SCIT Based Moving Target Defense Reduces and Shifts Attack Surface

Ajay Nagarajan[1] and Arun Sood[1, 2]

[1]Department of Computer Science, George Mason University, Fairfax, VA
[2]SCIT Labs, Clifton, VA

***Abstract* - Current approach to security is based on perimeter defense and relies on reactive systems like firewalls, intrusion detection and prevention systems. These systems require a priori information about attacks and vulnerabilities. McAfee reports identifying 100,000 new unique malware each day. Thus trying to prevent intrusions is becoming impractical. Although it is difficult to model and predict a hacker's moves, a defender might be able to make hacker's life harder by adopting Moving Target Defense (MTD) proactive security strategies. In this paper, we present SCIT, our MTD approach. We use Attack Surface assessment to evaluate SCIT. A system's attack surface is the subset of its resources that an attacker can use to attack the system. Manadhata uses attack surface reduction / shifting as means of assessing MTD. In this paper, we compare the dynamically changing Attack Surface for three system architectures (1) Static Systems; (2) Basic-SCIT and (3) Diverse-SCIT.**

***Index Terms* – Moving Target Defense, Attack Surface, Intrusion Tolerance, Proactive Recovery**

I. INTRODUCTION

Current Information Technology systems operate in a relatively static configuration and primarily focus on intrusion avoidance. For example, names, addresses, software stacks, networks, and various other configuration parameters remain static over extended periods of time. At the same time the variety of malware is increasing - McAfee reports [1] identifying 100,000 new unique malware each day. Thus preventing all intrusions is very hard. We believe that intrusions are inevitable. Current experience shows that in spite of prevention devices, the criminals are able to ex-filtrate data and damage systems. Industry studies by Verizon [2] and Mandiant [3] show that criminals are often in the compromised systems for months. According to the 2013 Verizon Business Data Breach Investigation Report [2], the average time an intruder resides in a system from initial compromise to the point of discovery is more than 34 days. The current static server approach is a legacy design striving solely for simplicity and performance despite the increasing concern of malicious exploitation of system vulnerabilities.

Moving Target Defense (MTD) is the idea of managing change across various system and network dimensions in order to increase the intruder work factor by increasing the intruder work complexity and decreasing visibility of systems to the intruders. Traditionally MTD strategies have presented two significant challenges to adoption. First, for the sake of security, MTD cannot ignore performance and end user productivity. Most customer facing systems don't have the luxury of adding security that slows down performance. Customers tend to move on if the experience is slow and tedious. Secondly, traditional MTD design generally consists of complex processes involving memory address randomization, network address shuffling, instruction set randomization and more [4]. All of these techniques are designed to prevent attacks and have the potential to be resource hogs thereby slowing down throughput.

SCIT based Moving Target Defense acknowledges that trying to prevent each intrusion is impractical. Therefore, we shift the emphasis to minimizing losses occurring from intrusions rather than preventing intrusions. SCIT systems are designed to be complementary to reactive systems [5]. Primary goal of SCIT-MTD is to reduce the intruder's window of opportunity to execute an attack and increase the costs of their foot-printing, scanning and attacking efforts. Since by design the SCIT-MTD attack surface of the system is constantly changing, the system vulnerabilities are difficult to exploit. The process of compromising a system involves identifying system vulnerabilities and customizing attacks to exploit them. Ever-changing attack surface presents a stiff challenge to the intruders. SCIT – MTD can be used with diversification approaches to further increase the attacker difficulty.

A. How SCIT (Self-cleansing Intrusion Tolerance) works

In [6] and [7], we presented SCIT, an intrusion tolerant technique that provides enhanced server security. SCIT research has focused on critical servers that are most prone to malicious attacks. The technique involves multiple virtual instances of server that are rotated and self-cleansed periodically irrespective of the presence or absence of intrusions. Self-cleansing refers to loading a clean image of the server's OS and application into the Virtual Machine. Rotation here refers to the process of bringing an exposed virtual server off-line, killing it, restarting it and in the meanwhile, bringing another virtual server online to assure availability. By doing so, in the event of an intrusion, the intruder is denied prolonged residence on the server. Once the virtual server's 'exposure time' to the Internet is completed, the virtual server instance is automatically rotated. This virtual

instance of the server is interchangeably referred to as virtual server throughout this paper.

