SCIT-DNS: Critical Infrastructure Protection through Secure DNS Server Dynamic Updates

Yih Huang, David Arsenault, and Arun Sood

Abstract—Domain Name Systems (DNS) provide the mapping between easily remembered host names and their IP addresses. While domain name information is typically created and updated off-line, dynamic DNS updates allow clients to manage domain names online, in real time. The current secure DNS standards (DNSSEC) require private keys to be kept online to sign dynamic updates, leaving private keys subject to network-based attacks.

In this work[*], we develop a secure *implementation* framework of DNS servers that voids the above requirement. Our approach, called Self-Cleansing Intrusion Tolerance (SCIT), strengthens DNSSEC through hardware redundancy. Our system uses a highly integrated cluster of DNS servers that constantly rotates the role of individual servers, handles one-server failures gracefully, confines the damages of successful intrusion to a limited time, and digitally signs dynamic updates by a c*lean* box (to be defined layer) using the DNS zone key while keeping the key offline at all times. It is our belief that the availability and integrity of critical communications infrastructure, such as DNS, far outweigh the costs of hardware redundancy.

In this paper, we present (1) the architecture of SCIT DNS clusters that achieves the above goals, (2) a secure Cluster Coordination Protocol (CCP) that servers in the cluster use to coordinate role changes without ever opening a port, and (3) the designs of our ongoing SCIT DNS cluster prototype. Preliminary experiences of our prototype show that role rotation and self-cleansing cycles are in the range of minutes, restricting the damages of even undetected but successful attacks to short time windows.

Index Terms—computer security, fault-tolerance,, information assurance, intrusion containment, self-cleansing systems

I. Introduction

As the world becomes ever more dependant on network-based services for critical applications, particularly via TCP/IP networks, it is vital to protect DNS from attacks and failures. Unfortunately, popular DNS implementations have been known to be the target of numerous vulnerabilities, exploits and attacks, as reported in [1, 15] and by the Cert Coordination Center (www.cert.org). To achieve the protection level desired for critical services such as DNS, we apply an enhanced version of the Self-Cleansing Intrusion Tolerance (SCIT) architecture [6, 7]. We believe that the SCIT approach can improve the security of a large class of servers that form part of the critical computing and communication infrastructure. In [6, 7] we present approaches for SCIT-based firewalls and web servers.

The Domain Name System (DNS) is essentially a distributed database [11, 12]. Each name server maintains the domain name information regarding a subspace, or a *zone*, in the DNS name space. Several predefined properties, or *resource records* (RR*)*, can be associated with a domain name. The most important is the “type A” RR, which contains the IP address of the domain. All the RRs pertaining to the domain names in 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 server, which periodically synchronizes its local DNS file with the master file. Secondary name servers respond to DNS queries but are not involved in maintaining the master file.

The DNS architecture was later enhanced with *DNS Security Extensions* (DNSSEC) to provide data origin authentication. With DNSSEC, each zone is equipped with (at least) a pair of public and private keys. The public key is configured into every client in the zone through a safe channel (e.g., manually by administrators). The private key is used to digitally sign RRs. The response to a DNS query includes the requested RRs and a “Sig RR,” the digital signature of the requested RRs. DNS clients verify the integrity and origin of received DNS data by checking accompanying signatures using the zone public key.

The integrity of the above approach depends on the secrecy of the participating private keys, which is best maintained by keeping the private keys off-line. In this way, even if a server is compromised, the hacker cannot procure the private key and temper with DNS data. (The hacker however could still perform other mischief, such as deleting the master file as a denial of service attack or using the server as a jump pad to machines inside the intranet.) It follows that signature computations must also be off-line. This perfectly suits those domain names created and managed manually via administrative procedures (one

This research is part of the Critical Infrastructure Protection Project funded by the National Institute of Standards and Technology (NIST). The authors are with the Department of Computer Science and the Center for Image Analysis, George Mason University, Fairfax, VA. E-mail: {huangyih, darsenau, asood}@cs.gmu.edu.

registers for a new domain name, fill out a form, pay a fee, etc.). Offline signature computation however is incompatible with dynamic domain name updates, where RRs are updated in real time upon online requests from clients [13, 14, 16].

