Secure, Resilient Computing Clusters: Self-Cleansing Intrusion Tolerance with Hardware Enforced Security (SCIT/HES)

David Arsenault, Arun Sood, Yih Huang
Computer Science Department
George Mason University
Fairfax, Virginia, USA
{darsenau, asood, huangyih}@cs.gmu.edu

Abstract

The formidable difficulty in securing systems stems in large part from the increasing complexity of the systems we build but also the degree to which we now depend on information systems. Complex systems cannot be fully verified under all possible conditions. Self-Cleansing Intrusion Tolerance (SCIT) servers go through periodic cleaning. SCIT can be used to create secure and robust cluster of servers without the impossible requirement of having perfect security on each server in the cluster. In this paper[†]*, we identify six SCIT security primitives that must be satisfied. We present a SCIT hardware enhanced (SCIT/HES) implementation that guarantees the incorruptibility of SCIT operations.*

1. Introduction

Networks and the systems that run on them have become essential to the operation of business enterprises, functioning of the global economy, and national security [1]. Despite decades of research on computer and network security systems are besieged by a myriad of security threats—viruses, worms, malware, denial of service attacks, data thefts, and software vulnerabilities.

Complex systems cannot be fully verified under all possible conditions, and thus unseen bugs and vulnerabilities can and do make their way into production software releases. The commercial drives to add more features and ship products quickly only exacerbate the problems. Another factor contributing to the poor state of system security today is connectivity. In [1] this point is captured with a simple equation: *Ubiquitous Interconnectivity = Widespread Vulnerability.*

These mounting security challenges necessitate a reexamination of the fundamental assumptions of present computer security defenses: Do we really know all enemy tactics and techniques? Do we know about all of the potential vulnerabilities that exist in our systems? The only prudent conclusion is there are attacks that could evade even the most state-of-art and constantly updated defensive systems.

In [2, 3, 4, 5] we developed Self-Cleansing Intrusion Tolerance (SCIT) with focus on creating a remediation mechanism for a broad set of system unknown and undetected attacks. In a SCIT system multiple servers are configured to perform some service such as e-mail, web or DNS servers. The SCIT cluster architecture allows individual servers to occupy different roles at different times without affecting the availability of the service that the cluster is performing. The lifecycle of an individual server begins with the cleansing role and then progresses through one or more online service roles. Constantly repeating this lifecycle ensures that each server is not online indefinitely and that it is periodically cleaned. By limiting the length of time a server is exposed online, SCIT reduces the window of opportunity for an attacker to exploit known or unknown vulnerabilities. Further, by periodically cleaning each server, an attacker's opportunity to establish a foothold to use for further or future attacks is severely restricted. In [5, 6], we have developed primitive forms of hardware enhancements for use by SCIT DNS systems. A cluster-wide SCIT management algorithm was proposed in [7].

Of course, SCIT operations and mechanisms themselves could also be attacked by the enemy in attempts to disable server cleansing and stop role rotations. Software implementations of SCIT operations in particular leave open the possibility of being compromised and inevitably constitute a weak link in security. In this paper, we propose a hardware-based, generic framework that guarantees the incorruptibility of SCIT operations. We begin our design with the following two *zero-trust principles*:

(1) Software is malleable. All software can be potentially corrupted through communications. This is

[†] This research is supported by the NIST-funded Critical Infrastructure Protection Project and Sun Microsystems.

the only prudent assumption given the trend of increasing complexities and vulnerabilities in software. *(2) Online servers are corrupted. All online systems (those exposed to the external, public network) must be assumed to be compromised. This is the only prudent response to not knowing "all" system vulnerabilities and attack techniques.*

In this paper we propose six security proprieties of SCIT, termed SCIT Primitives (see Table 1). The primary contribution of this paper is a scalable hardware framework, termed SCIT with Hardware Enhanced Security (SCIT/HES), that complements the software components of SCIT in order to enforce and guarantee the proposed SCIT Primitives. Compared to our previous, relatively primitive designs of hardware enhancements, the design presented here further address the following issues.

- Generic. The design presented here does not assume any specific services provided by the SCIT cluster. It is designed for incorporation by generic server clusters to benefit from the intrusion tolerance features provided by SCIT.
- Scalability. The new design scales to large size clusters.
- The removal of single point of failure at the central SCIT management controller in the cluster.