Figure 1: SCIT Server rotation

The illustrative example in Figure 1 shows 3 different time periods. At any given time, there are five red servers online and three green servers being wiped clean. In each case a different set of servers is being cleansed. Eventually every server will be taken offline, cleaned and restored to its pristine state. SCIT technology can be used to build a variety of servers that meet enhanced security requirements. In this paper, we use SCIT approach to SCITize a web server, and assess its security performance.

B. *Common Security Evaluation Metrics and Attack Surface*

Measurement of security has been a challenge and is of practical importance to software industry. Today we commonly use two measurements to determine the security of a system: (1) at the 'code level', we count the number of bugs found (or fixed from one version to the next); (2) at the 'system level', we count the number of times a system version is mentioned in CERT advisories, Security Bulletins and Vulnerability Databases like MITRE CVE. Manadhata [8, 9] proposed Attack Surface as a security metric that focuses at the 'design level' of a system: above the level of code, but below the level of the entire system. Attack Surface is a metric to compare the relative security of two versions of the same system rather than the absolute security of a system. Given two versions, A and B, of a system, one could measure the security of A relative to B with respect to the system's attack surface. Intuitively, higher the attack surface, more the chances of the system getting compromised e.g., eliminating certain system features potentially makes it more secure.

Attack Surface assesses (a) system 'actions' externally visible to the system's users; and (b) system 'resources' accessed or modified by each action. The more actions available to a user or the more resources accessible through these actions, the more exposed the attack surface. The more exposed the attack surface, the more likely the system could be compromised. System Resources that contribute to Attack Surface (Attack Surface Components) are: (a) Entry Points – Each system has a set of methods that receive data items from the environment as input. These are the system's entry points. Example – an API method that accepts a user input; (b) Exit Points – A system's method that send out data items to the system's environment are the system's exit points. Example – a method that writes into a log file; (c) Channels – The channels are the means by which users / systems in the environment communicate with one another. Example – TCP / UDP sockets; (d) Untrusted Data Items – An attacker utilizes data items to send data to or receive data from the system. Example of such data items are files, cookies, registry entries etc.

The Formal Definition of Attack Surface is [8] "*The set M of entry points and exit points, the set C of channels and the set I of un-trusted data items are the system's resources that can be used by the attacker to compromise the system. Therefore, given a system S and its environment, the system's attack surface can be represented as the triple <M, C, I>".*

Attacks carried out over the years, however, show that certain system resources are more likely to be opportunities, i.e., targets or enablers, of attack than others. This leads to the idea of 'Weighted Attack Surface'. For example, services running as the privileged user root in UNIX are more likely to be targets of attack than services running as non-root users. Similarly, in Windows, applications, such as Internet Explorer and Outlook Express, with VBScript, JScript, or ActiveX controls enabled are more likely to be enablers of attack than if such scripting engines and controls were disabled. Since every system resource contributes unequally to the system's attack surface, author of [8] proposes the use of '*Damage Potential – Effort ratio'*. The amount of damage that can be done to the system by exploiting a particular resource is the damage potential of that resource. Similarly, the amount of work that the attacker would have to put in to use that resource as an attack tool defines the effort.

In summary, the computation of a system's attack surface involves the following three steps [8]:

1 Given a system, s, and its environment, E_s, identify a set, M^{Es}, of entry points and exit points, a set, C^{Es}, of channels, and a set, I^{Es}, of un-trusted data items of s.

2 Estimate the damage potential-effort ratio, $der_m(m)$, of each method $m \in M^{Es}$, the damage potential-effort ratio, $der_c(c)$, of each channel $c \in C^{Es}$, and the damage potential-effort ratio, $der_d(d)$, of each data item $d \in I^{Es}$.

3 The attack surface of s is defined by the triple:

$$\langle \sum_{m \in M^{Es}} der_m(m), \sum_{c \in C^{Es}} der_c(c), \sum_{d \in I^{Es}} der_d(d) \rangle \quad \textbf{(1)}$$

II. ATTACK SURFACE SHIFTING / REDUCTION AS A TECHNIQUE FOR MOVING TARGET DEFENSE

In [8] Manadhata formalized the notion of a software system's attack surface and proposed the use of system's attack surface measurement as an indicator of the system's security. Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. Hence larger the attack surface, the more insecure the system.

