An Intrusion Tolerance Approach to Enhance Single Sign On Server Protection

David Pham[1] and Arun K Sood[1,2]
[1]International Cyber Center and Department of Computer Science
George Mason University,Fairfax, Virginia
[2]SCIT Labs Inc, Virginia
{dpham6@gmail.com,asood@gmu.edu}

***Abstract*— Modern IT systems have evolved into complex distributed systems that support thousands of users, with each user requiring access to several applications. Single sign on (SSO) provides a convenient facility for managing user authentication such that a user only logs into a system once in order to gain access to many protected applications. For this reason it becomes vitally important to secure the SSO server. If an SSO server is compromised, it could potentially put many applications at risk at the same time. The current Intrusion Detection and Prevention systems have proven to be inadequate because the "bad guys" are always one step ahead. In this paper we present a new and innovative approach to SSO server security called "Self-Cleansing Intrusion Tolerance SSO" (SCIT). SCIT shifts the focus from detection and prevention to containing losses, by reducing the exposure time of the servers. Specifically, we present the results of an evaluation of the performance of a SCIT-ized SSO server. In this way we increase the dependability of the server and provide a new way to balance the trade-off between security and availability. We will show that SCIT provides increased security with little degradation in overall response time of the system.**

Keywords, SCIT; exposure time; virtualization; vmware; persistence; pro-active; single sign on; response time; central authentication service

I. INTRODUCTION

Today's enterprise information systems support thousands of users and provide a multitude of services that span large IT infrastructures. With so many users and systems in place there is a need for controlled access to different infrastructures and a fundamental level of security to prevent unwanted access from unauthorized intruders. Single Sign On is a common access control that allows users to authenticate and be granted access to many applications, thus limiting the number of times a user has to log on. A single sign on system manages the users' credentials and automatically synchronizes the user information across all independent systems that utilize the access control. In addition to user management SSO utilizes Secure Socket Layer (SSL) protocol to encrypt username and password for communication over HTTP. This mechanism helps to ensure secure data delivery from client to server because only the trusted parties have the key to decrypt the data.

Like other web applications, SSO is exposed to attacks from intruders determined to steal confidential data. Managing user login information is an integral part of SSO. As a result it becomes the focus for intruders who need just one successful breach to compromise the system, gain access to multiple applications and steal data. Current intrusion prevention or detection approaches require prior knowledge of all potential attacks and the software vulnerabilities. These approaches are effective against attacks after-the-fact but are limited in preventing future attacks.

Self Cleansing Intrusion Tolerance (SCIT) proactively manages risk to mitigate damage from undetected attacks based on the assumption that the threat is always prevalent. SCIT does not focus on preventing attacks rather the emphasis is on limiting the time for which an intruder has access to a system. SCIT approach is recovery driven and belongs in the class of proactive recovery [14] techniques.

SCIT servers exploit virtualization technology. SCIT virtual servers can be rapidly cloned, yielding a cost effective solution. SCIT controller manages the servers such that service is always available, and regularly restores servers to clean states. If a server clone is under attack, the window of opportunity for the attacker is limited due to the short exposure window. Thus, an intruder is forced to repeat his attack sequence. Increased frequency of attacks, in turn, increases the likelihood of detection by the system.

SCIT is based on the selection of a metric, exposure time, to determine the level of security. Exposure time defines the amount of time a server is connected to the Internet and vulnerable to intruders. A low exposure time translates to increased security because the attacker has a smaller time window to launch attacks and exploit the system vulnerabilities. On the other hand, a higher exposure time leads to less security but yields higher availability and performance.

In our experimental work we apply SCIT to a SSO system called Central Authentication System (CAS) developed at Yale University. Our previous work also evaluated e-commerce web servers [1] and SCIT; however, we further our research by demonstrating the effectiveness of SCIT on a critical system application, such as SSO, and address security concerns pertaining to the system. Unlike standard web applications that provide basic services, SSO adds a layer of security that dictates the system must have high availability—its failure can deny access to all systems unified under its domain. We evaluate the performance of the SCIT Central Authentication Service SSO utilizing a response time metric. We hypothesize that a lower exposure time yields better security, but at the cost of lower performance. Our experiments focus on measuring the response time variation in response to changing exposure times. Since SCIT is best suited for short duration transactions, we consider exposure time varying between 2

and 4 minutes. We show that there is minimal degradation in performance for 4 minutes.

