high level, configuration management systems work by storing configuration information on a centralized policy server that controls a set of managed clients. In general, the policy server will contain a set of template configuration files that it uses to create the actual configuration file for the managed clients based on information contained within its own configuration. Configuration management systems also generally support the ability to run predefined programs, scripts and execute predefined actions on the clients they are managing.

When ISE-T is integrated with any configuration management system, it no longer manages the individual machines. Instead of the managed clients being controlled by ISE-T, the configuration policy server is managed by ISE-T directly and the clients are managed directly by the configuration management system. This provides a number of benefits. First, it simplifies the complexity of comparing two different systems, as ISE-T can focus on the single configuration language of the configuration management system. Second, configuration system already have tools to manage the runtime state of their client machines, such as stopping and starting services and restarting them when the configuration changes. Third, many organization are already used to using configuration management systems; by implementing ISE-T on the server side, they can enforce the two-person control model in a more centralized manner.

3 ISE-T Architecture

To enable the two-person administrative control semantic, ISE-T provides three architectural components. First, as the two administrators cannot administer the system directly, they must be provided with isolated environments in which they can perform their administrative acts. To ensure the isolation, ISE-T provides container mechanisms that allow ISE-T to create parallel environments that are based on the underlying system that is being administered. This allows ISE-T to fully isolate each administrator's clone environment from each other and from the base system.

Second, we note that any persistent administrative action has to involve a change to the file system. If the file system is not affected, the action will not survive a reboot. Whereas some administrative acts only affect the ephemeral runtime state of the machine, the majority of administrative acts are of a more persistent nature. Therefore, to allow ISE-T to create two-person administrative control, the file system is a central component. ISE-T provides a file system that can create branches of itself as well as isolate the changes made to it. This enables the easy creation of the clone containers, as well as enabling the easy comparison of the changes performed to both environments.

Finally, ISE-T provides the ISE-T System Service. This service instantiates and manages the life-times of the administration environments. It is able to compare the two separate administration environments for equivalence to determine if the changes performed to them should be committed to the base system. ISE-T's System Service performs this via an equivalence test that compares the two administration environment's file sys-

tem modifications for equivalence. If the two environments are equivalent, the changes will be committed to the underlying base system. Otherwise, the ISE-T System Service will notify the two administrators of the discrepancies and allow them to fix their environments in the appropriate fashion.

3.1 Isolation Containers

ISE-T can leverage multiple different types of container environments, depending on the requirements of the administrators managing the system. In general, the choice will be between hardware virtual machine containers and operating system containers. Hardware virtual machines, such as VMware [24], provide a virtualized hardware platform that a separate operating system kernel runs on and provides a complete operating system instance. Operating system containers, such as Solaris Zones [18], on the other hand, are just isolated kernel namespaces running on a single machine.

For ISE-T, there are two primary difference between these containers. First, hardware virtual machines allow the administrators to install and test new operating system kernels as each container will be running its own kernel. Operating system containers, on the other hand, prevent the administrators from testing the underlying kernel, as there is only one kernel running, that of the underlying host machine. Second, as hardware virtual machines require their own kernel and a complete operating system instance to be started up, they take a significant amount of time to create the administration clones. On the other hand, operating system containers can be created almost instantly, allowing the administrators to quickly perform their actions. As both types of containers have significant benefits for different types of administrative acts, ISE-T supports the ability to use both. For most actions, administrators will prefer to use operating system containers, while still enabling them to get a complete hardware virtual machine when they desire to test kernel changes.

When ISE-T is integrated with a configuration management system, ISE-T does not have to use any isolation container mechanism at all, as the configuration management system already isolates the administrators from the client system. Instead, ISE-T simply provides each administrator with their own configuration management tree and let each individual administrator perform the changes.

3.2 ISE-T's File System

To support its file system needs, ISE-T leverages the ability of some file systems to be branched. Unlike a regular file system, a branchable file system can be snapshotted at some point in time and branched for future use. This allows ISE-T to quickly clone the file system of the machine being managed for both clone administration environments. As each file system branch is independent, this allows ISE-T to capture any file system changes in the newly created branch, by comparing the branch's state against the initial file system state. Similarly, ISE-T can then compare the set of file system changes from both administration clones against each other for equivalence.

