Closing Cluster Attack Windows Through Server Redundancy and Rotations

Yih Huang, David Arsenault, and *Arun Sood*
Department of Computer Science and Center for Image Analysis
George Mason University, Fairfax, VA 22030
{huangyih, darsenau, asood}@cs.gmu.edu

Abstract — **It is well-understood that increasing redundancy in a system generally improves the availability and dependability of the system. In server clusters, one important form of redundancy is spare servers. Cluster security, while universally recognized as an important subject in its own right, has not often been associated with the issue of redundancy.**

In prior work, we developed a Self-Cleansing Intrusion Tolerance (SCIT) architecture that strengthens cluster security through periodic server rotations and self-cleansing. In this work,[1] we consider the servers in the cleansing mode as redundant, spare hardware and develop a unified control algorithm that manages the requirements of both security and service availability. We will show the advantages of our algorithm in the following areas: (1) Intrusion tolerance through constant server rotations and cleansing, (2) Survivability in events of server failures, (3) Guarantee of service availability as long as the cluster has a minimum number of functioning servers, and (4) Scalability, the support of using high degrees of hardware/server redundancy to improve security and fault tolerance. We provide proofs for important properties of the proposed algorithm. The effects of varying degrees of server redundancy in reducing attack windows are investigated through simulation.

I. Introduction

It is widely accepted that increasing the level of redundancy in a system generally improves service availability and system dependability. System security, on the other hand, is recognized as a critical subject on its own but has not often been associated with the issue of redundancy. This separation is evident when managers consider further investments in hardware. Hardware investments, such as acquiring more computing powers, typically aim to improve services, to handle anticipated increases in workload, or to better assure service survivability at times of server failures — they are not expected to automatically strengthen the security of the system. As emphasized in Figure 1, one goal of this work is to establish the connection between cluster security and hardware redundancy in the form of spare computing powers and in the context of intrusion tolerance.

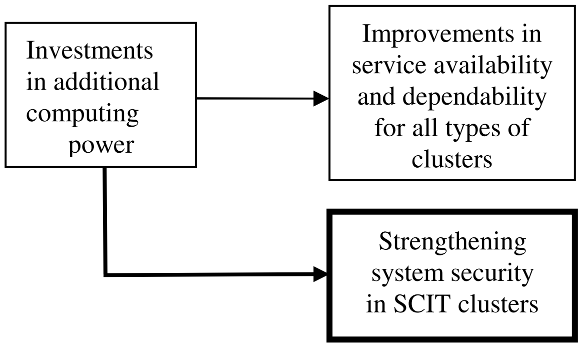

Investments in additional computing power
Improvements in service availability and dependability for all types of clusters
Strengthening system security in SCIT clusters

Fig.1: the relations between additional computing power with service availability and system security

The difficulty in securing computer systems stems in large part from the increasing complexity of the systems today and the constant innovation and morphing of attack techniques. Despite intense research on computer and network security, critical information processing systems remain vulnerable to attacks [1]. We believe that the trend warrants a new thinking in computer security: there will always be attacks that are sophisticated and stealthy enough to penetrate even the best security measures and evade the most advanced intrusion detection systems. It follows that a critical system must support intrusion prevention, detection, and *tolerance*, the last of which fends off, limits, or at least slows down the damages caused by successful but undetected attacks.

Our response to the intrusion tolerance problem is *Self-Cleansing Intrusion Tolerance*, or SCIT [2]. The underlying assumption of SCIT is that a server that has been performing services online and as a result exposed to attacks must be assumed compromised. Consequently, an online server must be periodically cleansed to restore it to a known clean

[1] This research is part of the Critical Infrastructure Protection Project funded by the National Institute of Standards and Technology.

state, regardless of whether an *intrusion is detected or not*. While this paranoid attitude may be overkill for an average server, it is perfectly appropriate for critical infrastructure servers or those whose breaches result in high pecuniary losses or even the compromises to national security. In [2,3,4,5] we presented our designs of SCIT-enabled firewalls, web servers, and DNS servers. We have also proposed hardware enhancements that guarantee the *incorruptibility* of server rotations [5,6]. The primary contribution of this paper is a server-rotation control algorithm to manage the requirements of security and service availability.

The effectiveness of SCIT depends on constant server rotations to limit the window which an intruder can stay in the system. The longer this *Intruder Residence Time* the greater the damage and loss. We anticipate that the loss curve will be an S-curve of the form in Figure 2. If the Intruder Residence Time is less than the low loss threshold, then the cost of the intrusion is low, while an Intruder Residence Time greater than the high loss threshold leads to near max loss. The low loss threshold reflects the fact that it takes some time for an intruder to probe system configurations, issue malicious commands, establish backdoors, install Trojan horse programs and so on in order to gain a foothold. The steep slope between the two thresholds indicates that the intruder is in the middle of achieving the "end goals," such as stealing sensitive information, rendering the service unavailable and/or tampering with important data.

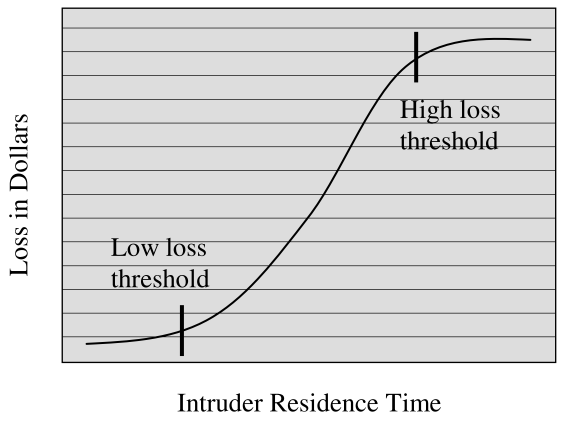


Fig. 2: Loss curve: Loss vs. Intruder Residence Time

Although there is no hard data for the loss curve in Figure 2, there are reports that can validate this curve. For example, in [7] it is reported that in the context of online banking, security experts believe that a theft of $5,000 to $10,000 can be carried out over a few weeks, while larger losses up to $1 million are likely to take four to six months.

It is emphasized that SCIT is not a substitute for the conventional defense systems against intrusion. Hardening system security raises the low-loss threshold by making it more difficult for the enemy to obtain a foothold. In the meantime, frequent server rotations reduce *Server Exposure Times*, the time window in which a server stays online and is inevitably exposed to attacks. A successful, undetected breach is *contained* if the server exposure time is shorter than the low loss threshold, that is, if the compromised server is rotated offline before the breach causes significant damage. The hardening of system security has been the subject of innumerable studies. In contrast, in this paper we focus on increasing security by reducing the attack window, that is, the amount of time any one server is exposed to potential attacks.

