

Securing DNS Services through System Self Cleansing and Hardware Enhancements

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 -- Domain Name Systems (DNS) provide the mapping between easily-remembered host names and their IP addresses. Popular DNS implementations however contain vulnerabilities that are exploited by frequent, targeted attacks. The software vulnerabilities of DNS together with the constant innovation and morphing of cyber attack techniques necessitate the consideration of the worst case scenarios: there will be successful but undetected attacks against DNS servers.

In this work¹ we develop a secure DNS architecture that contains the damage of successful, undetected attacks. This formidable end is achieved by constantly cleansing the servers and rotating the role of individual servers. Moreover, the server rotation process itself is protected against corruption by hardware. We will show the advantages of our design in the following areas: (1) protection of the DNS master file and cryptographic keys, (2) incorruptible intrusion tolerance, (3) high availability, and (4) scalability, the support of using of high degrees of hardware/server redundancy to improve both system security and service dependability. Due to the critical importance of DNS, such a dependable and intrusion-resilient design contributes significantly to the overall security of the Internet.

I. INTRODUCTION

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 [1]. 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 [2]. Because of the critical importance of DNS, many sites deploy dedicated backup servers which take over DNS services in face of the failures of online servers.

DNS was later enhanced with DNS Security Extensions (DNSSEC) to provide data origin authentication [3]. 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 [4,5,6,7]. 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 [7]. The situation is exacerbated by the constant innovation and evolution of network attack techniques. This trend and the critical importance of DNS lend consequence to the concept of *intrusion tolerance*: a highly dependable DNS system must fend off or at least limit the losses caused by successful undetected attacks, providing uninterrupted services in face of such attacks.

In prior work [8,9,10], we have developed the framework of Self-Cleansing Intrusion Tolerance, or SCIT, which constantly rotates and cleanses online servers in a server cluster, regardless of whether an intrusion is *detected or not*. In particular, we presented in [10] our earlier designs and prototyping experiences of SCIT-enabled DNS servers. In [11], we presented SCIT/HES (HES for Hardware Enhanced Security), which uses simple hardware enhancements to provide incorruptibility guarantees, or *SCIT Primitives*.

The primary contribution of this paper is the application of the SCIT/HES framework to DNS. Specifically, we present a second generation of SCIT DNS architecture, named DNS-HES (DNS with

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

Hardware Enhanced Security), that improves DNS security in the following areas.

- Protection of the zone master file and private keys — 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/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.
- High availability — the graceful handling of server failures.
- 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, DNS-HES *will* continue to contain their effects by server rotation and cleansing.
- Scalability — the support of using of high degrees of hardware/server redundancy to improve both system security and dependability.

We point out that the first two advantages above are also supported by our previous SCIT DNS design [10]. The additional properties of incorruptibility and scalability are derived from the new design presented in this paper.

The remainder of the paper is organized as follows. The concept of SCIT/HES is reviewed in Section II. We present in Section III the architecture of the DNS-HES and in Section IV the server rotation algorithm of DNS-HES. In Section V, we discuss the ramifications of DNS-HES on hardware redundancy, service availability, and system security. We give related work in Section VI and conclusion in Section VII.

II. SCIT/HES OVERVIEW

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. A central controller is used to manage server rotations and role assignments [11]. (In implementations, the controller could also be an off-the-shelf server; the name suggests its use not its construction.) A high-level view of a SCIT/HES server cluster is depicted in Figure 1.

When a server moves from the online mode to the cleansing mode, it reloads the operating system, service software and data from trusted media, followed by system auditing and recovery procedures. The trusted media can be any non-writable media or writable media combined with digital signatures to

verify the integrity of data. The reason for periodic cleansing is to force an online server to return to a known, clean state, whether an intrusion is detected or not. The underlying assumption is that there will be attacks that are sophisticated and stealthy enough to penetrate even the best security measures and evade the most advanced intrusion detection systems.

