Incorruptible Self-Cleansing Intrusion Tolerance and Its Application to DNS Security

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*— Despite the increased focus on security, critical information systems remain vulnerable to cyber attacks. The trend lends importance to the concept of *intrusion tolerance*: there is a high probability that systems will be successfully attacked and a critical system must fend off or at least limit the damage caused by unknown and/or undetected attacks.**

In prior work, we developed a Self-Cleansing Intrusion Tolerance (SCIT) architecture that achieves the above goal by constantly cleansing the servers and rotating the role of individual servers. In this paper[1], we show that SCIT operations can be incorruptibly enforced with hardware enhancements. We then present an incorruptible SCIT design for use by one of the most critical infrastructures of the Internet, the domain name systems. We will show the advantages of our designs in the following areas: (1) incorruptible intrusion tolerance, (2) high availability, (3) scalability, the support for using high degrees of hardware/server redundancy to improve both system security and service dependability, and (4) in the case of SCIT-based DNSSEC, protection of the DNS master file and cryptographic keys. It is our belief that incorruptible intrusion tolerance as presented here constitutes a new, effective layer of system defense for critical information systems.

***Index Terms*— computer security, self-cleansing system, intrusion tolerance, domain name system**

I. INTRODUCTION

A. Motivations of SCIT

Networks and the systems that run on them have become essential to the operation of business enterprises, functioning of the global economy, and the defense of the nation. Yet these critical information systems remain vulnerable even with the recently increased focus on security [1]. The problem stems in large part from the constant innovation and evolution of attack techniques. The increasing sophistication and incessant morphing of cyber attacks lend importance to the concept of *intrusion tolerance*: a critical system must fend off or at least limit the damages caused by unknown and/or undetected attacks.

An *unknown* attack is an attack based on new vulnerabilities, exploits and/or attack techniques that are yet unknown to the public. An *undetected* attack is a successful attack that evades intrusion detection mechanisms long enough to cause significant losses. The two concepts are related but not identical: while an unknown attack has greater opportunities to evade detection, an undetected attack could be based on previously known attack techniques. The two combined represents the "unknown unknowns" faced by today's critical information systems in cyber warfare.

Our response to these formidable challenges is Self-Cleansing Intrusion Tolerance, or SCIT. The underlying assumption is that all software is malleable and intrusion detection cannot absolutely detect all system breaches. It follows that a server that has been online and exposed to attacks must be assumed compromised. Consequently, an online server must be periodically cleansed to restore it to a known clean state, regardless of whether an *intrusion is detected or not*. In [2,3,4,5,6,7] we presented our designs of SCIT-enabled firewalls, web servers, and DNS servers.

B. Domain Name Systems

As part of the Internet infrastructures, the Domain Name System (DNS) is essentially a distributed database that maps easily-remembered host names to their numerical IP addresses [8]. Each name server maintains the domain name information regarding a subspace, or a zone, in the DNS name space. The domain names and IP addresses pertaining to a zone are stored in a master file, maintained by the primary name server of that zone. Each zone also has one or more secondary name servers, which periodically synchronize local DNS data with the master file. Secondary name servers respond to DNS queries but are not involved in maintaining the master file. DNS also supports the dynamic updates of domain names so that new domains can be added or the attributes of existing domains can be changed in a near real-time manner [9]. Because of the critical importance of DNS, many sites deploy dedicated backup servers which take over DNS services in face of online server failures.

[1] This research is part of the Critical Infrastructure Protection Project funded by the National Institute of Standards and Technology. The paper is based on "Securing DNS Services through System Self Cleansing and Hardware Enhancements," by Yih Huang, David Arsenault, and Arun Sood, which appeared in Proceeding First International Conference on Availability, Reliability, and Security (AReS 2006), Vienna, Austria.

DNS was later enhanced with DNS Security Extensions (DNSSEC) to provide data origin authentication [10]. With DNSSEC, each zone is equipped with (at least) a pair of public and private keys. The private key is used to digitally sign DNS data. Clients verify the origin of received DNS data by checking accompanying signatures using the public key. The integrity of the approach depends on the secrecy of the private keys.

Unfortunately DNS/DNSSEC implementations contain vulnerabilities and have been the target of numerous exploits and attacks [11,12,13]. Indeed, the SANS (SysAdmin, Audit, Network, Security) Institute places BIND, the most popular DNS/DNSSEC implementation, as the number one among the top 10 vulnerabilities to Unix systems [14].

C. Contributions

The focus of this paper is on a hardware driven implementation of SCIT. This is distinct by the primarily software approaches in [2,3,4]. The effectiveness of SCIT depends on fast self-cleansing cycles, thus restricting would-be attackers to short time windows to breach the system before restoration through self-cleansing. In response, an attacker may target SCIT itself in an attempt to defeat this last line of system defense.

In this paper we investigate the survivability of SCIT under unknown and/or undetected attacks and present hardware solutions to enforce several incorruptible properties of SCIT operations, called *SCIT Primitives*. We call the new generation of the SCIT framework SCIT/HES (for Hardware Enhanced Security). We then apply SCIT/HES to DNS services, resulting in the DNS-HES (DNS with Hardware Enhanced Security) architecture. Our design improves the security of critical, infrastructure servers in the following areas.