In the DNSSEC standards, signatures of dynamically updated RR are computed in one of the following two modes. In Mode A, a per-server private key is used to sign dynamic updates. The corresponding public key is stored in a Key RR associated with the domain name of the server and obtained by client through DNS requests. In this way, server compromises jeopardize only dynamically updated resource records. However, server keys are considered not as authoritative as zone keys [14]. In Mode B, the zone private key is kept online and used to sign dynamic updates. This leaves the key subject to exposure through network attacks. In the face of server compromises, the integrity of the entire master file is in question.

In this paper, we present a DNS implementation framework that eliminates the above concerns. Our approach, called Self-Cleansing Intrusion Tolerance (SCIT), uses a cluster of servers and constantly rotates the role of each server. At any point in time, a particular server may be the primary server, the secondary server, or in the process of rebooting and self-cleansing. We will show the advantages of our design in three areas: (1) offline zone private keys to sign dynamic updates, (2) high availability, the graceful handling of one server failures, and (3) intrusion tolerance, the damage confinement and automatic system recovery in the face of breaches.

The remainder of the paper is organized as follows. The concept of SCIT is presented in Section II, and the architecture of our SCIT-DNS system presented in Section III. In Section IV, we present the SCIT Cluster Coordination Protocol (SCIT-CCP), which servers use for role rotations. We detail our prototype design in Section V and conclude this work in Sections VI.

II. SCIT Overview

SCIT uses periodic restarting of the system which 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. During the cleansing process a backup server is brought online to provide uninterrupted services. In many aspects, SCIT is close to high-availability computing, which uses backup systems to ensure continued customer services in face of primary server failures [18, 4, 2, 10]. With SCIT, null failure events are introduced to force periodic takeover by the backup server and the cleansing of the primary server.

The reason for periodic cleansing is to force an online server to return to a known, clean state, regardless the detection of, *or lack thereof,* intrusion. The underlying assumption is that there always are cyber attacks that are sophisticated and stealthy enough to penetrate even the best security measures and be beneath the radar of the most advanced intrusion detection systems. While this "paranoid" attitude is obviously overkill for an average Joe's PC, it is perfectly appropriate for critical infrastructures, such as DNS. Just like a software tester's job is to assume the existence of bugs and contrive to uncover them, SCIT assumes undetected security breaches in online systems and takes actions accordingly.

In previous work, we have built a SCIT firewall prototype that constantly alternates between two identically configured firewall servers and a SCIT web server that alternate between two servers [6, 7]. In these prototypes, cleansing cycles are typically less than 10 minutes. While occasional packet losses do occur during handovers, they cause little, if any, perceivable service disruption owing to the retransmission mechanisms of TCP/IP. We notice that DNS provides retransmissions even when UDP is used.

III. SCIT-DNS Cluster

A *SCIT cluster* is a set of server boxes connected through one or more local area networks, working cooperatively and/or alternately to perform a set of predefined services. Our DNS SCIT cluster consists of three identical DNS servers running in a DMZ configuration behind a firewall, as shown in Figure 1.

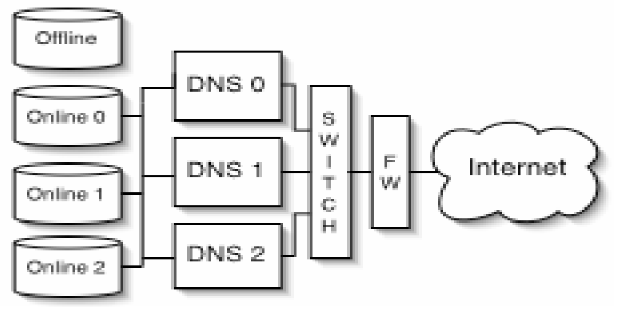

Figure 1: SCIT-DNS Cluster

The DNS zone running the cluster advertises *two* IP addresses of DNS servers, a primary name server address and a secondary name server address. It also advertises two domain names for the primary and secondary servers, for instance ns1.demo.com and ns2.demo.com. At the time of this writing, we do not support the optional tertiary DNS service. Generalizations to accommodate more than three servers are part of our ongoing work.