The remainder of the paper is divided into 7 sections. In Section II, we describe Central Authentication Service (CAS). We describe the SCIT CAS design in Section III. In Section IV, we describe the security enhancements that SCIT provides. Section V discusses our test methodology, and the test results are analyzed in Section VI. Section VII discusses related and future work. Finally, the conclusions are in Section VIII.

II. CENTRAL AUTHENTICATION SERVICE

The Central Authentication Service (CAS), originally developed by Yale University, is an open source solution that provides a way for applications to authenticate users [12]. CAS is based upon the server-client paradigm, where the server is the authentication server and the client is the user web browser that is requesting authentication.

A. CAS Sequence and process flow

CAS consists of three components: the client web browser, the web application that is protected, and the authentication server.

The flow process, depicted in Figure 1, begins with the client browser requesting access to the application. Before the client can access the application it must authenticate with CAS. After a successful authentication the client is redirected to the protected application(s). Below is a step-by-step description of the protocol.

1. On initiation of the session, the client web browser accesses the web portal and attempts to launch the application at http://portal.scit.com/application.
2. Over a secure connection, the application redirects the client to the CAS server login page. The client is prompted for username and password.
3. Upon successful authentication, the CAS server generates an in-memory cookie called a "Ticket Granting Cookie" (TGC) and sends this back to the client. This cookie identifies the client as the one who logged in successfully and expires automatically when the browser closes or when the session time limit is exceeded.
4. The client attempts to access the application with the ticket.
5. The application attempts to validate the client with the CAS by opening a HTTPS connection and passing the ticket information along with the name of the application for validation. The CAS checks its internal database for a match between the supplied ticket and the requested service. If a match is successful, the CAS returns the username to the application.
6. The client is granted access to the application and any other application under the CAS domain. Access is valid until the user logs off or the browser is closed at which time the ticket expires.

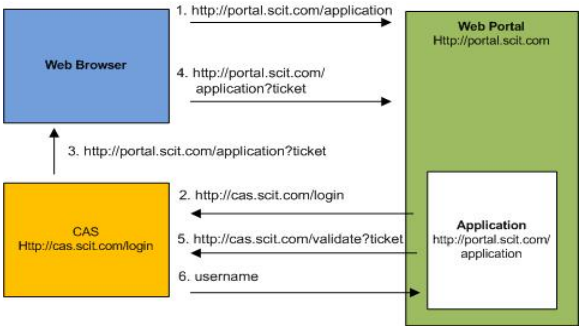

Figure 1. CAS Authentication

III. SCIT CENTRAL AUTHENTICATION SERVICE

SCIT CAS did not require any change to CAS code. SCIT leverages virtualization technology to rotate pristine virtual servers and applications at regular intervals. There are three components that comprise SCIT and each has to be implemented in order to SCIT-ize CAS:

- Virtualization layer using VMWare –virtualization reduces cost of hardware. Our experiments use VMWare, but SCIT is independent of the virtualization technology.
- Persistent memory – persistent session information must be stored and disseminated.
- SCIT controller – software component that manages the rotation and exposure time of virtual machines.

We discuss the integration of each component and how they are applied to the CAS in the following sections.

A. SCIT Controller

The SCIT controller is a central component of SCIT that controls the rotation and exposure time of the virtual machines (VM).

During one cycle of rotation each VM is in one of the following states:

- Active: virtual machine is online and accepts/processes incoming requests.
- Grace Period: virtual machine processes any pending requests from the active state, but does not accept any new requests.
- Inactive: virtual machine is offline and ready to be cleansed, defined as the process of deleting the virtual machine.
- Live Spare: virtual machine has been restored to a pristine state and is ready to become active. The server is off-line.

