ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

An Intrusion Tolerance Approach for Internet Security

G.Srinivas Reddy[1], Prof.T.Venkat Narayana Rao[2], D V S Nagendra Kumar[3]

Department of C.S.E, Mahatma Gandhi Institute of Technology, Hyderabad, A.P, India [1, 3]

Department of C.S.E, Guru Nanak Institutions Technical Campus, Ibrahimpatnam, Hyderabad, A.P, India[2]

***ABSTRACT*: The Internet has become essential to most enterprises and many private individuals. However, both network and the computer systems connected to it are still vulnerable to attacks which are becoming more frequent than ever. To face this situation, traditional security techniques are insufficient and fault-tolerance techniques are becoming increasingly cost-effective. Nevertheless, intrusions are very special faults, and this has to be taken into account when selecting the fault-tolerance techniques. In classical dependability, fault tolerance has been the workhorse of many solutions. Classical security-related has less privileged solutions with a few exceptions towards intrusion detection and prevention. The paper focuses on the fundamental concepts fault tolerance and security. The main strategies and mechanisms for architecting IT systems are discussed in the study along with the recent advances in secured distributed IT system architectures.**

Keywords: Intrusion, diagnosis, policies, checksum, regime.

I. INTRODUCTION AND REVIEW

There is a significant facet of research on distributed computing architectures, methodologies and algorithms, both in the fields of dependability and fault tolerance to ensure security and information assurance. A new approach has slowly emerged during the past decade, and gained impressive momentum recently i.e. intrusion tolerance (IT). This relates to the notion of handling react, counteract, recover and mask a wide set of faults encompassing intentional and malicious faults i.e. collectively called intrusion. This may lead to failure of the system security properties if nothing is done to counter their effect on the system state. In short, instead of trying to prevent every single intrusion, but tailor the system trigger mechanisms that prevent the intrusion from generating a system failure. Dependability is the system property that integrates such attributes as reliability, availability, safety, security, survivability and maintainability.

A. Fault prevention and Fault tolerance

Fault prevention is attained by quality control techniques employed during the design and manufacturing of hardware and software [1]. Such techniques include structured programming, information hiding, modularization, etc., for software, and rigorous design rules for hardware. Shielding, radiation hardening, etc., intend to prevent operational physical faults, while training, rigorous procedures for maintenance, 'foolproof' packages, intend to prevent interaction faults. Firewalls and similar defences intend to prevent malicious faults. Fault tolerance is intended to preserve the delivery of correct service in the presence of active faults [12]. It is generally implemented by error detection and subsequent system recovery [5,8]. Error detection originates an error signal or message within the system. An error that is present but not detected is a latent error**.** Concurrent error detection are pre-emptive error detection are two classes of error detection techniques.

B. Fault handling and Fail-controlled systems

Fault handling involves four steps

- ***Fault diagnosis***, which identifies and records the cause(s) of error(s), in terms of both location and type,
- ***Fault isolation***, which performs physical or logical exclusion of the faulty components from further participation in service delivery, i.e., it makes the fault dormant,
- ***System reconfiguration***, which either switches in spare components or reassigns tasks among non-failed components,
- ***System re-initialization***, which checks, updates and records the new configuration and updates system tables and records.

Fail-controlled systems are designed and implemented so that they fail only in specific modes of failure described in the dependability requirement and that to an tolerable extent. A system whose failures are, to an acceptable extent, halting failures only is a fail-halt or fail-silent system. Some mechanisms of error detection are directed towards both malicious and accidental faults (for example memory access protection techniques) and schemes have been proposed for

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

the tolerance of both intrusions, physical faults as well as for tolerance of malicious logic, and more specifically of viruses, either via control flow checking or via design diversity[13]. The section II discusses the architecture, servers and proxies of the system. Section III discuses the implementation issues and section IV discuses finding of the proposed system. Section V discusses summary of work in conclusion.

II. ARCHITECTURE

The architecture, shown in Figure.. consists of a redundant tolerance proxy bank that arbitrate requests to a redundant application server bank, with the entire configuration monitored by a variety of mechanisms to ensure content integrity including intrusion-detection systems (IDSs). The proxies and application servers are redundant in capability but diverse in implementation, so that they are unlikely to be simultaneously vulnerable to the same attack. All platforms and complex interfaces within the system are instrumented with a diversity of monitors based on signature engines, probabilistic inference, and symptom detection [11]. Given the reports from this monitoring subsystem, the management function undertakes a variety of tolerance policy responses [5-7]. Responses include enforcing more strict agreement protocols for application content but with a reduced system bandwidth, filtering out requests from suspicious clients, and restarting platforms or services that appears corrupt.