Each of the three DNS servers in the cluster cycles through three operating states: (1) Primary DNS, (2) Secondary DNS, and (3) Offline for cleansing and integrity checking. At any point in time only one of the DNS servers will be operating in any one mode (see Figure 2). When a server enters an operation mode, it assumes the identities required by that mode. For instance, when a newly cleansed server becomes the primary, it claims the IP address and domain name of the primary name server. (An IP address can be dynamically claimed by a machine with the VRRP protocol or Gratuitous ARP messages.) DNS clients use either of the two server identities when communicating with cluster. The server that assumes the identity specified in a query at time of receiving the query answers that query. It is worth emphasizing that the bindings between DNS server domain names and respective IP addresses do not change. What will change is the machine that assumes each identity.

Also shown in Figure 1 are backend storages, including an offline storage and three online storages. The *Offline Storage* cannot be accessed by any server that is presently serving DNS clients in the public network. It is used to store the zone private key and the master file. An *Online Storage* is accessible by all servers, including those that are online. It is used to temporarily store requests for dynamic DNS updates. Backend storages can be implemented as NFS servers or storage devices attached to storage area networks. They are considered part of the intranet. To access data in backend storages, a server must first *connect* to the storage and later *disconnect*. In implementation terms, connections can be, for example, SSH connections or NFS mounts. We use SSH in our prototype design.

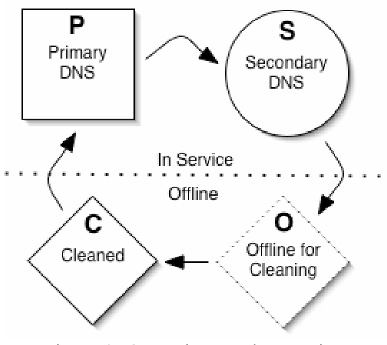

Figure 2: Operating Mode Rotations

To illustrate the interactions among the three servers and backend storages, we will go through a complete cycle of SCIT operation, using arrows to show the communications paths in each stage of operation. In Figure 3(a), Server 1 is the current primary server (labeled 'P' in the figure). It answers DNS queries and receives requests of dynamic updates from the Internet. However, the processing of update requests is postponed. Rather these requests are merely stored in Online Storage 1. In the mean time, Server 0 (labeled 'C') has just rebooted and Server 2 is the secondary (labeled 'S') and answers DNS queries. As shown in the figure, Server 0 connects itself to Offline Storage and Online Storage 2. It obtains the zone private key from Offline Storage, retrieves the pending update requests from Online Storage 2 (the reason of this particular online storage will become clear when the cycle completes), processes them according to DNSSEC, updates the master file in Offline Storage, and saves a copy of the newly updated master file in its local storage. Server 0 then disconnects from Offline Storage and eliminates the zone private key from its local disks and memory. It is now ready to initiate a role rotation by exchanging control messages with other servers through the SCIT-CCP protocol, discussed in Section IV.

In Figure 3(b) we show the configuration after the above role rotation, whereby Server 0 assumes the role of primary, Server 1 becomes the secondary and Server 2 reboots and restarts its lifecycle. In this configuration, new requests of dynamic updates are stored in Online Storage 0 by Server 0. The newly rebooted and cleansed Server 2 processes pending updates (those received by the previous primary and stored in Online Storage 1, commits the updates to the master file in Offline Storage, and saves a copy of the master file locally, before it initiates a role rotation.