In previous research, we noted that Server Exposure Times can be reduced by employing more computing power to speed up server rotations [5]. The contribution of this paper is a unified control algorithm for use in SCIT clusters to manage the requirements of both cluster security and service availability. We will provide proofs to the following properties of the algorithm:

- Minimum Service Guarantee. With arbitrary server failures the cluster maintains predefined minimum service availability as long as the cluster still has a required number of (functioning) servers. Many fault tolerance designs provide similar features.
- Server Rotation Guarantee. Server rotations, the primary security defense of SCIT, continue with arbitrary server failures as long as the cluster has one server more than the required number of servers to meet the minimum service requirement.

Moreover, while it is well understood that increasing server redundancy improves fault tolerance, its effectiveness in closing attack windows will be investigated through a simulation study. Results of this study show that attack windows are less than 5 minutes using the same level of redundancy as the primary-and-backup setup. Even shorter windows can be achieved by adding more computing power, either in the form of more powerful processors in individual servers to speed up self cleansing or in the form of more spare servers in the cluster to speed up server rotations.

The remainder of the paper is organized as follows. Our previous work on SCIT is reviewed in Section II. The contributions of this paper are presented in Sections III to V. Specifically, we define in Section III the configurations parameters for use by the SCIT control algorithm and present in Section IV the proposed algorithm itself. The results of our simulation study are presented in Section V. Studies in software rejuvenation and high availability are closely related to this work and discussed in Section VI. Conclusions are given in Section VII. We provide in an appendix the proofs of the two guarantees mentioned above.

II. SCIT Overview

Shown in Figure 3, a SCIT cluster comprises a set of interconnected servers that cooperatively provide a set of predefined services. Specifically a SCIT cluster supports $\mathcal{M}$ service roles, denoted by $\mathcal{R}_1$ to $\mathcal{R}_{\mathcal{M}}$, using $\mathcal{M}+1$ or more servers. Without loss of generality, in this presentation we assume an assignment of one role per server. Each server in the cluster periodically switches between two modes: online servicing clients in a designated role (depicted as a shaded circle in Figure 3), and offline for cleansing (depicted as a clean circle). A central controller is used to coordinate server rotations and role assignments.

In SCIT, the central controller also maintains the communication configuration depicted in Figure 3, where the controller keeps two-way communication paths with cleansing servers but only one-way paths to reach online servers. Consequently, an online server or any node outside the cluster can reach neither the controller nor those servers in cleansing. It is this complete, hardware-enforced isolation of the central controller from the outside world that underlies its ability to orchestrate the operation of the server cluster. Online servers of course must have two-way communications with clients outside the cluster. We emphasize that the arrows in Figure 3 represent permissible directions in communication; they do not mandate dedicated communication channels.

The communication restrictions in Figure 3 are enforced by the setup shown in Figure 4. In the figure, we use solid lines to represent network links for TCP/IP message exchanges (such as Ethernet cables) and dashed lines to represent wires/fibers that conduct electromagnetic control signals. As seen in Figure 4(a), the controller has two signal lines to reach each server: a reset line and a toggle line. A reset signal forces the server to reboot. The toggle signal controls two switches that are always in opposite states. A toggle signal followed by a reset switches the online server in Figure 4(a) to the offline mode in Figure 4(b). In this setup, a server can either receive messages from outside (and thus is subject to attacks) or send messages to the controller, but never both at the same time. With such physical isolation from external influence, it can be shown that the server rotation and cleansing mechanisms of SCIT is shielded from external influence and attacks [6]. The survivability of SCIT server rotations against server failures is the contribution of this work.

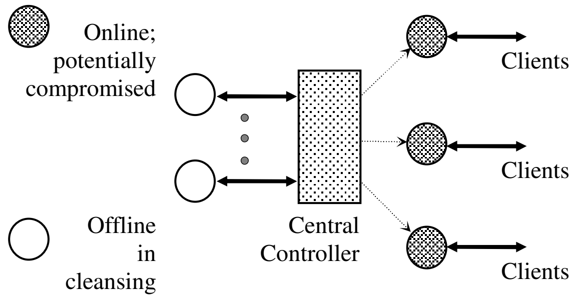
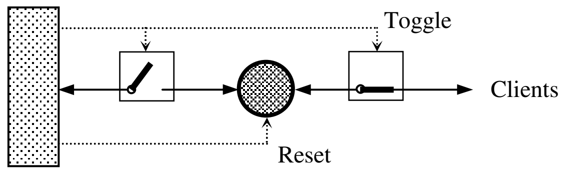
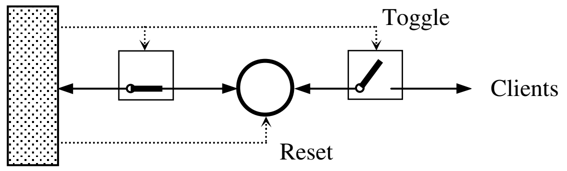


Fig. 3: A High-level view of SCIT



(a) a server presently online



(b) offline server after toggle signal and reset signal

Fig. 4: Enforcing SCIT communication restrictions using simple hardware controls

Part of the rotation process is to bring online servers offline and reset/reboot them to initiate cleansing procedures, in order to return the servers to a well-defined clean state, called the *baseline* state. At a minimum such a state includes system binaries, system configuration files, critical utilities, service binaries, and service configuration files. Many services may include (part of) application data in their baselines. In our SCIT DNS prototype, the baseline state covers the DNS master file and cryptography keys. In our SCIT web prototype, the baseline state includes static HTML pages and web scripts. The offline mode is also suitable for performing many traditional security tasks, such as security updates and system auditing for intrusion/abnormity detection. Indeed, the integration of SCIT operations with traditional system defenses constitutes part of our ongoing research.

We do not expect the concept of SCIT to be compatible with all types of computing systems. Specifically, we have discussed in [2] that SCIT is generally applicable to

- Services that are stateless in nature. Stateless services, such as HTTP, DNS, directory services, and NFS, enable the requests from the same client to be handled by different servers.
- Stateful services that involve short sessions. Many backend office and transaction-oriented services satisfy this requirement.