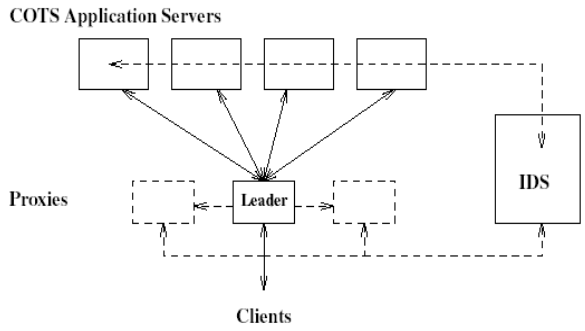

Figure 1. Schematic view of the intrusion-tolerant server architecture

A. *Assumptions*

It is assumed that attackers do not have physical access to the configuration. We assume that no more than a critical number of servers is in undetected compromised state at any given point of time. Our agreement protocols assume that all non-faulty and non-compromised servers give the same answer to the same request. Thus, the architecture is meant to provide content that is static from the end user's [4]. The reply to a request can be the result of substantial computation, on a condition that the same result be obtained by the different application servers as shown in figure 1. Target applications include plan, catalog, and news distribution sites. The system content can be updated periodically, by suspending and then resuming the proxy bank, but we do not address specific mechanisms for doing so. Survivable storage techniques can be used in the future to build a separate subsystem to handle write operations [9], while retaining the current architecture for read requests. The architecture focuses on availability and integrity, and does not address confidentiality. Usually we do not defend against insider threat or network flooding denial of service attacks.

B. *Architecture Components*

i. *Application Servers*

In this architecture, the domain-specific functionality visible to the client and is provided by a number of application servers. These provide equivalent services, but on diverse application software, operating systems and platforms, so that they are unlikely to be vulnerable to common attacks and failure modes. They include IDS monitoring but are otherwise ordinary platforms running diverse COTS software. In our instantiation these servers provide Web content. For our content agreement protocols to be practical we assume there are at least three different application servers; a typical number would be five or seven. However, the venture can add as many of these as desired, increasing the overall performance and intrusion-tolerance capabilities.

ii. *Tolerance Proxies*

The central components of our architecture are one or more tolerance proxies. Proxies mediate client requests, observe the state of the application servers and other proxies, and dynamically adapt the system operation according to the reports from the monitoring subsystem. One of the proxies is designated as the leader. It is responsible for filtering, sanitizing, and forwarding client requests to one or more application servers, implementing a content agreement protocol that depends on the current management, while balancing the load. In the presence of perceived intrusions, increasingly rigorous regimes are used to validate server replies. The regime is selected according to a chosen policy, depending on reports from the monitoring subsystem and on the outcome of the agreement protocol currently in use. Optional auxiliary proxies monitor all communication between the proxy leader and the application servers, and are themselves monitored by the other proxies and the sensor subsystem . Our current design and implementation focuses on the case of a single proxy. The tolerance proxies run only a relatively small amount of custom software, so they are much more agreeable to security solidification than more complex application servers, whose security properties are also more difficult to verify.

iii. *Intrusion Detection System*

The third main component of our architecture is an intrusion-detection system (IDS), which analyzes network traffic, the state of the servers and to report suspected intrusions. Some IDS modules execute on one or more dedicated hardware platforms, while others reside in the proxies and application servers.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

iv. Functional Overview

When a client request arrives, the following steps are performed:

Step 1: The proxy leader accepts the request and checks it, filtering out malformed requests.

Step 2: The leader forwards the request, if valid, to a number of application servers, depending on the current agreement regime.

Step 3: The application servers process the request and return the results to the proxy leader. If sufficient agreement is reached, the proxy forwards the content to the client.

Step 4: The regime is adjusted according to the outcome of the content agreement and reports from the monitoring subsystem.

Step 5: The auxiliary proxies, if present, monitor the transaction to ensure correct proxy leader behavior.