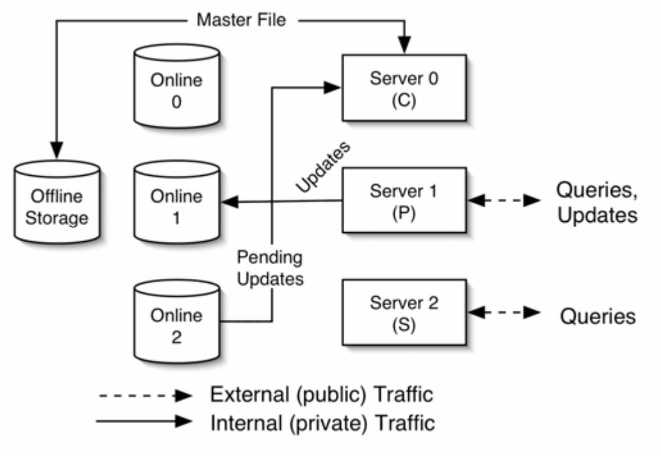

Figure 3(a): Server 1 as the Primary

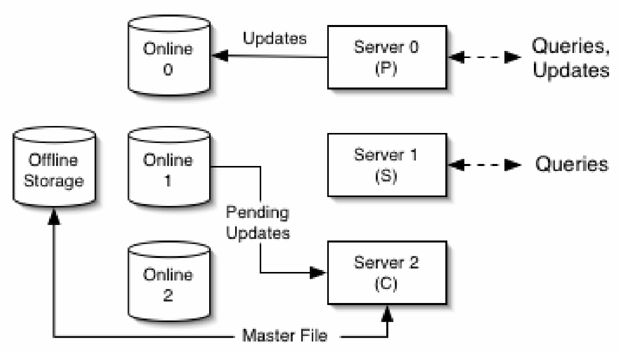

Figure 3(b): Server 0 as the Primary

In Figure 3(c) we show the configuration after the second rotation, whereby Server 2 becomes primary, Server 0 becomes secondary and Server 1 restarts its lifecycle. Arriving update requests are now stored in Online Storage 2. After rebooting, Server 1 processes the pending updates in Online Storage 0 and updates the master file in Offline Storage. Server 1 then initiates a role rotation, and the system returns to the configuration in Figure 3(a), where pending requests collected in this phase and stored in Online Storage 2 are processed by Server 0.

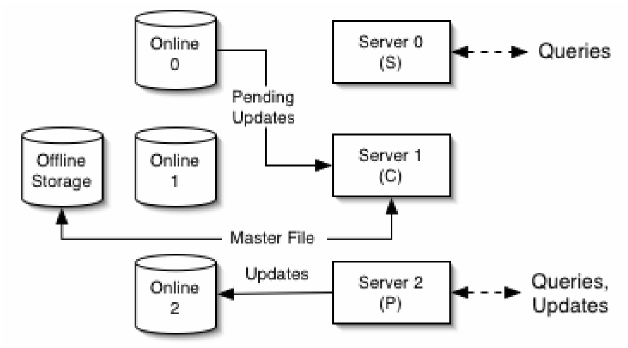

Figure 3(c): Server 0 as the Primary

We are now ready to present the lifecycle of SCIT DNS servers, starting from the first task immediately after rebooting. In the Lifecycle of SCIT-DNS Server, all arithmetic is in modulus 3.

Lifecycle of SCIT-DNS Server i, $0 \le i \le 2$:

1. Perform system integrity checking and, if necessary, recovery procedures. This mainly involves checking the signatures of important system files and directories.
2. Connect to the Offline Storage and retrieve the zone private key.
3. Connect to Online Storage i-1 and process, according to RFC2136, the pending requests of dynamic updates stored there. The master file is updated in the process and the signatures of affected RR recomputed using the zone private key.
4. Make a copy of the newly updated master file in the local file system of Server i.
5. Disconnect from both the Offline Storage and Online Storage i-1.
6. Eliminate all traces of the zone private key by reinitializing disk blocks and memory pages used to store the key.
7. Connect to Online Storage 1, for storage of incoming DNS dynamic updates when Server i becomes the Primary DNS server.
8. Use the CCP ROTATE_ROLE message to initiate a role rotation and then claim the IP address of the primary server.
9. Server i assumes the role of the Primary DNS server, which: answers DNS queries using its local copy of the master file, receives dynamic update requests and stores them in Online Storage i, and honors the Secondary DNS server's requests for DNS data synchronizations.
10. Upon receiving a ROTATE_ROLE CCP message, claim the IP address of the Secondary server.
11. Server i assumes the role of the secondary server. It will answer DNS queries but reject dynamic update requests. It synchronizes its local DNS data with the new primary in accordance with [12].
12. Upon receiving a GO_OFFLINE CCP message, reboot.

We point out that in Figures 3(a) through 3(c) there are no communications paths from outside to Offline Storage at all times. The zone private key is always offline. Neither is there any communications path from outside to the servers labeled 'C', those that have been newly cleansed but not yet gone online. Without communication paths, it is (virtually) impossible to breach such servers from the public Internet. We say a server is *clean* if it has completed Steps 1 to 8 of a lifecycle.