However, while a classical branchable file system allows one to capture the changes, it does not allow one to efficiently discover what has changed, as the branch is a complete file system namespace. Iterating through the complete file system can take a significant amount of time, as well as place a large strain on the file system and decrease system performance. To allow ISE-T to use a file system efficiently, it must provide two features. First, it must be able to duplicate the file system to provide each administrator with their own unique and independent file system to perform their changes on. Second, it must provide a way to easily isolate the changes each administrator makes to the file system to easily test the changes for equivalence. To meet these requirements, ISE-T creates layered file systems for each administration environment, where multiple file systems can be layered together into a single file system namespace for each environment. This enables each administration environment to have a layered file system composed of two layers, a single shared layer that is the file system of the machine they are administrating, as well as a layer that will contain all the changes the administrator performs on the file system.

To support the creation of the layered file system, ISE-T has to solve a number of file system related problems. First, it must support the ability to combine numerous distinct file system layers into a single static view. This is equivalent to installing software into a shared read-only file system. Second, as users expect to be able to interact with the layered file system as a normal file system, such as by creating and modifying files, ISE-T has to enable the layered file system to be fully modifiable. In a related vein, the third problem ISE-T has to solve is that end users should also be able to delete files that exist on the read-only layer. However, end users should also be able to recover the deleted files by reinstalling or upgrading the layer that contains the deleted. This is equivalent to deleting a file from a traditional file system, but reinstalling the package that contains the file to recover it.

To solve these problems, ISE-T leverages union file systems. Unioning file systems enable ISE-T to solve the first problem as they allow the system to join multiple distinct directories into a single directory view, as shown in Figure 2. These directories are unioned by layering directories on top of one another. For example, when two directories are unioned together, one directory contain-

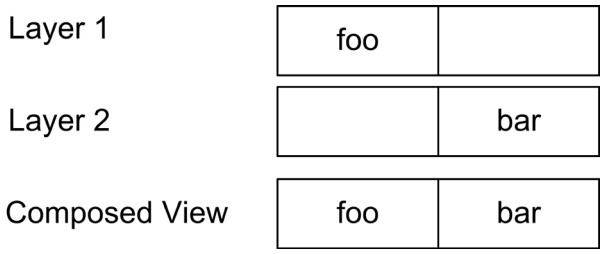

Figure 2: Unioning Namespaces

ing the file `foo` and the other containing the file `bar`, the unioned directory view would contain both files `foo` and `bar`. To provide a consistent semantic, most union file systems only allow one layer, namely the topmost to have files added to it. At the same time, if a file that already existed is modified, the union file system changes the underlying file directly, in whatever layer of the union it existed previously.

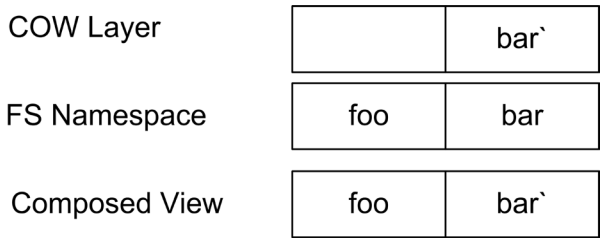

Figure 3: COW functionality

To solve the second problem, union file system can be extended [27] to enable them to assign properties to the layers, defining some layers to be read only while others can be read-write. This results in a model that borrows from copy-on-write (COW) file systems, where a modifying a file on a lower read-only layer will cause it to be copied to the topmost writable layer, as shown in Figure 3. For instance, in the above example, a blank cow writable layer can be layered on top of a read only layer containing `foo` and `bar`. If, in the course of usage, file `bar` get modified to `bar`` it will be *copied up* to the top most layer before the modification occurs. When a file is created or modified, it is written to the private read-write layer enabling the layered file system to be differentiated through file system changes.