C. Monitoring Subsystem

The monitoring subsystem includes a diversified set of complementary mechanisms, including the IDS. The information collected by the monitoring subsystem is aggregated into a global system view, used to adapt the system configuration to respond to suspected or detected threats and malfunctions, as described in the Section. Diversity helps make the monitoring subsystem itself intrusion-tolerant, since it may still be effective if some of its components fail.

i. Intrusion Detection

Our intrusion-detection systems feature diverse event sources, inference techniques, and detection paradigms. They include EMERALD host, network, and protocol monitors, as well as embedded application monitors. Different sensors cover different portions of the detection space, and have different detection rates, false alarm ratio and operational conditions. Their combination allows detecting more recognized attacks, as well as anomalies arising from unknown ones [10]. The advantages of heterogeneous sensors come at the cost of an increased number of alerts. To significantly manage them, they must be aggregated and correlated. Alert correlation can also detect attacks consisting of multiple steps.

ii. Content Agreement

The proxy leader compares query results from different application servers, according to the current agreement regime, as described above. If two or more results fail to match, this is viewed as a suspicious event, and suspect servers are reported [3].

iii. Challenge Response Protocol

Each proxy periodically launches a challenge response protocol to check the servers and other proxies. This protocol serves two main purposes: It provides a control that checks the live-ness of the servers and other proxies. If a proxy does not receive a response within prescribed delay after emitting a challenge, it elevate an alarm. The protocol checks the integrity of files and directories located on remote servers and proxies. The integrity of application servers is also verified indirectly by content agreement, as mentioned above [2]. However, a resolute attacker could take control of several servers and modify only rarely used files. The proxy could check that each response corresponds to the specified challenge by keeping a local copy of all sensitive files and running the same computation as the server, but this imposes an extra administrative and computational load on the proxy. Instead, we can utilize the fact that servers and proxies are periodically rebooted as a measure for software rejuvenation.

iv. Online Verifiers

As part of the design process, we express the high-level behavior of the proxy as a reactive system that can be formally verified. An abstraction of the system is described using a finite-state omega-automaton and the properties of interest are specified in temporal logic. The high-level specifications can be formally verified using model checking. However, this does not guarantee that the implementation (the concrete system) meets the corresponding requirements. To fill this gap, we introduce online verifiers, which check that the abstract properties hold while the concrete system is running, by matching concrete and abstract states. If an unexpected state is reached, an alarm is raised. Only temporal safety properties can be checked in this way, however, the challenge response heartbeat described here provides a complementary live-ness check. The online verifiers are generated by annotating the proxy Java program source. Since safeness is not guaranteed, and only high-level properties are checked, this does not detect lower-level faults such as buffer overflows.

D. Adaptive Response

We now describe how the system responds to state changes reported by the monitoring subsystem described in the previous section.

i. Agreement Regimes and Policies

The role played by the proxy leader is to manage the redundant application servers. The proxy decides which application servers should be used to answer each query, and compares the results. The number of servers used trades of system performance against confidence in the integrity of the results. Figure 2 and 3 presents the main steps in the content agreement protocol executed by the proxy leader. This protocol is parameterized by an agreement regime, which must specify, at each point in time:

- Which application servers to forward the request to.
- What constitutes sufficient agreement among the replies which servers, if any, to report as suspicious to the monitoring subsystems. The most important regime should be dynamically adjusted in response to alerts, a policy that specifies the action to take next if no agreement is achieved, and which regime to use in response to various events. A policy must also specify how to respond if intrusions or other undesirable

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

conditions are detected, and when to return to a less stringent agreement regime. The transition to a stricter regime can also occur as a result of administrative action.

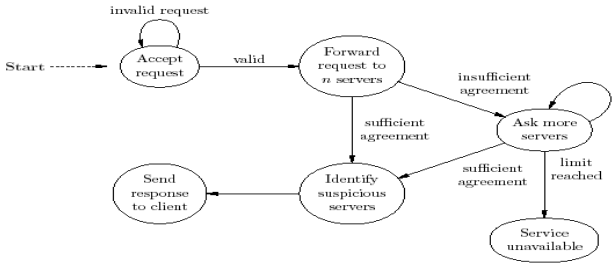

Figure 2. Generic content agreement protocol

ii. Responses to Alerts

Alerts are indications of possible attacks or compromises. Experience shows that all systems connected to the Internet are regularly attacked, or at least probed. IDS components generate a large number of legitimate alerts that do not always indicate system compromise, as well as numerous false alarms [15]. Other parts of the monitoring subsystem generate alerts as well: the content agreement and challenge-response protocols provide alerts that identify likely compromises. Although alerts can arise from direct detection of an attack, many report symptoms of a compromise that has already occurred [10]. While attack detection usually indicates only the possibility of a compromise, symptom detection can reliably recognize a compromise after it has occurred.

iii. Multi-proxy Protocols

Our implementation has focused on detection and response protocols for multiple application servers mediated by a single leader proxy, and no auxiliary ones. However, we are developing the general case, where redundant proxies mitigate the weakness presented by the leader as a single point of failure. Our multiproxy design includes three proxies and is intended to tolerate the compromise or failure of one of them. Proxies communicate with each other via a multicast channel implemented using a local Ethernet link as shown in figure. The decision to expel a proxy leader or auxiliary requires unanimity between the two others. If they do not agree, the accuser is suspect but not immediately shut down [11]. After a delay, the accuser may persist and reinitiate the expel protocol. After a fixed number of "false accusations, the accuser is itself considered faulty and restarted. This reduces the risk of prematurely removing a non-compromised proxy that has accused another by mistake.