- High availability — the graceful handling of server failures.
- Scalability — the support of high degrees of hardware/server redundancy to improve both system security and availability.
- Incorruptible intrusion tolerance — the operations of SCIT/HES cannot be compromised by remote attacks. There could still be successful, undetected attacks on online servers. However, SCIT/HES will continue to limit the resulting effects and losses by server rotation and cleansing.
- Protection of the zone master file and private keys in the case of DNS-HES — the zone master file and DNSSEC private keys are never exposed to the public Internet and therefore are not subject to remote attacks. With many, if not all, DNS and DNSSEC implementations, the master file and some private keys must be kept online to support secure dynamic domain name updates. These requirements are nullified by the DNS-HES architecture.

We point out that the first two advantages above are also supported by our previous, software-based SCIT designs [2,3,4]. The additional properties of incorruptibility and scalability of SCIT/HES (and DNS-HES in particular) are derived from the new designs presented in this paper, which in turn is based on concise and preliminary results published in [5,6].

It must be emphasized that SCIT systems (including the hardware-enhanced designs presented here) do not exclude the use of intrusion prevention and detection technologies, but rather add another layer of defense, extending the idea of "defense-in-depth" through periodic system cleansing.

The remainder of the paper is organized as follows. We give related work in Section II and review SCIT basics in Section III. We present the SCIT/HES framework in Section IV. Specifically we list a set of criteria, called SCIT Primitives, which collectively ensure the incorruptibility of server rotation and cleansing cycles, and in the same section describe the hardware enhancements to SCIT in order to enforce said primitives. As an example, we apply SCIT/HES in Section V to the DNS service and present the architecture of DNS-HES clusters. We give in Section VI the DNS server rotation algorithms for use by DNS-HES. In Section VII, we discuss the ramifications of SCIT/HES on hardware redundancy, service availability, and system security. We give concluding remarks in Section VIII.

II. RELATED WORK

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 [15].

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 [16,17,18,19]. 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). Examples of high-availability systems include DNS servers, NFS servers, authentication services, firewalls, IPsec gateways, and virtual private network gateways. The methodology presented in this paper can be used to protect these servers.

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 [20,21,22,23] for use by server clusters.

III. SCIT REVIEW

A SCIT cluster comprises a set of interconnected servers that cooperatively provide a predefined service. Any server in the cluster periodically switches between two modes: online servicing clients (which are outside

the cluster) and offline for cleansing. Either a central controller or a distributed control mechanisms using a Cluster Communication Protocol (CCP) can be used to coordinate server mode rotations [4]. A high level view of SCIT cluster operations is depicted in Fig. 1.

Part of the rotation process is to bring online servers offline. Next, the system is rebooted to initiate cleansing procedures in order to return to a well-defined clean state. At a minimum such a state includes system binaries, system configuration files, critical utilities, service binaries (BIND binaries, Apache binaries, etc.), and service configuration files. Many services may include (part of) application data as well. For SCIT DNS servers, for instance, the clean state also covers the DNS master file and cryptography keys. For the SCIT web servers, the clean state covers static HTML pages and web scripts. In many applications, audit functions can also be performed on offline servers.

IV. Hardware Enforced Incorruptibility

When considering unknown and undetected attacks, one must assume that the self-cleansing process is also subject to attacks. It is indeed possible to interfere with the operations of SCIT. To interfere with the self-cleansing process after rebooting, an attacker could install Trojan horse copies of system/cleansing utilities, hack startup tasks/processes, or even tamper with the bootstrapping procedure of the operating system. The successful completion of cleansing does not guarantee the assumption of a desired online service either. The server could already have been under attack while cleansing, and going online inevitably involves communications, giving opportunities for unpredicted breaches. Lastly, if an online server has been taken over by attackers, the process of counting down to the next rebooting could be interrupted, stopping SCIT cleansing cycles all together.

A. SCIT Primitives

In the following we present a set of properties, or primitives, so that if a given SCIT cluster satisfies these properties, then it is said to be SCIT incorruptible. In the discussion an exposed node refers to an online server inside the SCIT cluster or any computer/server outside the cluster. Among the primitives below, Primitives P1 to P4 apply to all SCIT designs while Primitive P5 applies to only those with a central controller.

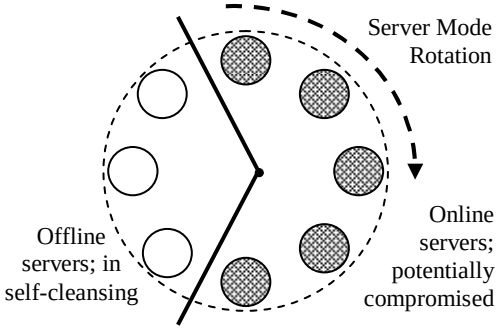


Figure 1. A high-level view of SCIT cluster operations

P1. **Inevitability of periodic server cleansing**. A server will be rebooted and subsequently cleansed within a predetermined[2] length of time.

P2. **No communications from exposed nodes to cleansing servers**. Consequently, a cleansing server is not subject to remote attacks. Notice that we do not disallow cleansing nodes to send messages/signals to exposed nodes.