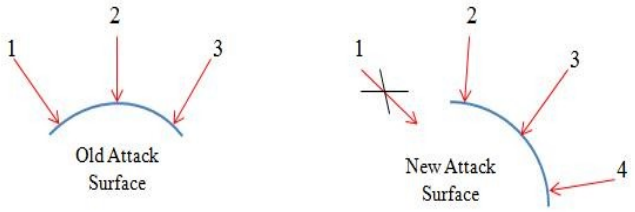


Fig 2: Attack Surface Shifting

In [9], the author considers a scenario where system administrators are continuously trying to protect their systems from attackers. As shown in Fig. 2, if a defender shifts a system's attack surface, then old attacks that worked in the past, e.g., attack 1, may not work anymore. Hence the attacker has to spend additional effort to make past attacks work or find new attacks, e.g., attack 4. Hence, the interaction between the defender and the intruder here can be viewed as a two player game where the action of one player has a consequence on the other. Thus, reducing or shifting a system's attack surface functions as MTD. This works in favor of the defender to increase the intruder's work factor randomly.

Attack Surface of a system can be reduced or shifted by Disabling, Modifying and / or Enabling the system's features [9]. Disabling the existing features reduces the number of entry points, exit points, channels, and data items, and hence reduces the number of resources that are part of the attack surface. Modifying the features changes the damage potential-effort ratios of the resources that are part of the attack surface, e.g., lowering a method's privilege or increasing the method's access rights reduces the resources' contributions to the attack surface measurement. The enabled features increase the attack surface measurement by enabling new features and adding more resources to the attack surface. When existing features are disabled and new features enabled, the attack surface shifts. Table 1 presents four illustrative scenarios to highlight the impact of disabling, enabling or modifying features on a system's attack surface:

Scenarios	Features	Attack Surface Reduction	Attack Surface Shift
A	Disabled Existing	Yes	Yes
B	Enabled New	No	No
C	Enabled New Disabled Existing	Yes	Yes
D	Enabled New Disabled Existing	No	Yes

Table 1: Possible Scenarios to Reduce and Shift the Attack Surface

A. *Dynamic Attack Surface*

The Attack Surface of a production system increases with time. For example, the number of open sockets may increase because of programming oversight. In typical operations, the application of a security patch reduces the Attack Surface, while a patch that increases functionality increases the Attack Surface. Thus, the Attack Surface is a dynamic property. The SCIT approach constantly restores software to a pristine state, and thus dynamically reduces the Attack Surface.

III. TEST BED EXPERIMENT

In this paper, we use Attack Surface assessment to evaluate impact of our MTD solutions. We compare the dynamically changing Attack Surface for three security configurations (1) Static systems; (2) Basic-SCIT and (3) Diverse-SCIT. Static systems adopt traditional reactive systems; they are sitting ducks that are indefinitely online. System cleansing and recovery is generally manually triggered or by an IDS/IPS. We compare Static systems with two flavors of SCIT - MTD: Basic-SCIT which loads the same pristine image every-time a virtual server is self-cleansed; and Diverse-SCIT loads clean images of diverse implementations of the same service during the SCIT cycle. We use the Microsoft Attack Surface Analyzer [10] to perform all attack surface assessment.

A. *Configuration of System used for Test Bed Experiment:*

Gateway P-7805u
Intel Core 2 Duo CPU P8400 @ 2.26GHz
4 GB RAM
64-bit Windows Vista Home Premium Service Pack 2

B. *Assumptions made for analysis*

Since it is not plausible to determine the 'Damage Potential - Effort Ratio' of every existing system resource (there are hundreds of them); we assume they are all equal. This is similar to the approach taken in [8]. On applying this logic to *Equation 1* above, we arrive at

Attack Surface Size = Total Number of Attack Surface Components

C. *Attack Surface Components*

For the purposes of our Test Bed Experiment, we use the Microsoft Attack Surface Analyzer [10]. In our experiment we assume that the following components make up the Attack Surface of a system. This is not intended to be comprehensive but summarizes the key components of any system's attack surface: (a) *Running Processes* – Process is an executing program; (b) *Executable Memory Pages* – Data Execution Prevention (DEP) is a system-level memory protection feature which enables the system to make one or more memory pages non-executable. Non-executable memory pages make it harder for the exploitation of buffer overruns. Therefore, fewer executable memory pages is better; (c) *Windows* – In a graphical Windows-based application, window is the area of the screen which interacts with the user by receiving input and displaying output; (d) *Kernel Objects* – An object is a collection of data that the OS manages. Kernel Objects are objects that are part of the kernel-mode operating system, for example: symbolic links, registry keys; (e) *Services* – Windows service is a program running in the background similar to a Unix Daemon; (f) *Drivers* – Software that enables