III. IMPLEMENTATION

The local support dimension of the architecture consists essentially of the operating system augmented with appropriate extensions. We have adopted Java as a platform-independent and object-oriented programming environment thus our middleware, service and application software modules are constructed to run on the Java virtual Machine (JVM) run-time environment. The run-time support thus includes abstractions of typical local platform services such as process execution, inter-process communication, access to local persistent storage, and protocol management [15]. Our instantiation of the architecture provides intrusion tolerant Web services. The Web servers used are Apache 1.5, Microsoft IIS 5.0 running under MS Windows 2000. Figure 3 shows the main components of our proxy implementation. The regime manager is responsible for executing the content agreement protocol.

A. Monitoring Subsystem

The implemented monitoring subsystem includes a variety of intrusion detection sensors and alert correlation engines, as follows:

- Network-based sensors detect a variety of network attacks and probes in real time. These sensors run on a dedicated machine that monitors the traffic between the clients and the proxy, and the private subnet between the proxy and the application server bank.

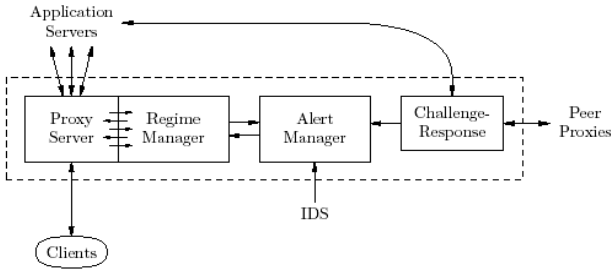

Figure 3. High-level view of proxy implementation

eXpert-Net is a suite of network-based sensors, each of which focuses on a specific network protocol. It features an attack knowledge base and inference engine developed using a forward-chaining rule-based system generator, P-BEST. eXpert-Net can detect complex attacks and variations of known attacks. For example, by performing session and transaction reconstruction, eXpert-HTTP, a sensor for monitoring Web traffic, detects attacks that would be missed if the analysis were performed on a per-packet basis [14].

B. Content Agreement using MD5 Checksums

A basic function performed by the proxy is checking that the pages returned by two or more application servers match. To improve the efficiency of this process, we use MD5 checksums, which the servers compute for each page served [13]. These checksums are cryptographically strong: producing a fake page that matches a given MD5 checksum is an intractable problem given the current and foreseeable state of the art. When comparing content from several servers, only the MD5 checksums need to be retrieved from all but one, which is queried for both checksum and content. The proxy verifies that the checksums match; if so, it also verifies that the received content matches the common MD5.This has the following

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

advantages: Internal network bandwidth and proxy memory requirements are reduced

- When querying multiple servers, it is more efficient to compare the checksums than the full n pages; verifying a single MD5 is relatively inexpensive (linear-time in page size)
- The leader proxy can keep a cache of checksums, to be checked when lower agreement regimes are used. If cache hits occur, the proxy can operate at a higher assurance level despite using fewer application servers for content.

C. Policy Implementation

Our implementation supports a variety of policies based on a generalization of the simple agreement regimes. In general, each regime is specified by a pair (n, k), where n is the number of servers to query, and k is the minimum number of servers that yield sufficient agreement in that regime [14]. The client request is forwarded to n servers, and a response is considered correct if there is agreement between k of them; otherwise, the function δ is used to identify a new pair (n^1; k^1), which dictates the new regime, querying n^1 – n extra servers. This is repeated until satisfactory agreement is obtained, or the panic state is reached, in which case the system is considered too corrupted to function. The alert manager is notified of the content agreement results. Implemented policies can range from efficiency-conscious ones that initially ask only a few servers and query one additional server when needed, to integrity-conscious ones that query more servers initially and immediately query all servers when in doubt.

IV. RESULTS AND CONCLUSION

Our intrusion-tolerant architecture combines a variety of traditional security mechanisms, as well as concepts from fault tolerance and formal verification. The figures 4. and 4.a to 4.l clearly summarizes these mechanisms, and the protective functions they play and outcomes in terms of snapshots which include activities and results pertaining to proxies , attacks ,clients and cache states etc.