The remainder of the paper is organized as follows. We present in Section 2 the hardware components of the SCIT/HES framework. We elaborate in Section 3 the ways in which the SCIT Primitives in Table 1 are enforced and their implications on system security. We present in Section 4 the per-server state transitions in SCIT/HES and summarize in Section 5 the advantages of SCIT/HES. The issues of scalability and points of hardware failures are addressed in Section 6. We give related work and conclusion in Sections 7 and 8, respectively.

Table 1. SCIT Security Primitives

(1) Guaranteed Periodic Reset and Cleansing
(2) Protected Files Load From Trusted Storage Source
(3) Guaranteed Role Assumption in a known state.
(4) Security-Critical Operations Executed Only in off-line Clean State
(5) Guaranteed Isolation of Online Servers (from the internal networks and trusted storage)
(6) Guaranteed Isolation of Secure Controller

2. Secure Controller and Trusted Interface

The SCIT/HES framework uses a centralized controller and a small set of simple, non-programmable logic circuits in the communication interfaces. It allows for the use of standard server hardware while providing guaranteed security "primitives" upon which the ultimate security of the system depends. The high-level architecture of SCIT/HES is depicted in Figure 1. In the figure, we show only the detailed setup between the central controller and a particular Server X. The same setup applies to all the other servers in the cluster, omitted in Figure 1 for clarity. We point out that the issue of single point of failure caused by the central controller will be addressed later.

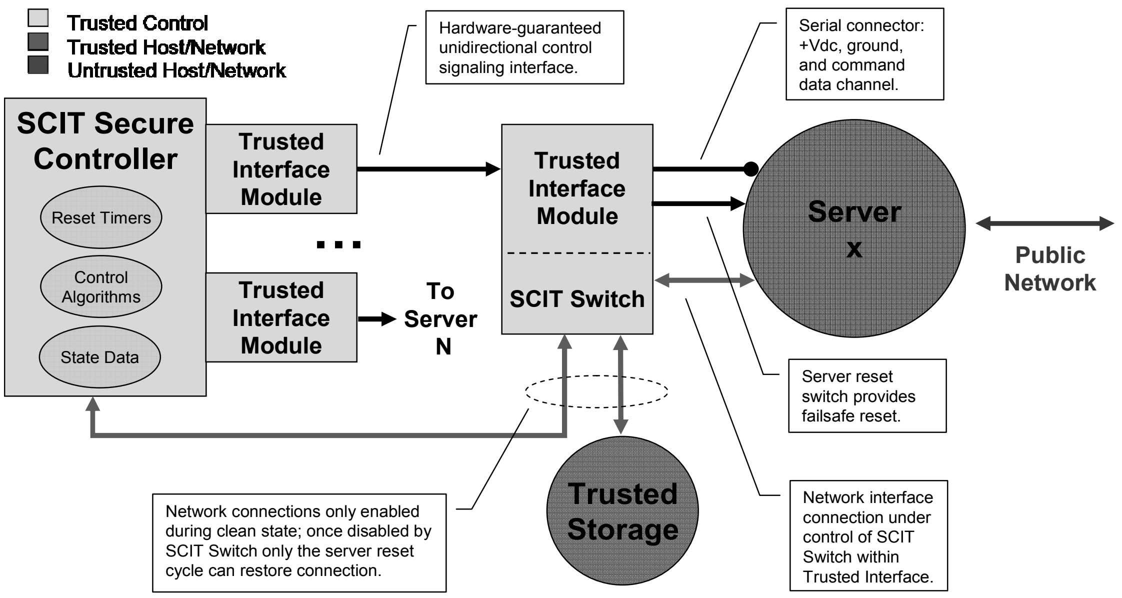
Trusted Control
Trusted Host/Network
Untrusted Host/Network
SCIT Secure Controller
Reset Timers
Control Algorithms
State Data
Trusted Interface Module
Trusted Interface Module
To Server N
Hardware-guaranteed unidirectional control signaling interface.
Trusted Interface Module
SCIT Switch
Serial connector: +Vdc, ground, and command data channel.
Server x
Public Network
Server reset switch provides failsafe reset.
Network interface connection under control of SCIT Switch within Trusted Interface.
Trusted Storage
Network connections only enabled during clean state; once disabled by SCIT Switch only the server reset cycle can restore connection.

