http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (3): 385-390, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.385.390

Virtualization-based Recovery Approach for Intrusion Tolerance

Jian-Hua Huang, Jun Huang, Rong Li and Xiao-Ming Li
School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

Abstract: It is well known that increasing redundancy in a system generally improves the availability and dependability of the system. In this study, we present a Virtualization-based Recovery for Intrusion Tolerance (VRIT) architecture that strengthens cluster's availability and dependability through periodic and event-driven recovery. By periodically reverting each virtual server to a pristine state, the VRIT cluster can limit the online exposure time of all servers, ensuring that even undetected attacks will be thwarted or at least be limited. Anomaly detection engines are installed in every virtual server to enable event-driven recovery within a fixed recovery cycle. Accumulated intrusion reports will prompt the compromised servers to be reverted earlier. A control algorithm is designed to manage both security and service availability. Experimental results demonstrate good performance of the algorithm.

Key words: Virtualization, intrusion tolerance, recovery, virtual servers

INTRODUCTION

In spite of all efforts to increase system security in the past decade, security intrusions are still commonplace events. Traditional intrusion prevention or detection methods require prior knowledge of all potential attack modalities and software vulnerabilities. These methods are good at fighting yesterday's wars, but are totally ineffective against current and future threats. The increasing sophistication and incessant morphing of cyber attacks lend importance to intrusion tolerance, a critical system must thwart or at least limit the damage caused by unknown and undetected attacks.

Intrusion tolerance systems (Cachin and Poritz, 2002) are able to tolerate a limited amount of faulty nodes. A system with n replicas usually can tolerate up to f<n/3 replicas that fail in arbitrary, malicious ways. However, over a long lifetime, it is most likely that compromised replicas exceed this limit. Regardless of whether an intrusion is detected or not, if replicas are periodically recovered from potential penetrations to a known clean state and the recovery is performed sufficiently often, then an attacker has not enough time to compromise replicas. In this study, we propose a event-driven recovery approach to accelerate the recovery when intrusions are detected, so system security is strengthened to a higher degree.

RELATED WORK

Intrusion tolerance is an important factor in building applications that withstands security attacks. There are several such architectures for securing Web servers. Most of them are distributed architectures based on replication. SITAR (Wang *et al*., 2001) is a classic intrusion tolerance architecture. It focuses on distributed services built from COTS components as the protection target. MAFTIA (Nguyen and Sood, 2011) is a solution that helps survivability. It builds layers of trusted components and middleware subsystems from untrusted components such as hosts and networks.

A problem with classic intrusion-tolerant solutions based on Byzantine fault-tolerant replication algorithms is the assumption that the system operates correctly only if at most f out of n of its replicas are compromised. The problem here is that given a sufficient amount of time, a malicious and intelligent attacker can find ways to compromise more than f replicas and collapse the whole system. Some works show that a technique called proactive recovery (Castro and Liskov, 2002) can solve this problem. The idea is simple: a replica is periodically recovered to a known clean state to remove the effects of malicious attacks. Proactive recovery is a promising approach for building intrusion tolerance systems that tolerate an arbitrary number of faults during system lifetime (Fig. 1).

Virtualization is a technology that can simultaneously execute multiple operating system instances in isolated environments on a physical machine. Reiser and Kapitza, (2007) proposed that the hypervisor is used to initialize a new replica in parallel to normal system execution and thus minimize the time in which a proactive reboot interferes with system operation. As a result, the system maintains an equivalent degree of system availability

Corresponding Author: Jian-Hua Huang, School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