P3. **Completion of cleansing.** The cleansing procedure will be completed so that system is in a predefined clean state.

P4. **Guaranteed role assumption**. A newly cleansed server will assume a designated online role/identity.

P5. **No communications from exposed nodes to the central controller.** It is thus impossible to attack the central server at any time. Again, we allow the controller to send messages/signals to exposed nodes.

Not all SCIT designs satisfy these demanding requirements. In fact, with the assumption that software is eventually corruptible, an entirely software-based SCIT system cannot satisfy all the primitives and therefore is subject to compromises in its own operations. Next, we present a SCIT framework that is incorruptible as defined above through the use of simple hardware enhancements.

B. Hardware Enhancements

The SCIT primitives are best achieved by isolation. A server is completely shielded from external influence if it is "physically" cut off and if it does not process data left by online servers. To achieve cutting-off in an incorruptible way, simple hardware devices, such as on-off switches, are employed. We refer to the resulting framework as SCIT/HES (for Hardware Enhanced Security).

In SCIT/HES, a central controller is used to manage server rotations and role assignments. (In implementations, the controller could also be an off-the-shelf server box; the name suggests its use not its construction) It also maintains the communication configuration, shown in Fig. 2, where the controller keeps two-way communication paths with clean servers but

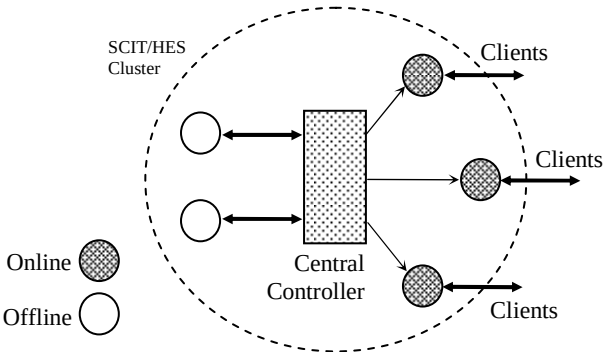


Figure 2. Communication configuration of SCIT/HES

[2] For additional protection the server cleansing time could be random and change in each cycle. The time length in Primitive P1 is the longest cleansing cycle allowed.

only one-way paths to reach online servers. Consequently, an online server or any node outside the cluster (the exposed nodes) can reach neither the controller nor those servers in cleansing. Online servers of course must have two-way communications with clients outside the cluster. We emphasize that the arrows in Fig. 2 represent permissible directions in communication; they do not mandate dedicated communication channels.

Due to server rotations, the configuration of a SCIT/HES cluster is dynamic. Communication paths must be cut and reestablished when servers switch between online and offline modes. In SCIT/HES, the central controller also manages communication paths. The setup between the controller and an individual server is presented in Fig. 3. 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 Fig. 3(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 Fig. 3(a) to the offline mode in Fig. 3(b). As such, a server can either receive incoming messages from outside (and thus is subject to attacks) or send messages to the controller, but never both at the same time. Clean servers receive inputs from only the central controller. Primitives P2 and P5 are thus fulfilled.

With the central controller shielded from any form of external influence and in charge of periodically resetting servers, Primitive P1 is fulfilled.

Primitive P3 concerns the bootstrapping and cleansing procedures used by SCIT servers after receiving reset signals. A simple way to enforce Primitive P3 is that the entire system uses read-only storage for system state. For instance, all system components and service software are stored on a CD-ROM. In this case, rebooting and loading the system from the CD-ROM is the entire cleansing process; afterward the system is in a known clean state. This solution has potential performance problems, due to the slowness of optical drives. Also, many services require the predefined "system state" to cover (some) data.

A more flexible approach is to bootstrap a server from a read-only device. However the application data and the binaries that the server needs to perform its online functions are stored in a write-able storage, called the root hard drive. When the server enters an online mode, it switches to the hard drive as the root file system. The self-cleansing procedure checks the integrity of the root hard drive and restores it to a clean state if corruptions are detected. In the self-cleansing mode, all executables (the kernel, utilities, system check tools, etc.) are retrieved from the read-only device. We notice that before a server goes online, it cannot be reached by exposed nodes and all its configurations and binaries are from read-only storage; hence the fulfillment of Primitive P3.

Lastly, it is the (never exposed) central controller that assigns a role to a newly cleansed server. While receiving directions of its new role from the controller, the server is still disconnected from exposed nodes. The process of role assumption is not subject to external influence, and Primitive P4 is fulfilled.

