Design and Implementation of Acceptance Monitor for Building Scalable Intrusion Tolerant Systems [*]

Rong Wang[1], Feiyi Wang[1] and Gregory T. Byrd[2]

[1]*Advanced Networking Research, MCNC, Research Triangle Park, NC 27709*
[2]*Department of ECE, NC State University, Raleigh, NC 27695-7911*

Abstract

Intrusion detection research has so far concentrated on techniques that effectively identify the malicious behaviors. No assurance can be assumed once the system is compromised. Intrusion tolerance, on the other hand, focuses on providing minimal level of services, even when some components have been partially compromised. The challenges here are how to take advantage of fault tolerant techniques in the intrusion tolerant system context and how to deal with possible unknown attacks and compromised components so as to continue providing the service. This paper presents our work on applying one important fault tolerance technique, acceptance testing, for building scalable intrusion tolerant systems. First, we propose a general methodology for designing acceptance testing. An Acceptance Monitor architecture is proposed to apply various tests for detecting the compromises based on the impact of the attacks. Second, we make a comprehensive vulnerability analysis on typical commercial-off-the-shelf (COTS) web servers. Various acceptance testing modules are implemented to show the effectiveness of the proposed approach. By utilizing the fault tolerance techniques on intrusion tolerance system, we provide a mechanism for building reliable distributed services that are more resistant to both known and unknown attacks.

Keywords

Fault Tolerance; Intrusion Tolerance; Acceptance Testing.

[*]This work is sponsored by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) under contract N66001-00-C-8057 from the Space and Naval Warfare Systems Center - San Diego (SPAWARSYSCEN). The views, opinions and findings contained in this paper are those of the authors and should not be construed as official DARPA or SPAWARSYSCEN's positions, policy or decision.

1 Introduction

Network security research [15, 27] has in general emphasized making information systems secure by keeping intruders out. Confidentiality and integrity have been achieved by encrypting critical information and limiting access to it only to authenticated users. However, since no security precautions can guarantee a system not be penetrated, once a system is compromised or even just under attack, it will be left in a vulnerable and unpredictable state, which is not acceptable for mission critical applications or services. As a second line of defense, intrusion detection and response research [2, 4, 16, 21] has mostly concentrated on known and well-defined attacks. This narrow focus of attacks has accounted for both the successes and the limitationes of many commercial intrusion detection systems (IDS). A number of well respected research and commercial IDS have been evaluated at MIT Lincoln Labs in the past two years [19]. The results showed that new and novel attacks present formidable challenges to these systems.

In order to overcome the above problems, the SITAR (Scalable Intrusion-tolerant Architecture for Distributed Services) [29] architecture (see Figure 1) is proposed to provide a framework to build intrusion tolerant system for distributed services. It has the following novel aspects: (1) We focus on one generic class of services (network-distributed services built from COTS components) as the target of protection. Specifically, we discuss the framework in a web service context to make our presentation tangible. (2) Two specific challenges are addressed in this architecture. The first is how some of the very basic techniques of fault tolerance (e.g., redundancy, diversity and acceptance test) apply to our target. The second is how we deal with external attacks and compromised components, which exhibit very unpredictable behavior compared to accidental or planted faults. (3) Our dynamic reconfiguration strategies will be based on the intrusion tolerant model built within the architecture.

In the SITAR architecture, the Acceptance Monitor plays an important role in detecting compromises caused by external attacks and reporting these compromises through intrusion triggers. Its main functionality is acceptance testing, which applies pre-designed application-specific tests to both requests and responses and detects system compromises primarily based on the impact of the attack. The results of the acceptance testing can help to determine (1) what kind of the compromises the system has, (2) which software component is possibly involved, and (3) whether the response generated by the software component is still valid.

The rest of the paper is organized as follows. In Section 2 we propose a general Acceptance Monitor design methodology and a generic monitor architecture to effectively apply tests on various COTS service. Section 3 describes the implementation of Acceptance Monitors for web services based on the generic Acceptance Monitor design. Section 4 presents the experimental analysis on detecting web server compromises and discusses detection coverage of the Acceptance Monitors that we designed. Section 5 discusses the related work. Finally, we make conclusions and discuss the future work in Section 6.

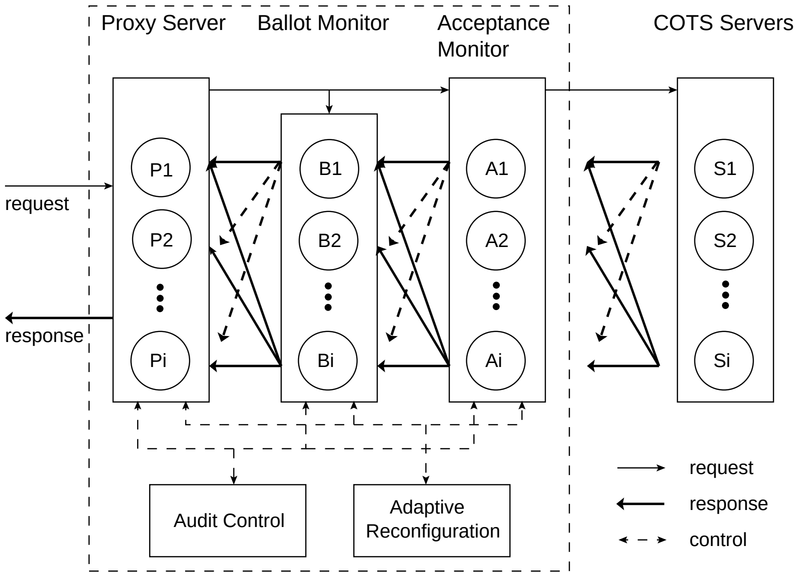

Figure 1: Overview of SITAR architecture

2 Acceptance Monitor Design

As mentioned above, the main functionality of an Acceptance Monitor is to detect the compromises of COTS servers through acceptance testing. With the result of acceptance testing, the system adaptively reconfigures its components so that the impact caused by attack can be masked and the damage can be recovered.

