An Intrusion-Tolerant Mechanism for Intrusion Detection Systems

Liwei Kuang and Mohammad Zulkernine
School of Computing
Queen's University, Kingston
Ontario, Canada K7L 3N6
{kuang, mzulker}@cs.queensu.ca

Abstract

In accordance with the increasing importance of intrusion detection systems (IDS), users justifiably demand the trustworthiness of the IDS. However, sophisticated attackers attempt to disable the IDS before they launch a thorough attack. Therefore, to accomplish its function, an IDS should have some mechanism to guarantee uninterrupted detection service even in the face of IDS component failures due to attacks. In this paper, we propose an intrusion-tolerant mechanism for network intrusion detection systems (NIDS) that employ multiple independent components. The mechanism monitors the detection units and the hosts on which the units reside and enables the IDS to survive component failure due to intrusions. As soon as a failed IDS component is discovered, a copy of the component is installed to replace it and the detection service continues. We implement the intrusion-tolerant mechanism based on the CSI-KNN-based NIDS and evaluate the prototype in the face of component failures. The results demonstrate that the mechanism can effectively tolerate intrusions.

1 Introduction

As more and more valuable information is distributed and transferred through networks, intrusion detection has become more important than ever. With the increasing importance of Intrusion Detection Systems (IDS) in network security, the IDSs may become a primary target of attacks [1]. Experienced attackers tend to first disable the IDS in order to conceal their subsequent behaviors. Therefore, it is necessary to ensure the continuity of IDS services even under attacks.

Since successful attacks (intrusions) are a kind of fault that is launched on purpose by malicious users to subvert legitimate services, the goal of intrusion tolerance is to enable the application to survive the effects of intrusions [5]. An Intrusion-Tolerant System (ITS) is an application that is able to achieve this goal through an intrusion-tolerant practice or mechanism. Basically, ITSs such as ITDB [6] and SITAR [7] employ an extra intrusion detection component to discover intrusions. Thus, an intrusion-tolerant IDS is an intrusion detection application that has an intrusion-tolerant mechanism to withstand the consequences of attacks. However, the intrusion-tolerant IDS is somewhat special because the mechanism protects the IDS service instead of the object that the IDS monitors. For example, when an intruder crashes an intrusion-tolerant IDS, its intrusion-tolerant mechanism will undertake a recovery and restore IDS service. However, if the attacker crashes the monitored networks, the intrusion-tolerant IDS could detect the intrusions but would be unable to restore network service.

As a newly emerging field, intrusion-tolerant IDS is not addressed adequately. Some papers [2][3][4] discuss the issue of establishing mechanisms to respond to attacks. However, the mechanisms proposed by those papers are isolated from the IDS infrastructure. The effectiveness and performance of the mechanisms are not rigorously evaluated across IDSs.

In this paper, we propose an intrusion-tolerant mechanism that is built over a network intrusion detection system (NIDS) and seamlessly integrated with the NIDS framework. The NIDS employs multiple service-based classifiers. Each classifier processes a specific network service (e.g., http) and detects intrusions. Because the classifiers work independently, they can be distributed across different machines in the same network segment. Based on this infrastructure, an intrusion-tolerant mechanism is developed to guarantee the NIDS detection service even in the component failure because of attacks. Since the intrusion-tolerant mechanism is independent of the intrusion detection method, it can be applied to other NIDSs with similar

0-7695-3102-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ARES.2008.68

frameworks. However, in this paper, as an example, we use the CSI-KNN (Combined Strangeness and Isolation measure K-Nearest Neighbors) based NIDS [8, 9].

This paper is organized as follows. Section 2 presents an overview of the NIDS framework. Section 3 outlines some related work. In Section 4, we present the proposed intrusion-tolerant mechanism. We illustrate how the mechanism tolerates the intrusions on NIDS components and present the performance evaluation in Section 5. Finally, Section 6 draws conclusions and discusses future work.