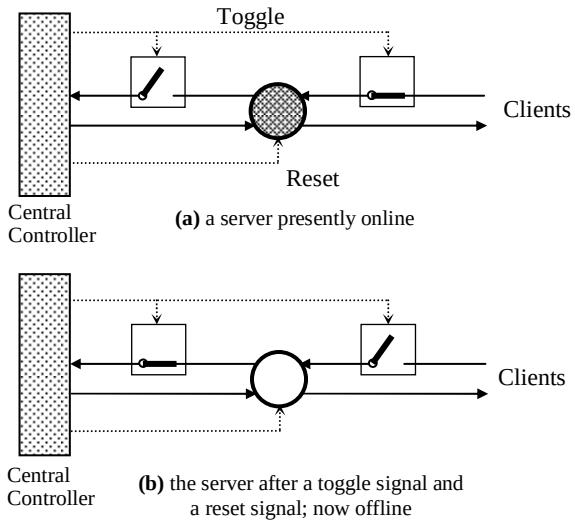


Figure 3. The hardware enhancements in SCIT/HES

V. DNS-HES CLUSTER

In this section, we apply the SCIT/HES frame work to one of the most critical infrastructure services of the Internet, DNS. In our design, a DNS-HES cluster comprises N≥4 identical DNS servers running in the configuration shown in Fig. 4. It is for use in a DNS zone to provide domain names in the zone to the rest of the Internet. A DNS-HES cluster makes available two servers for servicing clients on the Internet (labeled P for Primary and S for Secondary in the figure). It advertises two IP addresses, a primary name server address and a secondary name server address. The generalization to support a tertiary DNS server is omitted; its design and functionality are similar to those of the secondary server.

At any point in time only one of the servers will be operating in one of the following four modes: (1) Mode **P**: Primary DNS, communicating with clients using the primary IP address, (2) Mode **S**: Secondary DNS,

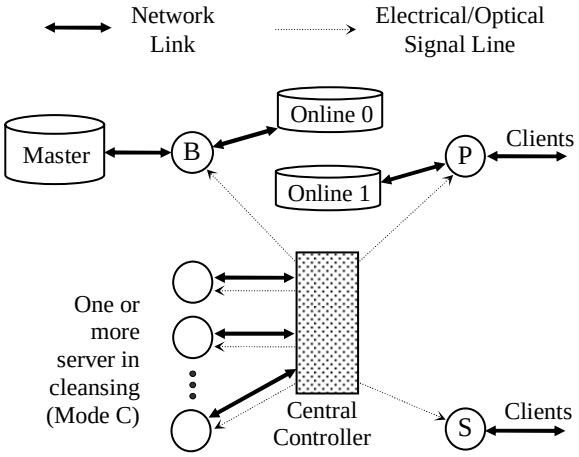
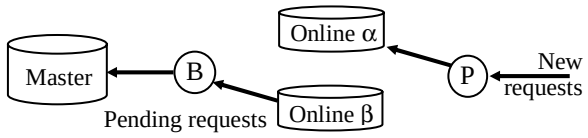


Figure 4. DNS-HES Cluster

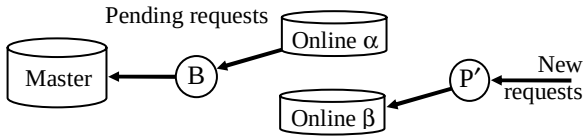
communicating with clients using the secondary IP address, (3) Mode **C**: Offline for self-cleansing with no public IP address, and (4) Mode **B**: Backend server, processing pending dynamic DNS updates in the background with no public IP address. At any time, there is one primary, one secondary and one backend server. The three servers are said to be *on-duty*. The present primary and secondary servers are said to be *online*. The remaining N–3 servers in the cluster will be in cleansing. They are essentially redundant, backup servers. If the number of cleansing server is small (e.g., N–3=1 or 2), the level of redundancy is in line with those of many important DNS sites where each online DNS server has a dedicated backup server. DNS-HES provides the options of using higher degrees of redundancy to improve both security and service availability.

Also shown in Fig. 4 are three backend storages: a Master storage (named after its primary contents, the DNS master file) and two Online storages. The Master storage also stores the DNSSEC private keys. It cannot be accessed by any server that is presently online. An Online storage is used to temporarily store requests for dynamic DNS updates and must be accessible by the online primary server. Backend stores can be implemented as NFS servers or storage devices attached to storage area networks (such as iSCSI hard drives). They are considered part of the intranet, and a network connection is required for a server to access a backend store.

In addition to resolving the IP addresses of domain names, a primary server also accepts the requests of dynamic domain name updates. In DNS-HES the processing of these requests is delegated to the backend server, as illustrated in Fig. 5. In Fig. 5(a), incoming requests are stored in Online storage α, where α is either 0 or 1. We use β to denote "the other" Online storage. After a new primary is rotated online, as in the case of Fig. 5(b), it stores new requests of dynamic updates on Online storage β, whereas the backend server processes pending requests in Online storage α, left by the previous primary server in Fig. 5(a). After a second primary rotation, the configuration returns to Fig. 5(a). We note that the Master storage is inaccessible from the public Internet in either configuration. We also point out that rotations of the backend server do not change the use of online storages. The change depicted in Fig. 5 is triggered by only the rotation of the primary.



(a) New requests stored in Online α; processing pending requests from Online β



(b) After a rotation of the primary, new requests stored in Online β; processing pending requests from Online α

Figure 5. Processing dynamic DNS requests; P′ in (b) emphasizes that a different server is now the primary

To service DNS data to the Internet, a primary or secondary DNS server has to download a copy of the master file from Master storage to their local file systems. In this arrangement, local copies of the master file on online servers are exposed to remote interferences (for instance, a local copy could be tampered with or removed entirely by a successful attacker). The master copy, stored in the Master storage, nevertheless is never exposed. Dynamic DNS updates are performed on the master copy by the backend server, which is not connected to the Internet either. Private keys used to sign dynamic domain name changes are also stored in the Master storage and accessed only by the backend server. The DNS master file and the DNSSEC private keys are in this way shielded from direct cyber attacks.