This layering model also provides a semantic that directory entries located at higher layers in the stack obscure the equivalent directory entries at lower levels. Continuing the example, both layers now contain the file `bar`, but only the top most layer's version of the file is visible. To provide a consistent semantic, if a file is deleted, a white-out mark is also created on the top most layer to ensure that files existing on a lower layer are

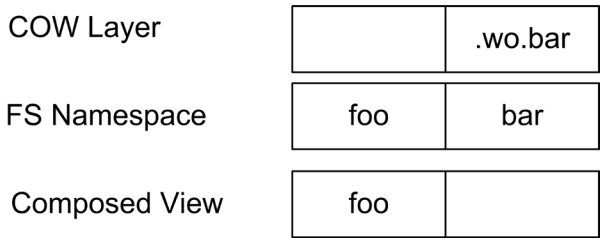

Figure 4: White-Out Support for Deletion

not revealed, as shown in Figure 4. Now, if the file `bar` were deleted, it would not allow the `bar` on the lower layer to be revealed. The white-out mechanism enables obscuring files on the read-only lower layers, simply by creating white-out files on the topmost layer.

ISE-T's layered file system provides the ability for multiple independent views of a file system to be in an active modifiable state at the same time, while confining each view's modifications to itself by providing each file system with an independent COw layer. To provide a simple example, imagine one has a directory that one wants to branch into two distinct views. This implies that processes operating in one view would be able to modify the files, without the changes causing any effect in the other view, and vice versa. This model can simply be implemented by ISE-T with the above union file system semantic. ISE-T creates two distinct views of the directory by creating two distinct ISE-T branched file system mounts. Since all modifications will cause files to be copied to the top most directory, it enables one to simply contain each views modifications into its own space. Finally, as each COW layer isolates the changes that were performed to each file system, ISE-T can easily determine which files it has to compare for equivalence.

3.3 ISE-T System Service

ISE-T's System Service has a number of responsibilities. First, it manages the lifetimes of each administrator's environment. When administration has to be performed, it has to setup the environments quickly. Similarly, when the administration session has been completed and the changes committed to the underlying system, it removes them from the system and frees up their space. Third, it evaluates the two environments for equivalence by running a number of equivalence tests to determine if the two administrators performed the same set of modifications. Finally, it has to either notify the administrators of the discrepancies between their two environments or commit the equivalent environment's changes to the underlying base system.

ISE-T layered file system allows ISE-T system's service to easily determine which changes each administra-

tor made, as each administrator changes will be confined to their personal layer of the layered file system. To determine if the changes are equivalent, ISE-T first isolates the files that it does not care about, and that will not be committed to the base system. This is currently limited to the administrator's personal files in their branch, such as shell history. Instead of just removing them, ISE-T saves them for archival and audit purposes. ISE-T then iterates through the files in each environment, comparing the file system contents and files directly against each other. If each administrator's branch has the equivalent set of file system changes, ISE-T can then simply commit a set to the base system. On the other hand, if the files contained within each branch, are not equivalent, ISE-T flags the differences and reports to each administrator what the differences are. The administrators can then confer with each other to ensure that they perform the same steps, so that they will create the same set of files to commit to the base system.

Determining equivalence can vary based on the type of file and what is considered to be equivalent. For instance, a configuration file modified by both administrators with different editors can visually appear to be equivalent, but can differ from each other if one uses spaces and another used tab characters. These files can be equivalent, as they would be parsed by applications in the same manner, but would be different when examined on a character by character level. However, there are some languages (e.g., Python) where the amount of whitespace matters; this can have a large effect on how the script executes. On the other hand, two files that have exactly the same file contents can have varying meta data associated with the file, such as permission data, extended attributes or even the multiple types of time data associated with each file. Similarly, some sets of files should not matter for equivalence, such as the shell history that recorded the steps the administrators took in their respective environments, and in general the home directory contents of the administrator in the administration environment. ISE-T prunes these files from the comparison, and never commits them to the underlying system.

Taking this into consideration, ISE-T's prototype comparison algorithm determines these sets of differences.

1. Directory entries which do not exist in both sets of changes are differences.
2. Every directory entry that does not have the same UID, GID, and permission set are different.
3. Every directory entry that is not of the same file type (Regular File, Symbolic Link, Directory, Device Node, or Named Pipe) are different

For directory entries that are of the same type, ISE-T performs the appropriate comparison.

- Device nodes must be of the same type
- Symbolic links must contain the same exact path
- Regular files must have the same size and the exact same contents

There are two major problems with this approach. First, this comparison takes place at a very low semantic level. It does not take into account simple differences between files that make no difference in practice. However, without writing a parser for each individual configuration language, one will not easily be able to compare equivalence. Second, there are certain files, such as encryption keys, that will never be generated identically, even though equivalent actions were taken to create them. This can be important, as some keys are known to be weaker and a malicious administrator can construct one by hand.

