International Journal of Computer Science and Informatics

Volume 3 | Issue 3 Article 7

January 2014

CONATION OF A STANDARD INTERRUPTION-LENIENT MANNER FOR WEB SERVERS

SK. JOHNY BASHA
CSE Department, Universal College of Engineering, Guntur, Andhra Pradesh, India,
shaikhjanibasha@gmail.com

B. BHANU PRATAP
CSE Department, Universal College of Engineering, Guntur, Andhra Pradesh, India,
bhanupratapcse@gmail.com

E. SRINIVAS
CSE Department, Java Institute of Science, Hanmakonda, Andhra Pradesh, India,
srinivaseku81@gmail.com

K.S.S. RAM PRASAD
CSE Department, Universal College of Engineering, Guntur, Andhra Pradesh, India, sivsai@gmail.com

CONATION OF A STANDARD INTERRUPTION-LENIENT MANNER FOR WEB SERVERS

[1]SK. JOHNY BASHA, [2]B. BHANU PRATAP, [3]E. SRINIVAS & [4]K.S.S. RAM PRASAD

[1, 2, 4] CSE Department, Universal College of Engineering, Guntur, Andhra Pradesh, India
[3] CSE Department, Java Institute of Science, Hanmakonda, Andhra Pradesh, India
Email: shaikhjanibasha@gmail.com, bhanupratapcse@gmail.com, srinivaseku81@gmail.com,sivsai@gmail.com

Abstract— Now-a-days, more and more information systems are linked to the Internet and offer Web interfaces to the general public or to a limited set of users. This paper proposes a standard manner to implement interruption-lenient Web servers. This manner is based on idleness and diversification principles in order to increase the system resilience to assaults: usually, an assault targets particular software, running on a particular platform, and fails on others. The manner is composed of redundant proxies that mediate client requests to a redundant bank of diversified application servers. The idleness is deployed here to increase system availability and integrity. To improve presentation, adaptive idleness is applied: the idleness level is selected according to the current attentive level. The manner can be used for static servers, that is, for Web distribution of stable information (updated offline) and for fully dynamic systems where information updates are executed immediately on an online database. The feasibility of this manner has been demonstrated by implementing an example of a travel agency Web server, and the first presentation tests are satisfactory, both for request execution times and recovery after incidents.

Keywords- *COTS (commercial off-the-shelf), reliability, protection, interruption - lenient, survivability.*

I. INTRODUCTION

Everybody agrees now that the Internet has become essential in everyday life. People use the Internet to work, to exchange information, to make purchases, etc. This growth of the Internet use has unfortunately been accompanied by a growth of malicious activity in the Internet. More and more vulnerabilities are discovered, and nearly every day, new protection advisories are published. Potential assaulters are very numerous, even if they represent only a very small proportion among the hundreds of millions of Internet users. The problem is thus particularly tricky: on one hand, the development of the Internet allows complex and sophisticated services to be offered, and on the other hand, these services offer to the assaulter many new weaknesses and vulnerabilities to exploit.

Almost all traditional approaches for building secure systems only focus on preventing assaults to be successful. Such approaches are becoming insufficient when used in the context of open networks like the Internet, which are characterized by frequent appearance of new assaults. Current systems are so complex that it is impossible to identify and correct all their vulnerabilities before they are put in operation. Thus, preventive approaches require regular updates of some components of the system as soon as a new vulnerability is discovered, that is, nearly every day. Consequently, keeping the system defenses up to date is becoming a full-time task. Furthermore, protection updates of some components

may lead to degradations of the service that they provide due to incompatibility with previous versions

or reduction in functionality. It is clear that the preventive approaches are not sufficient: it is necessary to build systems that survive assaults, because it is not possible to stop them all.