The role of the Acceptance Monitor in the system architecture is shown in Figure 1. An Acceptance Monitor applies acceptance testing on client requests and the server responses. When an Acceptance Monitor detects any compromise, it generates intrusion triggers for the Adaptive Reconfiguration Module. Based on the content of the intrusion triggers, the Adaptive Reconfiguration Module takes corresponding actions such as isolating compromised components, reconfiguring system resources and enforcing new security policies. In addition to the compromise detection, an Acceptance Monitor also determines whether or not the server response is valid. It gives the response's validity result to Ballot Monitors so the Ballot Monitors can pick only valid responses. To improve fault tolerance, SITAR introduces redundancy and diversity of COTS service. This redundancy and diversity requires one or more Acceptance Monitors to participate in the detection by processing requests and responses. The redundancy level and the work assignment among Acceptance Monitors for different COTS servers are determined by the Adaptive Reconfiguration Module and the

Proxy Server.

When we design the Acceptance Monitors for generic COTS services, we need to consider two important aspects: (1) what kind of acceptance testing measures can be used and (2) how the Acceptance Monitors can efficiently collaborate with other distributed components in the SITAR system.

2.1 Design methodology

2.1.1 Acceptance testing measures

In fault tolerance terms, acceptance testing is a programmer or developer-provided error detection measure in a software module [18], in the form of a check on the reasonableness of the results calculated. It usually consists of a sequence of statements that will raise an exception if the state of the system is not acceptable. If any exception is raised by the acceptance test, the module is said to have failed or been compromised. In our Acceptance Monitor, we include acceptance testing modules to detect the compromises caused by any errors, including both accidental faults malicious attacks. The source of the testing is mainly the responses from the COTS server. Different services requires different types of testing measures. We can broadly classify the testing measures into following categories.

a. Requirement test: In many cases, some conditions are imposed to complete a task. These conditions can be represented as an expected sequential order of events or a subset of given events. The requirement test is to make sure that the imposed conditions are satisfied. For example, if an attacker stored a back-door code that enables the server to send some message to a client without a relative request, we will observe some violations in the required sequential order.

b. Reasonableness test: Reasonableness test is used to detect software/system failures through pre-computed ranges, expected sequences of program states, or other relationships that are expected to be satisfied. Reasonableness checks are based on physical constraints, while satisfaction of requirements tests are based on logical or mathematical relationships.

c. Timing test: In fault tolerance, timing test is used in systems with time-sensitive components to determine whether the execution time meets the constraints. In our system, we can use timing test to detect the denial of service(DoS) [5] compromise in both time-sensitive and non-time-sensitive services. In time-sensitive services, restricted time parameters are defined for the COTS server's response arrival time. In non-time-sensitive services, reasonable time parameters for module execution are also given. Based on such timing checking, we can determine whether there is a DoS compromise or service degradation. Finding out the reasonable time parameters is not simple. A bad value can result in many false alarms. A learning process can help to estimate the reasonable range of time parameters.

d. Accounting test: The accounting test is used for transaction-based applications that involve simple mathematical operations. Examples are airline reservation systems, library records, inventory control and control of hazardous materials. A tally for both the total number of records and sum over all records of a particular data field can be compared between source and destination, whenever a large number of records are transmitted or reordered.

e. Coding test: Theoretically, coding test is based on redundancy in the representation of the target data we want to protect. The redundant check data are maintained in some fixed relationship with the non-redundant data representing the value of the payload. Errors that result from a corruption of either form of data can violate this relationship, and if the relationship does not hold, data corruption will be detected. Traditionally, the redundant data are within the data. In our system, the redundant data of the objects can be kept in the Acceptance Monitor because the Acceptance Monitor is more trustworthy. The problem of separating the redundant data from the object is that the redundant data will be out of date if the object has been modified legally. The representation of the redundant data can be the cryptographic checksum or the hash-code of the original data. Many algorithms are widely used for this purpose, such as CRC [22], MD5 [25], and SHA [24].

2.1.2 Communication and collaboration environment

An Acceptance Monitor supports the following communication primitives with other SITAR components:

a. Registering/unregistering for a given connection: When a client asks for a COTS service, it sends a connection request to the Proxy Server. Before setting up the connection, the Proxy Server needs to choose Acceptance Monitors to process the client's requests and COTS server's responses. After Proxy Server and Adaptive Reconfiguration Module decide the redundancy level and working assignment among available Acceptance Monitors, the Proxy Server multicasts the working assignment and waits for the confirmations from Acceptance Monitors. If some of the Acceptance Monitors fail to confirm, the Proxy Server chooses other available Acceptance Monitors or changes the redundancy level of Acceptance Monitors. An Acceptance Monitor that sends a confirmation will be a registered Acceptance Monitor for this new connection. When a COTS server stops working for this connection, the Acceptance Monitor that is testing this server needs to unregister for the connection.

b. Reporting validity of a server response to Ballot Monitors: As Ballot Monitors only pick up valid COTS server responses and Acceptance Monitors check the validity of COTS servers' responses through acceptance testing, Ballot Monitors needs to know from Acceptance Monitors which responses are valid.

c. Reporting compromises to the Adaptive Reconfiguration Module: In SITAR, the Adaptive Reconfiguration Module receives intrusion triggers, evaluates intrusion threat, tolerance objectives, costs, as well as performance impacts, and generates new configurations for the system. An Acceptance Monitor generates an intrusion trigger when it detects a compromised COTS server through acceptance testing. The trigger to the Adaptive Reconfiguration Module contains the information of how serious the threat is, which type of the compromise has been detected, and which server is compromised.

As SITAR components are distributed, a flexible and reliable communication environment is very important. When building such a communication environment, we also needs to make sure that the distributed collaboration environment should meet the scalability requirements of the SITAR components.

As distributed tuplespace [13] provides dynamic and flexible coalition environment for the distributed system, we choose to use it as our SITAR collaboration space. Tuplespace was first developed for the Linda coordination language [7]. It is a content-addressable shared memory. Each element of the space is a tuple, which is an ordered list of type-value pairs. To retrieve a tuple from the space, a template is provided, in which a subset of the tuple elements can be left blank. A matching tuple is one which matches the values provided by the template and the types of the blank entries in the template. Like tradition message passing, tuples are passed by value, not by reference, and may be freely copied.

JavaSpaces[1] [8] is a implementation of the tuplespace. It combines the concept of the tuple space with the Java language. The JavaSpaces services are built from the distributed programming facilities of Jini [14]. The tuples in JavaSpaces are objects which implement the Entry interface. JavaSpaces implements the following operations that control the contents of the tuple: *write* places an object into the space; *read* retrieves a copy of a matching tuple; *take* gets and removes a matching tuple from the space. The non-blocking forms of *read* and *take* are called *readIfExists* and *takeIfExists*. In addition to these basic operations, JavaSpaces provides three other mechanisms for space-based coordination: (1) *lease*, which is a time limit associated with a tuple, (2) *transaction*, which is a collection of operations that are performed atomically, and (3) *distributed event*, which allows an object to be notified if a particular type of tuple is added to the space.