Both of these problems can be solved by integrating ISE-T with a configuration management system and teaching ISE-T the configuration management system's language. First, these systems simplify the comparison by enabling it to focus on the configuration management system's language. Even though most configuration management systems work by creating template configuration files for the different applications, these files are not updated regularly and can be put through the stricter exact comparison test. On the other hand, when ISE-T understands the single language of the configuration management system, it can rely on a more relaxed equivalence test. Second, configuration management systems already have to deal with creating dynamic files, such as encryption keys. A common way configuration management systems deal with these types of files is by creating them directly on the managed client machines. As ISE-T understand the configuration management system's language, the higher level semantics that instruct the system to create the file will be compared for equivalence instead of the files themselves. However, a potential weakness of ISE-T is in dealing with files that cannot easily be created on the fly and will differ between two system administration environments, such as databases. For instance, two identical database operations can result in different databases due to the saving of a time-stamp, or the simple reordering of updates on the database server.

4 ISE-T for Auditing

Whereas the two-person control model that ISE-T provides to system administration is useful for providing high assurance that faults are not going to creep into the system, its expense can make it unusable in many situations. For example, since the two-person control model

requires the concurrence of two system administrators on all actions, it can prevent timely actions from being taken if only a single administrator is available. Similarly, whereas the two-person control model provides a high degree of assurance for a price, it would be useful if organizations could get a higher degree of assurance than normal with little extra cost. To achieve these goals, we can combine ISE-T's mechanisms with audit trail principles to create an auditable system administration semantic.

In auditable system administration, every system administration act that is logged to a secure location so that it can be reviewed for correctness at some point in the future. The ISE-T System Service creates clone administration environments for the two administrators and can capture the state they change in order to compare them for equivalence. For auditable system administration, ISE-T's mechanism can also be used. The audit system prevents the single system administrator from modifying the system directly, but require the creation of a cloned administration environment where the administrator can perform the changes before they are committed to the underlying system. Instead of comparing for equivalence against a second system administrator, the changes are logged so that they can be used by an auditor at some point in the future as well as immediately committed to the underlying system. Audit systems are known to increase assurance that malicious changes are not performed, as the malicious person knows there's a good chance his actions will be caught. Similarly, depending on the frequency and number of audits performed, it can help prevent administration faults from persisting for long periods of time in the system. However, it does not provide as high assurance a model as can be provided by the two-person control system, as the administrator can use the fact that his changes are committed immediately to create backdoors in the system that cannot be discovered until later.

Auditable system administration needs to be tied directly to an issue-tracking service. This allows an auditor to associate an administrative action with what the administrator was supposed to accomplish. Every time an administrator invokes ISE-T to administer the system, an issue-tracking number is passed into the system to tie that action with the issue in the tracker. This allows the auditor to compare the results of what occurred with what the auditor expects to have occurred. In addition, auditable system administration can be used in combination with the two-person control system when only a single administrator is available and action has to be taken in a more immediate fashion. With auditing, the action can be performed by the single administrator, but can be immediately audited when the second administrator becomes available. This helps the system maintain its higher level of assurance when immediate action has to be taken by a single administrator.

5 Experimental Results

To test the efficacy of ISE-T's layered file system approach, we recruited 9 experienced computer users with varying levels of system administration experience, though all were familiar with managing their own machines. We provided each user with a VMware virtual machine running Debian GNU/Linux 3.0. Each VM was configured to create an ISE-T administration environment that would allow the users to perform multiple administration tasks isolated from the underlying base system. Our ISE-T prototype uses UnionFS [27] to provide the layered file system needed by ISE-T. We asked the users to perform the eleven administration tasks listed in Table 2. The user study was conducted in virtual machines running on an IBM HS20 eServer blade with dual 3.06 Ghz Intel Xeon CPUs and 2.5GB RAM running VMware Server 1.0. These tasks were picked as they are indicative of common administration tasks, as well as including a common way a malicious administrator would create a back-door in the system for himself.

