T1B3 1100

Proceedings of the 2001 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 5–6 June 2001

SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services

Feiyi Wang, Fengmin Gong, Chandramouli Sargor,
Katerina Goseva-Popstojanova, Kishor Trivedi, Frank Jou

Abstract— **This paper presents a intrusion tolerant architecture for distributed services, especially COTS servers. It is motivated by two observations: First, no security precautions can guarantee that a system will not be penetrated; Second, mission critical applications need to provide minimal level of services even under active attacks or partially compromised. The emphasis of proposed architecture is on the *continuity of operation*. In this context, *effects* are more important than *causes* because a system will have to deal with and survive an adverse effect long before a determination is made as to whether the cause was an malicious attack, or a accidental failure. We utilize the techniques of both *redundancy* and *diversity* as our building blocks. Five critical components are defined in this proposed architecture: (1) Proxy Servers enforce the service policy specified by current intrusion tolerant strategy and policies determine which COTS servers the request should be "forwarded" to and how the final result be presented. (2) Acceptance Monitors apply validity checks to the response, optionally forward them to Ballot monitors along with indication of checking results. Acceptance monitors also detect signs of compromise on the COTS servers and generate intrusion triggers. (3) Ballot Monitors serve as "representatives" for the respective COTS servers to solve conflicts, and decide a final response through either a majority voting or Byzantine agreement process. The actual process taken will depend on the current level of detected security threat. (4) Adaptive Reconfiguration module receives intrusion trigger information from other modules (including Acceptance Monitors), evaluates threats, the tolerance objectives, cost/performance impact, and generate new configurations for the system. (5) Audit Control verifies the audit records and identifies abnormal behaviors in components by conducting periodic diagnosis tests.**

Keywords— **Intrusion tolerance, intrusion detection and response, distributed system security, adaptive reconfiguration, voting**

Feiyi Wang and Frank Jou are with Advanced Network Research Group, MCNC, Research Triangle Park, NC. Email: {fwang2,jou}@mcnc.org

Fengmin Gong is with Intrusion Detection Technology Division of IntruVert Network Inc. Email: fengmin@intruvert.com

Chandramouli Sargor is with CoSine Communications, Inc., Redwood City, CA. Email: Chandramouli.Sargor@cosine.com

Katerina Goseva-Popstojanova is a research associate in Duke University, Durham, NC. Email: katerina@ee.duke.edu

Kishor Trivedi is a professor in Duke University, Duhram, NC. Email: kst@ee.duke.edu

This work is sponsored by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) under contract N66001-00-C-8057

I. Introduction

In the past, network security research has in general emphasized on 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 system was 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 has mostly concentrated on known and well-defined attacks. This narrow focus of attacks has accounted for both the success and the limitation 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. The results showed that new and novel attacks present formidable challenges to these systems.

As soon as we focus our attention on the attacks themselves, we cannot expect to develop a general protection mechanism because all attacks are not well defined and there are always unknown attacks. Although intrusion tolerance must also deal with intrusions, it is inherently tied to the functions and services that require protection (i.e. to be made intrusion tolerant). It is this focus that makes intrusion tolerance the most promising approach to build our defense from. As a first advantage, we can now develop intrusion triggers by focusing on only those events that pose a threat to the services under consideration instead of on arbitrary events. Second, we can leverage many well developed techniques from the fault tolerant and dependable computing research. As a third advantage, newly developed intrusion tolerance techniques can eventually be used to build new information systems that will be invulnerable to intrusions (to the degree that desirable levels of services can be maintained regardless of intrusions). To overcome the inability of coping with unknown attacks, we need to shift attention from attacks or attacker themselves to the target of protection, which is inherently tied to the functions and services being provided. This implies that *the effect of attack is more important than the cause of the attack* because a system will have to deal with and survive

ISBN 0-7803-9814-9/$10.00

an adverse effect long before a determination is made as to whether the cause was a malicious attack, a natural failure, or human accident. The emphasis here is on the *continuity of operation* or provision of minimal level of services despite active attacks.