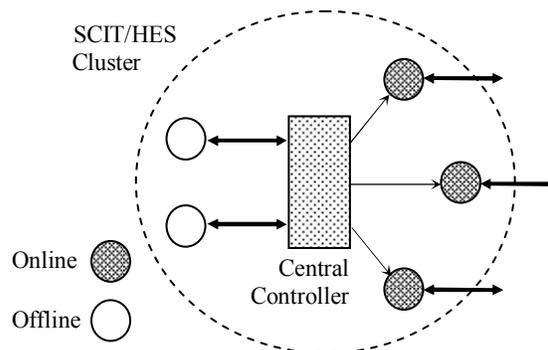


Fig. 1: A High-level View of SCIT/HES

In SCIT/HES, the central controller also maintains the cluster communication configuration shown in Figure 2, 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. Online servers of course must have two-way communications with clients outside the cluster. We emphasize that the arrows in Figure 2 represent permissible communications; they do not mandate dedicated communication channels.

Due to server rotations, the configuration shown in Figure 2 is dynamic. Communication paths must be cut and reconnected when servers switch between different modes. In SCIT/HES, the central controller also manages the connection and disconnection of communication paths. Simple on/off switches are introduced to achieve this end. The uses of these devices in the DNS-HES cluster will be discussed in the next section.

Our previous SCIT-DNS design is based on a distributed control scheme using software that runs on each machine [10]. Even with the best security designs and programming practices, software-based solutions are inevitably subject to ingenious attacks (and always will be). In the next section, we present a design that employs simple on/off switches to isolate SCIT operations from external influence. We will argue that the resultant DNS-HES architecture achieves the *incorruptibility* of intrusion tolerance.

III. DNS-HES CLUSTER

A DNS-HES cluster comprises $N \geq 4$ identical DNS servers running in the configuration shown in Figure 3. 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, 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. Furthermore, DNS-HES provides the options of using higher degrees of redundancy to improve both security and service availability.

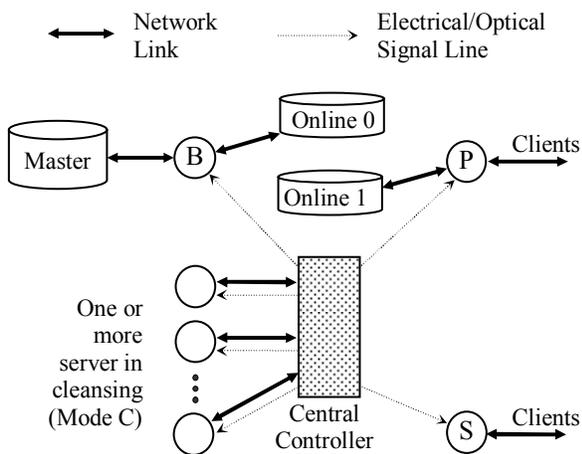


Fig. 3: DNS-HES Cluster

Also shown in Figure 3 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 Figure 4. In Figure 4(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 Figure 4(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 Figure 4(a). After a second primary rotation, the configuration returns to Figure 4(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 Figure 4 is triggered by only the rotation of the primary.

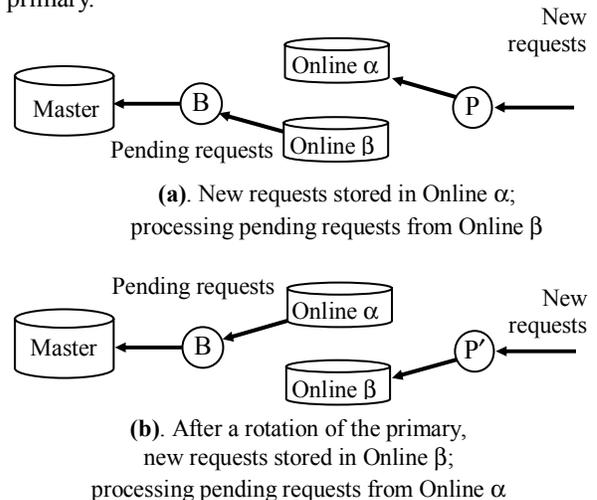


Fig. 4: Processing dynamic DNS requests; P' in (b) emphasizes that a different server is now the primary

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 SCIT/HES, the central controller also manages the communication paths of servers. The setup between the controller and an individual server is depicted in Figure 5. In the figure (and in Figure 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.