Attack Surface Component	Pristine Apache System	Apache System after 32 days	Apache System after 4 hour Exposure	Pristine Nginx System	Nginx System after 4 hour Exposure
Running Processes	76	79	77	76	77
Executable Memory Pages	25	46	27	21	26
Windows	183	265	199	180	189
Kernel Objects	513	520	513	513	516
Services	182	189	182	181	181
Drivers	274	284	274	274	274
TCP/UDP Ports	118	138	124	101	115
Named Pipes	133	146	136	133	134
RPC Endpoints	33	35	33	33	33
Attack Surface Size	**1537**	**1699**	**1565**	**1512**	**1545**

Table 2: Attack Surface Size comparison

the functionality of a physical or virtual device; (g) *Ports* – Ports are process specific communication endpoints of a system. Ports are associated with host IP addresses and the type of protocol used for communication as in TCP or UDP; (h) *Named Pipes* – A named pipe is a one-way or duplex pipe for communication between a pipe server and one or more pipe clients. A named pipe can facilitate inter process communication; (i) *RPC Endpoints* – Remote Procedure Call is an inter-process communication that allows a program to execute a procedure on a remote computer over the network. RPC Endpoints facilitate such communication; (j) *Objects with weak Access Control List (ACL):* These can be files, executables, registry entries etc. One example of a weak ACL is allowing non-administrators to modify files.

Sum of all of these components make up the Attack Surface Size of a system.

D. *Static Systems*

Static Systems are systems that do not incorporate proactive security strategies. In such systems, cleansing is generally triggered manually or by an alarm on discovering malicious activity. According to Verizon's Data Breach Investigation Report 2013 [2], the average time an intruder resides on the system from the point of initial compromise to the point of intrusion discovery is more than 34 days. In the case of a Static System, since there is no periodic self-cleansing or restoration, the attack surface of the system keeps on increasing with time as a result of normal system use.

To illustrate this, we compared the attack surface size of an Apache System (a) before use (pristine) and (b) after random usage for 32 days. Figure 3 illustrates the setup of the Apache System. Table 2 presents results from the attack surface analysis report. Columns 2 and 3 of Table 2 show the growth in Attack Surface Size of the static Apache system during usage.

Table 3 section (a) summarizes the security issues that arose during the 32 day usage. In other words, these security issues were not present in the Pristine Apache System but appeared in the Apache System after 32 days.

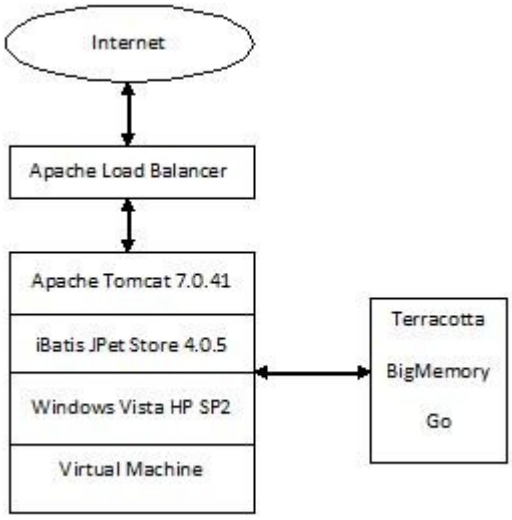


Fig 3: Apache System

E. *Basic-SCIT Setup*

In the Basic-SCIT setup, there are multiple virtual instances of the server out of which one or more are online at any given time. Once every period of time known as 'Exposure Time', the system proactively self-cleanses and rotates. Every time this happens, the Pristine Apache Image is loaded into the virtual instance that is about to go online. Figure 3 presents one such virtual instance. This setup is providing a Pet Store e-commerce application service through iBatis JPetStore 4.0.5. In our experiment, the 'Exposure Time' is 4 hours. Therefore, the attack surface size of the system can only increase till the point of self-cleansing. Thus there is an upper bound on the growth of the Attack Surface Size. After 4 hours, on self-cleansing, the size of the attack surface is reduced back to that of the Pristine Apache Image. This is cyclical and so the size of the attack surface is periodically reduced and kept manageable.