SITAR (Salable Intrusion-tolerant Architecture for Distributed Services) is our proposed architecture which aim to overcome above problems and 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 target of protection. Specially, we discuss the framework under web service context to make our presentation tangible. (2) Two specific challenges are addressed in this architecture. The first one is how some of the very basic techniques of fault tolerance (e.g., redundancy and diversity) apply to our target. The second is how we deal with the 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 intrusion tolerant model built within the architecture.

The rest of the paper are organized as follows. Section II describes the overall architecture design, core components and information flow. Section III, IV, V, VI, VII present design details on Proxy Server, Acceptance Monitor, Ballot Monitor, Audit Control and Adaptive Reconfiguration respectively. Section VIII discusses our experimental setup and preliminary testing results. Section IX makes the conclusion remarks.

II. SITAR Architecture

Figure 1 presents a logical view of the proposed service architecture. Everything within the dotted box is part of the newly proposed intrusion tolerant architecture. The block on the right consists of COTS servers. The basic assumption is that COTS systems are vulnerable to intrusions. The proposed architecture will enable us to build intrusion tolerant services out of the existing intrusion vulnerable servers. However, the architecture will not exclude intrusion tolerant servers from being integrated. In fact, if new servers are built as part of an intrusion tolerant service infrastructure, new server access protocols can be defined to explicitly support correlation of individual request and response. With such a protocol, Proxy Servers and Ballot Monitors can easily correlate concurrent service requests.

The general information flow for the architecture is as follows. Proxy Servers represent public access points for the intrusion tolerant services being provided (e.g. a decision support system for military command and control, or a transaction processing system for an E-commerce site). All requests come in to one of the Proxy Servers depending on the service needs. The Proxy Server enforces the service policy specified by the current intrusion tolerant strategy. The policy determines which COTS servers the request should be "forwarded" to, and how the results from these servers should be adjudicated to arrive at the final response. A new request by the Proxy Server to the appropriate COTS servers is made on behalf of the original client, as depicted by the thin lines from the Proxy Servers to the COTS servers. Relevant Ballot Monitors and Acceptance Monitors are also informed of this decision.

When the responses (signified by the thick lines from right to left) are generated by the COTS servers, they are first processed by the Acceptance Monitors. The Acceptance Monitors apply certain validity check to the responses, forwarding them to the Ballot Monitors along with an indication of the check result. The Acceptance Monitors also detect signs of compromise on the COTS servers and produce intrusion triggers for the Adaptive Reconfiguration module.

The Ballot Monitors serve as "representatives" for the respective COTS servers and decide on a final response through either a simple majority voting or Byzantine agreement process. The actual process taken will depend on the current level of detected security threat. The final response is forwarded to the Proxy Servers to be delivered to the remote client.

The Adaptive Reconfiguration Module (ARM) receives intrusion trigger information from all other modules, evaluates intrusion threats, the tolerance objectives, and the cost/performance impact, and generates new configurations for the system. Since it is assumed that any individual component can be compromised, the backup ARM is provided to guard against ARM becoming a single point of failure.

The Audit Control module provides means for auditing the behavior of all the other components in the intrusion tolerant system. All system modules maintain audit logs with signature protection. These logs can be verified through the Audit Control module. Additional diagnostic tests can be conducted through the Audit Control module.

Intrusion triggers are distributed among three sets of modules. The triggers in the Acceptance Monitors are responsible for detecting compromised COTS servers. The triggers in the Proxy Servers are for detecting external attacks, and the triggers in the Audit Control will help the security administrator to monitor the secure operation of all the new functional blocks in our architecture through active auditing.

It should be noted that each of the functional blocks, P_i, B_i, and A_i, only represents the logical function to be executed to satisfy a given service request. Multiple processes may run on a single physical processor, or each process may run on a dedicated processor. The decision will be made dynamically based on the request type, the current threat level, and the system load. Regardless of the process allocation, all these logical blocks, including the Audit

ISBN 0-7803-9814-9/$10.00

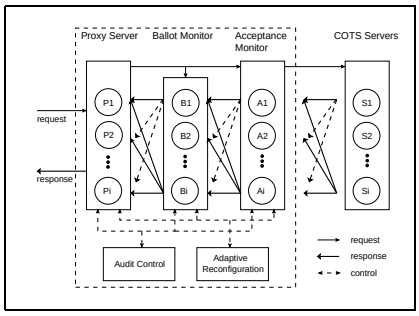

Fig. 1. A generic intrusion-tolerant service architecture

Control module and the Adaptive Reconfiguration module, will maintain full reachability among themselves. This reachability is critical for coordination among them and for implementing fully dynamic reconfigurations.