Each task was performed in a separate ISE-T container, so that each administration task was isolated from the others, and none of the tasks depended on the results of a previous task. We used ISE-T to capture the changes each user performed for each task in its own file system. We were then able to compare each user against each other for each of the eleven tasks, to see if they performed equivalent modifications or where their modifications differed.

For every test, ISE-T prunes the changes that were done to remove files that would not affect equivalence since they would not be committed to the underlying file system, as described in Section 3.3. Notably, in our prototype, ISE-T prunes the `/root` directory which is the home directory of the root user, and therefore would contain differences in files such as `.bash_history` amongst others that are specific to how they went about performing the task. Similarly, ISE-T prunes the `/var` subtree to remove any files that were not equivalent. For instance, depending on how an administrator would administer the system and what tools one would use, different files would be created, for instance a cache of packages downloaded and installed via the `apt-get` tool versus being downloaded and installed manually. The reasoning behind this pruning is that the `/var` tree is meant as a read-write file system for per-system usage. Tools will modify it; if different tools are used, different changes will be made. However, one cannot prune the entire directory tree as there are files or directories within it that are necessary for runtime use and those

Category	Description	Result
Software Installation	Upgrade entire system via package manager	Equivalent
	Install official Rdesktop package	Equivalent
	Compile and install Rdesktop from source	Equivalent
System Services	Install SSH Daemon from package	Not Equivalent (Not Desired)
	Remove PPP package using package manager	Equivalent
Configuration Changes	Edit machine's persistent hostname	Equivalent
	Edit the inetd.conf to enable a service	Not Equivalent (Not Desired)
	Add a daily run cron job	Equivalent
	Remove an hour run cron job	Equivalent
	Change the time of a cron job	Equivalent
Exploit	Create a backdoor setuid root shell anywhere	Not Equivalent (Desired)

Table 2: Administration Tasks

changes have to be committed to the underlying file system. Therefore, only those changes that are equivalent were committed, while those that are different were ignored. ISE-T also prunes the `/tmp` directory as the contents of this directory would also not be committed to the underlying disk. Finally, due to the UnionFS implementation, ISE-T also prunes the whiteout files created by UnionFS if there is no equivalent file on the underlying file system. In many cases, temporary files with random names will be created; when they are deleted, UnionFS will create a whiteout file, even if there is no underlying file to whiteout. As this whiteout file does not have an impact on the underlying file system, it is ignored. On the other hand, whiteout files that do correspond to underlying files and therefore indicate that the file was deleted are not ignored.

5.1 Software Installation

For the software installation category, we had the users perform three separate tests that demonstrated different ways administrators install software into the system. These tests were to demonstrate that when multiple users install the same piece of software, as long as they install it in the same general way, the two installations will be equivalent.

To demonstrate this, the users were first instructed to install the rdesktop program from its Debian package. Users could choose to download the package by hand and install it via `dpkg`, they could use `apt-get` to download it and any unfulfilled dependencies, or use the `aptitude` front end amongst many ways to perform this task. Most users decided to install the package via `apt-get`, but even those who did not made equivalent changes. The only differences were those in pruned directories, demonstrating that installing a piece of prepackaged software using regular tools will result in an equivalent system.

Second, the users were instructed to build the rdesktop program from source code and install it into the system. In this case, multiple differences could have occurred. First, if the compiler would create a different binary each time the source code is compiled, even without any changes, one would have a more difficult time evaluating equivalence. Second, programs generally can be installed in different areas of the file system, such as `/usr` versus `/usr/local`. In this case, all the testers decided to install the program into the default location, avoiding the latter problem, while also demonstrating that as long as a the same source code is compiled by the same toolchain, it will result in the same binary. However, some program source code, such as the Linux kernel, will dynamically modify their source during build, for example to define when the program was built. In these cases, we would expect equivalence testing to be more difficult as each build will result in a different binary. A simple solution would be to patch the source code to avoid this behavior. A more complicated solution would involve evaluating the produced binary's code and text sections with the ability to determine that certain text section modifications are inconsequential. Again, in this case the only differences were in pruned directories, notaby the `/root` home directory to which the users downloaded the source for rdesktop.