For that purpose, we propose in this paper a standard manner for interruption-lenient Web servers. We address, in particular, servers publishing information that is not confidential (that is, public access Web servers) but whose integrity and availability are critical. No particular assumption is made regarding the software run by the Web servers: they are commercial off-the-shelf (COTS) software, that is, components that are likely to have vulnerabilities. To make the server interruption lenient, we propose to use diversification in idleness. By diversification, we mean that the redundant components will be implemented with as much diversity as possible. We use these techniques to give the system the possibility of continuing its mission, even if some of its components are compromised. Therefore, while the corrupted components are repaired, the legitimate users continue receiving a correct service, even during assaults. We argue that diversification is very important, because assaults, in general, take advantage of specific vulnerabilities either in the operating system (OS) or in the application software. We assume that any

vulnerability affects particular software on a particular platform, and consequently, an assault exploiting vulnerability is dedicated to one platform and will be inefficient on the others: a buffer overflow exploit for PowerPC Linux will be ineffective on Windows XP running on x86. The servers that we consider are dedicated to the Web and provide no other service, so the only assaulters to consider are Web clients (connected through regular http requests) and server administrators. All other remote accesses are blocked by a firewall or are rejected by the servers.

The manner can be used for 1) fully static servers such as the Web distribution of static content that provide stable information, which can be updated offline, for example, a server publishing protection attentive, like www.cert.org, and 2) fully dynamic systems where the updates are executed immediately on an online database (for example, the Web Server of an Internet-based travel agency, where bookings, modifications, and cancellations are made in real time). Our manner is not conational for some specific Web applications such as search engines or auction servers that use farms of thousands of servers. To achieve such a high scalability, specific interruption-lenient techniques, dedicated to the targeted application, must be implemented, while our manner aims at being standard for large classes of Web server applications.

The main contributions of this manner are the following:

- The systematic use of diversification (hardware, OS, and application software),
- The notion of adaptive idleness according to the attentive level (particular attention was paid to efficiency by means of a low-overhead idleness/voting mechanism),
- The specific choice of multiple complementary detection mechanisms (these detection mechanisms, taken together, constitute a whole interruption detection system (IDS) much more powerful that traditional IDSs), and
- The realization of an interruption-lenient dynamic Web server using a database server that only tolerates accidental foibles.

The remainder of this paper is organized as follows: In Section 2, the concepts of foible and interruption tolerance are defined. Then, related work in the fields of interruption tolerance and survivable systems is given in Section 3. In Section 4, a standard interruption-lenient manner for Web servers is proposed. In Section 5, we present some presentation test results from this prototype. Section 6 concludes and presents future work.

II. PRINCIPLES OF INTERRUPTION LENIENCE

A dependable system [1] is defined as one that is able to deliver a service that can justifiably be trusted. Attributes of dependability include availability (readiness for correct service), reliability (continuity of correct service), confidentiality (prevention of unauthorized disclosure of information), and integrity (the absence of improper system state alterations). Protection is the concurrent existence of 1) availability for authorized users only, 2) confidentiality, and 3) integrity, with "improper" taken as meaning "malicious" rather than accidental. Foible-lenient techniques can be used to build dependable systems that are interruption lenient [10], that is, able to continue providing a secure service, despite the presence of malicious foibles, that is, deliberate assaults on the protection of the system. Such foibles are perpetrated by assaulters who make unauthorized attempts to access, modify, or destroy information in a system and/or to render the system unreliable or unusable. Assaults are facilitated by vulnerabilities, which are foibles in the requirements, specification, conation, implementation, and/or configuration of a system.

Assaults, vulnerabilities, and interruptions are defined as three types of interrelated foibles:

- **Assault:** This is a malicious interaction foible through which an assaulter aims at deliberately violating one or more protection properties or an interruption attempt.
- **Vulnerability:** This is a foible created during the development of the system or during operation, which could be exploited to create an interruption.
- **Interruption:** This is a malicious externally induced foible resulting from an assault that has been successful in exploiting vulnerability.

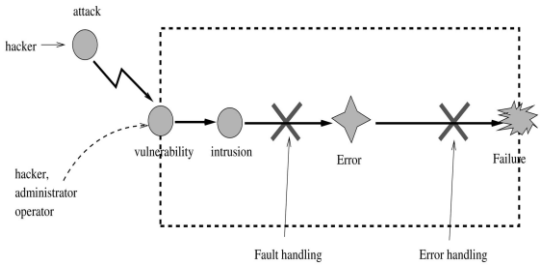

Fig 1: Foible Tolerance Mechanism

An interruption-lenient system is a system that is capable of self diagnosis, repair, and reconfiguration while continuing to provide a correct service to legitimate users in the presence of interruptions [4]. Such a system must be able to identify a wrong "state" or a wrong behavior to recover this situation and to avoid that an internal failure propagates to a system failure observed by the clients.