In each rotation cycle only one VM is online. The Load Balancer will at any given time point to the VM that is in the active state. The transition between states is shown in Figure 2.

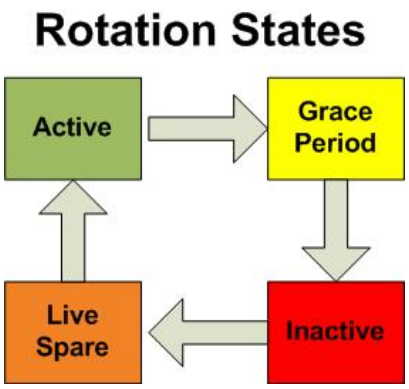

Figure 2. SCIT States

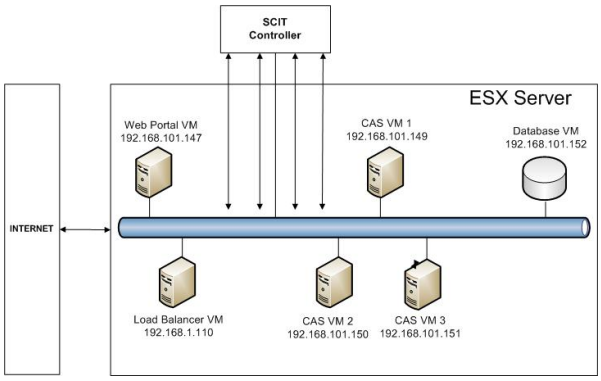

Figure 3. Test Environment Set-up

B. *CAS Virtualization*

We utilize VMware ESX, an enterprise-level product created by VMware Inc., to create an arbitrary number of virtual machines (VM) installed with the Slackware Linux operating system. Our setup consisted of three VMs. CAS is a portable web application that can be installed in any Java based web container. For our experiment we installed CAS under the Tomcat 5.5.12 web container on the VMs. Each VM had a 92 MB memory footprint. In order to manage client requests, an Apache Load Balancer installed on the CentOS operating system functions as a proxy to each VM.

C. *Persisting CAS session information*

When a CAS grants a TGC to an authorized client, the ticket is maintained in memory for the duration of the session. During a rotation cycle, if this information is not persisted, a client request will be forced to re-authorize with CAS.

Ticket information is persisted by implementing a storage mechanism for the ticket and the data is replicated by means of a network transport. In addition to storing the ticket information in-memory, we utilize JBossCacheTicketRegistry, a custom Java library that can be quickly configured to work with the CAS server [3]. Tickets are automatically deleted, stored and updated in an abstract registry within memory. The registry can be shared across the network to peer CAS servers. Multiple tomcats can be clustered to communicate via multicast such that peer servers are immediately aware of new and expired servers. Our experiment consisted of a cluster of three CAS servers that automatically replicated, via multicast packets, the ticket registry under the following conditions:

- A new peer joining the Tomcat cluster will receive a full registry update from existing peers.
- An incremental change to the ticket registry will initiate a replication to all peers.

D. *CAS Network Configuration*

Figure 3 shows the development environment used in the experiment. VMware ESX Server allows the management of all the virtual machines within the network.

IV. Additional Protection For SSO

All SCIT servers increase the system security. The premise of the SCIT approach is that by reducing the exposure time implies that the intruder has to spend additional effort to do damage. This concept is illustrated by the loss curve below. As the Intruder Residence Time increases, the losses increase.

To illustrate the additional protection provided by SCIT SSO we consider two scenarios:

- *Malware Deletion:* SCIT does not prevent the installation of malware on the exposed virtual server, but since the virtual server is restored to a pristine state every cycle, the malware is in effect deleted every rotation cycle. We emphasize that malware detection is difficult, and SCIT does the deletion without detection.
- *Reduced Data Ex-filtration:* A major weakness of the SSO approach is that the enterprise username – password information is accessible from one server. Thus strategies that reduce the data ex-filtration potential are very useful. SCIT SSO is most effective when it is used in conjunction with Intrusion Detection System / Intrusion Prevention System. These systems reduce the rate at which data can be stolen. The IDS/IPS ensures that the transfer rates are bounded – a rapid increase of the data flow on a connection is easily flagged and terminated. The SCIT operations, interrupts the data flow every rotation cycle, thus restricting the amount of data stolen. This in turn, forces the intruder to make repeated attempts, thus making the intrusion more easily detectable.