In both cases, but especially for the latter, server handovers may raise concerns. If an online server is rotated out in the middle of servicing a client, the client experiences service disruption. This problem can be overcome by introducing a *grace period*, T_{gp}, so that the central controller informs an online server of an impending rotation T_{gp} seconds in advance. During the grace period, the "retiring" server finishes processing existing requests while new requests are directed to a different server. To conform with the communication restrictions depicted in Figure 3, the task of communicating the impending rotation to an online server is performed with an interrupt signal from the controller to the server.

III. SCIT CONTROL PARAMETERS

In this section, we discuss the control parameters of a SCIT cluster, shown in Table 1, using as an example the 2DNS-3WEB SCIT cluster depicted in Figure 5. The 2DNS-3WEB cluster supports five roles: a primary DNS server (role P), a web server (role W), a second web server (role W′), a tertiary web server (role W″), and a second DNS server (role S). The purpose of this particular ordering will become clear later.

Table 1. SCIT Cluster Configuration Parameters

Symbol	Description	2DNS-3WEB
$\mathcal{M}$	The number of roles supported	5
$\mathcal{P}$	The rotation pattern	P-W-W′-P-W″-S
$\omega[i]$, $1 \le i \le \mathcal{M}$	The index of role $\mathcal{R}_i$	10, 10, 0, 0, 0
Ω_{min}	The minimum value of the SCIT cluster index Ω	20

Parameter $\mathcal{M}$. A SCIT cluster is configured to support $\mathcal{M}$ service roles. The 3DNS-2WEB cluster, for instance, supports five roles. A service role is said to be *active* if there is an online server servicing in the role; otherwise it is *inactive*. An inactive role is

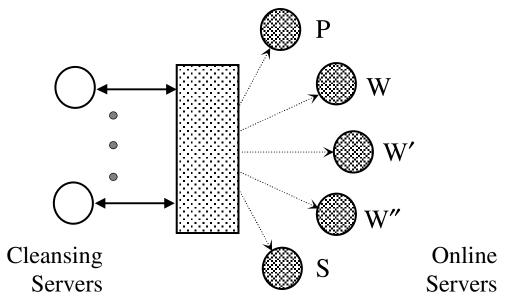


Fig. 5: The 2DNS-3WEB cluster

activated by designating a clean server to assume the role. An active role is *deactivated* by bringing its corresponding server offline without assigning a replacement. Rotating an online server offline and bringing a clean server online as its replacement is called a *swap*. We point out that the number of servers in the cluster, denoted by $\mathcal{N}$, is not an inherent attribute of a SCIT cluster. (In Figures 3 and 5, the total numbers of servers are not specific.) As we will see later, a SCIT cluster operates with any $\mathcal{N} \ge 0$ servers, with the obvious restrictions imposed by the given number of servers. For instance, a SCIT cluster with no servers cannot provide services, but its control algorithm (running on the central controller) continues to operate, waiting for servers to be introduced. On the other hand, we do expect a SCIT cluster to have $\mathcal{N} \ge \mathcal{M}+1$ servers in normal situations.

A Role-$\mathcal{R}_i$ Swap (or simply an $\mathcal{R}_i$ Swap), $1 \le i \le \mathcal{M}$, brings the server presently in role $\mathcal{R}_i$ offline for cleansing and rotates a clean server online to service in role $\mathcal{R}_i$. The 2DNS-3WEB cluster, for instance, implements 5 types of role rotations: P Swap, W Swap, W′ Swap, W″ Swap, and S Swap. Assume that at a given time the cluster contains 6 servers and that the cluster is presently in state

$$(\text{P, W, W}', \text{W}'', \text{S}, c),$$

where servers 1 to 5 are in roles, P, W, W′, W″, and S respectively and server 6 is in the cleansing mode (denoted as *c*). If the first server rotation is a P Swap, then the system enters state

$$(c, \text{W, W}', \text{W}'', \text{S, P}).$$

Further assuming that the next rotation is a W′ Swap, the system subsequently enters state

$$(\text{W}', \text{W}, c, \text{W}'', \text{S, P}).$$

Rotation Pattern $\mathcal{P}$. The central controller uses a rotation pattern to determine the sequence of role switches. For the 2DNS-3WEB cluster, the rotation pattern

$$\text{P}-\text{W}-\text{W}'-\text{W}''-\text{S}$$

dictates that the system cycles through P, W, W′, W″, and S swaps, in that order. That is, the first clean server replaces the server in role P, the second replaces the one in role W, and so forth. As a second example, the rotation pattern

$$\mathbf{P}-\text{W}-\text{W}'-\mathbf{P}-\text{W}''-\text{S}$$

performs the P Swap (in boldface above for emphasis) twice as often as others. We will demonstrate later that different rotation patterns can be used to fine tune the rotation frequencies for different roles according to their respective security requirements.

Index ω. In many applications, some service roles are more important than others. When a number of servers in a cluster fail, remaining, functioning servers must be assigned to the roles most critical to the overall service quality of the cluster. To model the relative importance of service roles, each role $\mathcal{R}$ is associated with a metric-less integer index, $\omega(\mathcal{R})$. The greater its ω index, the more important the role to the cluster. Consider the 2DNS-3WEB cluster again. If the cluster is left with only two functioning servers due to the failures of others, the two must serve as the primary DNS server and a web server. To reflect the importance of the primary DNS and one of web servers, we set the ω indices for the five roles as follows:

$$\omega(\text{P})=\omega(\text{W})=10, \text{ and } \omega(\text{W}')=\omega(\text{W}'')=\omega(\text{S})=0.$$

Notice that the particular values above are somewhat arbitrary. The point is their relative importance.

Minimum cluster index $\Omega_{\min}$. We define the *SCIT cluster index* at time t, denoted by $\Omega(t)$, to be the sum of the ω indices of those roles that are active at time t. We require that the Ω value of a SCIT cluster must be at all times greater than or equal to a predetermined minimum value, $\Omega_{\min}$. If we define $\Omega_{\min}$=20 for the 2DNS-3WEB cluster, then the cluster must meet the minimum requirement by always having one server in role P and one in role W, as long as there are two or more servers still functioning in the cluster. As such, a 2DNS-3WEB cluster with only two servers provides minimum but still useful services.