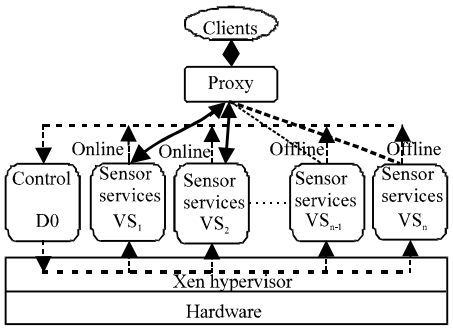

Fig. 1: Virtualization-based recovery for intrusion tolerance

without requiring more replicas than a traditional replication system. Resilient Web Service (RWS) (Huang *et al*., 2010) uses virtualization with feedback control to achieve fault tolerance, intrusion analysis and automatic recovery. It is transparent to the applications it supports and accesses to application source code are not needed to integrate into a RWS-based system. SCIT (Huang *et al*., 2006) is a self-cleansing intrusion tolerance architecture. Its aim is to provide high degree of security for Web servers by reducing server's exposure time. Therefore, an attacker will not get much time to explore the vulnerabilities of a target server to corrupt it. SCIT does not include intrusion detection mechanism. A pure SCIT-based system is not hardened in the face of attacks that can impact data confidentiality. For shorter windows of exposure, the SCIT system will experience higher overhead and potential user disruption from frequent recycling.

A more robust solution should borrow from the above discussed approaches. Our aims are to find compatible combinations of off-the-shelf, mature technologies to create resilient systems. An event-driven recovery approach based on virtualization will be discussed in this paper. Anomaly detection engines are installed in every virtual machine to enable event-driven reversion within a fixed reversion cycle. This allows, to bring an online virtual machine, offline at the time of intrusions to be detected.

VIRTUALIZATION-BASED RECOVERY ARCHITECTURE

Since Web servers are open to public access, they can be subjected to attack attempts by hackers. Flaws in network services are inevitable. When such flaws are triggered or exploited, corresponding services are compromised or even disrupted entirely. To address the issue, we present a Virtualization-based Recovery for Intrusion Tolerance (VRIT) cluster, shown in Fig 1. In the VRIT cluster server reversion is scheduled on a periodic and event-driven basis to restore servers to a pristine state.

VRIT uses Xen Cloud Platform (XCP) as the virtualization foundation. A XCP environment consists of several items: Xen hypervisor, domain 0 and guest domain U. Virtual machines in XCP are called domains. The Xen hypervisor is the abstraction layer of software that sits directly on the hardware below any operating systems. It is responsible for memory partitioning and CPU scheduling of the various virtual machines. Domain 0 has special rights to access physical I/O resources and manage the other domains. XCP supports virtual machine snapshot. The feature is used to revert virtual machines to a pristine state.

A VRIT cluster is a pool of virtualized server (VS) replicas that cooperatively provide a set of predefined services. Each server in the cluster is dynamically brought online or taken offline. At any time, some virtual servers are online serving requests and others are taken offline for reversion to clean state. Each client request is directed to active role servers by the Proxy in VRIT.

The domain 0 controls the creation and execution of the guest domains. VRIT uses the hypervisor to provide replication logic, while actual service replicas are executed in isolated guest domains. A Control Center (CC) is installed in the privileged domain for high security. the CC is used to manage server recovery and role assignments, including both periodic rotation and event-triggered recovery. Periodic rotation is to bring online servers offline for reversion to a pristine state. Such a state includes system configuration files, system binaries, critical utilities, service binaries and so on. Unlike system reboots which remove only memory corruption, VS recovery removes both memory and file system corruptions, including malwares, backdoors and Trojan horses. In normal conditions, VRIT executes periodic rotation recovery through an exposure timer set by the CC. Once a compromised server is detected, the CC will bring it offline and switch to an accelerating rotation mode for cleansing. At the same time, a spare clean server is brought online to accept requests from clients. The offline server is reinstalled operating system image and critical services. After reversion to clean state, the offline server is added to a ready server queue for online.

By periodically restoring each VS to a pristine state, VRIT can limit the online exposure of all servers, ensuring that even undetected, successful attacks will be thwarted. VRIT imposes an upper limit on VS's exposure time. When the exposure time reaches the limit, the VS is reverted even without intrusion reports which will prompt the VS

to be reverted sooner. Intrusion and anomaly sensors in each VS report observable events to the CC, such as system calls and attack alarms. The CC uses a two-level control mechanism for triggering different actions. First, the CC tries to take local actions with negligible system-wide impacts in a VS to resolve corruptions with minimal overhead. These local actions include restarting unavailable services and killing suspicious processes. Second, it is taken a weighted reversion schedule, where accumulated anomaly or intrusion reports from a given VS will prompt the CC to revert the VS earlier, while good reports will weight it back to its fixed reversion mode.