2 The NIDS Framework Overview

The NIDS framework proposed for intrusion tolerance is shown in Figure 1. It employs multiple network sensors. Each sensor is installed on a computer and captures the traffic in the network segment. Although each sensor has access to all the traffic of the network, the sensor is configured to process specific network services such as http and smtp. Network traffic of the same service are fed into a specific service-based classifier. The classifier analyzes network activities, raises alerts for suspicious network behaviors, and reports them to the IDS console. A detector is a collection of classifiers that are associated with a specific sensor. The detector is regarded as a component that provides intrusion detection for specific service types.

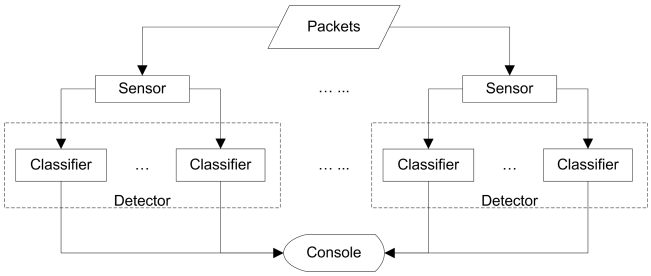

Figure 1. The NIDS framework

Over the NIDS framework, we propose an intrusion-tolerant mechanism that can tolerate the failure of detection components (classifier and detector) due to intrusions. The intrusion-tolerant mechanism takes advantage of the diversity of network data and the independence of classifiers. The mechanism monitors the detection components and restores the failed components on peer hosts.

This intrusion-tolerant mechanism is independent of the intrusion detection method and can be applied to any NIDSs with similar frameworks. The CSI-KNN-based NIDS follows the framework depicted in Figure 1. Therefore, we implemented and evaluated our intrusion-tolerant mechanism over the CSI-KNN-based NIDS.

3 Related Work

MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) [10] research project investigates a comprehensive approach for tolerating both accidental faults and malicious attacks in large-scale distributed systems. It proposes a multi-layer intrusion analysis topology based on a generic intrusion detection model. The objective of the topology is to address the core problems of intrusion tolerance in IDSs by improving the detection accuracy. In the topology, multiple sensors work in parallel and focus on different vulnerabilities. Analyzers on level one detect different kinds of malicious activities. Each analyzer on level two runs specific filtering rules [11] that automatically discard certain alarms generated by the underlying analyzers. On level three, analyzers either combine or compare the results from previous analyzers. Through aggregation, the analyzer correlates events containing information on different aspects and thus increases the detection coverage. The IDS can also detect subversion against itself by comparing the outputs of the various analyzers. Our intrusion detection topology that has only two layers possesses a relatively simple architecture. Each classifier has a similar internal architecture and make it is easy to implement. Moreover, the intrusion-tolerant mechanism protects detection components and guarantees detection services of the NIDS.

Shen *et al.* [2] propose a "Prairie Dog" system that supervises the IDS as well as itself. Basically, Prairie Dog is an IDS that monitors the behaviors of the target IDS. It consists of three types of monitoring components: the Integrity Checker (IC), the Intrusion Detection Monitors (IDMs), and the Neighborhood Watchers (NWs). The IC detects unauthorized modification and replacement of monitored files; the IDM monitors the service provided by the target IDS; and the NWs are responsible for supervising the IDMs. Prairie Dog employs redundancy on IDMs and NWs and can restore them in case of failure. It is claimed that Prairie Dog makes the supervised IDS attack-tolerant by protecting both the monitored IDS and itself. However, the tolerance mechanism is not complete as it takes no action to restore the monitored IDS even if the IDS is detected being subverted.