Without loss of generality, we assume that the ω indices of roles $\mathcal{R}_1$ to $\mathcal{R}_\mathcal{M}$ are in a descending order (not necessarily monotonically); hence the particular order we present the roles of the 2DNS-3WEB cluster earlier. Assuming that there are $\kappa<\mathcal{M}$ servers available, the task of meeting $\Omega_{\min}$ is accomplished by assigning the first $\kappa-1$ servers to roles $\mathcal{R}_1$ to $\mathcal{R}_{\kappa-1}$. Whether the last (and least important) role $\mathcal{R}_\kappa$ is activated by the last available server depends on whether $\Omega_{\min}$ is satisfied by the first $\kappa-1$ roles. If it is, then $\mathcal{R}_\kappa$ is not activated and one server is kept offline to trigger rotations. Otherwise, $\mathcal{R}_\kappa$ is activated and consequently all servers are used online in an attempt to meet $\Omega_{\min}$. The behavior of a SCIT cluster is undefined if its $\Omega_{\min}$ cannot be met by the available servers.

To conclude this section, we show in Table 2 the behaviors of the 2DNS-WEB cluster with various numbers of servers available in the cluster. Recall from Table 1 that the indices of roles P, W, W′, W″, and S are set to 10, 10, 0, 0, and 0 respectively. Moreover, $\Omega_{\min}$ is set 20.

Table 2. Behaviors of the 2DNS-3WEB cluster with different numbers of servers in the cluster

# of servers	Active roles	Ω	Rotation
1	*Undefined*	10	No
2	P and W	20	No
3	P and W	20	Yes
4	P, W, and W′	20	Yes
5	P, W, W′, and W″	20	Yes
≥ 6	P, W, W′, W″, and S	20	Yes

IV. The SCIT Control Algorithm

We present in Figure 6 the algorithm that the central controller of a SCIT cluster uses to manage server rotations, role assignments and server failures. The SCIT_Control algorithm uses variable Ω to keep track of the index value of the entire cluster, variable κ to remember the number of active service roles, and data structure Server_in_Role[i] to record the ID of the server presently online in role $\mathcal{R}_i$. The support of a grace period before resetting an online server is straightforward and omitted in presentation. As a quick starter, we summarize the algorithm in Figure 6 as follows: service roles are activated by Case ($\kappa<\mathcal{M}$) and deactivated by Case Otherwise. Server rotations are carried out by Case ($\kappa=\mathcal{M}$). Details ensue.

To initialize, the SCIT_Control algorithm sets both κ and Ω to 0 before it enters the Main Loop and waits for a ready clean server. This waiting step involves communications between the controller and servers in the cleansing mode, permissible in SCIT as seen in Figure 3. Two cases may occur when a clean server c is ready. If there are some roles still inactive ($\kappa<\mathcal{M}$), then the controller assigns c to the next role $\mathcal{R}_{\kappa+1}$, activating that role. Variable κ is increased by one, and the ω index of the newly activated role is added to the cluster index Ω. Under the normal circumstances where the cluster has more servers than

```
Algorithm SCIT _Control ()
Parameters: see Table 1
Variables:
    $\Omega$: the value of the SCIT cluster index ;
    $\kappa \le \mathcal{M}$: the number of active roles ;
    Server_in_Role[$i$], $1 \le i \le \kappa$: the ID of the
        server presently online in role $\mathcal{R}_i$ ;
Initialization:
    $\kappa = 0$, $\Omega = 0$ ;
Main Loop:
    /* Repeat the following steps */
    Wait for a cleaning server $c$ to report the
        completion of cleansing ;
    Case ($\kappa < \mathcal{M}$): // role activation
        $\kappa = \kappa + 1$ ;
        $\Omega = \Omega + \omega[\kappa]$ ;
        Server_in_Role[$\kappa$] = $c$ ;
    Case ($\kappa = \mathcal{M}$): // server rotation
        $j$ = next_rotation ($\mathcal{P}$, $\kappa$) ;
        Reset Server_in_Role[$j$] for cleansing ;
        Bring server $c$ online to assume Role $j$ ;
        Server_in_Role[$j$] = $c$ ;
Timeout Handler:
    /* timeout when awaiting a clean server */
    Case ($\kappa = 0$): do nothing and return ;
    Case ($\Omega - \omega[\kappa] < \Omega_{\min}$): do nothing and return;
    Case Otherwise:  // role deactivation
        $j$ = next_rotation ($\mathcal{P}$, $\kappa$) ;
        Exchange  Server_in_Role[$j$]  with
                  Server_in_Role[$\kappa$] ;
        Reset  Server_in_Role[$\kappa$] for cleansing ;
        $\kappa = \kappa - 1$ ;
        $\Omega = \Omega - \omega[\kappa]$ ;
```

Fig. 6: The SCIT Control Algorithm

roles, the first $\mathcal{M}$ clean servers are all processed by Case ($\kappa < \mathcal{M}$) to assume roles $\mathcal{R}_1$ to $\mathcal{R}_{\mathcal{M}}$. This is how a SCIT cluster bootstraps itself.

In Case ($\kappa = \mathcal{M}$), all roles are active and the completion of a server cleansing triggers a server rotation. When this occurs, the SCIT_Control algorithm invokes an auxiliary function, next_rotation($\mathcal{P}$, κ), to scan in the rotation pattern $\mathcal{P}$ for the next server swap involving an active role. (The second parameter κ indicates that roles $\mathcal{R}_1$ to $\mathcal{R}_\kappa$ are active.) The identified role swap is subsequently carried out, and the Server_in_Role data structure is accordingly updated.

Under the circumstances of multiple server failures, SCIT may have to operate without enough servers to support all $\mathcal{M}$ service roles. The lack of spare servers in a SCIT cluster manifests itself in the absence of ready clean servers reporting to the control algorithm. To detect the condition, a timer is used to limit the time to wait for a clean server. Generally when the timer fires, some less important roles are deactivated and corresponding online servers brought offline for use in future by more important roles. More specifically, there are three cases to consider when the timer fires.

Case ($\kappa = 0$): there is no role active. In this case, the timeout is ignored, and the SCIT central controller returns to the Main Loop. Under the extreme conditions where the cluster has no (functioning) servers, a second waiting-for-clean-server timeout will ensue, only to be ignored again. The events repeat themselves until at least one server is added to the cluster.

Case ($\Omega - \omega[\kappa] < \Omega_{\min}$): deactivating even the least important active role produces a cluster index Ω below the minimum requirement $\Omega_{\min}$, indicating that the service quality of the cluster would degrade to an unacceptable level. To prevent this from happening, the handler does nothing and leaves the system in its current state. In the 2DNS-3WEB scenario where the primary DNS and a web server are the only ones in the cluster, neither will be rotated and the system operates in a state that provides minimum but useful services without rotations.