It can be argued that the backend server and the Master storage are still subject to *indirect* attacks, whereby an attacker sends to the primary server dynamic DNS update requests specifically formatted to trigger vulnerabilities in the software running on the backend server. While this observation is correct, the consequences can easily be handled as follows. First we point out that the secrecy of the DNSSEC private keys are never jeopardized, for a backend server, compromised or not, does not have the communication paths required to send information back to the attacker. The attacker however may attempt to corrupt the DNS master file, residing on the Master storage (for instance, the attacker may attempt to delete the master file entirely). This attack can be easily prevented by storing the master file in obscure locations in the file system. In general, it is impossible to probe the configuration of the backend server from the public Internet due to the lack of communication paths, and therefore attacks against the server must be based on assumed, common-practice configurations. While we acknowledge the weakness of security by obscurity in general applications, it is perfectly suitable to the defenses of the backend server due to the impossibility of probing. In fact, the software running on the backend server can be compiled in a way that produces uncommon memory layouts, preventing attacks from taking over the software in the first place.

In DNS-HES, the setup between the controller and an individual server is depicted in Fig. 6. In the figure (and in Fig. 3 as well), we use solid lines to represent network links for TCP/IP message exchanges (such as Ethernet wires) and dashed lines to represent wires/fibers that conduct electrical/optical control signals. Specifically, the controller uses six control signals to enforce the operation modes of each server in the cluster. Among the six signals, the reset signal to a server forces the server to reboot. The remaining five signals to the server manage the communication configuration of the server. The five signals are discussed below.

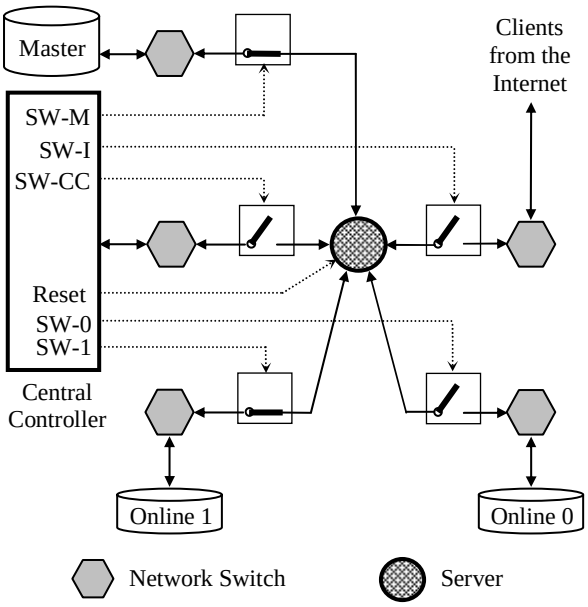


Figure 6. The hardware enhancements in DNS-HES between the central controller and an individual server.

1. The **SW-M** signal controls the connection of the server to the Master storage.
2. The **SW-I** signal controls the connection of the server to the public Internet.
3. The **SW-CC** signal controls the connection of the server to the central controller.
4. The **SW-*i*** signal, where *i* is either 0 or 1 controls the connection of the server to Online storage *i*.

As an example, the controller uses the signals in Fig. 6 to enforce the communication configuration of a cleansing server depicted in Fig. 7, where the server has network paths connecting to only the central controller. As we will see later, one use of the path is for the controller to communicate the role/identity of the server when it is ready to take on a duty. By the same token, the primary name server will be connected to the Internet and one of the Online storage, but disconnected from the rest of the cluster (see Fig. 8).

Implied in Fig. 6 are five local area networks within a DNS-HES cluster (in the figure, each LAN is represented

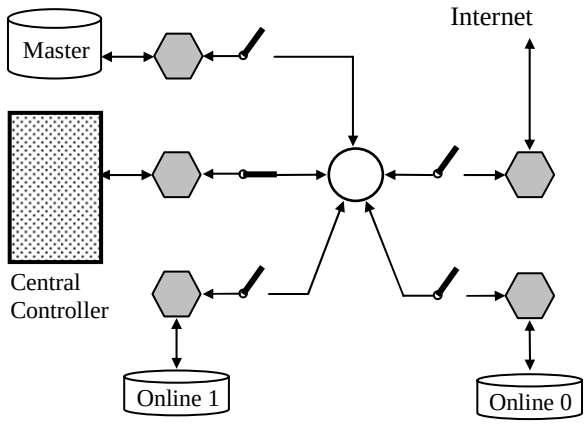


Figure 7. A cleansing server has access to only the central controller

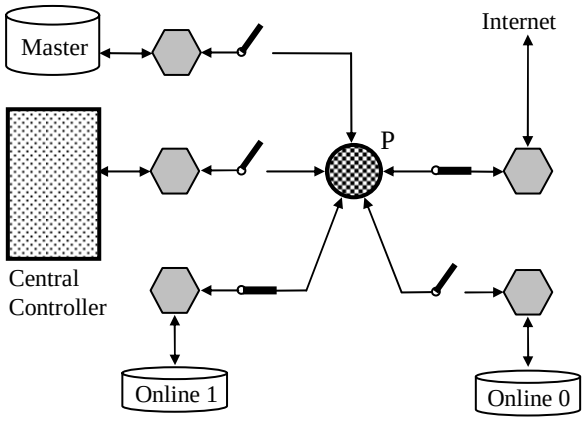


Figure 8. The primary server has access to only the public Internet and one of the Online storage (Online 1 in this example)

by its switch/hub, depicted as a hexagon). For instance, there is a LAN dedicated to the Master storage. Any server disconnected from that LAN has no access to the Master storage. The same applies to the LANs that connect to the central controller, the public Internet, and the two online storages.