The reference implementation of JavaSpaces, provided by Sun Microsystems, uses a centralized tuplespace server. Other implementations [20] or enhancements [12] provide for distributed JavaSpaces servers with replication for fault tolerance.

2.2 Generic Architecture of an Acceptance Monitor

According to the above methodology, we designed the generic Acceptance Monitors architecture as shown in Figure 2. An Acceptance Monitor contains a group of acceptance testing modules (we call them workers). Each worker checks for a unique connection within the monitor. A worker is created after its monitor registers for a connection. The number of workers in a monitor represents the number of connections it handles. A worker closes itself once the corresponding

[1] JavaSpaces and Java are registered trademarks of Sun Microsystems Inc.

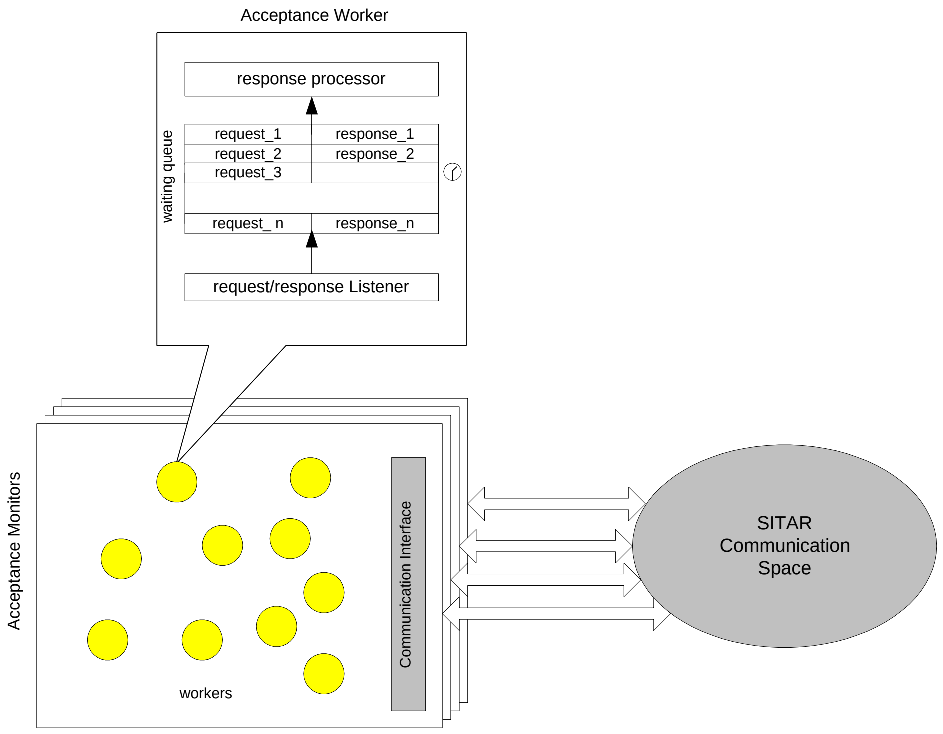

Figure 2: Generic Acceptance Monitor architecture

COTS server stops working for the connection. The communication interface is the interface between the Acceptance Monitor and the SITAR communication space.

The main functionality of each worker is to process requests and responses for a given connection. Inside each worker there are a response processor, a waiting queue, and a request/response listener. The logic flow of the worker components is also shown in Figure 2.

The response processor, shown in Figure 3, includes a group of acceptance testing units (we call them checkers) that apply acceptance testing for a given response. Different units, according to their names, uses different testing measures. The final testing results from the units will be collected by a result marshaller. Based on the testing result, the marshaller generates a validation report and intrusion triggers.The worker sends marshaled response via the response processor.

The waiting queue in the worker contains a list of requests or request/response pairs. When the request processor is idle, it retrieves a complete request/response pair from the waiting queue. When a request just arrives at the waiting queue, the timing checker is activated for its response. If the response does not arrive

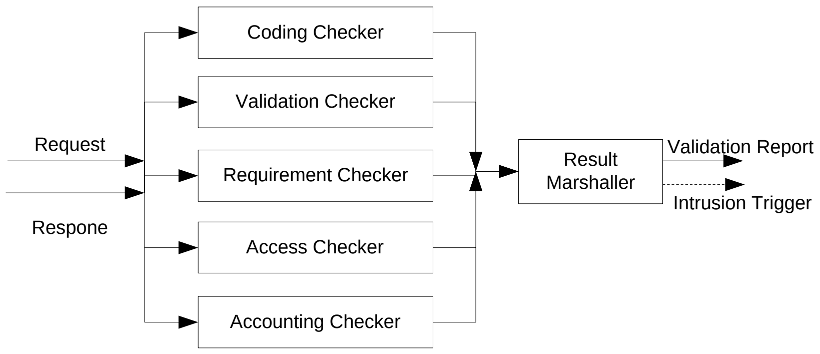

Figure 3: The structure of a response processor

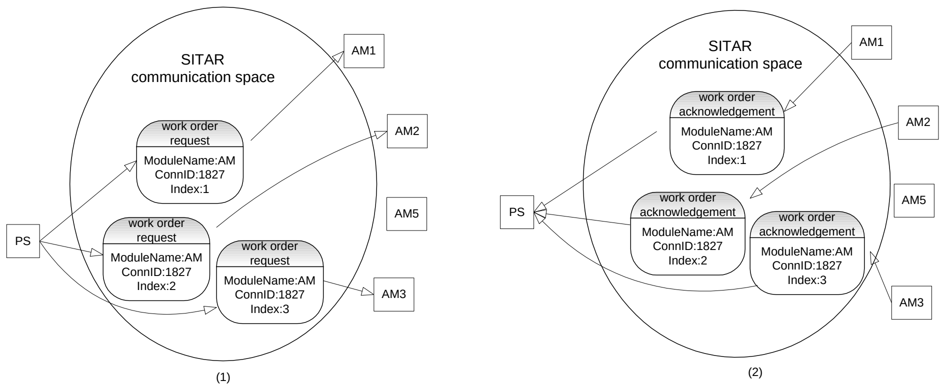

Figure 4: Registering for a connection

within the timeout threshold, the timing checker raises a timeout exception.