Our proposed intrusion tolerant system architecture does not require any change to the COTS client or the COTS server applications. In fact, it is completely transparent to both end users and server applications. In the following sections, we discuss each of the key architecture components in further detail.

III. Proxy Servers

In the intrusion tolerant architecture proposed above, the Proxy Servers constitute the set of machines that are visible to the end user and that provide the services in an intrusion tolerant manner. Typically, an end user will not contact a COTS server directly, in fact, the identities (IP addresses) of the COTS servers may not even be publicly known. As far as the end user is concerned, it is the Proxy Server that is providing the service. Many high availability project such as Piranha [1] is designed for Primary/Backup classical scenarios, as illustrated in Figure 2. SITAR Proxy Server aims to provide higher degree of tolerance through shared control on dynamic resource pool, as illustrated in Figure 3. We elaborate this design idea as follows.

As shown in Figure 1, a cluster of Proxy Servers will be utilized in our architecture. While each server will have distinct physical IP addresses that are tied to distinct physical network interfaces, they will also share a pool of virtual IP addresses amongst themselves. Only the virtual IP addresses are made known to the clients. There may be a single pool of virtual IP addresses or there could be one pool per intrusion tolerant service that is being provided. The main advantage in using virtual IP addresses is that it allows easy migration of addresses from one machine to another in case of a fault or intrusion. We will design techniques to share these virtual IP addresses among the active machines in the proxy cluster in such a fashion that as long as even one of the machines in the cluster is active, *all of the virtual IP addresses* advertised to the clients will be available. This is achieved by migrating virtual IP addresses from a faulty Proxy Server to the Proxy Servers that are functioning correctly. Migrating addresses directly makes it possible to do load balancing by moving virtual IP addresses from a heavily loaded machine to a lightly loaded one. Also, since clients access services using virtual IP addresses, migration also enables dynamic reconfiguration of proxies. For instance, under normal circumstances a specific service may be provided only on one Proxy Server. When under attack, the service could be migrated to all the proxies to improve survivability. It must be emphasized that all such migrations/reconfigurations are completely transparent to the end user.

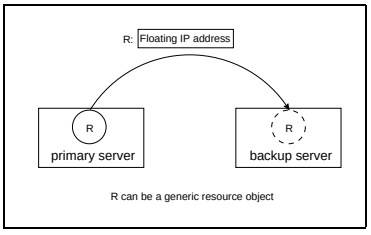

Fig. 2. Primary/backup operation mode

While migrating virtual IP addresses from one Proxy Server to another is in itself fairly simple, the main issue associated with migration is to ensure that the "state" associated with services being proxied is correctly migrated as well. For instance, a Proxy Server will need to maintain state to keep track of requests issued by clients, the particular set of servers, Ballot and Acceptance monitors used to fulfill specific requests, the virtual IP address used by the client to make the request and other such parameters. Clearly, one cannot expect a compromised or faulty Proxy Server to migrate this state upon detection of a fault or intrusion. The state information would therefore need to be shared in such a way that all of the Proxy Servers have a consistent view of the global shared state. There are many possible approaches to sharing such state information. For example, reliable multicast or shared memory techniques could be used to exchange such information. However, there are problems with both approaches. With reliable multicast, there can be a significant impact on network performance especially if the state does not need to be updated frequently. Shared memory implementations

tend to be platform dependent and stand in the way of design diversity. Our preliminary analysis indicates that Javaspaces [2] may be an appropriate mechanism for sharing global state and other control information.