Table 2 (columns 2 and 4) compares the Attack Surface Size of a Pristine Apache System with that of the Apache System after 4 hour use. After the system has been exposed for 4 hrs,

Security Issues on System Usage	Count
(a) Pristine Apache System VS Apache System after 32 days	
Executables with weak ACLs	17
Directories containing objects with weak ACLs	10
Registry Keys with weak ACLs	10
Processes with NX disabled	1
Services vulnerable to tampering	3
Services with Fast Restarts	1
Vulnerable Named Pipes	26
(b) Pristine Apache System VS Apache System after 4 hours	
Directories containing objects with weak ACLs	3
Processes with NX disabled	2
Services vulnerable to tampering	1
(c) Pristine Nginx System VS Nginx System after 4 hours	
Directories containing objects with weak ACLs	4
Services vulnerable to tampering	1

Table 3: Security Issues after System Usage (a) Apache System after 32 days; (b) Apache System after 4 hours; (c) Nginx System after 4 hours

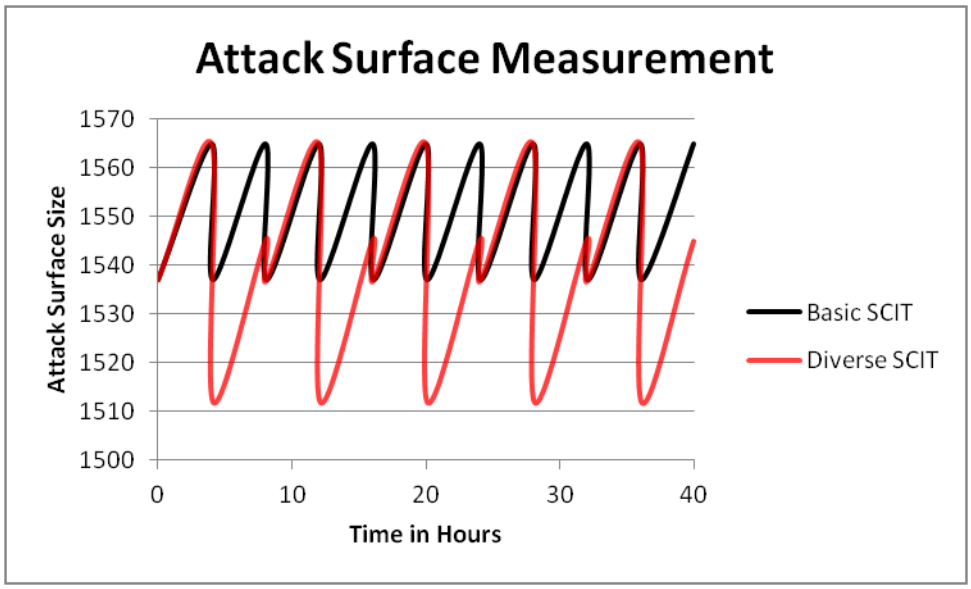
it is self-cleansed and rotated thereby reducing the Attack Surface Size back to that of the Pristine Apache System. From columns 2, 3 and 4, we emphasize that the growth in Attack Surface Size is much less in 4 hours as opposed to 32 days. Similarly, Table 3 sections (a) (b) shows that the count of security issues that arose over 32 days far outnumbered issues that arose from 4 hours.

Graph 1 (black series) presents the temporal attack surface of the Basic SCIT setup for the first 40 hours. The size of the system's attack surface increases for 4 hours when it is exposed and is reduced to that of the Pristine Apache Image on self-cleansing. This occurs iteratively, thereby always keeping the Attack Surface Size bounded to lower values.

F. Diverse-SCIT Setup

In Diverse-SCIT, we use a system with two diverse implementations of the iBatis JPetStore service. Virtual Server 1 uses the Apache Load Balancer with Apache Tomcat 7.0.41 and Terracotta Big Memory Go Caching; whereas Virtual Server 2 uses the Nginx HTTP Server with load balancer along with MemCached v1.4.15 to provide service.

Figure 4 presents two such virtual instances of the server, one with each configuration. In this setup, virtual servers are rotated in such a manner to alternate between the two configurations.