Figure 1. SCIT/HES Cluster Architecture

2.1. SCIT Secure Controller

As seen in Figure 1, the central element of a SCIT/HES cluster is the SCIT Secure Controller which manages a set of Trusted Interfaces that provide connections to each server in the cluster. The SCIT Controller is a programmable machine (server) that implements the control algorithms to govern the cluster-wide SCIT operations, including server role rotations. Using a centralized, programmable controller affords cluster designers flexibility and robustness that would be very difficult if not impossible to achieve with the fully software-based SCIT implementations. A central mechanism of control simplifies both administration and programming of the cluster which allows the cluster designer to modify, tune, and optimize the role rotation algorithms needed for a particular application. Timing of both role rotations and reset timers are also left to the cluster designer. From a design perspective we purposefully kept as much of the complexity of the cluster as possible in one place—the SCIT Secure Controller.

2.2. SCIT Trusted Interface Module

The SCIT Trusted Interface Modules (TIMs) in Figure 1 provide a unidirectional communications link from the SCIT Controller to each server in the cluster. At the server end of the Trusted Interface Module, the unit is connected to the server's serial port for power, ground, and data; a connection to the server's reset switch is also used to provide the failsafe reset function. The data sent from the Secure Controller to the server over this connection informs the server when a state change is needed (such as a reset signal to begin the cleansing process) and what state to transition to. The server-side Trusted Interface Module also contains a SCIT Switch component that provides the ability to physically cut-off the network connections from the server to the local network (which includes the SCIT Controller itself as well as the Trusted Storage server).

In this way, the Trusted Interface is the isolating mechanism that maintains the protected state of the SCIT Controller and at the same time allowing it to send messages to control the state of each server—this is a unidirectional link guaranteed by hardware. Using the Trusted Interface, the Controller manages the communications interface between the clean machine (a server that has been cleansed but not yet online), Trusted Storage machine, and the Controller itself. The SCIT Switch provides hardware-guaranteed isolation between each Server and the Trusted Storage unit and between the Controller and the Server. When the Switch is *closed* it enables the network connection between the Server and the other machines it connects to, but, when the switch set to the *open* position no network connection can exist. These form the foundations of the six SCIT Primitives, as discussed in the next section.

3. The SCIT Security Primitives

The first SCIT security primitive in Table 1 is the guarantee that servers always go through and complete the periodic self-cleansing process. Resetting the server system will force it to reboot, go through the cleaning process, and finally load known clean copies of all critical files from the trusted storage. This primitive is achieved through two mechanisms: the primary mechanism is the SCIT daemon process running on each server and the secondary (backup) mechanism is a hardware failsafe timer and reset switch, both to be discussed later.

The second security primitive requires that all files critical to the successful operation of the server are loaded from a read-only device such as a CD-ROM or from a Trusted Storage device which is cryptographically secured.

The third security primitive guarantees the initial service role assumption of a server transitioning from the ready (clean) state to the assigned online service role. The basis for this guarantee is the fact that a newly clean server has not been exposed to the outside world (external networks) and is thus free from any corruption. Since the SCIT daemon and other critical system software has been loaded from a known clean copy we can be assured that the server, when commanded by the Secure Controller, will transition into its intended service role as expected.

The fourth security primitive requires the processing of security-critical operations only during the clean state, before the server goes online and is exposed to the external network. Like the second primitive, previous SCIT architectures already implement this primitive but do so in software. SCIT/HES enforces the fourth security primitive by using SCIT Switches to manage the access to the Trusted Storage, which hold security keys and other critical data. Before moving a server online, the Controller will turn off the SCIT Switch of the server to physically prevent access to the Trusted Storage host and thus keys and other critical data. Any data that is stored on the Trusted Storage host and is accessed during the self-cleansing process will never be exposed to the public/enterprise network.