Snapshots

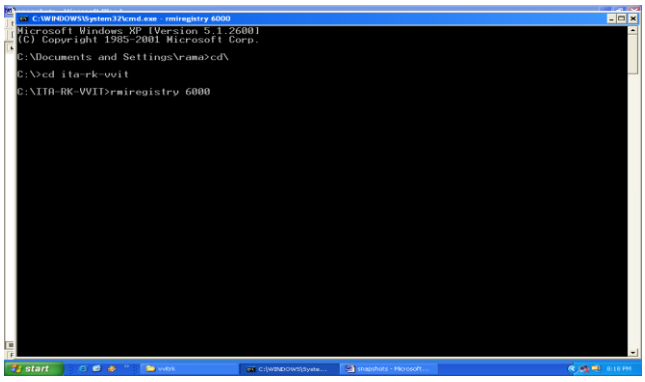

Figure 4. Running rmi registry

Figure 4. shows rmi registry at port number 6000.

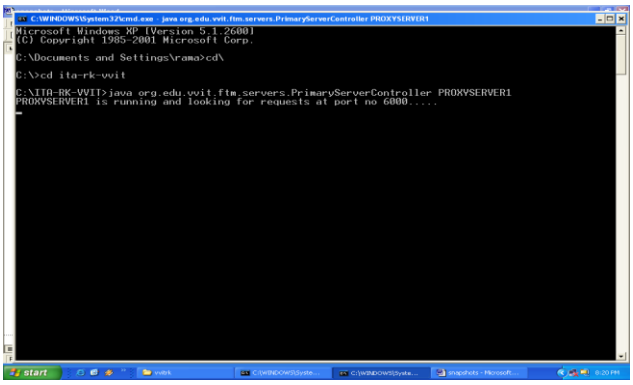

Figure 4a. Proxyserver1 initialization

Figure 4a. shows the initialization of proxyserver1, it is listening at port number 6000.

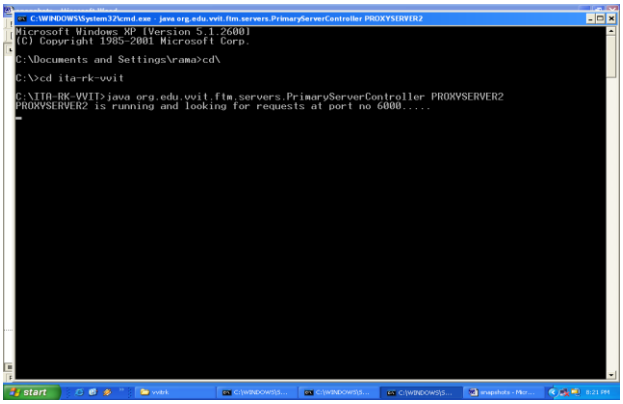

Figure 4b. Proxyserver2 initialization

Figure 4b. shows the initialization of proxyserver2, it is listening at Port number 6000.

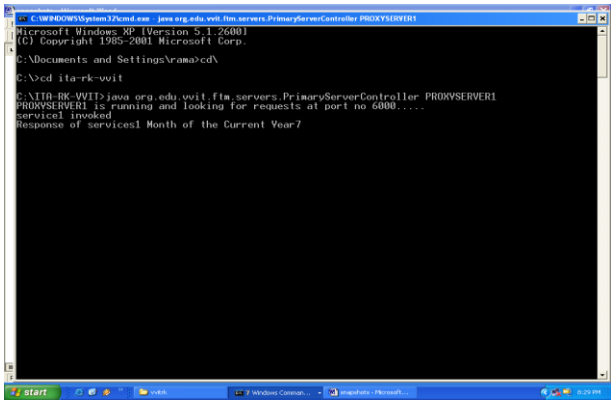

Figure 4c. State of proxyserver1

Figure 4c. shows the information available at proxyserver1 regarding service1 invocation by client1.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

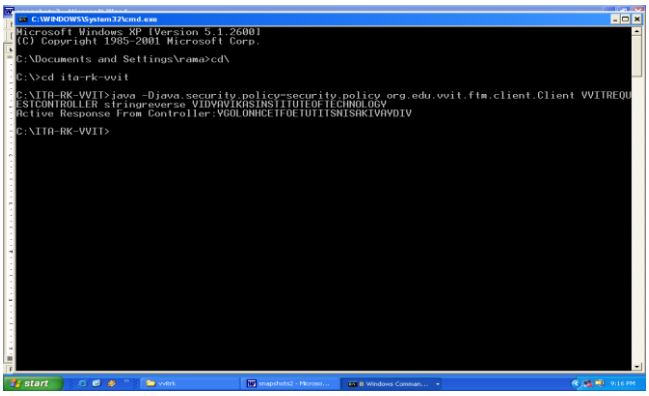

Figure 4d. Running client2 for string reverse service

Figure 4d. depicts the result of service3 (string reverse) provided to client2. The service is provided by any of the three proxies, which are listening for client requests. Here we are assuming that all proxies are functioning properly (i.e., None of the proxies are attacked).