To conclude this section, we summarize the benefits of the SCIT-DNS cluster as follows:

- *Key Protection.* Zone private keys are kept offline even though they are used to sign dynamic updates, ensuring the highest level security for dynamically updated RR. Primary server compromises will not expose zone private key and do not degrade data origin authentication.
- *High availability.* The system gracefully handles the failure of one server. In the presence of one server failure the system simply stops role rotations, because there will be no newly cleansed machine to send ROTATE_ROLE messages in Step 8 above. Dynamic updates are however postponed until the third server is repaired and activate role rotation again.
- *Intrusion Tolerance.* Even if a hacker manages to breach a server, say the current primary, she can inflict damage only for limited time. For instance, if the hacker deletes the online copy of the master file, stored in the primary server, services will be restored when the next primary server takes over. Trojan horse programs can be detected and removed through system auditing after rebooting [HuSK03, HoDu98]. If the hacker means to use the server as a jump pad to the intranet, she loses her foothold in a short period of time.

IV. Cluster Coordination Protocol

In this section we introduce the SCIT Cluster Coordination Protocol (SCIT-CCP) which manages the server role rotations within the SCIT-DNS cluster. Figure 1 illustrates the cluster architecture in which the protocol is designed to operate.

SCIT-CCP is designed to enable the intra-cluster communication necessary to effect the server role rotations while at the same time not providing connections between the DNS servers in the cluster. By not maintaining active connections between the servers in the cluster we do not keep any ports open other than those used for DNS transactions. This virtually eliminates the possibility of an attacker gaining control of one server in the cluster and using it as a foothold to compromise other machines in the cluster. To communicate our cluster management messages without intra-cluster connections we turn to a novel technique called port knocking [9].

Port knocking provides the means to transport messages across *closed* ports. A "port knock" is nothing more than an attempted connection to a port that is closed but is being monitored. By looking for predefined sequences of port knocks, servers can exchange critical control information without ever opening a port and exposing themselves to attacks. This mechanism is not used for data exchange. To use port knocking for our protocol's transport mechanism we first choose a range of ports to use for communication; we reserve ports 28,000 though 28,100 for this purpose. Secure implementations of port knocks should use per-machine firewalls, available on modern OS, to (1) block this port range and (2) suppress the sending of ICMP error messages to clients attempting to connect to these ports. One way to see port knocking activities is through monitoring firewall log files.

There are four messages defined in CCP (see Table 1). Each message is comprised of two identical knocks. Further, ACKs at the *message level* not the individual packet/knock level since our messages are all only two packets long and do not require sequencing.

Table 1
SCIT-CPP Messages and Ports

Message Type	*Dest. Port*
ROTATE_ROLE	28,010
ACK_ROTATE_ROLE	28,011
GO_OFFLINE	28,020
ACK_GO_OFFLINE	28,021

A knock in CCP is a TCP/SYN packet sent to a designated port on the receiving machine. The destination address of a knock is the IP address that is associated with the "role" of the intended receiver. The destination port number is determined by the type of the message as defined in Table 1. Besides the SYN flag, source address, destination address and destination port, other fields of the TCP/IP headers are not used.

We now step through one cycle of the protocol at the message level to illustrate its operation within the DNS cluster. Figure 4 depicts the CCP message exchanges.

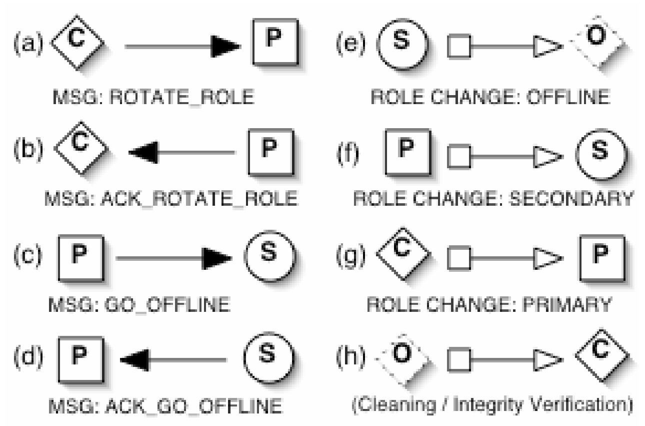

Figure 4: CCP Steps (a) – (h)

The rotation cycle begins with the Clean (C) server coming online ready to takeover the role as the Primary DNS server. The Clean server sends a ROTATE_ROLE message, Fig. 4, Step (a), to the current Primary (P) server which responds by sending back an ACK_ROTATE_ROLE message (Step (b)). Next, the Primary server sends a GO_OFFLINE message (Step (c)) to the current Secondary (S) server which replies with an ACK_GO_OFFLINE message (Step (d)).