The fifth security primitive is the guarantee that the SCIT Secure Controller is always fully isolated from the servers that are online performing their production roles. To enforce this guarantee a unique type of hardware switch—a "once after reset" switch—is implemented using a basic logic circuit. This switch is installed on all interfaces that connect the Controller to the servers in the cluster and regulates the direction of data flow on the interfaces. The "once after reset" switch operates between two and only two states: one state allows bidirectional data flow between the Controller and the server while the other state allows only unidirectional data flow from the Controller to the server. The Controller operates the switch via simple hardware-based signaling inputs which cause changes in the switch's state. The SCIT Switch powers up in the closed state which allows the server to connect to the Trusted Storage unit as it goes through the cleaning process. At this time, the server can also communicate with the Controller via the intranet connection to exchange any state data (if required by the applications being run in the cluster). Once the server is clean and ready to perform an online role, the SCIT Switch is set to the open position which prevents any communications on the network interface regardless of what state the network interface card is set to on the server. An attacker cannot turn the interface on and gain a network link outside the server into the rest of the cluster. Once the state of the switch is changed by the controller the only way to restore the state to enable communications on the link is a server reset which cycles the power to the switch. This is the so called "once after reset" property that ensures server isolation.

The sixth and final security primitive guarantees that the Secure Controller will always remain completely isolated from the rest of the SCIT network. The isolation of the Controller is accomplished in implementation terms by using a one-way optical link in the Trusted Interface Modules. Such a link provides absolute logical isolation of the Controller. As such the Secure Controller is free from all forms of network-based attack.

4. Server States and Role Rotations

Figure 2 shows the per-server state transition diagram in a SCIT/HES cluster. Here an important distinction must be made between the terms state and role. In a SCIT system a state means the condition of a SCIT server at a given point in time whereas the term role refers to the service a server provides to users (i.e. web, DNS, or file services).

A SCIT server begins with the warm rebooting (hardware reset) of the server, denoted as state [0] in Figure 2. The reboot starts the self-cleansing process by cleaning up all forms of corruption in memory. The server subsequently loads critical files from a read-only storage source (or a read/write storage that is cryptographically-protected source), reference state (1). Critical files, also termed "Protected files", include fundamental operating system files (kernel and other vital files), application executables, and important configurations files. These are the set of files that are required to be in a known, uncorrupted state in order to provide assured server operation. The server enters the final offline state (2) in the SCIT lifecycle. When commanded by the Secure Controller (via the Trusted Interface), the ready server enters a service role, state (3), during which the server provides designated service(s) to end users.

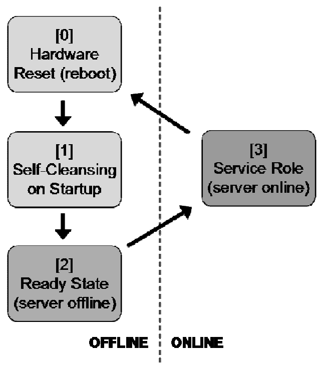

Figure 2. SCIT State Diagram

Upon completion of its service role, each SCIT server will return to state 0 after a reboot signal. The vital transition from state (3) to (0) is achieved by two independent mechanisms. Under normal conditions (no attack on server) the reboot signal is sent from the Secure Controller via the Trusted Interface and ultimately a SCIT daemon responds to the controller' signal and resets the server to begin the cleansing process. This reset method relies on the viability of the SCIT daemon, a software process, which is subject to potential subversion by an attacker. To eliminate this attack vector and provide absolute assurance that periodic self-cleansing occurs (SCIT Security Primitive 1), an independent mechanism known as the Trusted Interface Failsafe Reset Timer is implemented as a hardware-based timing circuit. Upon expiration, this timer circuit triggers a relay to activate the hardware reset switch on the server thus ensuring the periodic rebooting and cleansing of SCIT servers regardless the presence of software corruption.

5. Advantages of SCIT/HES

The general benefits of the SCIT/HES architecture, which implements a cluster using centralized control, are the strong physical/logical isolation of the online servers, the elimination of the dependency on distributed software for cluster coordination, and the removal of the need for intra-server communications (as required with distributed SCIT versions).

In summary, the centralized control architecture version of SCIT has the following properties, all of which enforce the six SCIT security primitives:

- Server reset (restart) cleans memory and begins the cleansing process.
- Server boot-up from trusted storage loads all files needed to guarantee the "clean" state of the server.
- Each server's lifetime (time in service) is finite, known, and ultimately guaranteed by the failsafe reset mechanism. Limiting the amount of time that each server is exposed to the outside world limits the potential for successful attacks.
- The SCIT Central Controller architecture eliminates the software risks associated with using a distributed algorithm to control the role rotations of the servers in the cluster.
- A guaranteed unidirectional communication channel from the Controller to each server in the cluster ensures the complete and absolute isolation of the Controller from all servers that are in the online (in service) state.