The mechanisms that can be used to make a system interruption lenient are directly inherited from the usual foible lenient mechanisms:

- error detection techniques,
- error handling techniques to avoid errors from propagating into a protection failure observed by the users
- Foible handling techniques to eliminate the causes of the detected errors.

These mechanisms are represented in Fig. 1 in the context of an interruption-lenient system.

III. AN INTERRUPTION-LENIENT WEB SERVER

This section presents in detail the manner of our interruption-lenient Web Server. This section begins with the foible model and threat assumptions and ends with a protection analysis of the whole manner.

3.1 Foible Model and Omen Assumptions

The manner aims at tolerating both accidental foibles and deliberate foibles. In particular, we consider the following foible classes:

- Temporary foibles (hardware or software). These can be removed by restarting the corrupted machine. For example, corrupted tables or programs will be reinitiated from clean copies at reboot.
- Permanent conation foibles (hardware or software). These can be tolerated through software and hardware diversification, for example, bugs or hardware manufacturing defects. Such foibles are activated each time the system reaches a particular state that triggers them. If such activation is sufficiently rare, it can be sufficient to restart the failing component. If the activation frequency becomes unacceptable, the component (hardware or software) must be repaired by patches or by replacement with a new version.
- Deliberate human-made foibles. These can be malicious or resulting from an engineering decision and can be tolerated, thanks to diversification.

Next, we present the assumptions underlying the conation of the manner:

- The firewall cannot stop all the assaults, but an assaulter has no means of modifying the firewall configuration.
- At any given moment, all the correct servers return the same correct answers1 to the same request. The consistency of the manner depends on this property and the determinism of execution.
- Assaults can be performed by clients or administrators. Administrators are local privileged users. The servers are in different administrative domains so that it is not possible for a single administrator to control or corrupt all the servers. Clients are remote users that access the Web servers by sending them http requests. Some authenticated clients may have more privileges to perform updates or access restricted data. For that, conventional authentication and authorization mechanisms (including the https protocol) can be implemented, but this is beyond the scope of this paper.

To face such assaults, each Web server and proxy must be administrated by a different person in such a way that a malicious administrator can successfully assault the server that he administrates but not the other ones. As his local administrator privileges do not give him any particular privilege on the other Web servers and proxies, he cannot corrupt them. The corruption of the whole system would require the malicious cooperation of most administrators, which is very unlikely.

- Assaults from clients can be performed only by sending http requests. All other requests can be filtered out by the firewall, and they would be rejected by the servers. So, http requests are the only way for a client to communicate with our system. Denial-of-service assaults by flooding the network with requests are not explicitly addressed by our manner: we consider that this problem must be treated by the network. Nevertheless, we can consider that the firewall can also identify such assaults and filter out the flooding requests so that they do not affect our system directly.
- The probability distribution of the assaults is not uniform, and assaults are usually much correlated events: an assault generally does not occur alone, and many assault attempts may occur in short time intervals. It is thus advisable to increase the tolerance capacity when an assault is detected so as to better tolerate an expected burst of assaults.

3.2 Web Servers: Idleness and Diversification

The manner is based on the principles of idleness and diversification. Idleness is used to increase system availability, and diversification is used to increase independence between redundant subsystems from the assaulter's point of view. Since most assaults take advantage of specific vulnerabilities in a particular OS, application software, or hardware platform, they are, in general, ineffective on others. So, the deployment of a redundant bank of diversified Web servers (hardware/OS/software diversification

level) should allow the system to continue providing acceptable service to users, even if parts of the system are corrupted. The Web servers provide the same services (that is, they typically return exactly the same html pages) but are running different application software (Apache or IIS) and OSs (Solaris, Linux, Windows 2000, or MacOS) on diverse hardware platforms (SPARC, Pentium, or PowerPC). They include IDS monitors but are otherwise ordinary platforms running diverse COTS software.