Siqueira and Abdelouahab [4] propose a mechanism called *System Fault Tolerant Agent (SFTA)* that can protect agent-based NIDSs. SFTA operates on the administration layer and maintains the integrity of the target IDS. It consists of three agents: the Sys-

tem Sentinel Agent (SSA), the System Fault Evaluation Agent (SFEA), and the System Replication Agent (SRA). The SSA refers to a Profile Database (PRDB) and discovers abnormal behaviors of intrusion detection agents. Once a compromised agent has been discovered, the information about the agent is submitted to SFEA, where a decision is made whether to create a new agent. The SRA executes the replication and adds or removes agents. The authors claim that SFTA can detect agent failure and ensure the continuity of the service even in the presence of accidental or malicious faults. Similar to [4], our intrusion-tolerant mechanism also uses multiple agents to detect and restore failed components. However, instead of using a profile database, we employ rules encoded in monitoring agents to identify the failure of IDS components. A process is responsible for maintaining the availability information of the IDS components and responds automatically according to security alerts raised by the monitoring agents.

4 Intrusion-Tolerant Mechanism

Figure 2 shows the anatomy of the proposed intrusion-tolerant framework over the CSI-KNN-based NIDS. The framework consists of four components: the Manager, Alert Agents, Maintenance Agents, and the Console. They are installed on a secure administration server called Station. The Manager has knowledge of the resources of the NIDS and maintains system reliability by coordinating with Alert Agents (AAs) and Maintenance Agents (MAs). The AA is responsible for monitoring the IDS components and collect intrusion alerts generated by classifiers. The MA is responsible for restoring failed IDS components. An AA is dispatched by the Manager to Host A. On the host, the AA checks the behaviors of the IDS components and reports back to the Manager. Then, the AA moves to Host B and executes the same checks. The AA returns to the Station after it has visited all hosts. The MA is responsible for restoring failed IDS components. The Manager process provides the MA with the necessary information and dispatches it to an operational host. After the MA finishes the recovery task, it also returns to the Station and terminates. The Console provides an interface to the security administrator to monitor the intrusion-tolerant mechanism.

Figure 3 shows the data flow between the components. The AA receives an Available Detection List (ADL) from the Manager and sends the Manager alerts. The ADL is a list of the addresses of all operational detectors (hosts) in the system. Following the addresses in the ADL, the AA visits each detector and

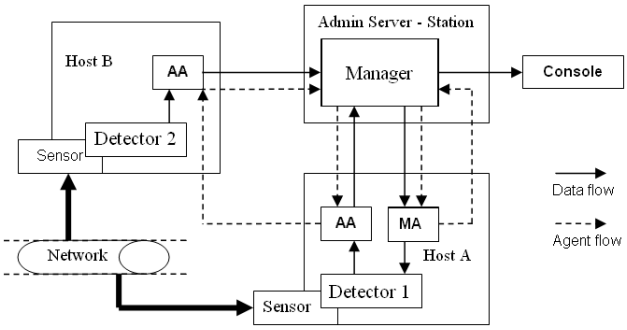

Figure 2. The anatomy of the intrusion-tolerant framework

performs its tasks. The Manager receives two types of alerts from the AA: *intrusion alerts* and *security alerts*. Intrusion alerts are generated by the classifiers when detecting network intrusions. Security alerts are generated by the AA when it discovers a failed IDS component. The MA receives the profile of the failed classifier from the Manager and restores the classifier on an operational detector. A profile describes the characteristics of a specific network service. A service-specific classifier consists of a profile and the CSI-KNN algorithm and detects network intrusions of some specific network service. As soon as the recovery is complete, the MA sends a recovery-complete message to the Manager. The Console receives alerts and messages from the Manager and outputs them.

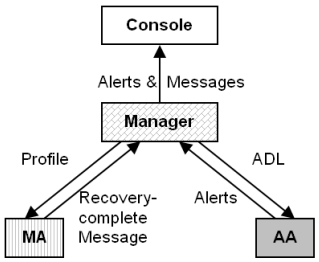

Figure 3. Data flow between framework components

4.1 Manager and Console

The Manager serves as a coordinator in the intrusion-tolerant mechanism. It analyzes the alerts and messages it receives and takes corresponding action. The Manager has two table's, *services* and *detectors*, which provide availability and resource knowledge about the NIDS. The *services* table contains informa-

tion on the classifiers. Each classifier has a unique entry in the table. As Table 1 shows, each entry has four columns which include the following information: the network service that the classifier processes, the status of the classifier, the detector that the classifier works on, and the maintenance status, which indicates if the classifier is undergoing recovery.