We see the first role change in Step (e) as the Secondary server immediately goes offline to begin the cleansing and integrity verification process. With the Secondary role now open, the Primary server transitions into this role by acquiring the IP address for the Secondary DNS server by a gratuitous ARP announcement to the cluster subnet (Step (f)). When the Clean (S) server sees the primary role vacated it broadcasts its own gratuitous ARP announcement to claim the IP address for the Primary DNS role (Step (g)). The final state transition (Step (h)) is the process of restarting the Offline (O) server followed by cleansing, integrity checking and master file updating. The result of this transition is the new Clean (C) server; the rotation process now repeats.

Lastly, the state transition diagrams for CCP knock processing are given in Figure 5. The diagrams in the figure illustrate the process used to send and process both the ROTATE_ROLE and the GO_OFFLINE messages. The "do action" box in the diagram refers to either the claiming of the IP address associated with a server role change or the restarting and cleansing process. Timeout lengths in the figure are those used in our own prototype. They can be adjusted for different implementations. In the next section we look at the specific implementation details for our prototype cluster which runs the SCIT-CCP protocol.

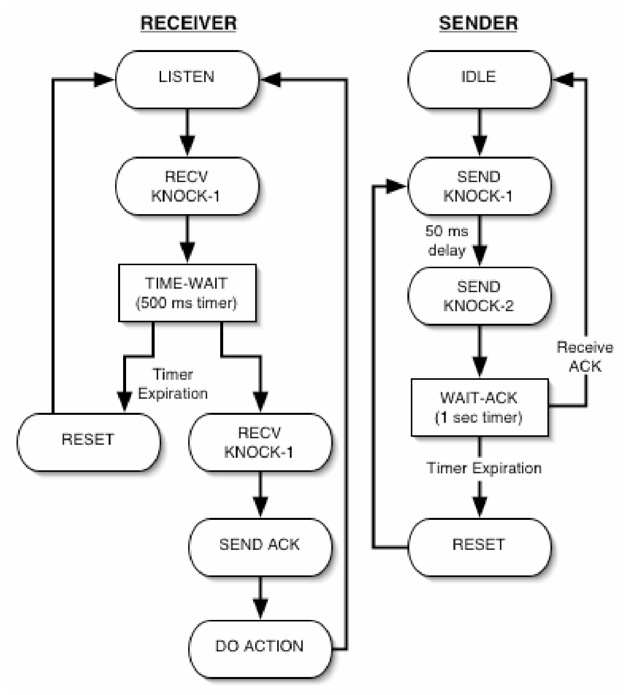

Figure 5: SCIT-CCP State Transition Diagrams

V. Prototype Design

In this section, we present the designs of our prototype implementation of the SCIT DNS architecture. Each DNS server in Figure 1 is an Intel/Linux PC in the prototype. The three servers are connected to a fast Ethernet switch which in turn is connected to a border router/firewall, also running Linux. Two public, routable IP addresses are used, one for each of the two active DNS servers. We use static NAT translation to provide a private, non-routable IP address for each of the DNS servers in the cluster. Linux IPTABLES is used to perform address translation and packet filtering. When a server is assigned a new role in the cluster, either as Primary DNS or Secondary DNS, it claims the private IP address for that role using a gratuitous ARP announcement on the subnet. The server lifecycle algorithm is coded in a shell script and executed immediately after booting.

Coordination of server roles is accomplished using the SCIT-CCP protocol described in the previous section. The processing of knocks, in accordance with Figure 5, is implemented in a SCIT-CCP daemon running on every server in the cluster. Each server in the cluster also runs a local firewall independent of the firewall box in Figure 1. The SCIT-CPP daemon on a server receives incoming knocks through monitoring the log file produced by this local firewall.

File integrity is monitored and maintained in our system through the use of Tripwire [8], which is a signature-based change detection package. The package is used to digitally sign important system files and directories (/sbin, /usr/bin, and so on). In Step 1 of the server lifecycle, these signatures are checked and in the cases of inconsistencies, corresponding files are recovered from a local read-only storage. These tasks constitute the "self cleansing" of SCIT DNS systems. We point out that the copy of the master file on the primary server is not signed. Corruption in this copy will be automatically corrected when the next primary server takes over – the new Primary will use a new copy obtained directly from Offline Storage (Step 4 of server Lifecycle).