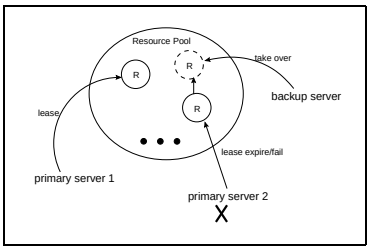

Fig. 3. Shared resource pool operation mode

Javaspaces is a space based distributed computing paradigm that provides a platform independent framework for implementing such coordination mechanisms. Unlike reliable multicast, a state object need not be multicast to all the Proxy Servers. Rather, it is simply written into the space and can be retrieved by any other server that is waiting for updated information. Since it is implemented in Java, it is naturally platform independent and supports design diversity. Moreover, many of the features developed for distributed computing can be utilized for providing higher degree of fault tolerance as well. For instance, Javaspaces defines a transaction operation. A transaction is a collection of operations that are performed atomically. Either all of the operations succeed, or none of them does. If all operations occur successfully, the transaction is said to be committed. If it is not committed, then the transaction is aborted. This is extremely useful for sharing state information. A Proxy Server may ***take*** the state object out of the space and after updating it to reflect new connection requests etc, will ***write*** it back into the space. However, if the Proxy Server crashes while it is updating the state object, the information is lost from the space. A transaction operation automatically ensures that this will not happen. The ***take*** operation from the space will only be committed if the subsequent ***write*** operation into the space is completed as well. Using Javaspaces for coordination will also make it possible to leverage research results from a recently funded DARPA project Yalta [3] at MCNC that is aimed at building a Javaspace based secure collaboration infrastructure in support of Dynamic Coalitions.

The Proxy Servers receive client requests and forward it to the actual COTS servers that will fulfill the request. Depending on the security posture of the overall system, this request might be forwarded to more than one server. Decisions on which servers to forward the requests to may be made on a per request basis (under high threat conditions) or on a session basis (under low threat conditions). The Proxy Server is responsible for coordinating the forwarding of client requests to COTS servers, correlating them with the responses from the Ballot Monitors and forwarding the final result back to the client. We will explore Javaspaces based mechanisms as one possible means of achieving the coordination between the Proxy Servers and other components in our architecture.

The Proxy Servers form the front line of our intrusion tolerant architecture and are most likely to be the targets of external attacks. These attacks can range from simple denial of service attacks to attacks that may compromise one or more of the Proxy Servers. Detection of attacks and possibly compromised Proxy Servers therefore requires for timely action to be taken to reconfigure the system and provide non-interrupted service to the clients. This is accomplished by deploying an IDS on each of the Proxy Servers. The IDS software on each proxy will continually monitor the network traffic for external attacks and will also monitor all other proxies to determine if they are behaving correctly. We intend to utilize and extend *existing research results* from our earlier projects in building the IDS system. Specifically, the recently completed JiNao [4] project at MCNC has lead to the development of a highly extensible intrusion detection system that uses rule based, protocol based and statistical based approaches to detect intrusions. The JiNao IDS can be tailored to our specific environment and deployed on each of the Proxy Servers. Procedures for exclusion/inclusion of physical machines from/into the proxy cluster will also be developed.

When the IDS on a Proxy Server detects an attack or compromised peer, the ARM will be notified. The ARM will evaluate this along with all security relevant information from other modules and decide on whether a reconfiguration is necessary. Reconfiguration for the Proxy Servers includes changing the level of access control imposed on clients, degrees of redundancy used to fulfill a client request and increased auditing.

IV. Acceptance Monitors

Acceptance Monitors process the responses from the COTS servers and apply acceptance tests on them. The responses, along with the results of the acceptance tests, are then forwarded to the Ballot Monitors. Another function of the Acceptance Monitors is to detect intrusions in the COTS servers and alert the Adaptive Reconfiguration module.