The VRIT ALGORITHM

Description of algorithm parameters: In this study, the VRIT cluster is assumed to support M service roles, denoted by R_1 to R_M. Suppose that each server is assigned one role, thus VRIT needs to use M+1 and more servers to provide M services. Our case is a 2DNS-2Web cluster. This cluster supports four roles: a master DNS server(P), a master Web server (W), a secondary Web server(W) and a secondary DNS server (S).

N is the total number of servers in the cluster, the cluster can do rotation operation when N = M+1, a rotation cycle includes to rotate an online server offline, reboot and cleanse the offline server to restore to its pristine state and bring a clean server online. Obviously, the number of clean servers is a key factor to affect the efficiency of server rotation. Suppose that when there is one clean server, the rotation time is T_c. When another clean server is added to the cluster, the rotation time will be decreased to $T_c/2$. If there are n clean servers, the time is T_c/n. As is well-known, the shorter the online time of the servers, the more difficult successful attacks. So it is good to protect the system from some novel and unknown attacks.

Rotation pattern Pa: A role R_i swap, 1 = I = M, rotates the present server running as role R_i offline for cleansing and takes a spare server online as R_i. In the case of 2DNS-2Web cluster, There are four swaps, including P swap, W' swap, W' swap and S swap. Assume that at a given time, there are 5 servers in state <P, W, W', S, C>, the identifiers of servers from 1 to 4 denote role P, W, W', S, the fifth server is in a spare state, denoted by C. If the first rotation mode is P swap, then the cluster will enter state <C,W, W', S, P>. If the next rotation is W' swap, subsequently the cluster is in state <W', W, C, S, P>.

The rotation pattern is used to decide the rotation sequence. For the 2DNS-2Web cluster, the pattern is P->W->W'->S, it means that the system rotates with this sequence. In the pattern P->W->P->W'->S, compared with the other roles, role P swaps for two times. In practical applications, the system can adopt different rotation patterns to adjust the rotation frequency according to relevant security level for different roles.

Table 1: Index value and rotation behavior

No. of servers	Active roles	M(t)	Rotation
1	Undefined	10	No
2	P and W	20	No
3	P and W	20	Yes
4	P, W and W'	25	Yes
≥5	P, W, W' and S	30	Yes

The role index value configuration: In many applications, some service roles are more important than others. Assume that the relative importance is denoted by ω and each role R is associated with an integer index value ω(R). The greater the ω value, the more important the role in the cluster. Table 1 gives the case of 2DNS-2Web cluster. Assume that due to the server failures, the cluster is only left with two normal servers which must serve as master DNS server and master Web server. In order to reflect the importance of the two master servers, we assign the index value ω for 4 roles: ω(P) = ω(W) = 10, ω(W) = ω(S) = 5. Define M(t) as the VRIT cluster index value, M(t) is the sum of index value ω of all active roles at time t. It is required that M(t) must be equal to or greater than a predefined minimum cluster index value M_{min}. If sets M_{min} = 20 in the 2DNS-2Web cluster, the cluster must always satisfy with the minimum service requirement by keeping P and W role servers online.

Suppose that the value ω is decreased sequentially from R_1 to R_M and there are k<M enabled servers, then the task of meeting M_{min} is to assign the former k-1 servers as R_1 to R_{k-1}. Whether the last role R_k is active depend on whether the former k-1 roles satisfy the M_{min} requirement. If the requirement is satisfied, role R_k will not be activated and an alternate server will be reserved for triggering rotation. Otherwise, the last server will be activated as role R_k to meet M_{min} and provide the minimum service guarantee as far as possible.