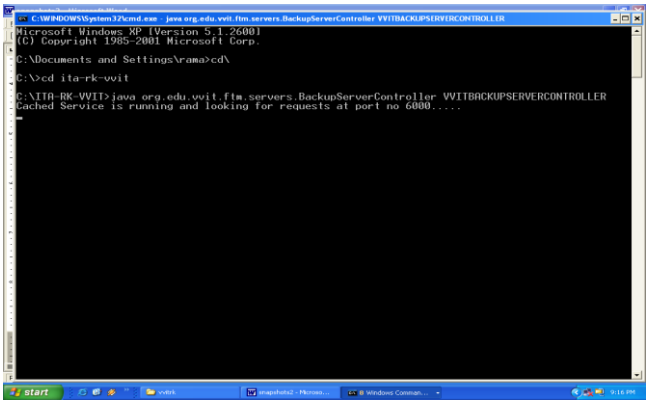

Figure 4e. State of backup server

Figure 4e. shows the outcome available at backup server for service3. When majority of the proxies functioning properly the service is provided by any one the proxy server.

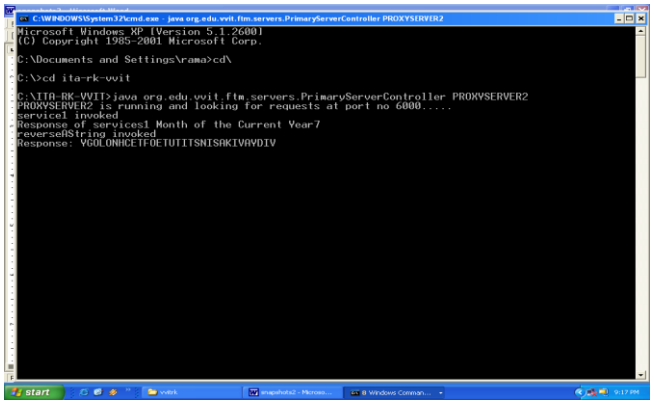

Figure 4f. State of Proxyserver2

Figure 4f. shows the information available at proxyserver2 regarding service3 invocation by client2

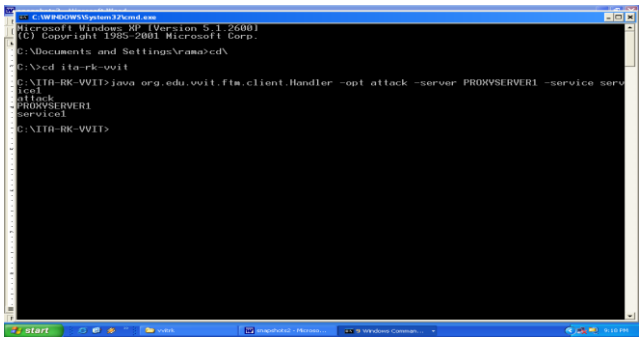

Figure 4g. Attack on proxyserver1 for service1

Figure 4g. shows the attack on proxyserver1 on servive1. If the clients, for service1, make any requests proxyserver1 may not generate the correct result.

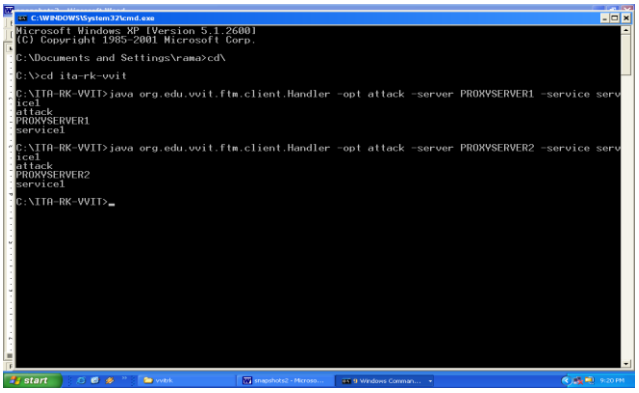

Figure 4h. Attack on proxyserver2 for service1

Figure 4h. shows the attack on proxyserver2 on servive1. If the clients, for service1, make any requests proxyserver2 may not generate the correct result.