Periodically resetting and cleansing each SCIT server provides three important security advantages that are unique to our system design. First, the server begins each lifecycle in a known clean state free from malware or corruption which provides for assured initial operation or the server. Second, by placing a finite time limit on the exposure of the server to potential attack via the external network—this bounds the probability of a successful attack damaging the SCIT cluster as well as limits the overall impact of an attack. Finally, by rebooting the server and loading protected files from a read-only source any malware or corruption that impacted the server while exposed to the external network are removed from the server without the need to rely on successful detection of the problems.

6. Refactoring for Robustness

While SCIT/HES provides numerous valuable security benefits unique to this architecture, there are several potential points of failure that should be factored out of the design to achieve a balance among security, availability, and robustness.

Centrally-controlled SCIT solves the server coordination and management issues nicely, but as an unwanted byproduct it added a potential critical point of failure in the Central Controller itself. Through refactoring our design we will show the desired balance between security and robustness.

Centralizing control introduces a scalability concern, that is, communication links must be established from the Central Controller to each server. At small scale this is not a problem, but as the scale of the SCIT/HES secure computing cluster approaches realistic enterprise-level size of twenty or more servers, the point-to-point flat controller link topology could become a challenge to scaling and managing the secure computing cluster.

Finally, as an aid to server management and more fine-grained control or server roles we paradoxically also seek to add a 2-way communication channel enabling each server to send signals to the Central Controller for auditing and server control feedback. The design challenge is to do this without violating one of the core tenants of the SCIT design—the guaranteed isolation of online servers from the Central Controller (Table 1: SCIT Primitive 6).

We begin our design refactoring with the Central Controller. As we have seen, the use of a Central Controller provides authoritative cluster control which is both an asset and a liability in terms of being a single point of failure. Figure 3 illustrates a more robust Controller architecture whereby there are now two Controllers (A and B) with one functioning as the active controller and the other serving as a hot failover spare controller. State synchronization between controllers necessary to enable a hot failover is maintained via a backside private LAN connection shown on the left edge of the diagram. The dual Controller setup is shown in context of the entire refactored SCIT/HES architecture shown in Figure 3.

With respect to the challenges of scaling a SCIT secure computing cluster to enterprise-class size of several tens or more servers, we introduce the concept of the SCIT Switchboard. The Switchboard provides several important capabilities that directly impact cluster scalability and management: addressing support for an arbitrary number of Secure Interfaces and simple message queuing. Note that we purposefully renamed the *Trusted Interface Modules* from the original design in Figure 1 to the *Secure Interfaces* here in Figure 3. This was done to help differentiate the two because the refactored Secure Interfaces now support two-way data traffic as we will soon see.

The SCIT Switchboard logically connects both Controllers to an arbitrary number of Secure Interfaces; but each Controller-to-Switchboard interface consists of only two links: an outbound (from the Controller's perspective) Control connection (C) and an inbound Feedback connection (F). Regardless of how many servers are present in a cluster, there will only need to be these four backside connections between the Switchboard and the two SCIT controllers—a significant gain with respect to scaling is the result.