Control algorithm description: The critical component of the VRIT cluster is the CC, which is responsible for all transection, including server rotation, server recovery, role assignment and etc. The CC algorithm uses M(t) to track the cluster index value, variable k records the number of active server roles and variable C_n is the number of cleaned servers. Rotation time is T. Queue OID is used to record cleaned server's ID and the spare server queue DS. A Short Session Storage (SSS) that adopts the high speed memory technology based on the NAM, which is built and ruined in an intranet server, is used to store each online server's session data which has not finished yet. The clean server to be taken online will read the session data from the SSS and go on with the session.

At initial phase, k = 0, M(t) = 0 are set. The algorithm scans and reads the total number N of servers in the cluster, C_n = N, T = T_c/(N-M), then boot servers online to be relevant roles. When C_n = 0, the CC is always in the state of seeking enabled servers. When C_n>0 and there is a clean server C, the CC queries the index values predefined server roles ω(R) to find out whether the roles meet the minimum requirement and then boot C online as role R_i in sequence. At the same time, M(t) = M(t)+ω(R_i), k = k+1, C_n = C_n-1, save the ID of just online server to the OID and delete the relevant ID from DS. If the number of servers just meets the minimum requirement, it means at this moment C_n = 0 and the system can provide services, but without rotation. If C_n = 1, the value of M(t) will be checked. When M(t) = M_{min}, the next clean server is not activated firstly, it will be used as a rotation server. If C_n-1-M>0, the CC can activate the clean servers as other secondary role servers. When all role servers are online, the rest of clean servers will enter the quene DS and wait for rotating online.

At normal proactive rotation phase, the VRIT cluster runs according to the predefined T and Pa. Before rotation, at least one clean server in the cluster is waiting to swap with online servers. When there is only one C in the cluster, the rotation interval is T_{sc}. Before an online server is offline, it needs to check whether the value of C_n is greater than zero. When C_n>0, the server can be rotated offline and enters clean state. At the same time, C_n = C_n-1, boots another C_j online as role R_j, starts a timer, replaces the ID of the former offline server with the latter online server in the OID and deletes the ID of the latter online server from the DS. In addition, after the offline server finishes the clean work, the CC sets C_n = C_n+1 and adds the ID of this clean server to the DS. Before the next rotation, if the timer is timeout, it indicates that this server has failed and needs to be taken offline by force. In this process, when an anomaly is detected, whatever it is the required exception or responsive exception, if the number of the anomalies detected equals the predefined value, the detection module will directly send exceptional signal to the CC, then the CC firstly queries what services the present role server provides and adjusts the rotation pattern. It is better to defense a hacker from attacking this kind of service by prior speeding up the rotation of this role server for more times. In our cluster, the general rotation pattern is P->W->W'->S, when a hacker attacks the DNS server, the rotation pattern becomes P->W->P->W'->S, even P->W->P->W'->P->S. After a long time, if there are not any abnormal records about the former type of attacks in the audit log and the CC will adjust the rotation pattern back to the predefined mode again.

The servers themselves have some faults. We consider the worst situation. When a role server has a fault, it needs to be rotated offline and a clean server is taken online, C_n = C_n-1. When the bad offline server is not recovered in time and there is not any clean server in the DS at the moment, the CC will rotate some secondary servers offline to ensure the rotation continuously, M(t) = M(t)-ω(R_i). When M(t) = M_{min} and there is no clean server in the cluster, the system provides the minimum basic services without rotation. Actually the system has gone back to the initial phase.

The SIMULATED EXPERIMENT AND PERFORMANCE ANALYSIS

The simulated experiment investigated the situation that the servers provide the services by working with the algorithm based on the proactive and reactive rotation under the known and unknown attacks and the exposure time and service effection were analyzed. In order to achieve the contrastive data, the rotation time and pattern would be adjusted. The experimental circumstance is a 2CPU Xeon 2.4 GHz Dell Server, with 16 GB memory. The system platform is XCP based on XenServer 5.6. Eight virtual Servers (VSs) are used to build the VRIT cluster. 2 VSs are assigned as DNS servers, 2 other VSs as Web servers, the rest as the spare servers for swap and rotation. We conducted the experiment according to the performance test method proposed by Huang *et al*. (2006). In the experiment we choose the standard clean time T_{sc} = (n×T_{dns}+m×T_{web})/(m+n). When m = n = 2, obviously T_{sc} = 255 sec. In the redundancy respect, we compare S = 1 with S = ½ (it indicates that the count of spare servers is twice as the role servers). In the rotation pattern respect, three kinds of pattern are compared. The system mixed the two factors together during the experiment.