3.3 Adaptive Idleness point

In order to minimize the presentation degradation of the system, the notion of adapted idleness is introduced. The rule is the number of Web servers that process each client request. If the manner includes N Web servers and if the rule is n, it means that only n among N servers process the request. The rule n is selected according to the assault density. When the system is under an active assault, the rule is increased automatically to improve the interruption tolerance capacity. This allows a graceful degradation of presentation in order to reinforce the robustness of the system. The rule will be decreased if a time-out expires without any anomaly being detected. Consequently, the number N of Web servers has to be selected so that a sufficient presentation is provided for an expected assault rate, corresponding to a nominal rule. Typically, the nominal rule can be 1 (simplex) for noncritical applications or 2 (duplex) or 3 (triplex) for more critical applications (see Fig. 2 for duplex and triplex rules).

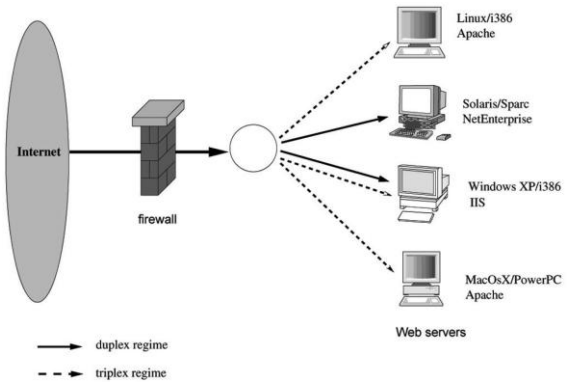

Fig 2: Duplex and Triplex Rules

3.4 Active Data Management

Since requests are not processed by all Web servers (except in the full N rule), an update request cannot modify consistently all the Web servers. If a request updates some data on one server, the same modification should be applied on the other servers in a consistent and atomic way. Let us consider the following example: The system is composed of four Web servers and applies the duplex rule. Two update requests are accepted: the first one is executed by servers 1 and 3, and the second is executed by servers 2 and 3. At the end of these requests, the four databases are inconsistent. This is a quite simple scenario: more complex situations can be easily imagined. A simple solution to this problem is to reject all update requests. In that case, the information stored in the Web servers can only be modified offline once a day, for example, on all the servers and to switch all the servers at the same time between the two versions of the data. This is acceptable for static content Web servers such as protection attentive repositories.

However, if we consider the example of a travel agency Web server, the data managed by such a server must be updated in real time (for booking, modification, and cancellation). The Web servers must thus be able to access and update a database in real time. If each Web server uses its own database, the atomicity and consistency problems remain. Thus, to authorize online updates in a safe functioning rule, we have decided that only the rarely updated data are to be kept replicated on each Web server. The data that are likely to be frequently modified are stored on a centralized foible-lenient database server.

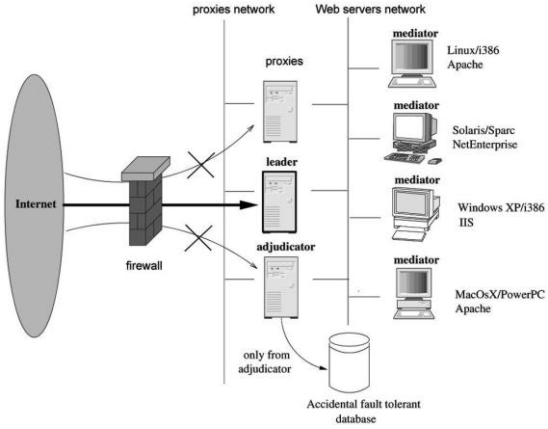

Fig 3: An intrusion tolerant architecture for Web Servers

By "foible lenient," we mean that the database server is sufficiently internally redundant so that it tolerates foibles and thus ensures that the database queries that it receives are always correctly executed. It can integrate replicated databases or use any other usual foible-lenient technique. Such foible-lenient database servers are commercially available from major vendors. The main problem is then to protect the database in such a way that no illegal database query, requested by a corrupted Web server, is executed. Our solution is based on the use of a particular proxy, the adjudicator.