The dynamic nature of SCIT servers enables further enhancement of the system security. SCIT approach can incorporate diversity to further confuse the attacker. For example, on each rotation, it is possible to change the face of the server installing a different operating system, or using a different version of the application, or randomizing address space layout [11]. This approach also makes it more difficult for one virtual server to infect another virtual server on the host network.

Another point of vulnerability in SCIT SSO is malicious alteration of the session information that is shared among all

the virtual servers. This is also a weakness in existing SSO solutions. The SCIT solution does not provide any additional capability for the intruder to alter the session data. The constant rotation potentially reduces the scope of the alteration. We note that the session information is relatively small and well structured. This facilitates the validation of the data as it transfers from one server to the other.

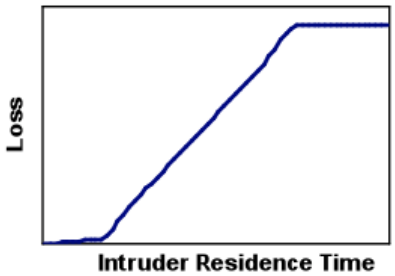

Figure 4. Loss Curve

V. TEST METHODOLOGY

We measured the effectiveness of SCIT by evaluating the impact on the latency of service. Open System Testing Architecture (OPENSTA), an open source load generation and testing tool, was utilized to generate workloads against the CAS servers and analyze the performance impact. The program simulated large volumes of users performing multi-step actions that consisted of authenticating with SSO in order to access a simple static web server.

In each of our test sets, we adhere to the following criteria:

- The workload is measured in terms of the number of users per minute (U). Three cases of 20, 40, and 80 users per minute were simulated—each case modeled a small enterprise system with varying degree of users.
- To minimize the impact of random behavior we conducted three test runs for each U. The duration (D) of each test is three times the exposure time (E), D = 3 * E. This choice ensures that each virtual server is tested at least once. The workload is generated in batches – each batch of N user requests is released every 10 seconds. The total users per minute are U = N * 6, and the total number of requests in a scenario (T) is given by T = U * D.
- Three different exposure times of 2, 3, and 4 minutes were tested. We also tested a single CAS server with No Rotation (NR) to provide the baseline results for comparison.
- A total of 36 test samples were performed for SCIT SSO.
- Each user session consists of a series of requests and responses from the client to the CAS server. Between each request we model an average "think" time of four seconds for a total of eight seconds. Think time is the average time required for the actions such as the user entering their username and password. The series of requests and responses are broken down into the following steps:
  1. User browses to the web portal.
  2. User login and TGC validation is broken into the following sequence:
     a. Launching the protected application
     b. Enter username and password
     c. Receiving a TGC
     d. Validating the TGC by launching a second protected application.

VI. PERFORMANCE RESULTS

There is a cost associated with increased security. For a SCIT SSO system the performance degradation, can be attributed to higher system overhead requiring additional computing cycles. We assert that utilizing SCIT with SSO will produce slight degradation, but the benefit is higher level of security of user identity information. Longer exposure time yields performance at a level equal to a system that does not utilize SCIT.

Exposure Time	Users per Minute	Avg. Response Time (secs)	STD Dev.
2 mins	20	11.63	0.08
2 mins	40	12.39	0.07
2 mins	80	12.94	0.09
3 mins	20	11.04	0.07
3 mins	40	11.34	0.24
3 mins	80	11.68	0.03
4 mins	20	9.62	0.02
4 mins	40	9.88	0.48
4 mins	80	9.83	0.04
NR	20	9.05	0.02
NR	40	9.29	0.04
NR	80	9.41	0.02

TABLE I. AVERAGE RESPONSE TIME SUMMARY

A total of 36 test samples were performed for SCIT SSO. For the SSO baseline we use the No Rotation (NR) case, and test SCIT SSO with three values of exposure time: 2 minutes, 3 minutes, and 4 minutes. Varying user workloads were injected into the system and the average response time of each request was recorded to measure the performance. A summary of the results is shown in Table 1. The average response time is computed over all 3 runs, and the low standard deviations confirm consistency. It is worth noting that the results factor in the average "think" time fixed at eight seconds. Under a live production scenario, some users may only require one to two seconds to login while others require more time.