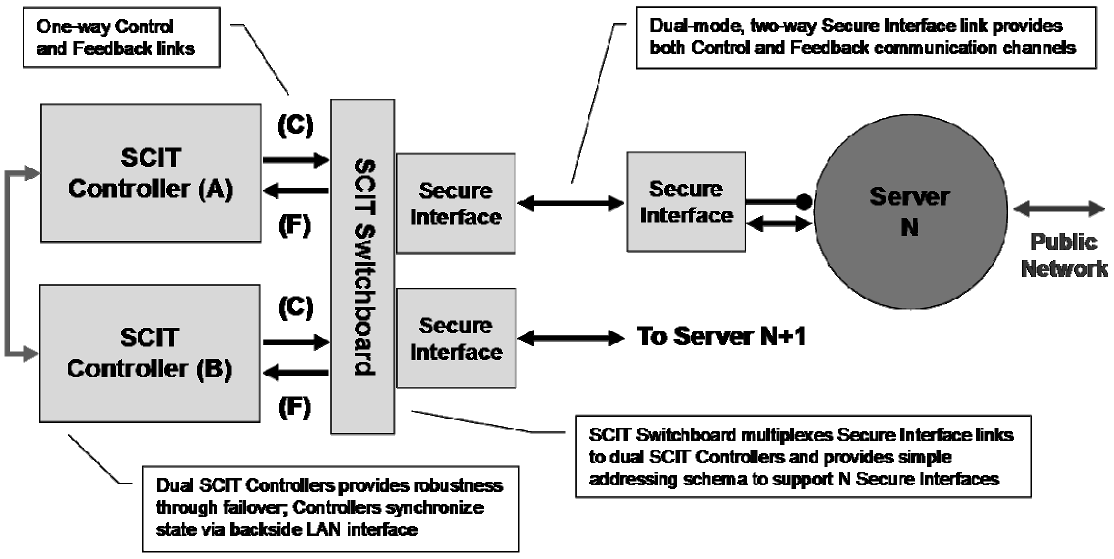
One-way Control and Feedback links
Dual-mode, two-way Secure Interface link provides both Control and Feedback communication channels
SCIT Controller (A)
(C)
(F)
SCIT Switchboard
Secure Interface
Secure Interface
Server N
Public Network
SCIT Controller (B)
(C)
(F)
Secure Interface
To Server N+1
SCIT Switchboard multiplexes Secure Interface links to dual SCIT Controllers and provides simple addressing schema to support N Secure Interfaces
Dual SCIT Controllers provides robustness through failover; Controllers synchronize state via backside LAN interface

Figure 3. Refactored SCIT/HES Architecture

Before discussing Switchboard addressing and message queuing in detail, we need to introduce the new dual-mode Secure Interface that enables bidirectional data traffic flows that is, both to and from the Controller (via the Switchboard). As a major change from the original architecture design we now use the same link for both control messages and server status or feedback messages. The notion of making this critical communications link operate as a dual-mode channel enables the same security benefits as the original design yet allows data to flow back to the Controller, but only under the specific conditions of the *feedback mode* of operation. In our original design, these links are always outbound and unidirectional carrying data from the Controller to each server in the cluster; under the refactored architecture this is now the *control mode* for the link.

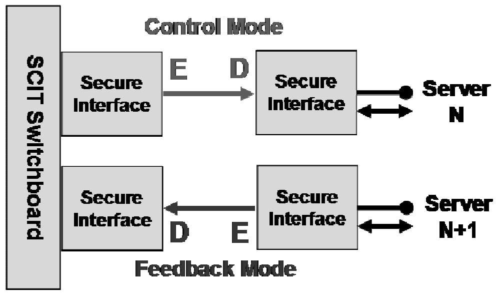

Figure 4. Control and Feedback Modes

The dual-mode Secure Interface links are really two hardware-enforced unidirectional links which share a single communications channel—either electrical or optical; the choice of which type of communications path to use is left to those implementing the system. Using an optical link for explanation of the links operation, we have an emitter/detector module at each end of a fiber optic cable that links the two Secure Interface modules. In *feedback mode*, the communication path is from server to Switchboard (and ultimately to the Controller). The opposite direction of data flow is used for *control mode* as shown in Figure 4. Maintaining this configuration is simply a matter of enabling the emitter at the server side and the detector at the Switchboard side of the link; the mode of the link is hardware controlled by a simple switching circuit that enables one mode or the other but not both at once. *Feedback mode* is enabled once per server self-cleansing cycle after restarting but before being put online. Feedback mode communication occurs after the server is returned to a known state, and this mode is terminated before the server is exposed to the internet, thus ensuring the isolation of the SCIT controller from online servers. Once feedback mode is switched off by the Switchboard, the interface maintains a *control mode* configuration until the next server cycle.

With an understanding of how the dual-mode Secure Interface links work, we can now return to our discussion of how the Switchboard operates. Let us begin with how messages are structured and addressed.

Since a SCIT secure computing cluster will have N servers and since we have already stated that each controller only maintains a single control and single feedback connection to the Switchboard, some form of message addressing is required to route messages to and from the correct server. In keeping with our example of using an optical coupling for the Secure Interfaces, we'll use a simple pulse-based coding for our messages. We need only two fixed length components in our messages: an address and a code, so our messages look like this: `[address][code]`. The specifics of the message content are left to implementation.