3.5 Proxies

To manage idleness and make it transparent to the clients, tolerance proxies are included in our manner, as depicted in Fig. 3. These proxies have an essential function in the interruption tolerance policy (ITP). They mediate client requests, monitor the state of the

Web servers and the other proxies, dynamically select the rule according to the attentive level, and protect the database. These proxies are also diversified (diverse OSs and diverse platforms are used). The software that they run is not diversified, but it is a very simple code, formally verified offline (during development) and checked online (by runtime verifiers). Furthermore, the proxies are hardened by turning off all the nonessential services of their OSs. The kernels of these OS are particularly secured (by integrating a *Pax* or *grprotection* patch [8] for a Linux kernel for example). The communication protocols between proxies are very simple and static (the format of all messages is always the same). The leader impersonates all the Web servers from the client point of view. The IP address of the leader IPL is the only public IP address, and it is considered as "the" IP address of the virtual Web server. The leader is also responsible for balancing the load among the Web servers.

The leader receives the replies from the Web servers and, according to an agreement protocol (AP), decides the response to return to the client. If the responses are not consistent, an alarm is raised by the leader. Each proxy (leader or other) monitors the other proxies and the Web servers by means of various detection mechanisms described. The proxies decide and apply together the tolerance policy in response to the alarms. The proxies analyze the alarms and take adequate decisions about the tolerance policy according to the attentive level. Responses to interruptions include enforcing a more stringent rule, filtering out requests from suspicious clients, and restarting servers and proxies that are suspected of being corrupted. If the leader is declared corrupted, then a protocol is executed by the other proxies to elect a new leader (automatic proxy reconfiguration). The IP address of this new leader changes to IP_L.

One other proxy is elected to become the adjudicator. The adjudicator is in charge of controlling all accesses to the database server. In fact, it coordinates, checks, and filters the database queries received from the Web servers.

3.6 Attentive Supervisor

Fig. 4 illustrates the proxy manner, which is composed of a specific module, depending on the proxy role and the attentive supervisor that is common to all proxies.

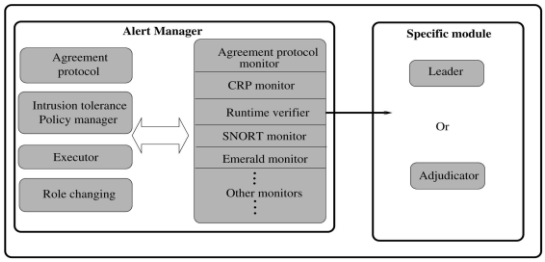

Fig 4: Proxy Architecture

The role of the proxy may be as the leader, the adjudicator, or the auxiliary. There is no specific module for the auxiliary role. In that case, the proxy only executes the attentive supervisor function; that is, it permanently checks and evaluates locally the states of the other proxies and servers and stores its evaluation of these states (trustworthy, suspected, or corrupted; see Fig. 6).

The attentive supervisor includes one monitor for each detection mechanism included in the manner and several other functions: an AP function, an ITP supervisor function, an executor function, and a role-changing function. The main role of the attentive supervisor is to process the attentive raised by the diversified detection mechanisms according to their credibility.

As shown in Fig. 5, each monitor waits for messages from the corresponding attentive source or from the same monitors on the other proxies. For example, the SNORT monitor is waiting for "SNORT attentive," the "AP monitor" is waiting for AP attentive, etc. When one of these monitors receives an attentive, it verifies the local current state of the suspected components. If it is already considered as corrupted, the attentive is ignored. If it is considered as trustworthy, then the attentive is broadcast to the same monitors on the others proxies, and a request is sent to the AP function to start a vote among all the proxies about this component.

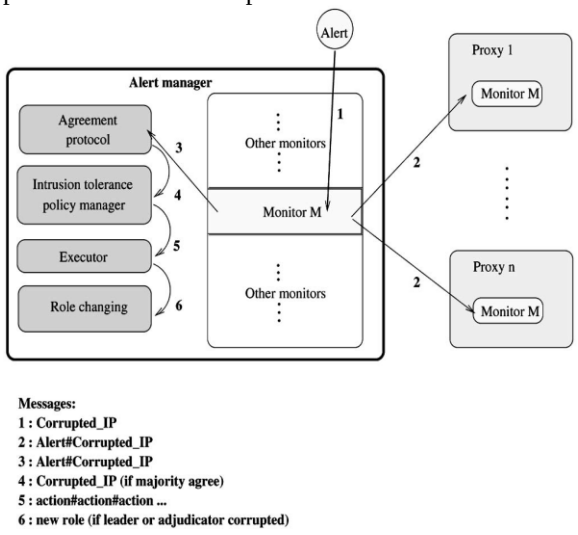

Fig 5: Different steps of attentive processing

The AP function is responsible for carrying out votes between all the proxies in order to find a

common decision about a suspected component. If a majority agrees on the corruption of a component, a request is sent to the ITP supervisor to generate the list of countermeasures adequate to the corrupted component's role. This list is generated by the proxy that initiated the vote and is broadcast to all the proxies.