Table 1. Table structure of *services*

Column name	Description
service	Service name
detector	The address used to access the detector
s_availability	Status of the classifier
maintenance	Maintenance status

Each detector has a unique entry in the *detectors* table. In the NIDS, each host has one detector and a proxy server to receive the agents (AA and MA). Thus, the detector status is basically associated with the status of the host and the proxy server. Each entry in the *detectors* table contains the following information: the address of the proxy server through which mobile agents can access the detector (host) and the status of the detector. The structure of the *detectors* table is shown in Table 2. Through the two tables, the Manager knows the status of all classifiers and detectors.

Table 2. Table structure of *detectors*

Column name	Description
detector	The address used to access the detector
d_availability	Status of the detector

The Manager receives alerts and messages through a proxy server and decides what actions should be taken. When receiving a message, the Manager first outputs the message to the Console, then acts according to the content of the message. For intrusion alerts, it simply saves the alerts. For security alerts, the Manager updates the *services* and *detectors* tables and dispatches MA(s) for recovery. Since the intruders can attack either the classifier or the detector, the AA monitors both components and generates two different security alerts: *service alert* and *detector alert.* A service alert is raised when the failure happens on an individual classifier, whereupon the Manager dispatches an MA to restore the classifier. A detector alert is raised when a failure happens on the host level and makes all classifiers of the detector unavailable. As soon as a detector alert is raised, the Manager sends MAs to restore the detector. Since a detector may contain many classifiers, the corresponding number of MAs is generated to restore each classifier. If the Manager receives a recovery-complete message from the MA, it updates the information about the restored classifier on the two tables. The Manager generates a new AA when it receives a get-home message from the returned AA, so there is always an Alert Agent monitoring the system.

4.2 Alert Agent (AA)

The AA is responsible for checking the status of classifiers and detectors and collecting intrusion alerts. When taking the route specified in the ADL, the AA executes the following tasks. The AA checks the status of the host when it moves to the host. If the machine is not available, it raises a detector alert and skips the failed detector. Otherwise, it starts checking the status of each classifier on the host and generates a service alert if it detects a classifier failure. The AA also disables the classifier so that no more incorrect information is generated. The intrusion alerts are collected and sent to the Manager only when the detector and classifier are both verified. After all detectors in the ADL are visited, the AA returns to the Station and sends a get-home message to the Manager.

4.3 Maintenance Agent (MA)

The MA is designed to restore a failed classifier. The Manager dispatches an MA to an operational detector on which the new classifier is to be built and activated. After the recovery is complete, the MA sends a message to the Manager. The message contains the information that the classifier has been restored so that the Manager can perform a corresponding update on the two tables and maintain resource information about the NIDS. Once the MA has restored the classifier and informed the Manager of this fact, it returns to the Station and terminates.

5 Implementation and Evaluation

With the intrusion-tolerant mechanism in place, the CSI-KNN-based NIDS can tolerate classifier and detector failures caused by intrusions. Although the detection components can fail due to other faults such as operational mistakes, intrusions are the major cause of the failures of NIDSs. Given that, the proposed intrusion-tolerant mechanism accomplishes its intrusion tolerance tasks by detecting and masking the failure of the detection components.

5.1 Tolerating Intrusions on Detectors

Since the status of a detector component is associated with both the proxy server and the host on which it resides, when an intrusion successfully crashes a host

or a proxy server in the NIDS, a detector failure happens. To verify the system's ability to tolerate a failed detector, we simulate a Denial-of-Service (DoS) attack against the detector. We overload the proxy server on the victim host by using a process that keeps sending requests to the listening port of the proxy server. Finally, it makes the proxy server fail to respond to legitimate requests.