Case Otherwise: if neither of the above two cases applies, then task of the SCIT central controller is to deactivate the least important active role in order to revive rotations. To this end, the control algorithm consults the rotation pattern $\mathcal{P}$ to determine the next type of swap j. Rather than immediately rotating the server in role j offline, however, the algorithm exchanges the server in role j with that in role κ and brings the new role-κ server (the old role-j server) offline. The old role-κ server now services in role j. In this way, server rotations continue to cycle through the rotation pattern with minimum loss in Ω.

The Exchange step above must be implemented with care to conform with the communication restrictions of SCIT (Figure 3). The old role-κ server must be brought offline first by a toggle signal, before the controller communicates its new role. A second toggle signal brings the old role-κ server online to service in role j.

Lastly, we present the properties of the SCIT_Control algorithm. We use $\mathcal{N}_{\min}$ to denote the minimum number of servers required to achieve $\Omega_{\min}$ ($\mathcal{N}_{\min}=2$ for the 2DNS-3WEB cluster). Precisely,

$\mathcal{N}_{\min}$ is the smallest integer that satisfies $\sum_{i=1}^{N_{\min}} \omega(\mathcal{R}_i) \geq \Omega_{\min}$.

- Server Rotation Guarantee. With arbitrary server failures, server rotations continue if there are $\mathcal{N} \geq \mathcal{N}_{\min}+1$ functioning servers in the cluster.
- Minimum Service Guarantee. With arbitrary server failures, $\Omega \geq \Omega_{\min}$ is maintained at all times (after an adjustment period) if there are $\mathcal{N} \geq \mathcal{N}_{\min}$ functioning servers in the cluster.

Proofs of the two properties are presented in an appendix.

V. PERFORMANCE STUDIES

In this section, we investigate the effects of varying degrees of server redundancy on security. Our primary metric is Server Exposure Times, the attack windows for would-be intruders.

The Server Exposure Times in a SCIT cluster depends primarily on two factors: the number of spare servers, denoted by $\mathcal{S}$, and the time that each individual server uses to bootstrap and cleanse itself after a reset signal. For the first factor, we consider all offline servers as spares, including the servers in the middle of self-cleansing and those that have completed cleansing and are ready to go online. The degree of server redundancy is measured by the ratio of the number of spare servers to that of service roles supported by the cluster, referred to as the $\mathcal{S}/\mathcal{M}$ ratio.

For the second factor, our SCIT prototypes indicate that server cleansing times ranges from 60 to 90 seconds for DNS servers, to up to 7 minutes for web servers with 40Mbytes of static web pages included in the baseline state. When cleansing times are in the 60 to 90 seconds range, the bootstrapping time of the underlying operating system is the dominating factor. The longer cleansing times of the SCIT web servers reflect the overheads to check the digital signatures of the static web pages by a 1GHz Pentium III processor. To avoid the issues of hardware dependency, we report the Server Exposure Times in terms of a *Standard Cleansing Time*, T_{sc}, defined to be the average cleansing times of all service roles supported by the cluster.

Consider a minimum SCIT cluster with one online server and one spare server. The T_{sc} of the cluster is the self-cleansing time of the web server. In this cluster, a rotation occurs every T_{sc} seconds, and the server exposure times are also T_{sc} seconds. In terms of the degree of redundancy, this cluster is similar to the primary-and-backup setup used by many highly important and/or infrastructural services. Because SCIT also targets these services, we use the $\mathcal{S}/\mathcal{M}$ ratio 1 as the reference in comparisons.

We created a simulation program using the CSIM package [8] to investigate the behaviors of SCIT clusters with varying numbers of spare servers. In each simulation run, we start a SCIT cluster with $\mathcal{M}+\mathcal{S}$ servers being booted up at time 0. The cluster then bootstraps itself as described in Section IV and rotates servers following a given rotation pattern $\mathcal{P}$. The simulation continues until $\mathcal{P}$ is repeated 10,000 times. (For instance if a rotation pattern includes 5 server swaps, the respective simulation involves 50,000 swaps.) Results presented below are the averages over the 10,000 repetitions of the given $\mathcal{P}$. In all simulation runs, we note that results have long stabilized before the completion of 10,000 repetitions.

We present in Figure 7 the results of the 2DNS-3WEB cluster using the round-robin rotation pattern

$$\text{P–W–W}'\text{–W}''\text{–S}.$$

As seen, the average exposure times across all service roles conform to the equation $T_{sc} \times \mathcal{M}/\mathcal{S}$. With one spare, there is one swap for each role $\mathcal{R}$ every repetition of the rotation pattern and hence the average exposure times of $5T_{sc}$. Moving from $\mathcal{S}=1$ to $\mathcal{S}=2$ reduces the exposure times by half. The $\mathcal{S}/\mathcal{M}$ ratio of 1 (five spare servers) produces the average exposure times of one T_{sc}. Using the (somewhat outdated and consequently over estimated) cleansing times of our SCIT prototypes, T_{sc} is 288 seconds, the average cleansing times of 2 DNS servers (90 seconds) and 3 web servers (420 seconds). That is, with $\mathcal{S}/\mathcal{M}=1$, an intruder has less than 5 minutes to penetrate system defenses and cause damages before having to restart again. The attack window can be further reduced to less than 2.5 minutes with $\mathcal{S}/\mathcal{M}=2$. We conclude that, using $\mathcal{S}/\mathcal{M}=1$ as the benchmark, lower degrees of redundancy still produce relatively short attack windows, and extremely short attack windows can be "bought" by higher degrees of redundancy.