In our prototype we segregate the requests coming into the name server using a *DNS wrapper* program which borrows from the idea of Dr. Venema's TCP Wrappers [17]. The wrapper listens on port 53 (TCP/UDP) and acts as a proxy for named, a popular open-source implementation of DNS/DNSSEC. It passes quires to named but stores incoming dynamic update data in a temporary file on the online storage machine associated with the Primary DNS server. As discussed in Section 3, the updates are not incorporated into the master file until they are verified [14] during the next cleansing cycle.

Each backend storage in Figure 1 is implemented as an Intel/Linux machine with local hard drives. The connection between a server and a backend storage machine is established as follows. A port knocking message is used to begin the process. Two knocks on port 28,030 on a backend storage machine opens a designated port for the server shell script to establish a SSH channel. Such connections are used to give clean servers access to the master file in the offline storage and pending dynamic update requests stored in an online storage. The current primary server also establishes a connection to an offline storage to store incoming update requests.

Lastly, we briefly discuss the limitations of the current protocol and prototype design with an eye toward our future research. We have presented a cluster architecture that incorporates three servers in a round robin rotation scheme. In the future we will extend our cluster model to an arbitrary number of servers and may utilize rotational schemes other than straight round robin. A potential enhancement to SCIT-CPP includes the ability to carry data payloads for message origin authentication. With the presence of payloads in CCP messages, error handling will also likely be needed. We will also explore the potential value of building a SCIT cluster using pairs of SCIT servers as individual cluster nodes. In such a configuration, each node in the cluster would be a pair of SCIT servers alternates roles between providing service and cleaning as in our prior work [6, 7].

VI. Conclusion

The SCIT DNS cluster architecture provides high-availability, intrusion tolerance, and guaranteed DNS integrity through periodic restarting, cleansing and integrity checking of each server in the cluster. To achieve this we operate the cluster such that each server rotates through each role shown in Figure 2. By limiting the time any server is running and exposed to the public Internet we limit the amount of exposure to potential attack in terms of time. That is if an attacker is able to corrupt either the Primary or Secondary DNS server it will soon be restarted and cleaned, ejecting the hacker, and repairing any damage done to critical files. We protect the integrity of the master file by segregating DNS dynamic updates from name resolution requests, storing the dynamic updates in a temporary storage until they can be verified and committed to the master file during the cleaning process.

SCIT-DNS architecture augments the security afforded to DNS services over and above the DNSSEC enhancements. Through hardware redundancy, segregation of dynamic DNS updates, server role rotations, and the periodic cleansing process, SCIT DNS ensures high availability and master file data integrity even in the face of unknown or undetected attacks. A more resilient and reliable DNS service contributes significantly to the protection of network-based critical infrastructure systems.

REFERENCES

[1] 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.

[2] 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. [CIDF] Common Intrusion Detection Framework, www.gidos.org

[3] Eronen, P. and Sars, J. Applying decentralized trust management to DNS dynamic updates. In Proceedings of the NordU/USENIX 2001 conference. Stockholm, Sweden.

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

[5] M. Hosmer and M. Duren, "Detecting subtle system changes using digital signatures," Proceedings of Information Technology Conference, pages 125—128, (Syracuse, NY) Sep. 1998.

[6] 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.

[7] 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.

[8] Gene H. Kim and Eugene H. Spafford, "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool," in *Proceedings of USENIX Applications Development Symposium*, (Toronto, Canada), April 1994. Also see www.tripwire.com.

[9] Krzywinski, M. 2003. Port Knocking: Network Authentication Across Closed Ports. SysAdmin Magazine 12: 12-17. www.portknocking.org

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

[11] Mockapetries, P. Domain names - concepts and facilities. Internet RFC1034, 1987.

[12] Mockapetries, P. Domain names implementation and specification. Internet RFC1035, 1987.

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

[14] D. Eastlake. Domain Name System Security Extensions. Internet RFC 2137, April 1997.

[15] Vixie, P.: DNS and BIND security issues. In Proceedings of the 5th Usenix Security Symposium. Salt Lake City, UT, 1995.

[16] Wang, X., Huang, Y., Desmedt, Y., and Rine, D. Enabling secure on-line DNS dynamic update. In Proceedings of the 16th Annual Computer Security Applications Conference. New Orleans, Lousiana, USA, 52—58, 2000.

[17] Venema, Wietse. "TCP Wrappers." 1997. htp://www.porcupine.org.

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