In a SCIT system there are no arbitrary data payloads that are exchanged between Controller and the servers but rather status codes that denote *commands* when sent by the Controller or denote *status indicators* when sent by a server. By using a code-based system for communication we avoid the perils of allowing arbitrary data payloads to be exchanged between the Controller and the servers. This design contributes greatly to the isolation and thus safety of the Controller.

Given our simple message format of `[address][code]`, we can implement simple message routing and queuing within the Switchboard. Each Secure Interface has an associated feedback queue used to collect incoming feedback messages from its attached server. A command queue exists for each of the two Controllers to collect outbound command messages sent by controllers. Similarly there is a master feedback queue for each Controller that holds all of the feedback messages sent by servers in the cluster.

A simple logical address-to-physical link mapping schema is implemented by the message router within the Switchboard, see Figure 5. FIFO buffers exist on each Switchboard side of the Secure Interface to remove the need to deal with contention within the Switchboard.

Message processing is straightforward. For feedback messages, processing involves the Switchboard reading (if data exists) each buffer in turn and moving a copy of all received messages into each Controller feedback queue. Each Controller periodically polls its feedback queue for incoming messages.

For command messages, the process begins with the primary Controller issuing a command by pushing onto its command queue in the Switchboard. The message router reads this queue and using the `[address]` portion of the message, decides which physical link to use to forward the message; it does so by writing the `[code]` portion of the message to the correct Secure Interface buffer. In *command mode*, the Secure Interface immediately signals the code points placed in its buffer, thus issuing the command to the server.

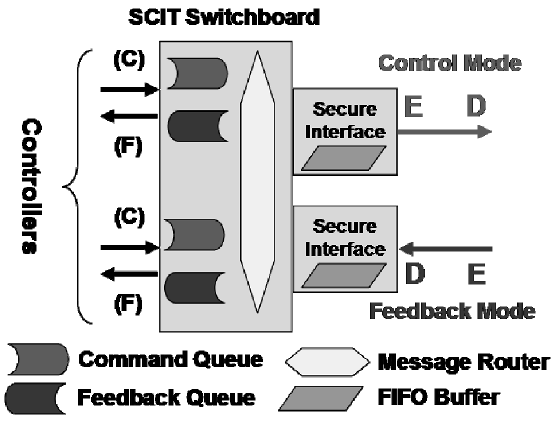

Figure 5: SCIT Switchboard Internals

Advantages of Refactored SCIT/HES

Our refactored SCIT/HES architecture addresses each of the challenges we specified in the last section. Adding a second Controller and keeping it synchronized with the primary Controller enables a hot failover in the event the primary Controller malfunctions. This redundancy eliminates the Controller as a single point of failure.

Adding a SCIT Switchboard with a simple but effective addressing and message routing schema, we greatly increase the capability to expand the SCIT cluster to multiple tens of servers or more to suit enterprise-scale computing needs.

By architecting hardware-enforced dual-mode links in the Secure Interfaces, we enable servers to communicate status messages back to the Controller without losing the critical Controller isolation that underlies the cluster's ultimate security. Use of feedback messages enables the Controller to make allocation decisions as the cluster operates.

By refactoring our centralized-control design we have retained the strong security benefits it affords while introducing increased robustness and flexibility.

7. Related Work

The concept of intrusion tolerance has been previously explored in [8, 9, 10, 11]. Our assumption that undetected intrusions are inevitable and must be treated as an inherent problem of clusters is similar to that of Recovery Oriented Computing, which considers software and human errors as the norm and handles them by isolation and redundancy [12].

Simple forms of server rotations have previously been employed in high-availability systems, where backup servers rotate online to ensure uninterrupted service in face of primary server failures [13, 14, 15, 16]. SCIT systems share many design challenges with high-availability systems, such as the seamless server transitions and sharing of server identities.

We point out that in many server clusters the term "server rotation" often refers to "rotating online servers in servicing arriving clients," typically for the purpose of workload sharing; such rotations are not related to the work presented here.