The request/response listener listens on the incoming request and outgoing response associated with the connection. When a request is received, it needs to pass a parser first. If a request fails the parsing, the worker drops the request. Otherwise the request will be queued. When the worker finishes parsing the request, it generates a request validation report indicating whether or not this request is valid.

As we discussed in section 2.1.2, an Acceptance Monitor collaborates with other SITAR components through a JavaSpaces. The communication interface of an Acceptance Monitor supports these collaborations. The entry designed for collaboration and their flow are as follows:

- A Proxy Server multicasts the working assignment by writing "work order request" entries into the JavaSpaces. An available Acceptance Monitor takes one of the entries and writes a "work order request acknowledgment" entry for registration. See Figure 4. When an Acceptance Monitor stops

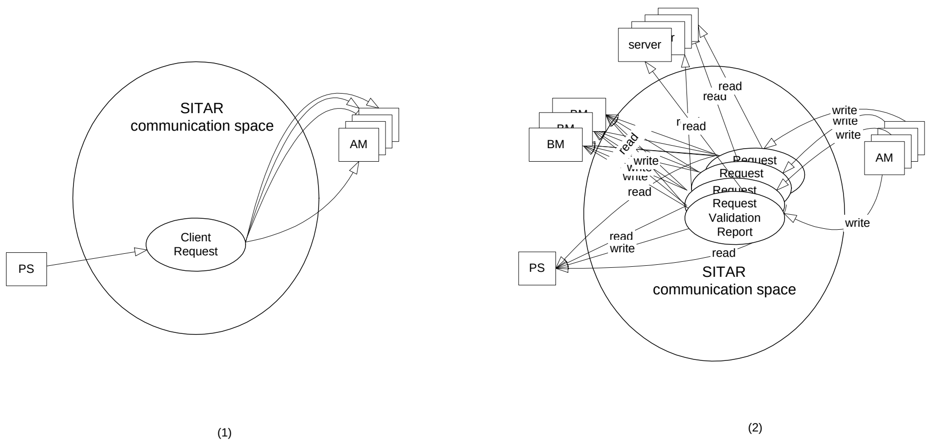

Figure 5: Protocol of verifying a request

working for a connection, the interface puts a “connection close alert” into the JavaSpaces.

- The interface read a client request from the JavaSpaces and writes back a “request validation report” in which there is a boolean field named “valid”. The valid field with “true” indicates that the request passed the parsing successfully. Otherwise the request fails the parsing. See Figure 5.
- The communication interface reads server responses. After checking them it writes back “response validation report” entries. As the “request validation report”, the “true” value in “valid” field in a response report indicates that the response is acceptable. Otherwise it is not acceptable. These report entries will be taken by Ballot Monitors. See Figure 6.
- When a worker detects any compromise of a COTS server, it writes an intrusion trigger entry through the communication interface into the JavaSpaces. The element of the entry includes: connection ID, Acceptance Monitor’s index number, COTS server’s physical name, trigger type and log message.

As current JavaSpaces lacks security features such as confidentiality, authentication and authorization checking, we plan to build an improved collaboration environment on secured JavaSpaces which is developed in Yalta [11] project.

To summarize, we have introduced the generic architecture of the Acceptance Monitor, which is designed to accomodate a variety of COTS services. Since the acceptance testings determine what a COTS server “should do” or “should not do”, Acceptance Monitors are highly application dependent. When we are implementing Acceptance Monitors for specific COTS service, we need to know which acceptance testing measure is appropriate for this service, in another words, which acceptance testing unit is to be activated in a response processor.

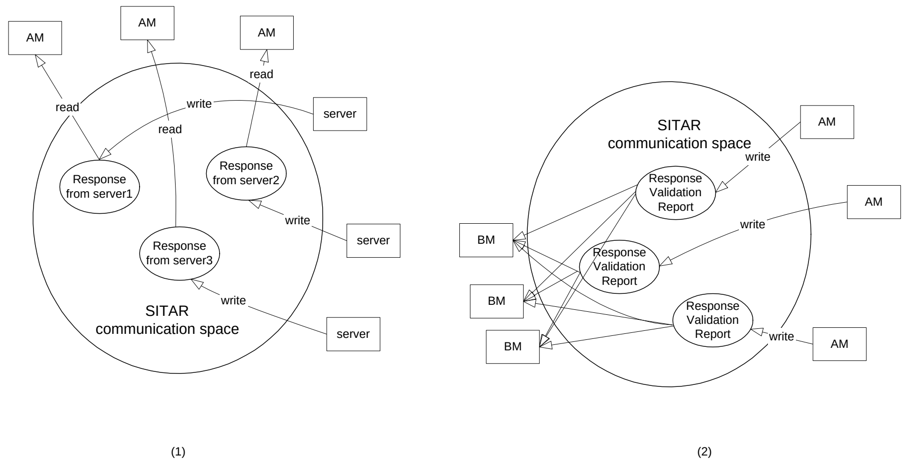

Figure 6: Protocol for verifying a response

Also, different COTS services use different protocols, so the parser within the request/response listener must be adapted for the target COTS service.

3 Implementation of Acceptance Monitors for Web Servers

Based on the generic Acceptance Monitor design, we implemented Acceptance Monitors for web servers. To determine which acceptance testing measures are appropriate for detecting compromise of web servers, we need to know what kinds of faults reside in a web server and what are the possible security compromises related to those faults. To do so, we investigated most of the web server's vulnerabilities advised in Bugtraq [26], which is a full disclosure moderated mailing list archive. This archive contains the detailed discussion and announcement of computer security vulnerabilities: what they are, how they are exploited, and how to fix them. This work is conducted in two phases. First, web server compromises are classified based on their possible impacts. Second, acceptance testing modules for web servers are implemented and verified through experimental analysis.

3.1 Vulnerabilities and Compromises for Web Servers

The Apache [3] and IIS [23] servers are chosen in our study since they are the most commonly used web servers in the Internet and they have many add-on software modules developed to support various web-based services. From 1996 to March 20, 2001, 41 Apache-related vulnerabilities and 84 IIS-related vulnerabilities were reported. We analyzed most of the cases, as shown in Table 1 and

Table 1: Apache related vulnerabilities: the table is filled with Bugtraq ids, `dir` means directory disclosure, `L` means local attack

	DoS	integrity	confidentiality	command execution	others
input validation	552		2100 1587 1488 1084 1081 968 1814	2023	1912 886
boundary condition	1642 1066 552 192	1191		2048 1861 1570 1109 307 286 2252	
access validation	657	1876	582(dir) 2280(dir) 1193 1057 689 167 189 190 149	529	1565
failure to handle exceptional condition	2843 1608 1819 2453 1476 579 2440 2441 1868 1190 2218 1089		1021 882		
configuration		2110 1818		1065 194	
design	521	658 1181	1174(dir) 1832 1499 1734 1108 978 559 447 229 2074		
race condition		501(L)			
unknown	465 193		195		1594 1595

2. Based on the investigation, we found the following types of vulnerabilities residing in web severs: (1) Boundary condition error; (2) Access validation error; (3) Input validation error; (4) Failure to handle exceptional conditions; (5) Race condition error.