The recovery process is initiated as soon as a detector alert is raised. The intrusion-tolerant mechanism employs forward recovery. Unlike backward recovery that rolls back to the error-free checkpoint, forward recovery needs no previous state. It attempts to "go forward" and try to make the state error-free by taking the necessary actions. In our mechanism, forward recovery is realized by building a new classifier on a "good" detector. We assume that the new classifier on a good host should start in an error-free state. Although forward recovery is simple and effective, it has a drawback. Since the restored component goes forward, it loses the information during the outage.

As Figure 4 shows, the intrusion-tolerant mechanism conducts a forward recovery for the classifiers on the failed detector. When Detector2 is detected to have failed, all classifiers (C2 and C3) on the detector are discarded, and corresponding classifiers are built on Detector1. The Manager supplies the information (profiles) for the new classifiers and dispatches Maintenance Agents (MAs) to execute the recovery tasks. Since in CSI-KNN-based NIDS a classifier is a process that applies the CSI-KNN algorithm on an associated profile, the profile is necessary to build the new classifier. The MA reads the profile from the Manager and brings it to Detector1. From the view point of the NIDS, the services of the classifiers migrate from one detector to another. From the view point of NIDS users, the intrusion tolerance is transparent. As the process is executed automatically by the intrusion-tolerant mechanism, the users are not affected by component failure.

The intrusion-tolerant mechanism updates the *services* and *detectors* tables based on migration of services. Before the failure of Detector2 happens, the *services* and *detectors* tables are shown in Table 3 and Table 4. In the *services* table, the service column records the name of the service-based classifier, and the detector column contains the address of a proxy server through which an AA can access the detector. The address has two parts: the name of the host on which the classifiers are installed, and the port that the proxy server listens to. The s_availability column indicates the status of the classifier, which is set to one when this classifier is working, and zero otherwise. The value of maintenance column is set to "true" when the classifier

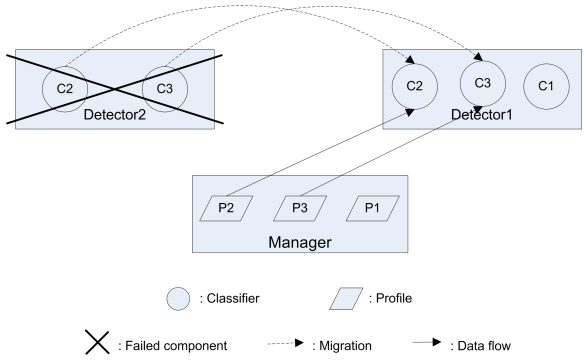

Figure 4. Forward recovery of the failed detector

is undergoing recovery, otherwise, it is set to "false".

Table 3. Example of the *services* table

service	detector	s_availability	maintenance
c1	\\host1:5555	1	false
c2	\\host2:6666	1	false
c3	\\host2:6666	1	false

In the *detectors* table, the detector column has the address of each detector in the NIDS, and the d_availability column records the number of active classifiers on the detector. In Table 4, d_availability value of the second entry is "2" because there are two active classifiers ("c1" and "c2") on that detector.

Table 4. Example of the *detectors* table

detector	d_availability
\\host1:5555	1
\\host2:6666	2

After the Manager receives the recovery-complete messages from "c2" and "c3" MAs, it updates the *services* and *detectors* tables. The results after update are shown in Table 5 and Table 6.

In the services table, the restored classifiers "c2" and "c3" are updated with the new detector names; their availability status is set to active; the values of the maintenance status are set back to "false", indicating that the classifier is active. The availability of the detector is also updated. As shown in Table 6, the availability value is set to "3" based on the number of active classifiers. After the update, the three classifiers are back to work and the detector failure has been masked.

Table 5. The *services* table after the receipt of recovery-complete messages

service	detector	s_availability	maintenance
c1	\\host1:5555	1	false
c2	**\\host1:5555**	**1**	**false**
c3	**\\host1:5555**	**1**	**false**

Table 6. The *detectors* table after the receipt of recovery-complete messages

detector	d_availability
\\host1:5555	**3**
\\host2:6666	0

5.2 Tolerating Intrusions on Classifiers

A classifier in the CSI-KNN-based NIDS is a process that employs the CSI-KNN algorithm and a service-specific profile to detect network intrusions. The attacker can subvert classifiers to prevent the NIDS from providing detection services. In the experiments, we simulate an intrusion scenario in which an attacker removes essential files of the classifier by misusing his/her privilege. As a result, intrusion detection for the associated service is not delivered.