An acceptance test is a programmer or developer provided error detection measure in a software module, in the form of a check on the reasonableness of the results calculated, which follows the execution of the module [5]. It usually consists of a sequence of statements which will raise an exception if the state of the system is not acceptable. If any exception is raised by the acceptance test, the module concerned is said to have failed or been compromised. In our case, the module may correspond to the application on the COTS server or the server itself. The acceptance test forms the basis for the recovery block scheme for fault

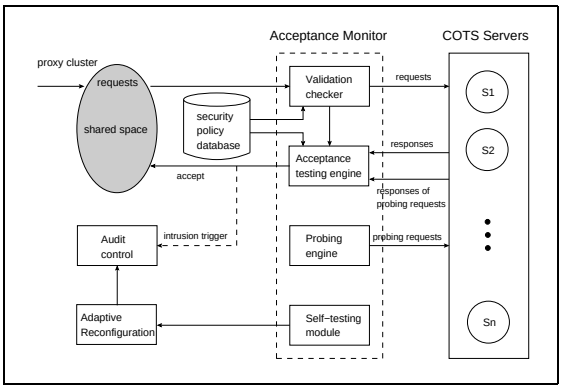

Fig. 4. Architecture of Acceptance Monitor

tolerance in software.

Acceptance tests may be devised such that they test for what a module should do or should not do. One may be simpler or provide a higher degree of independence compared to the other. Currently, no methodology exists for deciding on the most appropriate type of test for a given situation.

Acceptance tests are highly application dependent. The application has to be understood thoroughly in order to devise meaningful tests. The choice for the type of test is also often dictated by runtime, storage, and error detection requirements. Most acceptance tests may be classified into the four categories described below. However, some of these may not be directly applicable for certain services unless suitable changes can be made to the COTS servers.

A. Satisfaction of requirements

In many cases, conditions that must be met at the completion of a module execution, are imposed by the problem statement or the specifications for that module. These conditions can be used to construct the acceptance tests. Independence is a very important consideration in the design of acceptance tests. It may be a difficult and subtle problem to devise an independent satisfaction of requirements.

B. Accounting tests

The kind of applications accounting checks are suitable for, are transaction based applications which 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 the 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. Practices like double entry bookkeeping have been instituted in financial computing and are also applicable in other high volume transaction applications.

C. Reasonableness test

Reasonableness tests detect software/system failures by using precomputed ranges, expected sequences of program states, or other relationships that are expected to be satisfied. Reasonableness tests are based on physical constraints, while satisfaction of requirements tests are based on logical or mathematical relationships. Reasonableness tests are adapted for control or switching systems where physical constraints can determine the range of possible outcomes. Another type of reasonableness test is based on progression between subsequent states. Tests for reasonableness of numerical or state variables are quite flexible and effective for constructing fault tolerant process control software. They permit acceptance criteria to be modified as a program matures.

D. Computer runtime checks

Runtime checks comprise those provided by most current computers as continuous execution sequences which are often hardware implemented. They detect anomalous states such as divide by zero, overflow, underflow, undefined operation code, end of file, or write protection violations. These runtime tests can also serve as additional acceptance tests that cover much wider area and detect more subtle discrepancies. When a runtime error is detected, a bit in a status register is set and control is transferred to an alternate routine. Data structure and procedure oriented tests which are embedded in special software or in the operating system can also be incorporated by runtime checks. State chart based design can perhaps be exploited for state based runtime checks.

In the case of our architecture, we need to determine the acceptance tests based on the proposed applications. A particular category of acceptance tests might be the best suited for a given application while devising certain categories of tests may not be possible. If our architecture is used for a Web sever, accounting tests may not be possible. In this case, the satisfaction of requirements test may be the best suited. Specifications like response time for requests or file format can be used for constructing acceptance tests.

In many of the above examples, the acceptance test may fail due to software failure, deliberate alteration of input or internal data, or compromise of the system. Hence, the dividing line between system reliability and security often becomes blurred. An advantage of this is that techniques for improving system reliability can be used for improving security, and vice versa. Hence acceptance tests can be carefully devised to detect both module failures and system intrusions.

As mentioned earlier, Acceptance Monitors are also designed to detect intrusions in the COTS servers. This can be achieved by using known intrusion detection methodologies like user profiling and application profiling. Historical

system behavior is used as a reference and any deviations or anomalies can be detected using suitable algorithms [6]. It is also possible to design a system which adapts to changes in normal behavior. The challenge in this kind of user or application profiling is to determine the critical parameters to monitor. Some examples of monitored parameters may be CPU activity, number of active users, response time of the application, disk space and memory usage.