Since we are focusing on intrusion tolerance by analyzing the responses, it is necessary to analyze the possible impacts of the exploitations. So we came up with a classification of the web server's compromises based on their impacts, shown in Figure7.

Through the case study and classification, we found out that, although the possible errors share some common attributes (i.e., many can be classified into the same error type), the actual exploitation of each error type may have many different forms. However, various exploitations of many errors have the same impact. This fact motivates us to concentrate on analyzing the impacts from the responses.

3.2 The implementation of an Acceptance Monitor for web server

Based on the compromise analysis, we implemented following acceptance testing units for web servers:

Requirement checker: This checker predicts possible status of each field of the response according to the request and the COTS server's current conditions. If the response is not within the expected status, the checker will report the exception. The requirement checking process is shown in Figure 8. The checker divides an HTTP request into atomic requests. A condition analyzer generates

Table 2: IIS related vulnerabilities: the table is filled with Bugtraq ids, `dir` means directory disclosure, `L` means local attack

	DoS	integrity	confidentiality	command execution	others
input validation			2503 2407 1284 2518 2504 2060 1896 1990 2300	2286 776 629	
boundary condition	1988 1821		2409 2206 1707 1557	1570	2410(root privilege) 1876(upload)
access validation		2171 1457		2376	
failure to handle exceptional condition	1868 1760 2216				
configuration	2454(L)	1238	1023		
design			1531 337 2205 1728 1532		
race condition		2182(L)			
unknown	306		1548 649 98		1575(root privilege)

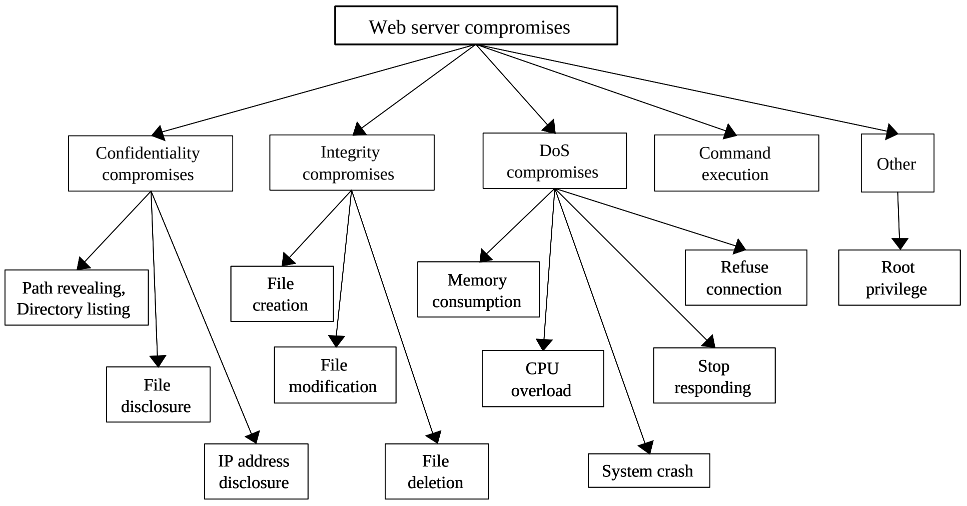

Figure 7: Web server compromise classification

possible response patterns for each atomic request. These responses patterns include (1) the possible HTTP response patterns when a request is satisfied, (2) the HTTP response patterns when a request is not satisfied, and (3) no response. Except no response, each pattern gives a reasonable scope HTTP response header status, such as the status code, and the length of the possible payload. When the patterns are generated, a comparator tries to match the real HTTP response with the patterns. If the HTTP response does not match any possible combination of the patterns, the checker generates a exception.

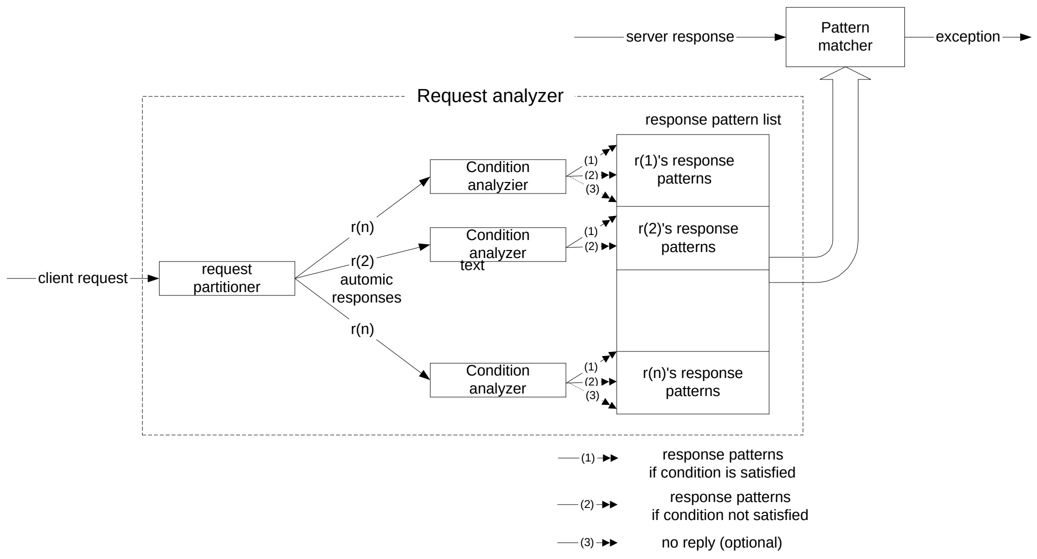

Figure 8: Requirement checking process

Coding Checker: This checker is used to test if the signature of the response's data matches the expected signature of a corresponding file that should be generated by the server. In each Acceptance Monitor, a small cache is maintained to dynamically store information of most frequently visited web pages of the relevant web server. Information for each page includes file length, last modified date, hash code of the file content, and algorithm for the hash code computation. To obtain such information, Acceptance Monitors assume that a web page first delivered by a web server is not the defaced page and extract information from the page. Even under some circumstance a coding checker's cache loads a defaced information, it will be detected by Ballot Monitors through voting. The functionality of coding checker is to detect some confidentiality and integrity compromises in the web servers. The coding checking process is shown in Figure 9.