The schema of the classifier recovery in the NIDS is shown in Figure 5. In the diagram, once classifier C2 in Detector2 is discovered to have failed, the classifier is discarded. A new C2 is restored on Detector1. To the NIDS, the service of C2 has migrated from Detector2 to Detector1. Since C3 is not detected to have failed, the service of C3 is not changed.

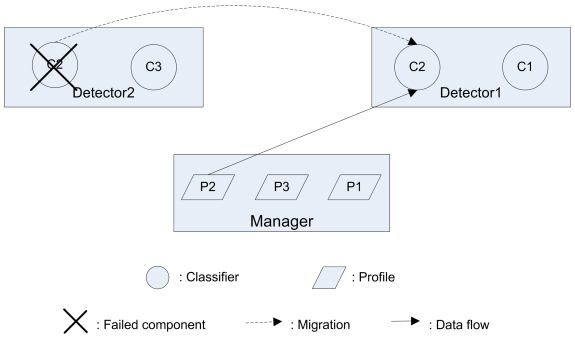

Figure 5. Forward recovery of the failed classifier

The intrusion-tolerant mechanism updates the two tables similar to its handling to a detector failure. However, to turn off the failed classifier and guarantee only one copy of classifiers works at the same time in the NIDS, configuration files on the two hosts are updated too. A configuration file contains the information of the classifiers on each host. In the configuration file, the service column corresponds to the service-based classifier and the s_availability column shows the availability status of the service. Like the s_availablity columns in the services table, this column is set to one when the classifier is active, and otherwise to zero. Through the configuration files, the NIDS triggers the active classifiers and the AA checks and collects intrusion alerts only from those active classifiers.

The configuration files after recovery is shown in Table 7(a) and 7(b)). The failed classifier "c2" on "host2" is deactivated as soon as the classifier failure is detected by the AA. (Table 7(b)). The "c2" classifier is set to active on "host1" when the MA build a new "c2" there. The two configuration files (Table 7(a) and 7(b)) are complementary to each other after recovery. Hence, it is always only one classifier of the service running in the NIDS.

Table 7. Configuration files after "c2" recovery

(a) The configuration file on "host1"

service	s_availability
c1	1
c2	**1**
c3	0

(b) The configuration file on "host2"

service	s_availability
c2	0
c2	**0**
c3	1

5.3 Recovery Performance

The intrusion-tolerant mechanism employs forward recovery to restore failed classifiers. Although the forward recovery is simple and effective, it has the drawback of information loss during the recovery. Therefore, we performed experiments to evaluate the recovery performance of the MA. The recovery starts when the Manager receives a security alert and ends when the recovery-complete message is received. Other than the influence of the network and the host where the MA restores the failed classifier, we observe that the recovery performance is mainly affected by the size of the profile. Thus, we chose four different sizes of profiles (20K, 50K, 100K, and 200K) and ran the recovery of each profile ten times. Figure 6 shows how the recovery time varies with profile size. Even the recovery time varies slightly from run to run, in general, we notice that the recovery time increases with the size of the profile. The experimental results show that even if the MA restores a large profile the time taken by

the forward recovery is short enough that information loss during the outage could be very small. Of course, besides the profile size, other factors such as network traffic and responding time of the target host will also affect the recovery time.