If either the acceptance test fails or the user/application profiling triggers an alert, both the Ballot Monitor and the Adaptive Reconfiguration Module can in turn be alerted so that the entire system adapts to the situation.

V. Ballot Monitors

The Ballot Monitors are responsible for deciding on a final response through either a simple majority voting/adjudication or Byzantine agreement process for a given request. Transformations (e.g. calculating the secure hash value of a result) are needed before the voting/adjudication process can begin. Three main transformations can be considered.

- Level 1: Fletcher checksum

The checksum can be implemented by using the Fletcher checksum algorithm, which is used in the OSI network and transport layers [7]. The main reasons for using the Fletcher checksum are: (1) it allows us to have a single numerical value to compare and (2) it is easy to calculate, yet it can detect patterns of data corruption different from the standard Internet ones-complement checksum.

- Level 2: MD5 checksum

This algorithm [8] takes as input a message of arbitrary length and produces as output a 128bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. In essence, MD5 is a way to verify data integrity, and is much more reliable than the Fletcher checksum described above.

- Level 3: Keyed MD5 checksum

A further refined MD5 algorithm called Keyed MD5 checksum is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public key crypto-system such as RSA. In our context, the server's response, be it ASCII or binary, will be encrypted with issuer's private key. This level of security is generally not necessary unless we assume that Acceptance Monitor can be compromised and Ballot Monitor can only trust their input with authentication.

The voting/adjudication algorithms need to adapt to both multiple-level robustness and dynamic reconfigurations. For multiple-level robustness, we assume that one or more voting machines can suffer from faults ranging from crash failure (the simplest) to Byzantine failure (the most sophisticated). We are particularly concerned with how the responses are fed into each voting machine. If each voting machine is mapped to a single server process, i.e., it only receives responses from that server, we need to run an agreement algorithm to get a vote on the final result. However, if each voting machine can "see" all the responses from every server, no agreement algorithm is really needed, i.e., each voting machine can arrive at a final vote independently.

Ballot Monitors support additional flexibilities for providing dependable responses to the Proxy Servers. One option is to designate a trusted voting machine as the announcer who is entrusted with presenting all the voting results to the Proxy Server. The second option is to have a dynamic announcer election process so that different Ballot Monitors may serve as announcers for different requests. The third option is for all the voting machines to present their own version of the result and leave it to the Proxy Server to decide the final answer. Intrusion tolerance and performance tradeoffs are important considerations for selecting among the three options.

VI. Audit control

Audit is defined as the independent examination of records and activities to ensure compliance with established controls, policy, and operational procedures, and to recommend any indicated changes in controls, policy, or procedures [9]. Audit records or audit trails are the chronological logs of the system/component activities. They contain information regarding activities such as logins, command executions, file accesses, etc. By maintaining a record of these activities it is possible to detect abnormal behavior when it occurs in the system/component [10].

In the proposed architecture, each of the components, i.e., the Proxy Servers, the Acceptance Monitors, the Ballot Monitors and the Adaptive Reconfiguration module, maintains its own audit record. The function of the audit control module is to verify the audit records and to identify abnormal behavior in the components by conducting periodic diagnostic tests.

We consider tracking different activities in each of the components depending on their functions. Keeping track of the application usage in such a case would help to detect unauthorized usage. Audit logs on a given module will provide intrusion information on failed attacks on that module, as well as information for intrusion activities coming from other modules that have interacted with this module. When using audit data to detect intrusions, a factor to be considered is the level at which the data should be collected. Audit records generated at a higher level may allow low level attacks go undetected, while low level audits may generate far too much information than can be meaningfully processed.

The Audit Control module also maintains results of di-

ISBN 0-7803-9814-9/$10.00

agnostic tests for all the components. It maintains test suites for singular components or functions as well as the required responses to the tests. The Audit Control generates requests, legal or illegal, to the components and detects abnormal behavior by verifying the responses to the requests. The processed responses are forwarded to the Adaptive Reconfiguration module for further action. For instance, Audit Control will allow security administrators to initiate requests to Proxy Servers, where the correct responses are known a priori. By comparing the responses returned with those expected, faulty or compromised components can be identified.