Timing checker: A timing checker is immediately activated when a request arrives at the waiting queue of a acceptance worker. According to the contents of the request, the timing checker determines a reasonable timeout value for the

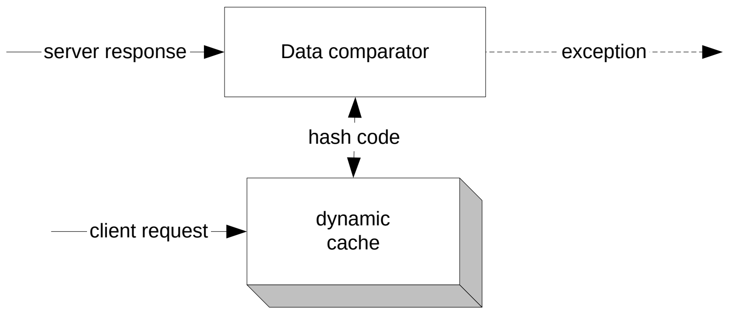

Figure 9: Coding checking process

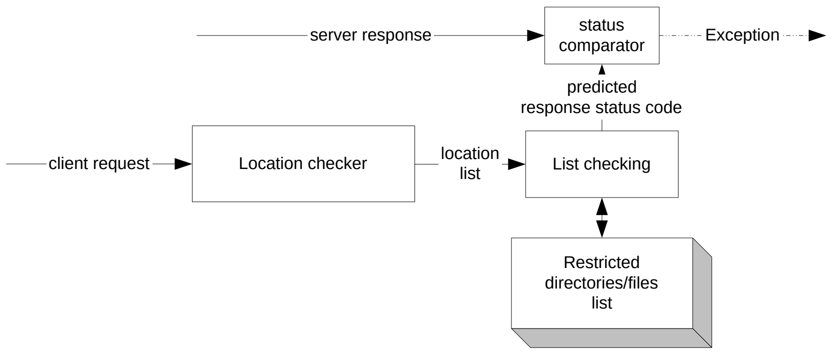

Figure 10: Access checking process

response. If the response does not arrive before the timeout, the timing checker generates a timeout intrusion trigger. Otherwise it will be deactivate by the worker when the response arrives. This timing checker usually can detect denial of service compromise.

Access checker: Access checker uses reasonableness test measure on the response. It checks if the response violates any access control policies. If it does, the access checker raises exception. The access checking process is shown in Figure 10 . When a HTTP response's status code indicates that a client's request is satisfied, the access checker analyzes the request and generates a list of visiting locations the server used for the requested service. If any of the locations is not authorized for the client, then there must be some access control violation and the checker raises an exception.

Validation checker: Validation checker also uses reasonableness test measures on the response. The validation checker has a reasonable range of each field of an HTTP header. If the value in a header field does not scale down to its reasonable range, the checker raises its exception. The validation checking process is shown in Figure 11.

In addition to the above acceptance testing units' implementation, we also

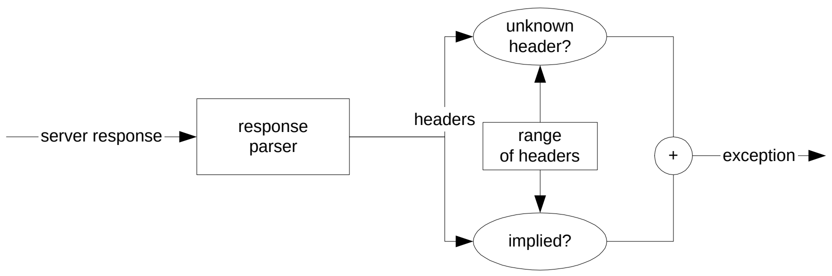

Figure 11: Validation checking process

used HTTP request parser as the request parser within the request/response listener of a worker. The Acceptance Monitor framework we implemented is based on the generic architecture of an Acceptance Monitor.

To verify how Acceptance Monitors can detect compromises successfully, we did some case studies and experiments on fault injection and attack simulation. One of the recent experiments, introduced in the following section, is the CodeRed [9] fault injection and effect detection. For detection of other known compromises, the testing and case study results are summarized in Section 4.4.

4 Experimental Analysis: Fault Injection, Compromises Detection and Detection Coverage

In this section, we performed experimental analysis on SITAR prototype. First, we used CodeRed attack as an example to show how Acceptance Monitors can successfully detects the compromise. Then, we discuss the detection coverage for web services based on our case studies and experimental results.

4.1 CodeRed attack

In July 2001, a virus, named CodeRed [9], was widely spreading throughout IIS Web servers on the Internet via the IIS .ida buffer-overflow vulnerability [10] that was published early in June, 2001. A CodeRed worm has following behaviors once it resides in a vulnerable IIS web server:

- Send a “GET” request back to the attacking machine to indicate that the local machine has been infected successfully.
- Generates a list of random IP addresses and spreads itself by probing remote machines.
- Launches a Denial-of-Service attack against www1.whitehouse.gov from the 20th-28th of each month.
- Defaces web pages on vulnerable web servers with the phrase “Hacked by Chinese”.