Graph 1: Temporal Attack Surface - Basic SCIT and Diverse SCIT

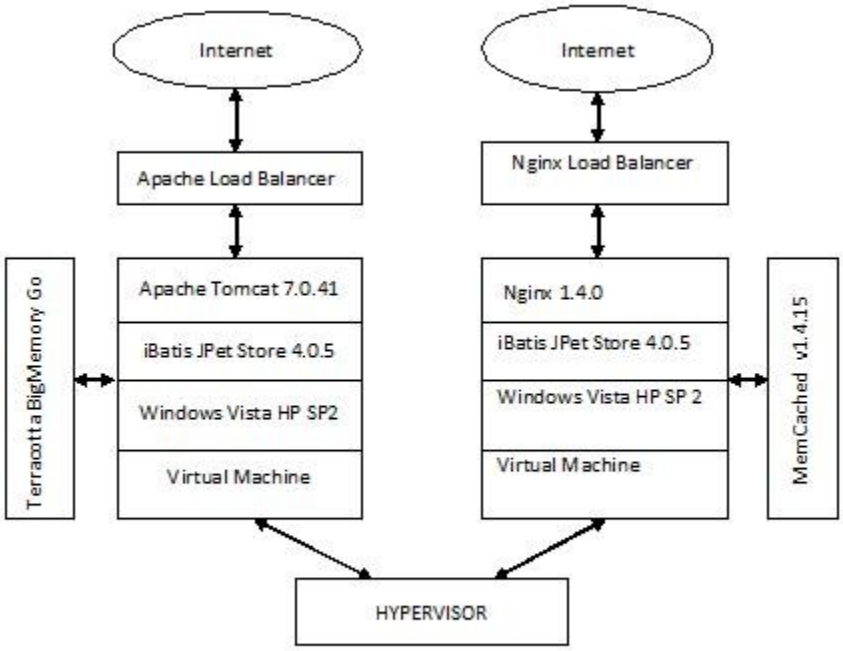


Fig 4: Two Virtual Instances of the Diverse SCIT setup

As shown in Graph 1 (red series), on each self-cleansing, the size of the system's attack surface alternates between that of Pristine Apache image and Pristine Nginx image.

Table 2 (columns 5 and 6) compares the Attack Surface Size of the Pristine Nginx System with the Nginx System after 4 hour use. Table 3 section (c) lists the security issues that arose during exposure.

Security Issues	Count
Issues present in Apache System but not in Nginx System	
Directories containing objects with weak ACLs	4
Processes with NX disabled	1
Services vulnerable to tampering	1
Issues present in Nginx System but not in Apache System	
Directories containing objects with weak ACLs	1
Processes with NX disabled	1

Table 4: Security Issues unique to each configuration

In addition to reducing the Attack Surface Size periodically, this setup also shifts it. Table 4 emphasizes this shift by presenting security issues that are unique to each setup. This adds another layer of complexity to the intruder since identifying system vulnerabilities with ever changing attack surface is a challenge. An attack that used to work with the previous configuration no longer works on rotation due to the diverse implementation of the new virtual instance.

3. Conclusion:

In this paper, we use Attack Surface assessment to evaluate the impact of our SCIT MTD solutions. We compared the dynamically changing Attack Surface Size for (1) Static Systems; (2) Basic-SCIT and (3) Diverse-SCIT using a test bed experiment. We present results of the experiments that show changes in attack surface size along a timeline for the three different security configurations. The results support our hypothesis that SCIT is an effective means to provide Moving Target Defense by reducing / shifting attack surface periodically, thus making the hacker task harder.. With Basic-SCIT, by moving the target virtual server, we force the intruder to restart the attack all over again. Furthermore with Diverse-SCIT, due to Attack Surface shifting, some of the attacks that worked before no longer work after self-cleansing and restoration.

Acknowledgement
This research was partially supported by Office of Naval Research grant N00014-13-1-0017.

References:

1. McAfee "Infographic: The State of Malware 2013"
2. Verizon Business "Data Breach Investigation Report 2013"
3. Mandiant "APT1: Exposing one of China's Cyber Espionage Units" report
4. David Evans, Anh Nguyen-Tuong, John Knight "Effectiveness of Moving Target Defenses" Chapter 2, Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats 2011
5. Ajay Nagarajan and Arun Sood, "SCIT and IDS Architectures for Reduced Data Ex-filtration" 4th Workshop on Recent Advances in Intrusion-Tolerant Systems, Chicago,IL, USA, June 28 2010.
6. Yih Huang, David Arsenault, and Arun Sood. "Incorruptible System Self-Cleansing for Intrusion

Tolerance". *Performance, Computing, and Communications Conference, IPCCC 2006*.
7. Quyen L. Nguyen and Arun Sood, "Comparative Analysis of Intrusion-Tolerant System Architectures", IEEE Security and Privacy. Preprint. Accepted for publication August 2010
8. Manadhata, P.K.: An attack surface metric. Ph.D. thesis, Carnegie Mellon University (2008)
9. Manadhata, P.K. "Game Theoretic Approaches to Attack Surface Shifting", Moving Target Defense II
10. Microsoft Attack Surface Analyzer http://www.microsoft.com/en-us/download/details.aspx?id=24487