Figure 4 summarizes the average response time for specific exposure times. For each exposure time we tested 3 workloads – 20, 40 and 80 users per minute. Each workload is represented by a bar and the bars for a exposure time are clustered. The data shows an inverse relationship between response time and exposure time. – response time reduces as

exposure time increases, and is minimum for the No Rotation case. For example, for 80 users per minute case, the response time for 2 and 4 minute exposure is higher than No Rotation by 38% and 4% respectively. The security protection for 4 minute exposure is less than the protection for less exposure time, but much better that the current static systems. In summary, lower exposure time requires more rotations cycles per hour, and this requires more CPU cycles which, in turn, degrades server performance. The degradation was more noticeable with extremely low exposure time, for example one minute, and results varied from very high average response times to the severe case where requests simply timed out due to lack of response from the server. Overall we conclude, SCIT's impact on SSO performance is low, and the increased security justifies the slight degradation in performance.

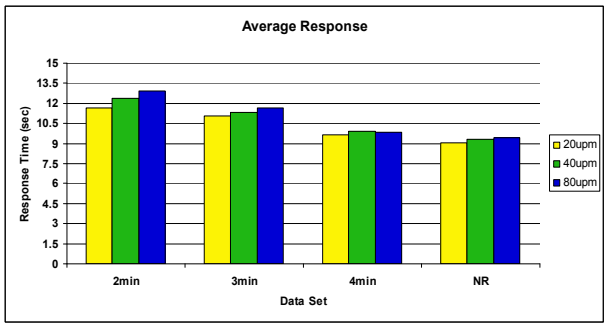

Figure 5. Average Response

VII. Related And Future Work

Political and economic motivation has lead to increased attacks and increase in successful intrusions. Typical way to mitigate the risk has been to adopt a risk prevention posture. However, over the last decade additional emphasis has been placed on approaches that attempt to contain the losses. OASIS (Organically Assured and Survivable Information Systems) [6] was a DARPA sponsored research effort to develop architecture for survivability and intrusion tolerance. It appears that this was the first major effort in this area that paved the way for new architectures. In this section we mention just a few efforts:

- MAFTIA: Malicious- and Accidental-Fault Tolerance for Internet Applications is a European project that aims to build conceptual models, mechanism and protocols for achieving tolerance [10].
- BBN and UIUC designed a complex architecture called DPASA. Several studies of DPASA have been reported. [9].
- SITAR: Scalable Intrusion-Tolerant Architecture for Distributed Services was developed at Duke University aims to use redundancy to reconfigure systems to increase security [7]. SITAR is the natural extension of the Software Rejuvenation research undertaken by this group [8]. We note that, this is one of the few efforts at using quantitative techniques to analyze this problem.

The SCIT concepts have been developed over the last 6 years. The SCIT web site [13] provides pointers to the publications on this architecture. Recently, we have built prototypes and demonstrated recovery from malicious attacks [4]. In another paper, the performance of SCIT Web servers has been analyzed [1].

In the future, we want to apply SCIT to other servers (Email, DNS, etc) as well as explore methods to reduce exposure time. We are also investigating ways to integrate SCIT and SSO with the SAML protocol [15] to standardize the communication of authentication information.

VIII. Conclusion

SSO, as a method of access control that facilitates the authentication process for users, is the gateway to accessing confidential user information. Attackers with this information hold the key to stealing personal data and inflict massive damage upon systems. In December 2009, Google announced that they were the focus of attacks originating in China [16]. Further investigation into the attacks revealed that the attackers were not only trying to break into Google mail accounts of human activists rather, a single sign on system called Gaia was also the target. Gaia contains passwords of millions of Google accounts and services (e.g., Google Doc, Gmail) available to account holders.