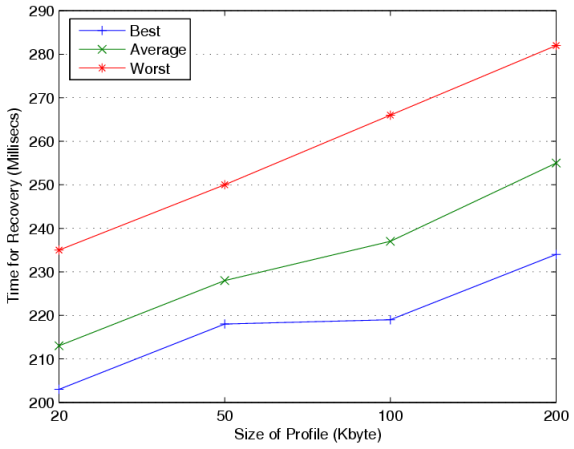

Figure 6. Increase of recovery time with the size of profiles

5.4 Detection Latency

We are interested in the time consumed by an AA visiting all detectors since it affects the time to detect a failed component. In the intrusion-tolerant mechanism, the AA serves as a watchdog timer, utilizing the encoded rules to check the status of detectors and classifiers. We assume that the checking rules are effective, so that the detection latency for any failed classifier is no more than one AA dispatching period – that is, the interval between the dispatching of two AAs. In the intrusion-tolerant mechanism, the AA dispatching period is one calling period (CP) of the classifier plus the AA round trip time. Since the calling period is set based on the NIDS, we calculate the AA round trip time to estimate the maximum detection latency.

Apart from the influence of network conditions and response time of target hosts, the round trip time of an AA can be computed using Equation 1. Here, T is the total time an AA spends for a patrol. It starts when the Manager dispatches the AA and ends when it receives a get-home message from the AA. t_i is the time that the AA spends on each host, t_0 is the time that the AA spends between two hosts, and N is the number of hosts. We first analyze how t_0 varies with the number of hosts. In our experiments, an AA is dispatched to a number of hosts where classifiers are set as inactive. Thus, we can calculate t_0 since the AA does not need to process any classifiers on its patrol. As Figure 7 shows, the curve of the AA patrol time roughly follows a linear increase with the number of hosts.

$$T = \sum_{i=1}^{N} (t_i + t_0) \quad (1)$$

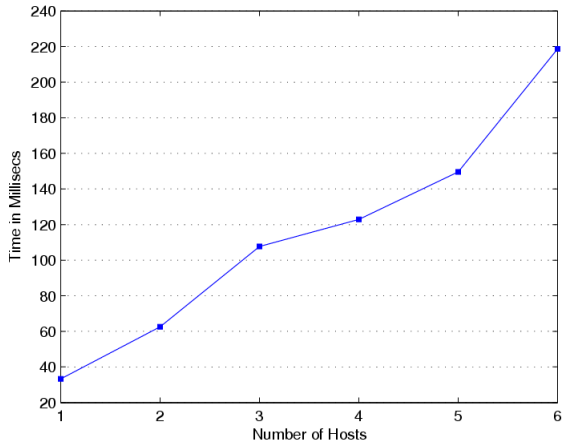

Figure 7. Increase of t_0 with the number of hosts

When an AA gets to a host, it checks the status of each classifier and processes intrusion alerts. We exclude the influence of the time to perform a status check since it is very short. Thus, the time that the AA consumes on each host is mainly affected by the processing time of intrusion alerts. We observe that t_i, the time consumed by an AA on a host, varies with the number of alerts. Therefore, we perform experiments in which the AA processes different numbers of intrusion alerts. Each intrusion alert is set to 100 bytes. Theoretically, t_i should increase linearly with the number of alerts, since the AA processes the alerts with the same size independently.

With the results from the experiments, we can calculate the AA round trip time T for different numbers of hosts and alerts based on Equation 1. In our experiments, the total time of an AA round trip over six hosts with 100 intrusion alerts on each host is 0.55 second. This result is close to the time calculated by Equation 1, which is about 0.52 second. Since the calling period (CP) is set for real-time detection, it should be a small value, perhaps a few seconds. Then, the dispatching period of the AA, which is the CP plus the AA round trip time, is still a small value based on the experimental results. Thus, when the NIDS has several hosts and a few hundreds of intrusion alerts, the detection latency is short. However, we should note that network environments and host performance may affect the detection latency. For example, in a congested network,

the time required to move between two hosts will be longer than what we observed in the experiments.