1. The **SW-M** signal controls the connection of the server to the Master storage.

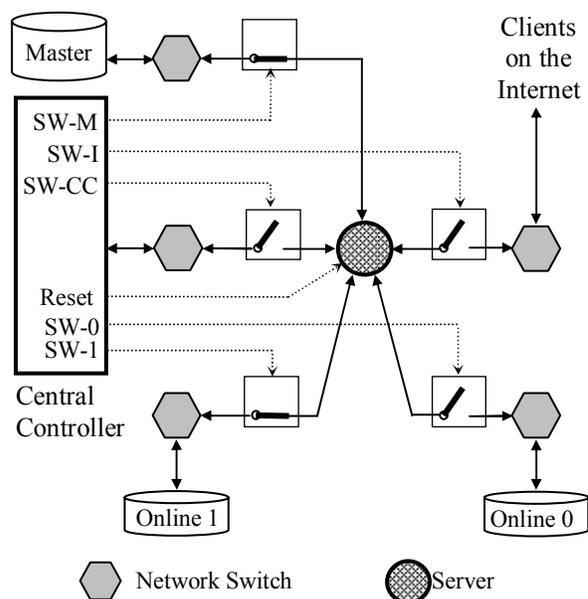


Fig. 5: The hardware enhancements in DNS-HES between the central controller and an individual server.

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 Figure 5 to enforce the communication configuration of a cleansing server depicted in Figure 6, 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 secondary name server will be connected to the Internet and disconnected from the rest of the cluster.

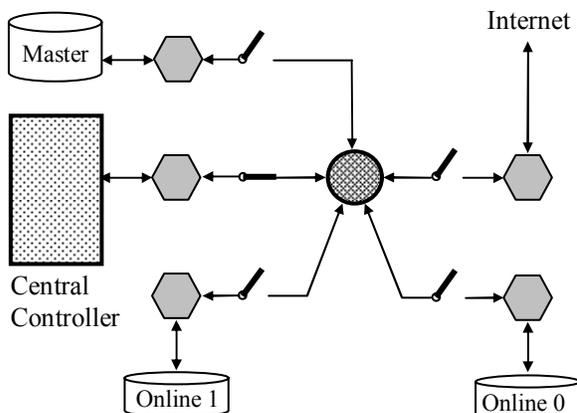


Fig. 6: a cleansing server has access to only the central controller

Implied in Figure 5 are five local area networks within a DNS-HES cluster (in the figure, each LAN is represented 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.

IV. SERVER ROLE 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 Figure 7. In its first 3 steps, the Secondary Swap routine disconnects the present cleansing server from the Internet, resets the server, and establishes the network path from the server to the central controller. The server is now in 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 DNS server. To perform its new role, server *c* needs an up-to-date copy of the zone’s master file. In Step 4, 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.

```

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;

```

Fig. 7: The Secondary-Swap routine, which rotates a new secondary DNS server online

Shown in Figure 8 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 the 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 backend’s access to the Master storage and Online storage α . In Steps 3 and 4, the controller resets the present backend 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*. 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 Figure 9. The central controller invokes the routine in order to rotate a clean server *c* online to replace the present primary server. Recall from Figure 4 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.

```

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
   $\alpha$  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- $\alpha$ [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- $\alpha$ [c];
  9. Set B to c;

```

Fig. 8: The Backend-Swap routine

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 the 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 Figure 4 that a Primary Swap affects the networking configuration of the backend server. As seen in Figure 4, the backend server processes pending dynamic DNS updates from Online storage β before the swap and must change to Online

storage α after the swap. Such are the effects of Steps 13 and 14 in Figure 9.

```

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
   $\alpha$  in [0,1]: the online storage that has been
      used by the present primary P.
Local Variable:
   $\beta$  in [0,1] : the inverse of  $\alpha$  (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- $\alpha$ [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- $\beta$ [c];
  11. Signal “on” to SW-I[c];
  12. Set P to c;
  // Switch backend B from Online storage  $\beta$  to  $\alpha$ 
  13. Signal “off” to SW- $\beta$ [P];
  14. Signal “on” to SW- $\alpha$ [P];

```

Fig. 9: The Primary-Swap routine

We present in Figure 10 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. The online storage used by the initial primary is Online storage 0. The steps whereby servers enter their respective initial roles 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 depend 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 V.

```
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
   $\alpha$  in [0,1]: the offline storage used by the
    present primary DNS server.
Initializations:
  P=0, S=1, B=2;
   $\alpha=0$ ;
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  $\alpha$ .
  For a Secondary Swap:
    Call Secondary-Swap(c).
  For a Backend Swap
    Call Backend-Swap (c).
```

Fig. 10: The Central-Control routine

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 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.