First, consider a minimum VRIT cluster which includes one online server and one spare server, the parameter C_n/M is used as the reference. Just like the description of the algorithm, M+C_n VSs firstly start their system by themselves in each experiment situation. After the CC boots M role servers online, it will rotate the servers according to a predefined rotation pattern. There are three typical rotation patterns: P_1 = (P->W->W'->S), P_2 = (P->W->P->W'->S), P_3 = (P->W->W'->W ->S). In the normal situation, the cluster adopts P_1. When a compromised DNS server is detected, the CC will adjust the pattern to P_2. When Web attacks are detected, the CC adjust the pattern to P_3. In this way, the experiment rotates repeatedly for 1000 times. The server swap number is 4000 times under P_1. Under pattern P_2 or P_3, the swap number is

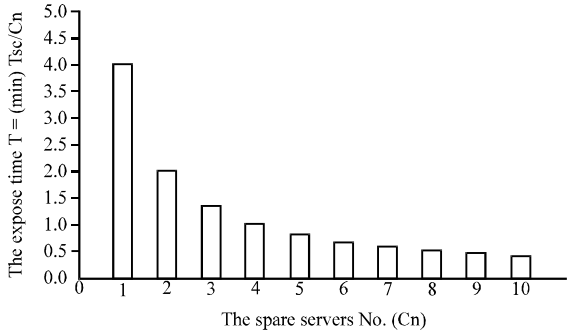

Fig. 2: Server exposure times under pattern P_1

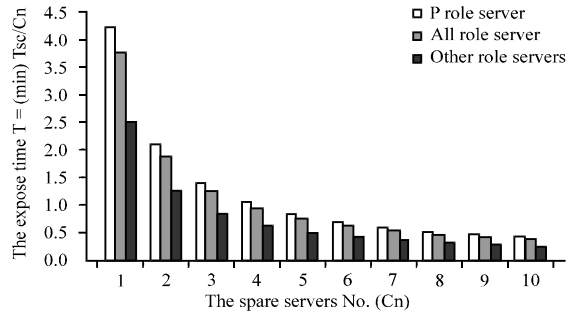

Fig. 3: Server exposure times under pattern P_2

5000 times. The average rotation interval or the exposure time is different in different S. When S = 1, the rotation interval $T = T_{sc} \times S = T_{sc}$. When S = 1/2, $T = T_{sc}/2$. The experiment rotates repeatedly for 1000 times under P_1, in the whole process, the results are steady, the system is also steady. The results are presented in the Fig. 2.

Next we use an impartial rotation pattern, it means that security priority is assigned to role P or role W. When the DNS server is attacked, if pattern P_2 is adopted and there is only one spare server in the cluster, compared with previous situation, the exposure time of role P is evidently decreased from $4T_{sc}$ to $2.52T_{sc}$. While the exposure time of the other servers is just increased a little, from $4T_{sc}$ to $4.21T_{sc}$. This impact is very little. In fact, the average exposure time of all servers is $3.75T_{sc}$. The benefit of this kind of rotation pattern is that before the rotation cycle is timeout and anomalies reach the predefined limit, the CC will rotate the possibly compromised server offline in time and take a clean server online to substitute the offline server. Therefore, the passive defense problem of only periodically rotating the servers can be solved. Fig. 3 shows the relationship between exposure window and spare server under P_2 without attacks.