VI. DNS Server Rotations

In a DNS-HES cluster, there are three types of server role rotations. A *Primary Swap* brings the present primary DNS server offline for cleansing and rotates a clean server online to be the new primary. A *Secondary Swap* brings the present secondary DNS server offline for cleansing and rotates a clean server online as the new secondary. A *Backend Swap* resets the present backend server for cleansing and designates a clean server to be the new backend server.

Consider a DNS-HES cluster with 4 servers, servers 0, 1, 2, and 3. Assume that the cluster starts with the configuration (P,S,B,C), where server 0 is the primary (P), server 1 the secondary (S), server 2 the backend (B), and server 3 in cleansing (C). If the first rotation is a Primary Swap, then the system enters configuration (C,S,B,P). Further assuming that the next rotation is a Backend Swap, the system subsequently enters configuration (B,S,C,P).

The central controller uses a rotation pattern to determine the sequence of role swaps. For instance the rotation pattern 'PSB' dictates that the system cycles through a Primary Swap, a Secondary Swap, and a Backend Swap. As a second example, the rotation pattern 'PSPB' dictates that the system cycles through a Primary Swap, a Secondary Swap, a Primary Swap, and a Backend Swap. For reasons to be explained later, PSPB is chosen as the default rotation pattern.

We are now ready to present the routines invoked by the central controller to carry out the three types of role swaps. Due to its simplicity, we start the discussion with the Secondary Swap routine, shown in Fig. 9. In its first 3 steps, the Secondary Swap routine disconnects the present secondary DNS server from the Internet, resets the server, and establishes the network path from the server to the central controller. The server is now in

```
Routine Secondary-Swap (c)
Parameters:
    c: ID of a cleansed server designated to
    become the new secondary server
Global Variables:
    S: ID of the present secondary DNS server
Steps:
  // Bring server S offline for cleansing
 1. Signal "off" to SW-I[S];
 2. Signal Reset[S];
 3. Signal "on" to SW-CC[S];
    // Bring server c online as the new secondary
 4. Signal "on" to SW-M[c];
 5. Send message "Role Secondary" to server c;
 6. Wait for c to download the master file;
 7. Signal "off" to SW-M[c];
 8. Signal "off" to SW-CC[c];
 9. Signal "on" to SW-I[c];
10. Set S to c;
```

Figure 9. The Secondary-Swap routine, which rotates a new secondary DNS server online

```
Routine Backend-Swap (c)
Parameters:
    c: ID of a clean server designated to
      become the new backend server
Global Variables:
    B: ID of the present backend server
    α in [0,1]: the offline storage used by the
      present backend server B.
Steps:
   // Bring server B offline for cleansing
 1. Signal "off" to SW-M[B];
 2. Signal "off" to SW-α[B];
 3. Signal Reset[B];
 4. Signal "on" to SW-CC[P];
    // Set up server c as the new backend
 5. Send message "Role Backend" to c.
 6. Signal "off" to SW-CC[c];
 7. Signal "on" to SW-M[c];
 8. Signal "on" to SW-α[c];
 9. Set B to c;
```

Figure 10. The Backend-Swap routine

cleansing and will be able to inform the controller of the completion of its self cleansing. Next the routine brings the designated clean server *c* online as the new secondary server. To perform its new role, server *c* needs an up-to-date copy of the zone's master file. (This is strictly speaking unnecessary, for a secondary name server must periodically synchronize its local DNS data with the primary server. Having the data readily available the moment it goes online improves performance however.)

In Step 4 of Fig. 9, the controller establishes the network path for server *c* to reach Master storage. In Steps 5 and 6, it sends a message to server c to convey its new role and waits for *c* to complete downloading the master file. Steps 7 and 8 disconnect the network paths from server *c* to the Master storage and the controller. The network path for server *c* to connect to the public Internet is connected in Step 9, and server *c* formally becomes the secondary DNS server in Step 10.

Shown in Fig. 10 is the Backend Swap routine, which the central controller invokes to designate a clean server *c* as the new backend server. Recall that a backend server processes the pending requests of dynamic DNS updates from one of the online storages and updates the master file stored in Master storage accordingly. In the routine, the present backend is assumed using Online storage α, where α is either 0 or 1. In Steps 1 and 2, the central controller disconnects the present backbend's access to Master storage and Online storage α. In Steps 3 and 4, the controller resets the backend server and reestablishes the connection between the server and the controller. The backend server is now in cleansing. To set up server *c* as the new backend, the controller sends a role message in Step 5, and subsequently in Step 6 cuts its network connection to server *c* (implied in Step 5 are the exchanges of acknowledgements for reliability before the disconnection in Step 6). In Steps 7 and 8, the controller establishes for server *c* the access to Master storage and Online storage α. Server *c* is declared the backend server in Step 9.