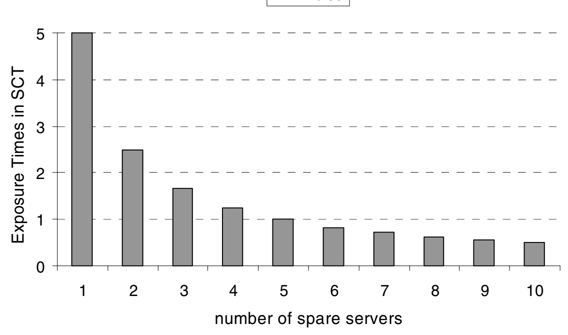


Fig. 7: Server exposure times of the 2DNS-3WEB cluster using the round robin rotation pattern

Next we use a "biased" rotation pattern that gives priority to the security of the primary DNS server (role P). Specifically, we use the rotation pattern

$$P–W–W'–P–W''–S,^2$$

to rotate servers in role P twice as often as others. The results of this rotation pattern are reported in Figure 8. Compared to the results in Figure 7, the second rotation pattern drastically reduces the exposure times of role P. In the case of one spare server, the exposure times of role P are reduced to $2.66\,T_{sc}$ from $5T_{sc}$. Not surprisingly, this drastic improvement is achieved at the expense of exposure times of other roles. Somewhat surprisingly though, the penalties to other service roles are not significant: with one spare server the average exposure times for roles other than P increase from $5T_{sc}$ in Figure 7 to only $5.31\,T_{sc}$ in Figure 8. Moreover, the exposure times averaged across *all* roles are actually reduced from $5T_{sc}$ to $4.43\,T_{sc}$, because of the significant decreases for role P and only marginal increases for others. With the biased pattern and an $\mathcal{S}/\mathcal{M}$ ratio of 1 (5 spare servers), the average exposure time of role P is $0.53T_{sc}$, while those of other roles are $1.06T_{sc}$ (or 153 and 306 seconds respectively when T_{sc}=288 seconds). Noticing its drastic benefits to the primary DNS and the insignificant penalties to others, we choose this biased pattern as the default rotation pattern for the 2DNS-2WEB cluster (see Table 1).

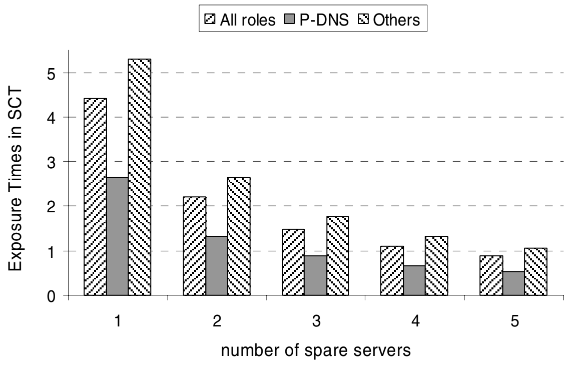


Fig. 8: Server exposure times of the 2DNS-3WEB cluster using a biased rotation pattern

Lastly, we investigate the possibilities of adding randomness to server rotations. We simulated a SCIT cluster that employs 200 servers to support 100 roles ($\mathcal{S}=100$) and uses a round-robin rotation pattern interposed by random swaps. More specifically, the cluster uses the rotation pattern

$$\mathcal{R}_1–X–\mathcal{R}_2–X–\mathcal{R}_3–X–\mathcal{R}_4–X– \ldots –\mathcal{R}_{\mathcal{M}}–X,$$

where X represents a Role-$\mathcal{R}_j$ swap and j is a uniform random variable from 1 to $\mathcal{M}$. The self-cleansing times in the simulation are generated by a uniform random variable from 60 seconds to 420 seconds, producing T_{sc}=240 seconds. The results are reported in Figure 9. In the figure, the height of a bar represents the probability for the duration of an *instance* of online service to fall within a 10-second interval. For example, the far left bar in the figure shows that the probability for a newly online server to stay online for 0 to 10 seconds is 0.0322. The largest probability 0.044 in the figure corresponds to the interval of 460 to 470 seconds (the (470, 480) interval is a close second), while the smallest value 0.0093 corresponds to the interval (410, 420). The average exposure time of all instances of online service is almost exactly 240 seconds, or one T_{sc}. However, the exposure times of individual instances of online servicing are relatively evenly distributed from 0 seconds to up to 500 seconds. We consider these results most encouraging. Indeed, there is a formidable challenge to breach a system within the attack windows varying unpredictably from 0 to 9 minutes. Exploiting randomness in various aspects of the SCIT framework constitutes part of our ongoing research.

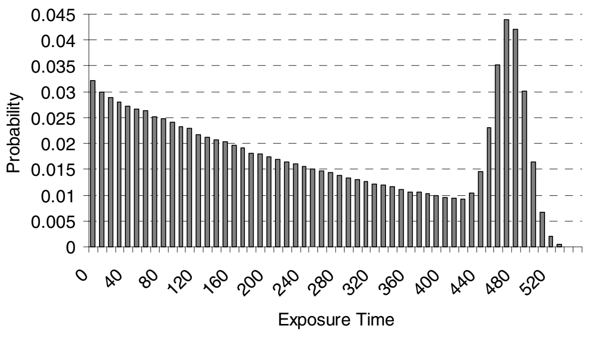


Fig. 9: The probability density function of the Server Exposure Times in the 100-Role SCIT cluster

VI. Related Work

The concept of intrusion tolerance has been explored in [9,10,11,12,13]. 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 [14].

[2] This simulation is intended to demonstrate the effects of different patterns. We are not advocating that the primary DNS is exactly twice as important as other roles such as web servers.

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 [15,16,17,18]. SCIT systems share many design challenges with high-availability systems, such as the seamless server transitions and sharing of server identities (IP and/or hardware addresses).

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. On the other hand, our server rotation and self-cleansing processes can be considered as a special form of software rejuvenation [19,20,21,22] for use by server clusters.

VII. Conclusion

The administrators of highly critical and/or infrastructure systems have long deployed backup servers to ensure service availability when server failures occur. The computing powers of the backup servers, by their nature, are left unused most of the times. We have presented in this paper a cluster control algorithm that takes advantages of the idle computing powers of backup/redundant servers to help strengthen system security. We have argued that the proposed algorithm survives arbitrary server failures while providing minimum service guarantees. It has been shown by simulation that the proposed algorithm limits the attack windows of would-be intruders to less than 10 minutes, using the degree of redundancy typically seen in primary-and-backup configurations. It has also been observed that extremely short attack windows can be "bought" by higher degrees of redundancy. We have thus established the connection of system security with hardware redundancy, adding a new dimension to secure cluster designs.

Appendix
Proofs of SCIT_Control Properties

In this appendix we investigate how the SCIT_Control algorithm handles server failure events. The events of new servers being added to the cluster or repaired servers returning to the cluster manifest themselves in the availability of clean servers in the Main Loop. The correctness of their processing is obvious due to the simplicity of the Main Loop. The failures of servers on the other hand may render the bookkeeping of the SCIT_Control algorithm incorrect and their handling requires further studies.

The state of a SCIT cluster comprises two parts. The *algorithm state* of the cluster is defined by the values of the data structures used by the algorithm, that is, the values of κ, Ω, and Server_in_Role[i], $1 \le i \le \kappa$. The *actual state* of the cluster includes the values of these variables in reality *plus* the number of servers in the cluster $\mathcal{N}$. A SCIT cluster is in a *consistent* state if its algorithm state conforms to the actual state.