Finally, we simulated some attacks and investigated the VRIT's performance and the influence for the whole services. When some anomalies are detected, the CC will adjust the rotation pattern to another pattern that is

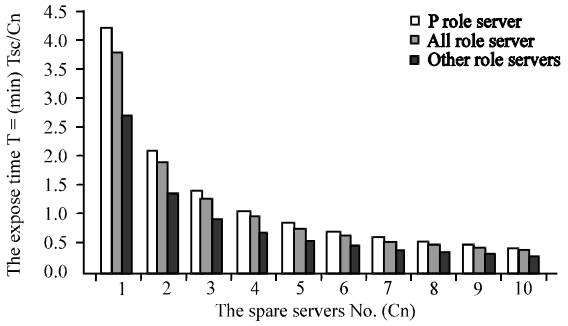

Fig. 4: Server exposure times in attacks

advantageous to accelerate the rotation rate of the DNS server or the Web server. The exposure time of role server being attacked can be decreased evidently, it means that a hacker has less time to attack the present role server each time. Therefore, the probability of successful attack becomes smaller and the novel and unknown attack will be contained greatly under this impartial pattern. At the beginning, the system runs normally under P_1. We simulate some traditional attacks, the rotation pattern will be changed from P_1 to P_2 or P_3. Like the second situation above, the average rotation time of role P is reduced from $4T_{sc}$ to $2.52T_{sc}$. At first, the system runs under P_1. After simulating some attacks, the CC will adjust the pattern to P_2. The system rotates for 50 times under this pattern, So attackers have no patience to attack this role server continuously. In the following time, the system needs to adjust the pattern back to P_1 manually. Attacking again will make the CC adjust the pattern to P_2. After simulating repeatedly for 1000 times, in this invasive circumstance, the average exposure time of P is $2.68T_{sc}$, the other role servers is $4.19T_{sc}$ and all role servers is $3.78T_{sc}$. The results in Fig. 4 indicate that the system runs steadily and its service performance is good.

Obviously, some conclusions can be achieved: the average exposed time of the prior patterns is smaller a lot than the general patterns, especially for the prior role servers and the time is decreased evidently. The exposed time of the role servers has some new changes due to the rotation pattern is adjusted in attacking. Compared with the general attack less situation, results show that the rotation and performance of the VRIT cluster with IDS is not influenced and the system runs steadily.

CONCLUSION

In this study, we have presented a virtualization-based recovery approach for intrusion tolerance. The experiment proved that this approach is feasible. It can not only accelerate the recovery when intrusions are

detected, but also provide good secure assurance for the architecture. In addition, it is easy to construct the VRIT cluster in practical application and it is convenient for administrators to maintain the system for the deeper security.

The future work is to optimize the CC, improve rotation efficiency, design a random rotation pattern to enhance the security level and study the possible security problem when transmit the data in the SSS to the new online server. In the premise of providing the normal services, it is more secure for the system to open few interfaces, so the rock-bottom service interfaces need to be studied further.

REFERENCES

Cachin, C. and J.A. Poritz, 2002. Secure intrusion-tolerant replication on the internet. Proceedings of the International Conference on Dependable Systems and Networks, June 23-26, 2002, Bethesda, USA., pp: 167-176.

Castro, M. and B. Liskov, 2002. Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20: 398-461.

Huang, Y., A.K. Ghosh, T. Bracewell and B. Mastropietro, 2010. A security evaluation of a novel resilient web serving architecture: Lessons learned through industry/academia collaboration. Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W), June 28-July 1, 2010, Chicago, pp: 188-193.

Huang, Y., D. Arsenault and A. Sood, 2006. Closing cluster attack windows through server redundancy and rotations. Proceedings of the 2nd International Workshop on Cluster Security (Cluster-Sec06), May, 2006, Singapore.

Nguyen, Q. and A. Sood, 2011. A comparison of intrusion-tolerant system architectures. Proceedings of the IEEE Security and Privacy, Volume 9, July-August, 2011, USA., pp: 24-31.

Reiser, H.P. and R. Kapitza, 2007. Hypervisor-based efficient proactive recovery. Proceedings of the 26th IEEE Symposium on Reliable Distributed Systems, October 10-12, 2007, Beijing, China..

Wang, F., F. Gong, C. Sargor, K. Goseva-Popstojanova, K. Trivedi and F. Jou, 2001. SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services. Proceedings of the 2001, IEEE Workshop on Information Assurance and Security, June 5-6, 2001, United States Military Academy, West point.