6 Conclusions and Future Work

In this paper, an intrusion-tolerant mechanism is proposed to guarantee the continuous service of network intrusion detection systems (NIDSs) even in the face of component failure caused by attacks. The intrusion-tolerant mechanism is implemented on the CSI-KNN-based NIDS. Through the continuous monitoring of detection units, the mechanism identifies failed components due to attacks and executes forward recovery. By using dynamic redundancy, our intrusion-tolerant mechanism requires no duplicate components to work at the same time. Moreover, the mechanism restores failed classifiers on peer hosts that are also used for intrusion detection in the NIDS. Therefore, it needs no additional backup machines.

A limitation of the intrusion-tolerant mechanism is that the mechanism is unable to detect unauthorized modification of configuration files. An attacker may change the configuration file and fool the Alert Agents (AAs) into refraining from checking the monitored classifiers. A function can be introduced to compare the configuration information on the local host with the *services* table. The other future works are planned as follows: more checking rules will be developed for the AA's to improve their ability to detect compromised components; an intelligent system will be employed to analyze the intrusion alerts generated by the NIDS and aid the intrusion-tolerant mechanism in treating the compromised components; and finally, a response mechanism is to be introduced in order to stop intrusions before a failure occurs.

7 Acknowledgments

We sincerely thank Bell Canada and NSERC (Natural Sciences and Engineering Research Council) of Canada for supporting this research.

References

[1] J. McHugh, A. Christie, and J. Allen. Defending yourself: the role of intrusion detection systems. *IEEE Software*, 17(5):42–51, 2000.

[2] Y. P. Shen, W. T. Tsai, S. Bhattacharya, and T. Liu. Attack tolerant enhancement of intrusion detection systems. *Proceeding of the 21st Century Military Communications Conference (MILCOM 2000)*, vol 1, pages 425–429, October 2000.

[3] D. Yu and D. Frincke. Towards survivable intrusion detection system. *Proceeding of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, vol 9, page 90299a, January 2004.

[4] L. Siqueira and Z. Abdelouahab. A fault tolerance mechanism for network intrusion detection system based on intelligent agents (NIDIA). *Proceeding of the 4th IEEE Workshop on SEUS-WCCIA*, vol 19(9), pages 49–54. IEEE CS Press, April 2006.

[5] P. Pal, P. Webber, R. E. Schantz, and J. P. Loyall. Intrusion-tolerant systems. *Proceeding of the IEEE Information Survivability Workshop*, pages 24–26, Boston, MA, USA, 2000.

[6] F. Wang, F. Jou, F. Gong, C. Sargor, K. G. Popstojanova, and K. Trivedi. SITAR: A scalable intrusion-tolerant architecture for distributed services. *Proceeding of Foundations of Intrusion-Tolerant Systems*, pages 359–367, 2003.

[7] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Alberta, Canada, 1998.

[8] L. Kuang and M. Zulkernine. An Anomaly Intrusion Detection Method Using the CSI-KNN Algorithm. *Proceeding of the 23rd Annual ACM Symposium on Applied Computing*, Cear, Brazil, March 2008, ACM Press.

[9] L. Kuang and M. Zulkernine. *A Dependable Network Intrusion Detection System Using the CSI-KNN Algorithm*. MSc Thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, August 2007.

[10] M. Dacier. Design of an intrusion-tolerant intrusion detection system. Technical Report D10, IBM Zurich Research Laboratory, 2002. (`http://www.maftia.org/deliverables/D10.pdf`).

[11] D. Powell and R. Stroud (editors). Conceptual model and architecture of MAFTIA. Technical Report D21, IBM Zurich Research Laboratory, 2003. Available at `http://www.maftia.org/deliverables/D21.pdf`.

[12] Safety and Mission Assurance. Reliability Block Diagrams. Technical Report QS-R-002, Marshall Space Flight Center, October 2004.

[13] Pankaj Jalote. *Fault Tolerance in Distributed Systems*. Prentice-Hall, Inc., NJ, USA, 1994.