We now examine the Primary Swap routine, shown in Fig. 11. The central controller invokes the routine in order to rotate a clean server *c* online to replace the present primary server. Recall from Fig. 5 that bringing a new primary name server online causes changes in the uses of online storages. Instead of using the same online storage to store incoming requests of DNS updates as the previous primary, the new primary switches to "the other" online storage. In the routine, the online storage used by the present (and soon to be replaced) primary server is denoted Online storage α and the other one is denoted Online storage β. In Steps 1 and 2, the controller disconnects the present primary from the Internet and cuts its access to Online storage α. The server is reset to enter the cleansing mode in Step 3, and its network path to the controller is reestablished in Step 4.

Starting from Step 5, the Primary Swap routine brings the clean server *c* online as the new primary. In Step 5, the controller establishes the network path for server *c* to reach Master storage. In Steps 6 and 7, it conveys to server *c* its new role and waits for *c* to download the master file. Subsequently in Steps 8 and 9, server *c* is disconnected from the Master storage and the central controller, respectively. Server *c* is connected to Online storage β in Step 10 and to the public Internet in Step 11. It is declared the primary name server in Step 12 and will store incoming requests of dynamic DNS updates in Online storage β.

Again recall from Fig. 5 that a Primary Swap affects the networking configuration of the backend server. As seen in Fig. 5, the backend server processes pending dynamic DNS updates from Online storage β before the

Routine Primary-Swap (*c*)
Parameters:
c: ID of a clean server; to become the primary
Global Variables:
P: ID of the present primary
B: ID of the present backup server
α in [0,1]: the online storage that has been used by the present primary P.
Local Variable:
β in [0,1] : the inverse of α (that is, the other Online storage)
Steps:
// Bring server P offline for cleansing
1. Signal "off" to SW-I[P];
2. Signal "off" to SW-α[P];
3. Signal Reset[P];
4. Signal "on" to SW-CC[P];
// Bring server *c* online as the new primary
5. Signal "on" to SW-M[*c*];
6. Send message "Role Primary" to server *c*;
7. Wait for *c* to download the master file;
8. Signal "off" to SW-M[*c*];
9. Signal "off" to SW-CC[*c*];
10. Signal "on" to SW-β[*c*];
11. Signal "on" to SW-I[*c*];
12. Set P to *c*;
// Switch backend B from Online storage β to α
13. Signal "off" to SW-β[P];
14. Signal "on" to SW-α[P];

Figure 11. The Primary-Swap routine

swap and must change to Online storage α after the swap. Such are the effects of Steps 13 and 14 in Fig. 11.

We present in Fig. 12 the Central Control routine executed by the central controller. The routine initially assigns server 0 as the primary server, server 1 as the secondary, and server 2 as the backend. Other servers in the cluster are in the clean mode. The online storage used by the initial primary is Online storage 0. The steps whereby servers enter their respective initial roles are straightforward and have been omitted in the presentation. The routine then enters an infinite loop where it waits for the completion of cleansing of any off-duty server. Once a ready clean server is found, the routine determines the type of the next role swap according to a pre-configured rotation pattern and subsequently invokes the corresponding role-swapping routine. We note that the Central Control routine does not depends on a fixed number of servers in the cluster. The routine to the contrary allows new servers to be added to the cluster or faulty servers to be removed from the cluster without reconfigurations. The implications of this feature are discussed in Section VII.

To conclude this section, we point out that in all of the above routines, the central controller exchanges TCP/IP messages with a server only after the controller has rebooted the server and has disconnected the server's connections to the public Internet and the rest of the

Routine Central-Control ()
Global Variables:
P: ID of the present primary DNS server
S: ID of the present secondary DNS server
B: ID of the present backend server
α in [0,1]: the offline storage used by the present primary DNS server.
Initializations:
P=0, S=1, B=2;
α=0;
Loop the following steps forever:
Wait for "any" cleansing server to complete cleansing. Call the server *c*.
Determine the type of the next server switch according to the rotation pattern.
For a Primary Swap:
Call Primary-Swap(*c*).
Invert α.
For a Secondary Swap:
Call Secondary-Swap(*c*).
For a Backend Swap
Call Backend-Swap (*c*).

Figure 12. The Central-Control routine

cluster. The controller in this way cannot be reached from the online servers or the public Internet, and consequently server rotations cannot be subverted by remote attacks.

VII. Redundancy, Availability and Security

Although SCIT/HES is not specifically designed for fault tolerance, it does handle some failures gracefully due to the use of hardware redundancy. Let us examine what happens when server failures occur. There are two possibilities. (1) If a server failure is caused by intrusion events or software errors, then the server will eventually be reset and cleansed. In this way, SCIT/HES succeeds in handling "soft" failures. (2) In the case of hardware failures, the sever will be reset by the controller at some point but cannot bootstrap the operating system or complete the cleansing procedure due to hardware dysfunctions. The consequence is that the server will not report to the central controller "cleansing completed" and thus will not be available for service.