In our arguments, we assume that up to a reference time point, T_0, the cluster is in a consistent state. Further, we assume that there will be no server failures after a second time point, $T_1 \ge T_0$. During the period (T_0, T_1), there is one or more server failure. Our goal here is to analyze the behaviors of the SCIT_Control algorithm after time T_1, given that the cluster still has enough servers to meet $\Omega_{\min}$. Simply put, we investigate the behaviors of the algorithm after a chaotic period of arbitrary server failures and analyze how it resolves the problems of inconsistency. One inconsistency problem is the *ghost* servers, each of which according to the Server_in_Role data structure is presently in some role $\mathcal{R}_i$ but in reality nonexistent due to failures in the (T_0,T_1) period.

We recall that $\mathcal{N}_{\min}$ is defined as the minimum number of servers to achieve $\Omega_{\min}$, that is, the smallest integer that satisfies $\sum_{i=1}^{N_{\min}} \omega(\mathcal{R}_i) \ge \Omega_{\min}$. Because of the requirement to meet $\Omega_{\min}$, we are interested only in those cases where $\mathcal{N} \ge \mathcal{N}_{\min} > 0$. As pointed out earlier, the behavior of a SCIT cluster is undefined if its $\Omega_{\min}$ requirement is impossible to meet. It turns out that the case of $\mathcal{N} = \mathcal{N}_{\min}$ needs a different treatment from the cases of $\mathcal{N} > \mathcal{N}_{\min}$. We start the analysis with the latter. Let us begin with a simple observation.

Observation 1. Assuming $\kappa = \mathcal{N}$, if there is a ghost server, there is a spare server.

To see this, consider again the 2DNS-3WEB cluster, whose $\mathcal{N}_{\min}=2$. The controller "believes" that the $\kappa=2$ roles are P and W. Assuming that the server which is supposedly in role P is a ghost, there must be another server that is not in any role to account for the $\mathcal{N}_{\min}=2$ servers in reality. In fact, the number of ghosts and that of the spares should always be equal.

Observation 2. With $\mathcal{N} > \mathcal{N}_{\min}$ servers in the cluster, condition $(\kappa<N)$ produces at least one rotation in future.

$(\kappa < \mathcal{N})$
$\Rightarrow$ there is at least one spare server
$\Rightarrow$ there will be at least one rotation hereafter

Observation 3. With $\mathcal{N} > \mathcal{N}_{\min}$ servers in the cluster, condition ($\kappa = \mathcal{N}$) produces at least one rotation in future.

($\kappa = \mathcal{N}$) and ($\mathcal{N} > \mathcal{N}_{\min}$)
$\Rightarrow$ all servers online (some may be ghosts)
$\Rightarrow$ timeout
$\Rightarrow$ Case ($\kappa = 0$) is impossible for $\kappa = \mathcal{N} > 0$.
$\Rightarrow$ Case ($\Omega - \omega[\kappa] < \Omega_{\min}$) is also impossible for $\kappa = \mathcal{N} > \mathcal{N}_{\min}$ and the definition of $\mathcal{N}_{\min}$.
$\Rightarrow$ Case Otherwise must occur
$\Rightarrow$ a role $\mathcal{R}$ is deactivated; $\kappa = \kappa - 1$

If role $\mathcal{R}$ was served by a ghost server
$\Rightarrow$ there is one spare server by Observation 1.
$\Rightarrow$ the spare will trigger a rotation in future

If role $\mathcal{R}$ was served by a real server
$\Rightarrow$ the sever has been reset for cleansing above
$\Rightarrow$ it will trigger a rotation after cleansing

Observation 4. With $\mathcal{N} > \mathcal{N}_{\min}$ servers in the cluster, condition ($\kappa > \mathcal{N}$) produces at least one rotation in future.

($\kappa > \mathcal{N}$) and ($\mathcal{N} > \mathcal{N}_{\min}$)
$\Rightarrow$ all servers online with $\kappa - \mathcal{N}$ ghosts
$\Rightarrow$ timeout
$\Rightarrow$ Case ($\kappa = 0$) is impossible for $\kappa > \mathcal{N} > 0$.
$\Rightarrow$ Case ($\Omega - \omega[\kappa] < \Omega_{\min}$) is also impossible for the precondition of this observation and the definition of $\mathcal{N}_{\min}$.
$\Rightarrow$ Case Otherwise must occur
$\Rightarrow$ a role $\mathcal{R}$ is deactivated, and $\kappa = \kappa - 1$
$\Rightarrow$ If the new $\kappa > \mathcal{N}$, we have the same condition of this Observation and the above events will repeat until κ is reduced to $\mathcal{N}$.
$\Rightarrow$ If the new $\kappa = \mathcal{N}$, there will be a rotation by Observation 2.

Property 1. If $\mathcal{N} > \mathcal{N}_{\min}$ at time T_1, then server rotations continue indefinitely.

At time T_1, the cluster is in state $\kappa > \mathcal{N}$, $\kappa = \mathcal{N}$, or $\kappa < \mathcal{N}$. Each guarantees a future rotation by Observations 2, 3, and 4. The state of the cluster after the rotation must also be one of the three states, guaranteeing further rotations indefinitely.

Property 2. If $\mathcal{N} > \mathcal{N}_{\min}$, then there exists a time $T_2 \geq T_1$ such that $\Omega(t) \geq \Omega_{\min}$ for all $t \geq T_2$.

We already show in Property 1 that server rotations will continue with the precondition. Server rotations are managed by Case ($\kappa = \mathcal{M}$) and Case Otherwise. Both follow the rotation pattern $\mathcal{P}$ to cycle through active roles (active as perceived by the algorithm). Ghost servers in active roles will eventually be identified by the rotation pattern for rotation and subsequently seen by the algorithm as being "reset and in cleansing." Being nonexistent in reality, these offline ghost servers will not return as clean servers in future. After rooting out ghost servers in active roles, the algorithm's Ω variable reflects the real Ω in the cluster. With Case ($\Omega - \omega[\kappa] < \Omega_{\min}$) the algorithm will eventually assign $\mathcal{N} - 1$ servers to the first $\mathcal{N} - 1$ roles to satisfy $\Omega_{\min}$. As such, the cluster eventually satisfies the minimum requirement of the cluster and stay so thereafter.