Our research attempted to augment and provide additional protection to SSO by integrating SCIT, a proactive recovery system. We measured performance by analyzing the latency of service as a function of exposure time and found that exposure time increases security however at the cost of performance. The degradation in performance, noted by higher latency, was more noticeable with lower exposure times due to the increase CPU cycles required to rotate the servers more frequently. Fortunately, by utilizing more powerful servers and hardware, this cost can be mitigated.

In the case of SSO, the largest threat is data theft. Although SCIT does not prevent an attack of this type, it does however limit the time an attack has access to data. We are addressing this limitation in our ongoing research and exploring prevention techniques that inhibit attackers. For example, Automatic Signature Generation is an area that SCIT can utilize to protect against malware and rootkits that allow attackers to take control of a system.

IX. Acknowledgment

This research was partially supported by a contract from Lockheed Martin and a contract from Virginia Center for Innovative Technologies.

X. References

[1] Anantha Bangalore and Arun Sood, "Securing Web Servers Using Self Cleansing Intrusion Tolerence (SCIT)", DEPEND 2009.

[2] Yih Huang and Arun Sood, "Self-Cleansing Systems for Intrusion Containment", Proceedings of Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN), New York City, June 2002.

[3] JBossCacheTicketRegistry API Documentation, http://developer.jasig.org/projects/cas/cas-server-integration-jboss/cas-server/cas-server-integration-jboss/apidocs/org/jasig/cas/ticket/registry/JBossCacheTicketRegistry.html, June 2008

[4] http://www.youtube.com/watch?v=gIN6JWInuv8

[5] Yih Huang, David Arsenault, and Arun Sood, "SCIT-DNS: Critical Infrastructure Protection through Secure DNS Server Dynamic Updates", Journal of High Speed Networking, vol 15 No 1, pp 5 - 19, 2006.

[6] Foundations of Intrusion Tolerant Systems, Edited by Jaynarayan H. Lala, DARPA Organically Assured and Survivable Information Systems (OASIS), IEEE Computer Press, 2003.

[7] Feiyi Wang Fengmin, Frank Jou, Fengmin Gong, Ramouli Sargor, Katerina Goseva-Popstojanova, and Kishor Trivedi, "SITAR: a scalable intrusion-tolerant architecture for distributed services", Foundations of Intrusion Tolerant Systems, Page(s):359 – 367, 2003.

[8] Kishor Trivedi, Gianfranco Ciardo, Balakrishnan Dasarathy, Michael Grottke, Andy Rindos, Rivalino Matias, and Bart Vashaw, "Achieving and Assuring High Availability", Proc. 13th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems/22nd IEEE International Parallel & Distributed Processing Symposium, April 2008.

[9] Michael Atighetchi, Paul Rubet, Partha Pal, Jennifer Chong, and Lyle Sudin, "Networking Aspects in the DPASA Survivability Architecture: An Experience Report", Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications, 2005.

[10] Ian Welch, John Warne, Peter Ryan, and Robert Stroud, "Architectural Analysis of MAFTIA Intrusion Tolerance Capabilities". Tech Rep CS-TR-788, Univ of Newcastle upon Tyne. Feb 2003.

[11] Address Space Layout Randomization, http://en.wikipedia.org/wiki/Address_space_layout_randomization, January 2010

[12] JASIG CAS, http://www.jasig.org/cas, June 2008

[13] Self Cleansing Intrusion Tolerance, http://cs.gmu.edu/~asood/scit, April 2010

[14] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo, "Resilient Intrusion Tolerance through Proactive and Reactive Recovery", LASIGE, Faculdade de Ciéncias da Universidade de Lisboa – Portugal

[15] Security Assertion Markup Language, OASIS Security Services, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, December 2009

[16] Google Single Sign On system exposed during December Attacks, http://www.tweaktown.com/news/14956/google_single_sign_on_system_exposed_during_december_attacks/index.html, April 2010