Incorruptible System Self-Cleansing for Intrusion Tolerance

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 problem stems in large part from the constant innovation and evolution of attack techniques. The trend lends importance to the concept of *intrusion tolerance*: 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, we show that, with simple hardware enhancements strategically placed in a SCIT system, incorruptible intrusion containment can be realized. We then present an incorruptible SCIT design for use by one of the most critical infrastructures of the Internet, the domain name services. It is our belief that incorruptible intrusion containment as presented here constitutes a new, effective layer of system defense for critical information systems.

Keywords
computer security, self-cleansing systems, intrusion containment, domain name system

I. INTRODUCTION

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. 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 [1,2,3,5] we presented our designs of SCIT-enabled firewalls, web servers, and DNS servers. The primary contribution of this work is the introduction of incorruptibility guarantees of SCIT, called SCIT Primitives, enforced by simple hardware enhancements strategically placed in a SCIT server cluster.

We point out that SCIT does not exclude the use of intrusion prevention and detection technologies, but rather adds another layer of defense, extending the idea of "defense-in-depth" through periodic system cleansing. 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. We first review SCIT basics in Section 2. We present, in Section 3, a set of criteria, called SCIT Primitives, which ensure the incorruptibility of system self-cleansing cycles. A generic design framework that satisfies said primitives is discussed in Section 4. As an example, we give in Section 5 the details of an incorruptible SCIT system for use by domain name services (it is an adaptation of our prior SCIT DNS design [3]). We conclude this work in Section 6.

II. 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 [3]. A high level view of SCIT cluster operations is depicted in Figure 1.

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

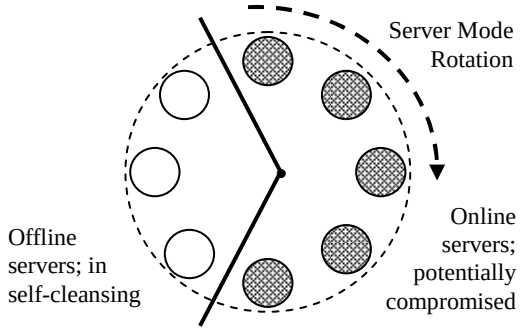


Fig. 1: A High-level View of SCIT Cluster operations

As a specific example, consider the SCIT DNS cluster shown in Figure 2. The cluster comprises *three* server boxes and advertises *two* IP addresses, a primary name server address and a secondary name server address. At any point in time only one of the servers will be operating in one of the following three states: (1) Primary DNS, communicating with clients using the primary IP address, (2) Secondary DNS, communicating with clients using the secondary IP address, and (3) Offline for self-cleansing with no public IP address.

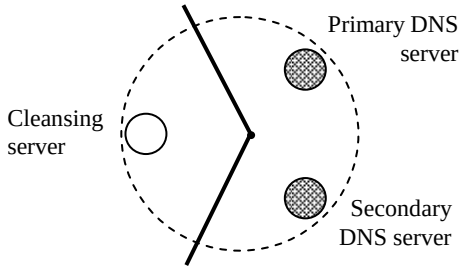


Fig. 2: The SCIT DNS Cluster

Part of the rotation process is to bring online servers offline. Next, system is rebooted to initiate cleansing procedures in order to return servers 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. In SCIT DNS, its 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.

We notice that a simple form of server rotation has previously been employed in high-availability systems, where backup servers rotate online to ensure uninterrupted service in face of primary server failures [4]. 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 existing high-availability systems include DNS servers, NFS servers, authentication services, firewalls, IPsec gateways, and virtual private network (VPN) gateways.

III. SCIT PRIMITIVES

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 role either. The server could already have been under attacks while cleansing, and going online inevitably involves communications, giving opportunities to unpredicted breaches. Lastly, but most importantly, an online server may be taken over by attackers. The process of counting down to the next rebooting could consequently be interrupted, stopping the SCIT cleansing cycles all together.

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 any 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.

P1. **Inevitability of periodic server cleansing**. A server will be rebooted and subsequently cleansed within a predetermined[1] 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

[1] 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.

subject to compromises in its own operations. In the next section, we present a SCIT framework that is incorruptible as defined above through the use of simple hardware enhancements.

IV. INCORRUPTIBLE SCIT WITH 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 Figure 3, where the controller keeps two-way communication paths with cleansing servers but only one-way paths to reach online servers. Consequently, an online server or any node outside the cluster (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 Figure 3 represent permissible directions in communication; they do not mandate dedicated communication channels.

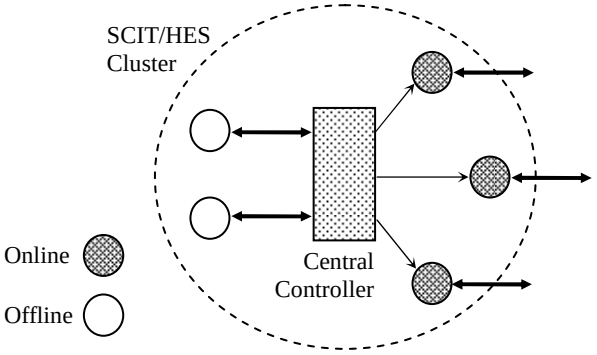
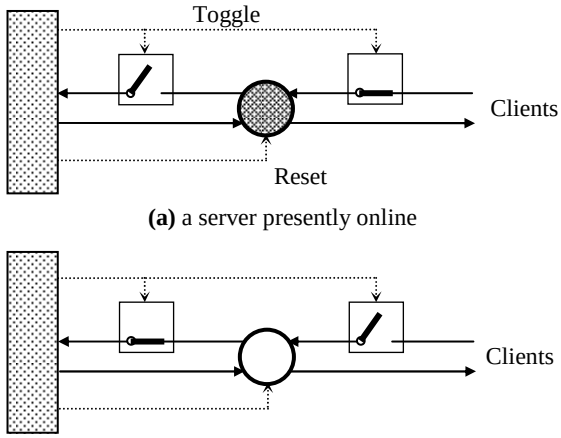