We now turn our attention to the special case of $\mathcal{N} = \mathcal{N}_{\min}$, where the server rotations will have to stop due to the lack of spare servers. We again start with some observations.

Observation 5. The condition ($\kappa < \mathcal{N}_{\min}$) is transient.

($\kappa < \mathcal{N}_{\min}$)
$\Rightarrow$ one or more spare servers in the system
$\Rightarrow$ Case ($\kappa < \mathcal{M}$) rotates a spare online
$\Rightarrow$ κ is increased by 1
$\Rightarrow$ if ($\kappa < \mathcal{N}_{\min}$) still holds, the above events will be repeated.
$\Rightarrow$ $\kappa \geq \mathcal{N}_{\min}$ eventually

Observation 6. With $\mathcal{N} = \mathcal{N}_{\min}$ servers, condition ($\kappa = \mathcal{N}_{\min}$) leads to one of the two outcomes below.

Outcome 1: System settles down in a consistent state
Outcome 2: κ becomes greater than $\mathcal{N}_{\min}$

When the system is in the given condition, there are two possibilities:

- Algorithm state is consistent with the real state of the cluster. In this case, we are done.
- If there is a ghost server, then there is a spare server in reality by Observation 1.

Let us consider what happens to the spare server.

- The spare server is rotated online by Case ($\kappa < \mathcal{M}$). In this case, κ is increased by one and we have the condition ($\kappa > \mathcal{N}_{\min}$). We are done.
- The spare server is rotated online by Case ($\kappa = \mathcal{M}$). In this case, κ is not changed and the above analysis about condition ($\kappa = \mathcal{N}_{\min}$) can be applied again. If Case ($\kappa < \mathcal{M}$) eventually occurs, we are done.
- If Case ($\kappa < \mathcal{M}$) never occurs, spare servers are rotated online only by Case ($\kappa = \mathcal{M}$). This however cannot go on forever, for the rotations will eventually go through all the roles presumed active by the algorithm. When a role corresponding to a ghost server is rotated by Case ($\kappa = \mathcal{M}$), a spare server assumes the role while the nonexistent ghost will not return as a clean server. A pair of ghost and spare servers is thus removed. A cycle of rotations through the

(presumed) active roles will root out all ghost servers and return the cluster to a consistent state.

Observation 7. With $\mathcal{N} = \mathcal{N}_{\min}$ servers in the cluster condition $\kappa > \mathcal{N}_{\min}$ produces server rotations.

First, there has to be inconsistency in the system with the above condition, for the SCIT_Control algorithm is using more servers than is actually available in the cluster. Because the algorithm believes it has more active roles than needed to meet the minimum requirement, it also believes that it can afford rotations. Specifically, Cases ($\kappa=0$) and ($\Omega - \omega[\kappa] < \Omega_{\min}$) will not occur, and timeouts are handled by Case Otherwise, which rotate servers following $\mathcal{P}$.

If there is any spare server, the server will be rotated online in the Main Loop. Otherwise, timeouts occur to produce rotations as argued above. We are done.

Observation 9. With $\mathcal{N} = \mathcal{N}_{\min}$ servers in the cluster, if there is any ghost server when $\kappa \geq \mathcal{N}_{\min}$, the ghost will be removed by server rotations.

Observations 6 and 7 guarantees server rotations if $\kappa \geq \mathcal{N}_{\min}$ and if there are ghost servers. As argued previously, ghost servers are removed after the rotation cycles through all active roles in $\mathcal{P}$.

Property 3. With $\mathcal{N} = \mathcal{N}_{\min}$ servers in the cluster, the system eventually settles down to the consistent state where $\kappa = \mathcal{N}_{\min}$ and $\Omega \geq \Omega_{\min}$ without server rotations.

At time T_1, the system is in condition $\kappa < \mathcal{N}_{\min}$, $\kappa = \mathcal{N}_{\min}$, or $\kappa > \mathcal{N}_{\min}$. By Observation 5 it will eventually be in the condition $\kappa \geq \mathcal{N}_{\min}$, under which ghost servers will be removed by Observation 9. Without ghost servers, the SCIT_Control algorithm uses the $\kappa = \mathcal{N}_{\min}$ available servers to meet the minimum requirement $\Omega_{\min}$.

ACKNOWLEDGMENT

We gratefully acknowledge the guidance provided by Robin Quinlan, Office of the Secretary of Defense. The importance of the trade-off between security and availability was highlighted in communications and discussions with her. Those exchanges resulted in some of the experiments reported in this paper.

REFERENCES

[1] President's Information Technology Advisory Committee (PITAC), Cyber Security: A Crisis of Prioritization, February 2005. Available at 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 City, 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," presented at the Trusted Internet Workshop Conference, Bangalore, India, December 2004. (Extended version to appear in Journal of High Speed Networking)

[5] Yih Huang, David Arsenault, and Arun Sood, "Securing DNS Services through System Self Cleansing and Hardware Enhancements," to appear in *Proceeding First International Conference on Availability, Reliability, and Security* (AReS 2006), Vienna, Austria.

[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] Sandeep Junnarkar, "Anatomy of a hacking", available at http://news.com.com/2009-1017-893228.html, May 2002.

[8] B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer eds, "CSIM19: A Powerful Tools for Building System Models," *Proceedings of the 2001 Winter Simulation Conference*, Arlington VA.

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

[10] 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.

[11] 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.

[12] 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.

[13] M. Atighetchi, P. Pal, F. Webber, R. Schantz, C. Jones, J. Loyall, "Adaptive Cyberdefense for Survival and Intrusion Tolerance," *IEEE Internet Computing*, vol. 08, no. 6, pp. 25-33, Nov/Dec, 2004.

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

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

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

[17] 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.

[18] 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.

[19] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, “Software Rejuvenation: Analysis, Module and Application,” In Proc. of the 25th Intl. Symposium on Fault Tolerant Computing, pp. 381—390, Pasadena, CA, June, 1995.

[20] William Yurcik and David Doss, "Achieving Fault-Tolerant Software with Rejuvenation and Reconfiguration," IEEE Software, July/August 2001, pp. 48-52.

[21] K. Vaidyanathan, R. E. Harper, S. W. Hunter, K. S. Trivedi, "Analysis and Implementation of Software Rejuvenation in Cluster Systems," in Proc. of the Joint Intl. Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 2001/Performance 2001, Cambridge, MA, June 2001.

[22] Khin Mi Aung, Kiejin Park, and Jong Sou Park, "A Rejuvenation Methodology of Cluster Recovery," Cluster-Sec, 2005.