Finally, we had the users upgrade the Debian stable system with all pending security updates. This was a more complicated version of the first test, as multiple packages were upgraded. Although differences existed between the environments of the users, the differences were confined to the `/var` file system tree and depended on how they performed the upgrade. This is because Debian provides multiple ways to do an upgrade of a complete system and those cause different log files to be written. As they all installed the same set of packages, the rest of the file system, as expected, contained no differences.

5.2 System Services

Our second set of tests involved adding and removing services: the users were instructed to install the ssh service and remove the PPP service. These tests were an extension of the previous package installation tests and were meant as a demonstration of how one would automatically start and stop services, as well as a demonstration of files we knew would be different and therefore fail equivalence testing.

For the first test, we instructed the users to install the SSH daemon. This test sought to demonstrate that ISE-T can detect when a new service is installed and therefore enable it when the changes are committed. This is demonstrated by the fact that in Linux systems, a System-V init script has to be added to the system to enable it to be started each time the machine boots. If the user's administration environment contains a new init script, ISE-T can automatically determine that the service should be started when this set of administration changes are committed to the base system. This test also sought to demonstrate that certain files are always going to be different between users if created within their private environment. This is demonstrated by the fact that the SSH host key for each environment is different. This is because it is created based on the kernel's random entropy pool that will be different for each user and therefore will never be the same if created in separate environment. A way to solve this would be not to create it within the private branch of each user, but instead have it be created after the equivalent changes are committed, for instance, the first time the service's init script is executed.

For the second test, we instructed the users to remove the PPP daemon. This test sought to demonstrate that there are multiple ways to remove a package in a Debian system and depending on the way the package is removed, the underlying file system will be different. Specifically, a package can either be `removed` or `purged`. When a package is removed, files marked as configuration files are left behind, allowing the packages to be reinstalled and have the configuration remain the same. On the other hand, when a package is purged, the package manager will remove the package and all the configuration files associated with it. In this case, the user's chose different ways to remove the package, and ISE-T was able to determine the differences for those that chose to remove or purge it.

5.3 Configuration Changes

Our third set of tests involved modifications to configuration files on the system and involved six separate tests. These tests could be subdivided into three categories. The first category was composed of simple file configuration changes. We first instructed the users to modify the host name of the machine persistently from `debian` to `iset`, which is accomplished by editing the `/etc/hostname` file. As expected, as this configuration change is very simple, all user modified the system's hostname in the exact same manner, allowing ISE-T to determine that all the systems were equivalent.

Next, we instructed the users to modify the `/etc/inetd.conf` file to enable the `discard` service. In this case, as the file is more free-form, their changes were not exact, and many were not equivalent. For example, some users enabled it for both TCP and UDP, while some users enabled it for TCP alone. Also, some users added a comment, while others did not. Whereas the first change is not equivalent, the second change should be considered equivalent, but this cannot be determined by a simple diff; one needs the ability to parse the files correctly to determine that they are equivalent, an ability our ISE-T prototype does not have. However, ISE-T was able to clearly report the differences that existed between the environments of users who performed this administration task differently.

The second set of tests involved setting up and removing cron jobs and was composed of three tests. First, we provided the users with a script in `/root` that we instructed them to install in the system in a manner so that it will be executed daily. In Debian there are two ways to have a cron job execute daily. The best way is to simply place the script in the `/etc/cron.daily` directory, as all the programs located within it are executed once a day. On the other hand, a user could use the `/etc/crontab` file and instruct it exactly what time each day to execute the program. We were looking for users to place the file in `/etc/cron.daily` which the large majority did; but one user who perhaps was not used to Debian modified `/etc/crontab` directly. Again, ISE-T was able to detect the difference when it existed and report the discrepancy. Similarly, we instructed users to remove a cron job that ran hourly. The cron job's script was already located in `/etc/cron.hourly` and not in `crontab`. All users deleted it from the `/etc/cron.hourly` directory, even the user who modified `/etc/crontab` in the previous test. Finally, we had an entry in `/etc/crontab` that executed a cron job at 4:35 AM every day, and instructed the users to change it to 5:35 AM. As expected, all users performed the same set of changes by modifying the `/etc/crontab` file.