8. Conclusion

Modern network-based systems are complex interconnected systems which inevitably contain known and unknown flaws. SCIT addresses this situation directly by mitigating the risks associated with known and unknown vulnerabilities in software using a secure clustering approach with self-cleansing processes, separation of duties, and server role rotations.

The SCIT Secure Controller architecture represents a significant evolution in self-cleansing systems and one that seeks to enable the construction of server clusters with provable security characteristics. We have shown that this centralized architecture yields guaranteed security properties but also provides important administrative and economic benefits.

- Clustering, role rotations, and self-cleansing provide security guarantees.
- Depending on security software in any *exposed part* of the system leads to an unknowable state of security.
- SCIT is designed to utilize generic server hardware and requires minimal modifications to this hardware to operate.
- Centralized control provides the advantage of easy administration of a SCIT server cluster.

The use of specialized hardware provides assurance that SCIT servers will operate in a predictable cycle regardless of any potential attacks. The hardware interfaces utilize simple, non-programmable logic circuits which are easy to assemble, inexpensive to build, and impervious to all network-based attacks—both known and unknown.

References

[1] PITAC, Cyber Security: A Crisis of Prioritization, February 2005. www.nitrd.gov

[2] Yih Huang and Arun Sood, "Self-Cleansing Systems for Intrusion Containment," Proceedings of Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN), New York, NY, June 2002.

[3] Yih Huang, Arun Sood, and Ravi K. Bhaskar, "Countering Web Defacing Attacks with System Self-Cleansing," Proceedings of 7th Word Multiconference on Systemics, Cybernetics and Informatics, pp. 12—16, Orlando, Florida, July 2003.

[4] Yih Huang, David Arsenault, and Arun Sood, "SCIT-DNS: Critical Infrastructure Protection through Secure DNS Server Dynamic Updates," Journal of High Speed Networking, vol 15 No 1, pp 5 – 19, 2006.

[5] Yih Huang, David Arsenault, and Arun Sood, "Securing DNS Services through System Self Cleansing and Hardware Enhancements", Proceedings First International Conference on Availability, Reliability and Security (ARES 2006), Vienna, Austria, April 2006.

[6] Yih Huang, David Arsenault, and Arun Sood, "Incorruptible System Self Cleansing for Intrusion Tolerance," to appear in Proceedings Workshop on Information Assurance 2006, Phoenix, Arizona, April 2006.

[7] Yih Huang, David Arsenault, and Arun Sood, "Closing Cluster Attack Windows through Server Redundancy and Rotations," Proceedings of the Second International Workshop on Cluster Security (Cluster-Sec06), Singapore, May 2006.

[8] Yves Deswarte and L. Blain and Jean-Charles Fabre, "Intrusion Tolerance in Distributed Computing Systems," IEEE Symposium on Security and Privacy, 1991.

[9] Pal P, Webber F, Schantz RE, and Loyall JP, "Intrusion Tolerant Systems," Proceedings of the IEEE Information Survivability Workshop (ISW-2000), 24-26 October 2000, Boston, Massachusetts.

[10] Pal P, Webber F, Schantz RE, Loyall JP, Watro R, Sanders W, Cukier M and Gossett J. "Survival by Defense-Enabling," Proceedings of the New Security Paradigms Workshop 2001, pp. 71-78, Cloudcroft, New Mexico, September 11-13, 2001.

[11] S. Dawson,J. Levy, R. Riemenschneider, H. Saidi, V. Stavridou, and A. Valdes, "Design assurance arguments for intrusion tolerance," In Workshop on Intrusion Tolerant Systems, Bethesda, MD, June 2002.

[12] Brown, A. and D. A. Patterson. "Embracing Failure: A Case for Recovery-Oriented Computing (ROC)," High Performance Transaction Processing Symposium, Asilomar, CA, October 2001.

[13] Peter S. Weygant, Clusters for High Availability, Prentice Hall, 1996.

[14] High-Availability Linux Project. www.linux-ha.org.

[15] Steve Blackmon and John Nguyen, "High-Availability File Server with Heartbeat," System Admin, the Journal for UNIX Systems Administrators, vol. 10, no. 9, September 2001.

[16] R. Rabbat, T. McNeal and T. Burke, "A High-Availability Clustering Architecture with Data Integrity Guarantees," Proc. of IEEE International Conference on Cluster Computing, 178–182, (Newport Beach, CA) Oct., 2001.