V. REDUNDANCY, AVAILABILITY AND SECURITY

Although DNS-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. If a server failure is caused by intrusion events or software errors, then the server will eventually be reset and cleansed. In this way, DNS-HES succeeds in handling "soft" failures.

In the case of hardware failures, the sever will be reset by the controller at some point but most likely 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. With $N \geq 4$ 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. As one can see, the availability of the system increases with the degree of server/hardware redundancy. This aspect of DNS-HES is similar to many fault tolerance designs.

With DNS-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 Figure 10). 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 DNS-HES with high-availability computing is this: In high-availability systems, hardware redundancy exists in the form of backup servers [12,13]. 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 powers 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 security. Thus it is desirable to subject the primary server to more frequent rotations. Assume again that the central controller finds a ready clean server every 10 minutes in a four-server DNS-HES cluster. The 'PSB' rotation pattern results in the present primary to be replaced within 30 minutes. The time is reduced to 20 minutes by the rotation pattern "PSPB;" hence its being chosen as the default. The time can be further reduced with more spare servers.

VI. 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 [14].

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 [12,13,15,16]. 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.

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 [17,18,19,20] for use by server clusters.

VII. CONCLUSION

We have presented a DNS cluster design that constantly rotates server roles and performs system cleansing in order to contain undetected intrusion. Our DNS-HES design uses simple hardware enhancements to physically isolate servers in cleansing from remote attacks. 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 DNS-HES work to guarantee a baseline integrity and the continuum of DNS services. Because of the critical importance of DNS and its well-known vulnerabilities to attacks, a dependable and intrusion-resilient DNS design such as DNS-HES contributes significantly to the overall security of the Internet.

REFERENCES

- [1] P. Mockapetris, “Domain names — Concepts and Facilities,” Internet RFC 1034, November 1987.
- [2] P. Vixie (editor), S. Thomson, Y. Rekhter, and J. Bound, “Dynamic Updates in the Domain Name System,” Internet RFC 2136, April 1997.
- [3] D. Eastlake, “Domain Name System Security Extensions,” Internet RFC 2535, March 1999.
- [4] P. Vixie, “DNS and BIND security issues,” in *Proc. of the 5th Usenix Security Symposium*. Salt Lake City, UT, 1995.
- [5] 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.
- [6] See DNS and BIND related advisories and incident notes published by CERT Coordination Center at <http://www.cert.org>.
- [7] The Twenty Most Critical Internet Security Vulnerabilities, available at <http://www.sans.org/top20>.
- [8] 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.
- [9] Yih Huang, Arun Sood, and Ravi K. Bhaskar, “Countering Web Defacing Attacks with System Self-Cleansing,” *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics*, pp. 12—16, Orlando, Florida, July 2003.
- [10] 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. (Also to appear in *Journal of High Speed Networking*)
- [11] Yih Huang, David Arsenault, and Arun Sood, “Incorruptible System Self Cleansing for Intrusion Tolerance,” accepted to appear in *Workshop on Information Assurance*, 2006.
- [12] Peter S. Weygant, *Clusters for High Availability*, Prentice Hall, 1996.
- [13] High-Availability Linux Project. www.linux-ha.org.
- [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] Steve Blackmon and John Nguyen, “High-Availability File Server with Heartbeat,” *System Admin, the Journal for UNIX Systems Administrators*, vol. 10, no. 9, September 2001.
- [16] R. Rabbat, T. McNeal and T. Burke, “A High-Availability Clustering Architecture with Data Integrity Guarantees,” *Proc. of IEEE International Conference on Cluster Computing*, 178–182, (Newport Beach, CA) Oct., 2001.
- [17] 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.
- [18] William Yurcik and David Doss, “Achieving Fault-Tolerant Software with Rejuvenation and Reconfiguration,” *IEEE Software*, July/August 2001, pp. 48-52.
- [19] 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.
- [20] Khin Mi Mi Aung, Kiejin Park, and Jong Sou Park, “A Rejuvenation Methodology of Cluster Recovery,” *Cluster-Sec*, 2005.