Fig. 3: Communication configuration of SCIT/HES

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 Figure 4. In the figure, we use solid lines to represent network links for TCP/IP message exchanges (such as Ethernet cables) and dashed lines to represent wires/fibers that conduct electromagnetic control signals.

As seen in Figure 4(a), the controller has two signal lines to reach each server: a reset line and a toggle line. A reset signal forces the server to reboot. The toggle signal controls two switches that are always in opposite states. A toggle signal followed by a reset switches the online server in Figure 4(a) to the offline mode in Figure 4(b). 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. Cleansing servers receive inputs from only the central controller. Primitives P2 and P5 are thus fulfilled.



(a) a server presently online

(b) the server after a toggle signal and a reset signal; now offline

Fig. 4: The hardware enhancements in SCIT/HES

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. Once a server is rebooted, it enters a bootstrapping process followed by cleansing routines. 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 disconnected from exposed nodes. The process of role assumption is not subject to external influence, and Primitive P4 is fulfilled.

V. INCORRUPTIBLE INTRUSION CONTAINMENT A DNS EXAMPLE

To illustrate the use of the SCIT/HES framework, we present its application in a critical part of the Internet, the domain name systems. A SCIT/HES DNS cluster includes three servers (S0, S1, and S2) and a central controller that executes the algorithm presented below to coordinate server rotations. There are two forms of communications in the algorithm. First, the controller uses unidirectional signals to reset a server and change its network communication paths. Second, the controller uses two-way TCP/IP message exchanges to communicate the designation of an online role/identity with a newly cleansed server. When receiving a role message from the controller, the server acknowledges only after it has completed its self-cleansing.

Centralized Server Rotation Algorithm

Variables:
- Toggle[0,2]: toggle signals to servers S0, S1, S2
- Reset[0,2]: reset signals to servers S0, S1, S2
- P: integer in [0,2] for the ID of the primary server
- S: integer in [0,2] for the ID of the secondary server
- C: integer in [0,2] for the ID of the cleansing server
- T1 and T2: two timers

Initializations:
- P = 0, S = 1, and C = 2.
- Set timer T1.

When T1 fires // switch primary with cleansing
- Send signal Reset[P].
- Send signal Toggle[P].
- Send role 'Primary' to Server[C].
- Wait for an ack from Server[C].
- Send signal Toggle[C]
- Swap the values of P and C.
- Set timer T2.

When T2 fires // switch secondary with cleansing
- Send signal Reset[S].
- Send signal Toggle[S].
- Send role 'Secondary' to Server[C].
- Wait for an ack from Server[C].
- Send signal Toggle[C]
- Swap the values of S and C.
- Set timer T1.

Following the algorithm, the central controller alternates between two timers T1 and T2 so that role rotations alternate between Primary↔Cleansing switches and Secondary↔Cleansing switches. The system starts with the configuration (P,S,C), where server S0 is the primary (P), S1 the secondary (S) and S2 in cleansing (C). (Due to space limits, the steps whereby servers enter their respective initial roles have been omitted in the above presentation.) After the first timeout event (T1 fires), the primary server S0 and the cleansing server S2 switch mode, resulting in the configuration (C,S,P). A complete cycle of server rotations is given in Figure 5.

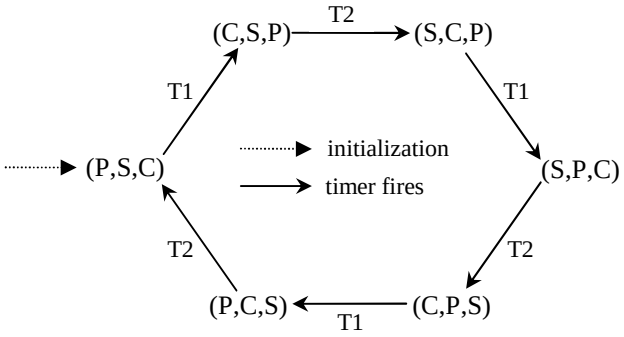


Fig. 5: A cycle of SCIT DNS server rotations

VI. CONCLUSION

We have shown that with simple hardware enhancements strategically placed in a server cluster, it is possible to build intrusion containment mechanisms that cannot be corrupted. We have presented a SCIT/HES DNS cluster as an example of our framework. It is our belief that incorruptible intrusion containment as presented constitutes a new, effective layer of defense for critical information systems against undetected and unknown attacks, the unknown unknowns in computer system security.

REFERENCES

[1] 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.

[2] 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.

[3] 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)

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

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