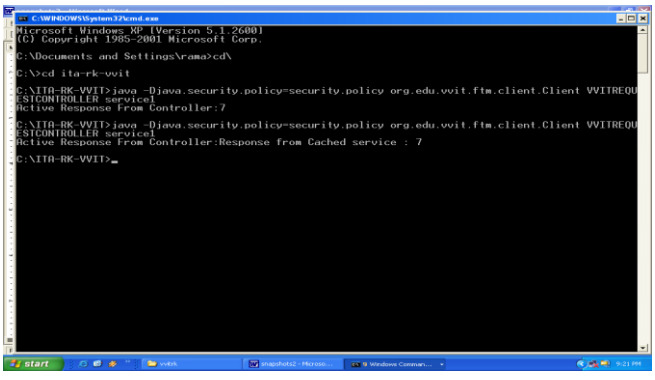

Figure 4i. Running client1 for service1

Figure 4i. Shows the result of services to client1. Here the service is provided by backend server (i.e., Cached Service), because among three servers two of them are already under the attack. When the result is generated by proxyserver1, proxyserver2, and proxyserver3 it will be compared with result available with cached service. Here in the above situation majority for the correct result is less than the

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

required one. So the result available with backend server will be served to the client.

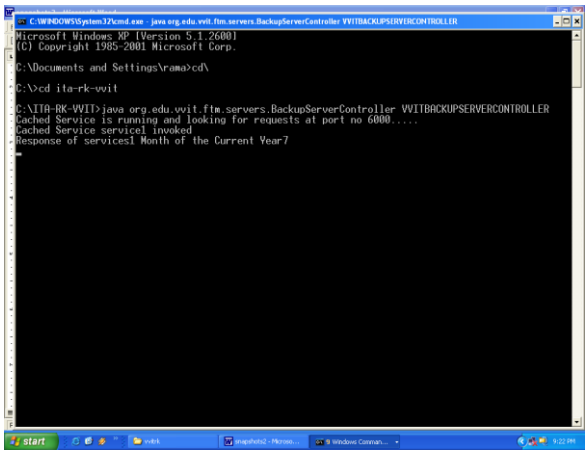

Figure 4j. State of the Cached service

Figure 4j. depicts that the result is sent from cached server not from the proxies.

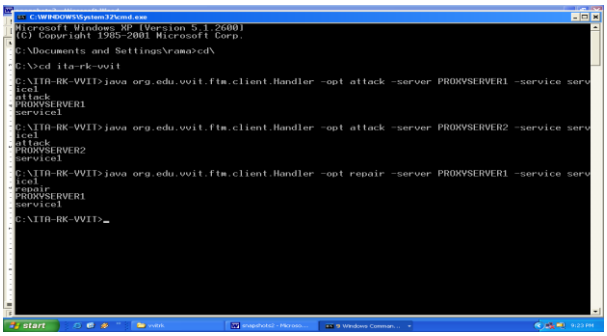

Figure 4k. Repair proxyserver1 for Service1

Figure 4k. shows the repair made to the proxyserver1 for service1. Now proxyserver1 is ready to generate correct results.

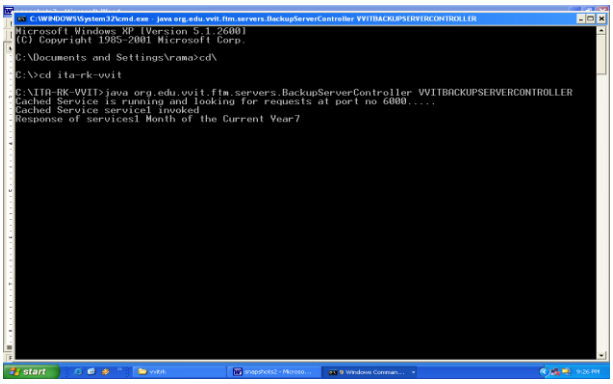

Figure 4l. State of the backup server

Figure 4l. shows that the service is provided by the controller not by the backup server. Below diagram shows that the result generated by proxyserver1. The result available with proxyserver1 is correct.