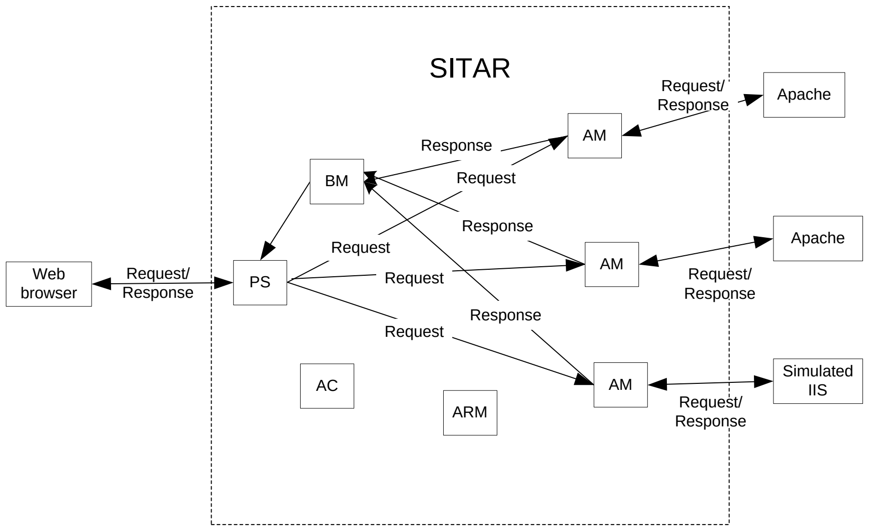

Figure 12: Experiment: scenario of SITAR system

4.2 Experimental environment

To validate the SITAR especially the Acceptance Monitors, we setup our test environment as shown in Figure 12. In the experiment, a simulated IIS web server is built on a windows machine. This simulated IIS web server is a modified mini_httpd [1], which implements all basic features of a small HTTP server. We injected faults in this server so that it can emulate the CodeRed worm's behaviors when it receives a given malicious HTTP request. We also set up two Apache servers on two Linux machines. These two Apache web servers are invulnerable to the CodeRed attack. Besides the three web servers, we set up three Acceptance Monitors, one Ballot Monitor, one Proxy Server and one Adaptive Reconfiguration Module. The configuration of the Proxy Server, Acceptance Monitors and Ballot Monitor for processing client requests and responses is also shown in Figure 12. Three available Acceptance Monitors and Web servers process all requests on a single connection between a client and the proxy. On the same connection, each Acceptance Monitor processes responses from one of the three web servers. The Ballot Monitor collects validation report from all the three Acceptance Monitors. The Adaptive Reconfiguration Module receives intrusion triggers from all three Acceptance Monitors.

4.3 Results

After we setup environment, we send a malicious request that represents the CodeRed attack to the simulated IIS server. As a result, when the server sends a "GET" request back to the client, the Adaptive Reconfiguration Module receives an intrusion trigger from the Acceptance Monitor that is bound with

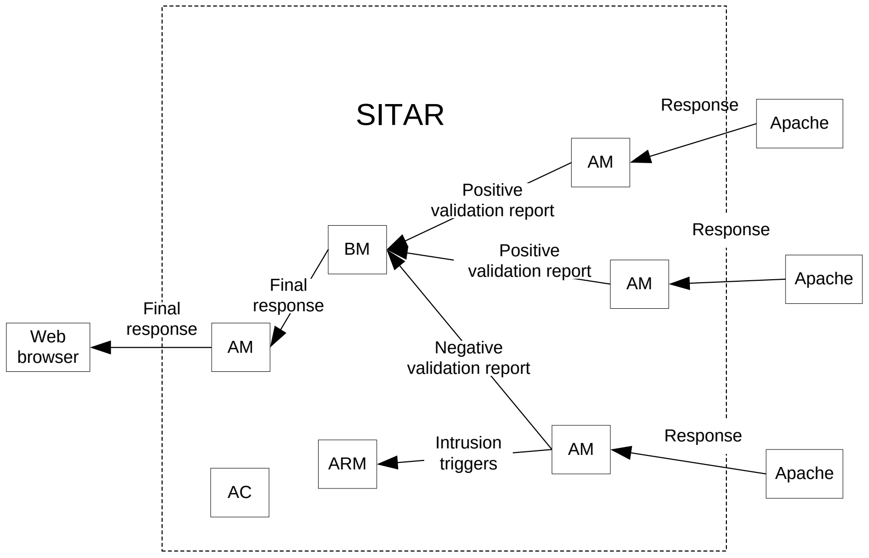

Figure 13: Experiment: verifying responses

the simulated IIS Server. The content of the intrusion trigger indicates that there is an exception raised by a requirement checker indicating the sequential order error of the response. After the simulated IIS is under attack, we visit normal web page from the web browser. Every time after request is sent, the web browser gets a normal web page from the one of the Apache servers and the Adaptive Reconfiguration Module gets an intrusion trigger from the Acceptance Monitor bounded with the simulated IIS server. This time the intrusion trigger indicates that a coding checker detected a defaced web page from the simulated IIS server. See Figure 13.

4.4 Acceptance testing coverage

After we did case studies [30] and experiments, we evaluated the coverage of compromise detection, which is presented in Table 3. All the DoS compromises can be detected through the timing check measure. Integrity compromises can be detected by coding check. In addition, the reasonable and requirement checks can handle the integrity problem caused by web server's input validation error, access validation error and failure to handle the exceptional conditions. Confidentiality compromises can be detected through the reasonableness check and requirement check. The requirement and reasonableness checks also can detect command execution compromises caused by access validation and the failure to handle exceptional condition. These test measures, however, cannot detect some categories of errors under some specific conditions, such as the confidentiality and command execution compromises caused by a boundary condition error.

Table 3: Acceptance testing coverage

Vulnerability	DoS	Integrity	Confidentiality	Command Execution
Input validation error	T,R,A,V	C,R,A,V	C,A,V	
Boundary condition error	T	C		
Access validation error	T,R,A,V	C,R,A,V	R,A,V	R,A,V
Failure to handle exceptional conditions	T	C,R,A,V	R,A,V	R,A,V
Configuration error	T	C	R,A,V	
Design error	T	C	R,A,V	
Unknown	T	C		

T: Timing checker C: Coding checker R: Requirement checker
A: Access checker V: Validation checker

5 Related work

Intrusion tolerance is a new technique that seeks to use the fault tolerance technology to build up systems that can provide secure and reliable service even when the system is under attack. In this section, we discuss some related research work in intrusion tolerance area.

The HACQIT(Hierarchical Adaptive Control for QoS Intrusion Tolerance) [28] project, developed by Teknowledge and UC Davis security Lab, is to build a intrusion tolerance COTS service by using fault-tolerance paradigms and by extending the techniques of intrusion detection to achieve error detection. This project relies primarily on error-detection methods where the errors are detected in the integrity or performance aspects of running processes through specification based analysis. To achieve effective recovery and reconstitution, HACQIT employs a hierarchical model in which each level in the hierarchy recovers as best as it can to a consistent states that is "close" to a state just prior to the attack, and then makes that recovered state available to the next higher level. Similar as SITAR technology, the HACQIT includes error detection mechanism as the base of system reconstitution. Different from the SITAR, it performs detection through violation of QoS specifications and specification based intrusion detection.