5.4 Exploit

In our final test, we asked the users to create a backdoor shell into the system. This test demonstrates that ISE-

T will detect when one of the two system administrator makes a rogue administration change to the system, as the other administrator will not have made that same administration change. We did not specify how or where the shell should be created, and left it up to the user. The users created setuid shells in many different locations, some as "dot files" to be hidden from a normal `ls` listing, and some just setuid existing shells. In all cases, as none of the users made this rogue modification in the same manner, ISE-T was able to detect the rogue changes as a difference that existed between all the environments.

6 Related Work

Two-person control mechanisms are known to provide high levels of assurance [22]. Multiple examples exist with regard to nuclear weapons. For instance, to launch a nuclear weapon, two operators must separately confirm that launch orders are valid and must turn their launch keys together to launch the missiles. In fact, every sensitive action concerning nuclear weapons must be performed by two people with the same training and authority [3, Chapter 2]. The same notion is applied in many financial settings: banks will require two people to be involved in certain tasks, such as opening a safe-deposit box [25], and companies can require two people to sign a check [8] over a certain threshold. This makes it much more difficult for a single person to commit fraud.

However, as far as we know, this mechanism has never been applied directly to system administration. In the Compartmented Mode Workstation (CMW), the system administration job is split into roles, so that many traditional administration actions require more than one user's involvement [23]. These demarcation of roles were first pioneered in Multics at MIT [12]. Similarly, the Clark-Wilson model was designed to prevent unauthorized and improper modifications to a system to ensure its integrity [4]. All these systems simply divided the administrators' actions amongst different users who performed different actions. This differs fundamentally from the traditional notion of two-person control where both people do the same exact action.

More recently, many products have been created to help prevent and detect when accidental mistakes occur in a system. SudoSH [9] is able to provide a higher level of assurance during system administration as it records all keystrokes entered during a session and is able to replay the session. However, while sudosh can provide an audit log of what the administrator did, it does not provide the assurances provided by the two-person control model. Even if one were to audit the record or replay it, one is not guaranteed to get the same result. Although auditing this record can be useful for detecting accidental mistakes, it cannot detect malicious changes. For instance, a file fetched from the Internet can be modified. If the administrators can control which files are fetched, they can manipulate them before and after the sudosh session. ISE-T, on the other hand, does not care about the steps administrators take to accomplish a task, only the end result as it appears on the file system.

Part of the reason accidental mistakes occur is that knowledge is not easily passed between the experienced and inexperienced system administrators. Although systems like administration diaries and wikis can help, they do not easily associate specific administration actions with specific problems. Trackle [6] attempts to solve this by combining an issue tracker with a logged console session. Issues can be annotated, edited and cross-referenced while the logged console session logs all actions taken and file changes and stores them with the issue, improving institutional memory. Although this can help prevent mistakes from entering the system due to enabling the less experienced system administrators from seeing the exact same steps a previous administrator took to fix a similar or equivalent issue, it does not prevent mistakes from entering and remaining in the system, nor does it prevent a malicious administrator from performing malicious changes.

ISE-T's notion of file system views was first explored in Plan 9 [17]. In Plan 9, it is a fundamental part of the system's operation. As Plan 9 does not view manipulating the file system view as a privileged operation, each process can craft the namespace view it or its children will see. A more restricted notion of file system views is described by Ioannidis [11]. There, its purpose is to overlay a different set of permissions on an existing file system.

Finally, a common way to make a system tolerant of administration faults is to leverage the semantic of file system versioning, as it enable you to rollback to a configuration file's previous state when an error was made. Operating systems such as Tops-20 [7] and VMS [15] include native operating system support for versioning as a standard feature of their file systems. These operating systems employ a copy-on-write semantic that involves versioning a file each time a process changes it. Other file systems, such as VersionFS [16], ElephantFS [19], and CVFS [21] have been created to provide better control of the file system versioning semantic.

7 Conclusions

ISE-T applies the two-person control model to system administration. In administration, the two-person control model requires two administrators to perform the same administration act with equivalent results in order for the administration changes to be allowed to affect the system that is being modified. ISE-T creates multiple paral-

lel environments for the administrators to perform their administration changes and then compares the results of the administration changes for equivalence. When the results are equivalent, there is a high assurance that system administration faults have not been introduced into the system, be they malicious or accidental in nature.