Intrusion detection mechanisms can be used in the Proxy Servers to detect intrusions directly from outside and in the Acceptance Monitors to detect compromises in the COTS servers. These additional enhancement can help Audit Control to analyze anomaly behavior and generate Intrusion alarms to the ARM and trigger system dynamic reconfiguration.

The IDS deployed at the Proxy Servers may keep track of the source and frequency of requests as well as the usage of service types. Normal patterns can be established and deviations will be detected to signal anomalies. IDSs that have firewall like capabilities are particularly useful at the Proxy Servers for functioning as a security filter.

The IDS at the Acceptance Monitors may model system activity and resource usage data like CPU utilization, amount of real/virtual memory, disk space and paging rates, and applications' response time, etc., to serve as potential indicators of malicious attacks on the COTS servers. One novel technique we are investigating is *software aging* based test [11]. The software aging is referred to the phenomenon in which the state of the software system degrades with time. The primary causes of this degradation are the exhaustion of operating system resources, data corruption and numerical error accumulation. This may eventually lead to performance degradation of the software or crash/hang failure or both. Software aging has been reported in widely used software like xrn, and also in specialized software. This shows quite a bit similarities with DOS (Denial of Services) attack, and the countermeasures developed from this area maybe applicable to the ITS system.

VII. Dynamic reconfigurations

A highly reconfigurable architecture enables multiple fault/intrusion tolerance strategies to exist and allows different levels of security requirements to be supported concurrently in the system. In general, different strategies can be defined by choosing the types of intrusion tolerance techniques, the degrees of redundancy, and the allocation of components, among other factors.

The design of our intrusion tolerant architecture takes into account the fact that the overall system will need to support various configuration options in support of varying levels of security postures. While each component is individually configurable, it is the responsibility of the Adaptive Reconfiguration Module to ensure that that the overall system configuration supports the desired security level. To this end, appropriate strategies for reconfiguration as well as methods to evaluate the security afforded by each possible configuration will need to be devised. We will use both formal and simulation techniques to evaluate overall system security.

COTS services offered via the Proxy Servers can be distributed among the servers based on the service itself, the current load on each Proxy Server or the desired redundancy level. For instance, a particular service may be offered only on a subset of Proxy Servers or on all of the servers. Thresholds for when to migrate services based on current load can be varied depending on service profiles or performance requirements. When under attack, the proxies can be configured to limit access to critical services to specific subnets (reflecting clients of greater importance). Access to noncritical services may be temporarily suspended until the system can recover from the attack or adopt a higher security posture.

Simple checksum schemes such as the Fletcher checksum have the advantage of fast computation and the ability to detect certain patterns of data corruption. However, under hostile circumstances when we can no longer trust a simple checksum value, a more robust checksum solution such as keyed-MD5 will need to be employed by the Ballot Monitor to meet the increased security requirements.

Single ballot voting is suitable in a relatively benign environment where optimum performance is the first priority. However, when notification is received from the ARM that one or more of the voting machines have been potentially compromised, a shift to distributed voting is imperative to increase the tolerance capability. Smooth dynamic adaptation and response under different configurations is well within the consideration of the proposed architecture and design of the Ballot Monitors.

One important consideration regarding strategies for intrusion tolerance is the actual level of security threat. The threat level directly impacts the level of active redundancy (e.g., replication level of servers) required to ensure dependable services. The Active Redundancy configuration, in which all redundant servers assigned to a given request are invoked in parallel all the time, is necessary if the threat level demands it. At the other end of the spectrum, zero redundancy is needed if there is no threat at all. An intermediate strategy will designate redundant servers as primary, secondary (called backup when the replication level is 2), tertiary, and so on. The primary server is invoked first to process the request and additional servers are invoked for a given request only when necessary. In the absence of faults (accidental or intentional) backup servers do not

ISBN 0-7803-9814-9/$10.00

need to be invoked at all. These servers can potentially be used to serve other requests. The downside, however, is when the server we choose to invoke first turns out to be compromised. This implies that one or more of the servers will have to be invoked successively before an acceptable result is obtained. The price paid in this case is the long latency in satisfying this particular request. This demonstrates another aspect of the cost/performance tradeoff in intrusion tolerance design – the overall service throughput versus the response time for individual requests.