Consider the case of DNS-HES. With N≥4 servers, the cluster continues to provide DNS services in face of N–3 "hard" failures. In the worst case of N–3 failures, server rotations stop, for the controller cannot find ready clean servers. The DNS service however is still provided by the remaining, on-duty servers, although dynamic DNS updates will not be reflected until rotations resume. In this way, the availability of the system increases with the degree of server/hardware redundancy. This aspect of SCIT/HES is similar to many fault tolerance designs.

With SCIT/HES, however, increasing the degree of redundancy also improves security. To illustrate this, assume that the self-cleansing procedure takes 10 minutes to complete (our previous SCIT prototypes indicate that this is a conservative assumption). With one cleansing

server, the controller waits for 10 minutes for the server to complete cleansing, which equates to a server swap every 10 minutes (see Fig. 12). With two spare servers, the controller is expected to find a clean machine in 5 minutes, doubling the rate of server rotations and reducing online servers' exposure to the Internet to half. In the case of successful breaches on the online servers, the window of a breach is also reduced by half. Even shorter rotation times and even stronger security can be achieved by introducing more servers to the cluster.

An interesting way to contrast SCIT with high-availability computing is this: In high-availability systems, hardware redundancy exists in the form of backup servers [16,17]. By its nature, a backup is idle most of the times. Its computing power is wasted unless the online server fails. With SCIT, redundancy exists in the form of servers in cleansing. This design in effect puts spare computing power to good use, such as self cleansing, system auditing, and intrusion recovery, for the sake of strengthening system security.

Due to its prominent role in DNS, the primary name server is generally considered most critical in DNS security. Thus it is desirable to subject the primary server to more frequent rotations; hence the selection of "PSPB" as the default rotation pattern in DNS-HES.

VIII. CONCLUSION

We have shown that with simple hardware enhancements strategically placed in a server cluster, it is possible to build intrusion tolerance mechanisms that cannot be corrupted. We have presented a SCIT/HES DNS cluster as an application of this framework. Precisely, we have shown that the processes of server rotation and cleansing are never "exposed" by physical isolation and thus not subject to remote attacks. In the case of DNS-HES, moreover, the same kind of protection through isolation also extends to critical data, including the DNS zone master file and DNSSEC private keys. While successful and undetected intrusions cannot be ruled out (probably never will be), intrusion tolerance mechanisms of SCIT/HES works to guarantee a baseline integrity and the continuum of services. It is our belief that incorruptible intrusion tolerance solutions such as the one presented here constitute a new, effective layer of defense for critical information systems against undetected and unknown attacks, the unknown unknowns in computer system security.

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 in Journal of High Speed Networking, Vol. 15, No. 1, 2006)

[5] Yih Huang, David Arsenault, and Arun Sood, "Securing DNS Services through System Self Cleansing and Hardware Enhancements," 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," 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] P. Mockapetris, "Domain names — Concepts and Facilities," Internet RFC 1034, November 1987.

[9] P. Vixie (editor), S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System," Internet RFC 2136, April 1997.

[10] D. Eastlake. "Domain Name System Security Extensions," Internet RFC 2535, March 1999.

[11] P. Vixie, "DNS and BIND security issues," in Proc. of the 5th Usenix Security Symposium. Salt Lake City, UT, 1995.

[12] Bellovin, S. M. Using domain name system for system break-ins. In Proceedings of the 5th Usenix UNIX Security Symposium. Salt Lake City, UT, 1995.

[13] See DNS and BIND related advisories and incident notes published by CERT Coordination Center at http://www.cert.org.

[14] The Twenty Most Critical Internet Security Vulnerabilities, available at http://www.sans.org/top20.

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

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

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

Yih Huang received the B.S. degree and M.S. degree in Computer Science and Information Engineering from Feng-Chia University, Taiwan (Republic of China), in 1985 and 1987 respectively. He received the Ph.D. degree in Computer Science from Michigan State University in 1998.

He is the Research Professor of Computer Science in the Department of Computer Science at George Mason University, Fairfax, VA. His research has been supported by DARPA, National Institute of Standards and Technology and private industry. His research interests include computer networking protocols, computer/network security, parallel and distributed processing, and micro-processor architectures.

Arun K. Sood received the B.Tech degree from the Indian Institute of Technology (IIT), Delhi, in 1966, and the M.S. and Ph.D. degrees in Electrical Engineering from Carnegie Mellon University, Pittsburgh, PA, in 1967 and 1971, respectively.

He is the Professor of Computer Science in the Department of Computer Science at George Mason University, Fairfax, VA, and the Director of the Laboratory for Interdisciplinary Computer Science. He has held academic positions at Wayne State University, Detroit, MI, Louisiana State University, Baton Rouge, and IIT, Delhi. His research has been supported by the Office of Naval Research, National Imagery and Mapping Agency, National Science Foundation, U.S. Army Belvoir RD&E Center, U. S. Army TACOM, U.S. Department of Transportation, and private industry. He was awarded grants by NATO to organize and direct advance study institutes in relational database machine architecture and active perception and robot vision. His research interests are in computer security, image and multimedia computing, signal processing, parallel and distributed processing, performance modeling and evaluation, simulation and modeling, and optimization.