DRAPER Laboratory developed Kinetic Application of Redundancy to Mitigate Attacks (KARMA) [17]. This project targets at: emploies only a small set of trusted components to protect a large set of untrusted unmodified COTS servers and databases; tries to minimize loss of data confidentiality and integrity in the presence of a successful attack on one of the servers and tolerates attacks whose specific signatures are now known a priori.

The ITSI (Intrusion Tolerant Server Infrastructure) [6] developed by Secure Computing uses independent network layer enforcement mechanisms to reduce intrusions, prevent propagation of intrusions that do occur, provide automated load shifting when intrusions are detected and support automated server recovery. In this architecture, a detection/initiating agent is used to send alert of spoofing violations, sniffing violations and matching on any filter rule that has alerting enabled. According to alert and PEN rules, the system will shift traffic among the server cluster.

Similar to SITAR project, these projects use dynamic configuration strategy and recovery technology to reconstitute system and mask the compromised component so that the system can still provide service with tolerate degradation level. Further more, all the projects, including the SITAR, uses error detection as the basic input of dynamic configuration.

Different with SITAR, all the above projects use rule-based or statistical-based intrusion detection technology to build error detection mechanism. The reason SITAR uses acceptance testing to detect error of compromised COTS services is, we hope to use acceptance testing to improve the detection on compromise caused by known attack. Through analyzing known impacts, we expect to handle both known and unknown intrusions.

6 Conclusion and future work

In this paper, we discussed the acceptance test methodology and the design of the Acceptance Monitors in the intrusion tolerant system context. A generic Acceptance Monitor architecture is proposed and the various acceptance test measures are discussed. Acceptance test by nature is highly application dependent. Fine tuning in our system helped to achieve high efficiency. To apply the Acceptance Monitor methodology for detecting the compromises of the web servers, we performed a detailed and comprehensive investigation of Bugtraq-advised vulnerabilities. Our conclusion is, although the exploitations of the errors can take quite different forms, the kinds of the compromise impacts are limited. So we classified those compromises according to their impacts and specified the acceptance test measures based on the classification results. Through case studies and experiments, we showed how the Acceptance Monitor can effectively detect the compromises and we also evaluated the coverage of compromise detection.

In our future work, we would like to pursue two directions. First, we plan to improve the communication infrastructure using secured JavaSpaces so we can have enhanced security even when the SITAR architecture itself is under attack. Second, we will implement an open architecture of acceptance testing units to enable the plug and play fashion of Acceptance Monitors.

Acknowledgments

The authors would like to acknowledge the earlier help from Dr. Fengmin Gong and Dr. Frank Jou and helpful discussion with Mr. Charles Killian.

References

[1] Acme Laboratories. mini_httpd - small HTTP server. `http://www.acme.com/software/mini_httpd/index.htm`, 2000.

[2] E. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace back, Traps, and Response*. Intrusion.Net Books, 1st edition, January 1999.

[3] Apache Foundation. `http://www.apache.org`.

[4] R.G. Bace. *Intrusion Detection: Technology Series*. Macmillan Technical Publishing, 2000.

[5] CERT Coordination Center. Denial of Service Attacks. `http://www.cert.org/tech_tips/`, 2001.

[6] Secure Computing Corporation. Intrusion Tolerant Server Infrastructure (ITSI). from DARPA OASIS PI Meeting Presentations, March 2002.

[7] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems.*, 7(1):80–112, January 1985.

[8] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: Principles, Patterns, and Practice.* Addison-Wesley, Menlo Park, CA, 1999.

[9] eEye Digital Security. .ida "code red" Worm. `http://www.eeye.com/html/Research/Advisories/`, 2001.

[10] eEye Digital Security Advisories. Windows 2000 and NT4 IIS .ASP Remote Buffer Overflow. `http://www.eeye.com/html/Research/Advisories/`, April 2001.

[11] Gregory T. Byrd, Fengmin Gong, Chandramouli Sargor, and Timothy J. Smith. Yalta: A Secure Collaborative Space for Dynamic Coalitions. In *IEEE 2nd SMC Information Assurance Workshop*, West Point, New York, 2001.

[12] Cisco Systems Inc. Cisco Scalable Infrastructure. `http://www.j-spaces.com/javaSpaces.htm`, 1999.

[13] Jr. James B.Fenwick and Lori L. Pollock. Issues and Experiences in Implementing a Distributed Tuplespace. *Software Practice and Experience*, 27(10):1199–1232, October 1997.

[14] K. Arnold, editor. *The Jini^{TM} Specifications.* Addison-Wesley, 2 edition, 2000.

[15] Charlie Kaufman, Radia Ferlman, and Mike Speciner. *Network Security: Private Communication in Public World.* PTR Prentice Hall Englewood cliffs, New Jersey, 1995.

[16] S. Kumar and E. H. Spafford. A Software Architecture to Support Misuse Intrusion Detection. In *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.

[17] DARPER Laboratory. Kinetic Application of Redundancy to Mitigate Attacks (KARMA). from DARPA OASIS PI Meeting Presentations, March 2002.

[18] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice.* Springer Verlag, 1990.

[19] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *DARPA Information Survivability Conference and Exposion, 2000. DISCEX'00. Proceedings*, pages 12–26, 2000.

[20] GigaSpaces Technologies Ltd. `http://www.gigaspaces.com`.

[21] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A Real-Time Intrusion Detection Expert System (IDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

[22] Marton and Frambs. A Cyclic Redundancy Checking (CRC) Algorithm. *Honeywell Computer Journal*, 5(3), 1971.

[23] Microsoft TechNet. Windows Web Servers. `http://www.microsoft.com/technet/iis`.

[24] U.S Department of Commerce. Secure Hash Standard. Federal Information Processing Standards Publication, April 1995.

[25] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[26] SecurityFocus. BugTraq Archive. `http://www.securityfocus.com`, 2001.

[27] William Stallings. *Cryptography and Network Security: Principles and Practice, 2nd edition.* Prentice Hall, 1999.

[28] Teknowledge and UC Davis. HACQIT: Hierarchical Adaptive Control for QoS Intrusion Tolerance.

[29] F. Wang, F. Gong, C. Sargor, K. Goševa-Popstojanova, K. S. Trivedi, and F. Jou. SITAR: A Scalable Intrusion Tolerance Architecture for Distributed Services. In *Proceedings of the IEEE 2nd SMC Information Assurance Workshop*, United States Military Academy, West Point, New York, June 5-6 2001.

[30] R. Wang. Intrusion Tolerant Systems Characterization and Acceptance Monitor Design. Master's thesis, NC State University, 2001.