VIII. Experimentation

We are performing several basic experiments over our test-beds at MCNC and Duke in order to fully evaluate the solution. The initial Web servers will include Netscape server and Apache server, running on PCs and SUN workstations. The attack tools will be drawn from a variety of sources, including existing tools from active and past MCNC projects, other researchers, and hacker sites on the Internet. After some experiments on each individual testbed, the two test-beds will be connected through NCNI GigaPOP network and additional experiments will be conducted between the two campuses. Three main types of experiments are planned.

- *Basic functionality testing.* This set of experiments will utilize controlled attacks that exercise intrusion triggers, intrusion tolerant modules, and the basic reconfiguration strategies. Our intent is to test and demonstrate the basic operation of the proof-of-concept capabilities. These experiments will be conducted independently on both testbeds at MCNC and Duke, with coordination between the two efforts.
- *Robustness testing.* This set of experiments will also use controlled attacks. The emphasis for these experiment is on exercising the system functions for longer durations and with more realistic application scenarios. Our aim is not at exhaustive testing of error conditions but rather to get a reasonable level of comfort with the robustness of the implementation. These experiments will also be performed independently on the two test-beds at MCNC and Duke.
- *Quantitative measures of performance.* This set of experiments is designed to explore and measure the performance limits of the prototype. We will experiment with several configurations of the system to sample the tolerance performance spectrum. We will attempt to correlate the analytical/simulation results with results from experimental measurements. These tests will be performed using both test-beds, connected via the NCNI GigaPOP. Possible measures to be studied include:
  - Basic latencies for the system to react to intrusion triggers.
  - Replication level and throughput tradeoffs for serving web page requests.
  - Scalability properties of the coordination and control design of the system.
  - Scalability and tradeoff properties of each of the key intrusion tolerant modules (Acceptance Monitor, Ballot Monitor, Proxy Server, Audit Control, and Adaptive Reconfiguration).

Some of these measures will be highly dependent on the testing conditions (e.g., system load). We will document as accurately as possible the testing environment to facilitate accurate interpretation of the results. There are several areas of IA&S programs with which we expect to find opportunities for integration such Intrusion Detection Systems and Fault Tolerant Networks. We expect to define or implement "hooks" for integration with other intrusion detection systems.

IX. Conclusions

This paper presented a scalable intrusion tolerant architecture for distributed services. This architecture consists the following novel aspects: (1) We focus on a generic class of services (COTS components) as target of protection. (2) We exercise basic fault tolerance techniques such as *redundancy* and *diversity* in security domain. (3) We explore dynamic configuration strategy for different level of tolerance needs and justify cost/performance tradeoff. (4) The tolerance capability applies to both external and internal attacks.

Acknowledgments

The authors would like to acknowledge the suggestions of many people in SITAR team: Rong Wang, Karen Litwin, and Travis Walsh.

References

[1] Redhat Piranha, "Load-balanced generic service clustering enviroment," `http://sources.redhat.com/piranha/`, 2000.

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

[3] Yalta, "Yalta:A Collaborative Space for Secure Dynamic Coalitions," `http://www.anr.mcnc.org/~yalta`, 2000.

[4] Y. F. Jou, F. Gong, C. Sargor, X. Wu, S. F. Wu, H. C. Chang, and F. Wang, "Design and implementation of a scalable intrusion detection system for the protection of network infrastructure," in *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 1999, pp. 422–434.

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

[6] R.A. Maxion, "Anomaly Detection for Diagnosis," in *Proc. of 20th IEEE Intl. Symposium on Fault Tolerant Computing*, 1990.

[7] A. Mckenzie, "ISO Transport Protocol Specificatioin. ISO DP 8073," RFC 905, Apr. 1984.

[8] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Apr. 1992.

[9] SANS, "Security Glossary," `http://www.sans.org/newlook/resources/glossary.htm`, 1999.

[10] Dorothy E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

[11] K.Vaidyanathan and K.S.Trivedi, "A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems," in *Proc. of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, Boca Raton, Florida, Nov. 1999, pp. 84–93.