We have implemented an ISE-T Linux prototype that creates parallel administration environments where separate administrators can perform changes, while not having administration rights on the machine itself. Our results from a user study demonstrate that many common administration tasks will result in equivalence when performed by isolated administrators without any communication between them. This demonstrates that the two-person control model can be applied to system administration by simply analyzing the results of the file system changes that occur in the environments created for the two administrators.

Acknowledgements

Paul Anderson, Andrew Hume, our paper shepherds, and Matthew Barr provided many helpful comments on earlier drafts of this paper, especially in the area of configuration management. This work was supported in part by NSF grants CNS-0426623, CNS-0717544, and CNS-0914845.

References

[1] *US DOD Joint Publication 1-02, DOD Dictionary of Military and Associated Terms (as amended through 9 June 2004).*

[2] P. Anderson. *LCFG: A Practical Tool for System Configuration*. Usenix Association, 2008.

[3] A. B. Carter, J. D. Steinbruner, and C. A. Zraket, editors. *Managing Nuclear Operations*. The Brookings Institution, Washington, DC, 1987.

[4] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *IEEE Symposium on Security and Privacy*, 0:184, 1987.

[5] Commission for Review of FBI Security Programs, William Webster, chair. Webster Report: A Review of FBI Security Programs, Mar. 2002.

[6] D. S. Crosta, M. J. Singleton, and B. A. Kuperman. Fighting Institutional Memory Loss: The Trackle Integrated Issue and Solution Tracking System. In *Proceedings of the 20th Large Installation System Administration (LISA 2006) Conference*, pages 287–298, Washington, DC, Dec. 2006.

[7] Digital Equipment Corporation. Tops-20 user's guide, Jan. 1980.

[8] M. S. Elmaleh. Nonprofit fraud prevention. `http://www.understand-accounting.net/Nonprofitfraudprevention.html`, 2007.

[9] D. Hanks. Sudosh. `http://sourceforge.net/projects/sudosh/`.

[10] J. Heller. *Catch-22*. Simon and Schuster, 1961.

[11] S. Ioannidis, S. M. Bellovin, and J. Smith. Sub-operating Systems: A New Approach to Application Security. In *SIGOPS European Workshop*, Sept. 2002.

[12] P. Karger. Personal Communication, May 2009.

[13] P. A. Karger and R. R. Schell. MULTICS Security Evaluation: Vulnerability Analysis. Technical Report ESD-TR-74-193, Mitre Corp, Bedford, MA, June 1977.

[14] O. Laadan, R. Baratto, D. Phung, S. Potter, and J. Nieh. DejaView: A Personal Virtual Computer Recorder. In *Proceedings of the* 21^{th} *ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2007.

[15] K. McCoy. *VMS File System Internals*. Digital Press, 1990.

[16] K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pages 115–128, San Francisco, CA, Mar./Apr. 2004. USENIX Association.

[17] R. Pike, D. L. Presotto, K. Thompson, and H. Trickey. Plan 9 from Bell Labs. In *Proceedings of the Summer 1990 UKUUG Conference*, pages 1–9, London, UK, July 1990. UKUUG.

[18] D. Price and A. Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *Proceedings of the 18th Large Installation System Administration Conference*, Nov. 2004.

[19] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, Dec. 1999.

[20] S. Soltesz, H. Pötzl, M. e. Fiuczynski, A. Bavier, and L. Peterson. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. *SIGOPS Operating System Review*, 41(3):275–287, 2007.

[21] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger. Metadata Efficiency in a Comprehensive Versioning File System. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, Mar. 2003.

[22] P. Stein and P. Feaver. *Assuring Control of Nuclear Weapons*. University Press of America, 1987.

[23] J. S. Tolliver. Compartmented Mode Worksation (CMW) Comparisons. In *Proceedings of the 17th DOE Computer Security Group Training Conference*, Milwaukee, Wi, May 1995.

[24] VMware, Inc. `http://www.vmware.com`.

[25] Wilshire State Bank. Safe deposit boxes. `https://www.wilshirebank.com/public/additional_safedeposit.asp`, 2008.

[26] D. Wise. *Spy: The Inside Story of how the FBI's Robert Hanssen Betrayed America*. Random House, 2002.

[27] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and Unix Semantics in Namespace Unification. *ACM Transactions on Storage*, 2(1):1–32, Feb. 2006.