V. CONCLUSION

The proposed system is adaptive, responding to alerts and intrusions, trading off performance against confidence in the integrity of the results. Our architecture allows a wide variety of response policies to be implemented, depending on the environmental assumptions and with cost-benefit analysis. We have presented an overview of the main concepts and design principles relevant to intrusion tolerant architectures. Given the current rate of attacks on Internet, and the large number of vulnerabilities in contemporary computing systems, intrusion tolerance appears to be a promising technique to implement more secure applications, particularly with diversified hardware and software platforms. There is a price to pay, since it is expensive to support multiple heterogeneous systems. However, this is probably the price that must be paid for security in an open, and in a uncertain world.

REFERENCES

1. M. Almgren and U. Lindqvist. Application-integrated data collection for security monitoring. In Recent Advances in Intrusion Detection (RAID 2001), volume 2212 of LNCS, pages 22{36. Springer-Verlag, Oct. 2001.
2. D. Curry and H. Debar. Intrusion detection message exchange format: Data model and extensible markup language (XML) document type definition, Nov. 2001. Work in progress.
3. Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In Proc. Intl. Symposium on Security and Privacy, pages 110{121. IEEE press, May 1991.
4. G. J. Holzmann. Design and Validation of Computer Protocols. Prentice Hall, Engelwood Cli_s, NJ, 1991.
5. Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: Analysis, module and applications. In 25th Symposium on Fault Tolerant Computing, pages 381{390. IEEE Computer Society Press, June 1995.
6. Real Secure server sensor policy guide version 6.0, May 2001. http://www.iss.net.
7. U. Lindqvist and P. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pages 146{161. IEEE press, May 1999.
8. U. Lindqvist and P. Porras. eXpert-BSM: A host-based intrusion detection solution for Sun Solaris. In Proc. of the 17th Annual Computer Security Applications Conference, Dec. 2001.
9. P. Liu and S. Jajodia. Multi-phase damage con_nement in database systems for intrusion tolerance. In Proc. 14th IEEE Computer Security Foundations Workshop, pages 191{205, June 2001.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

10. R. Permeh and M. Mai_ret. .ida \Code Red" worm. Security Advisory AL20010717, eEye Digital Security, July 2001. http://www.eeye.com/html/ Research/Advisories/AL20010717.html.

11. P. Porras. Mission-based correlation. Personal communication, SRI International, 2001. P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In National Information Security Conference, Oct. 1997.
12. P. Porras and A. Valdes. Live traffic analysis of TCP/IP gateways. In Proc. Symposium on Network and Distributed System Security. Internet Society, Mar. 1998.
13. G. R. Ranger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, S. Oguz, V. Pandurangan, C. A. N. Soules, J. D. Strunk, and J. J. Wylie. Survivable storage systems. In DARPA Information Survivability Conference and Exposition II, pages 184{195. IEEE Computer Society, June 2001.
14. R. Rivest. The MD5 message digest algorithm. Internet Engineering Task Force, RFC 1321, Apr. 1992.
15. L. Rodrigues and P. Verissimo. xAMp: a multi-primitive group communications service. In 11th Symposium on Reliable Distributed Systems, pages 112{121, Oct. 1992.

Biography

G. Srinivas Reddy, received M.Sc. in Applied Electronics from Osmania University, Hyderabad, India, holds a M.Tech in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India. He has 9 years of experience in teaching at various Engineering Colleges. He is presently Assistant Professor at Mahatma Gandhi Institute of Technology, Hyderabad, A.P, INDIA. He is currently working on research areas which include Digital Image Processing, Digital Watermarking, Data Mining, Network Security and other Emerging areas of Information Technology. He is pursuing Ph.D.

Professor T.Venkat Narayana Rao, received B.E in Computer Technology and Engineering from Nagpur University, Nagpur, India, M.B.A (Systems), holds a M.Tech in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India and a Research Scholar in JNTU. He has 21 years of vast experience in Computer Science and Engineering areas pertaining to academics and industry related I.T issues. He is presently Professor, Department of Computer Science and Engineering, Guru Nanak Institutions Technical Campus, Ibrahimpatnam, R.R.Dist., A.P, INDIA. He is nominated as Editor and Reviewer to 27 International journals relating to Computer Science and Information Technology. He is currently working on research areas which include Digital Image Processing, Digital Watermarking, Data Mining, Network Security and other Emerging areas of Information Technology.

D.V.S. Nagendra Kumar received his B.Tech, and M.Tech in Electronics and Communication Engineering from JNTU Hyderabad, A.P., India. He has taught at UG and PG level in various Engineering Colleges. He is presently Assistant Professor at Mahatma Gandhi Institute of Technology, Hyderabad, A.P, INDIA. He is currently working on research areas which include Digital Image Processing, VLSI Design and Radar signal processing.