The executor module is responsible for executing the countermeasures list. For example, if the corrupted component is the leader, the role-changing function will configure the new leader to be able to execute its new functionalities.

As shown in Fig. 6, the state of a component X can be TRUSTWORTHY, SUSPECTED, or CORRUPTED. When a monitor receives an attentive from the corresponding error detection mechanism, it can process or ignore this attentive according to two parameters: the current component state and the importance (weight) of the new attentive. Fig. 6 describes the evolution of the state according to the received attentive.

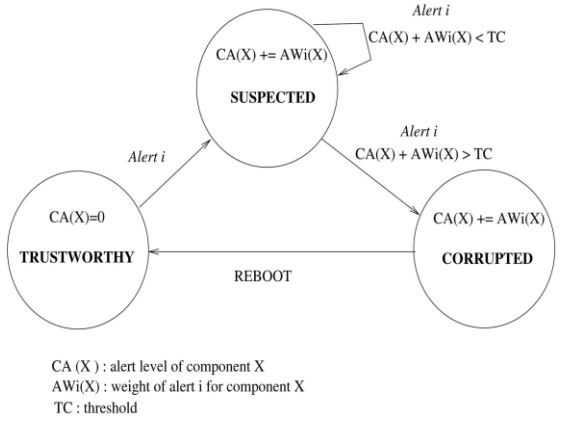

Fig 6: Different States of Component X

The weights assigned to the different attentive raised in the system should be evaluated by an experimental study and according to the credibility of each error detection mechanism.
We do not address this problem in this paper.

3.7 Protection Study

Let us summarize the protection features of our manner. It includes mechanisms for preventing, detecting, and tolerating interruptions. To prevent interruptions, 1) suspicious http and SQL requests are filtered out and sanitized (by the leader, the adjudicator, and the mediator), 2) a trusted firewall blocks all connections from the outside, except http requests to the leader, and 3) the OS of the proxies is particularly hardened.

To detect interruptions
1. Both proxies and Web servers include HIDS sensors,
2. The traffic on the internal network connecting proxies and Web servers is monitored by a NIDS,
3. The IDSs are very efficient, because the messages allowed on the Web servers network are limited to http or SQL requests and to monitoring service messages,
4. The CRP is run periodically by the proxies to check the integrity of critical processes and files,
5. The proxies execute runtime verifiers based on formal specifications [11], and
6. An AP is used to validate the server responses.

To tolerate interruptions, 1) the principle of diversification in idleness is applied so that a majority of components continues to provide a correct service to clients, even if a minority of components is corrupted, 2) the notion of adaptative idleness allows the tolerance capacity to increase when the assault density increases, and 3) any safe proxy can replace the leader or the adjudicator if it is detected as being corrupted, and any corrupted Web server or proxy can be rebooted from a read-only, trusted media. Now, let us take the point of view of an assaulter and explain why it is very difficult for him to make the whole system fail, even if he succeeds in corrupting a component.

3.7.1 Administrator Assaults

As we assume that servers and proxies are in different administrative domains (that is, administrated by a different person), a malicious administrator can successfully assault the server that he administrates but not the other ones. As his local administrator privileges do not give him any particular privilege on the other Web servers and proxies, he cannot corrupt them. The corruption of the whole system would require the malicious cooperation of most administrators, which is very unlikely.

3.7.2 Client Assaults

A client is a remote user accessing the Web server by sending http requests only. If such an intruder wants to corrupt the system, the first step of the assault is to select a target component (proxy, Web server, or database server) and determine the vulnerabilities that he will exploit.

If the assaulter targets a proxy, his task are difficult, because the kernel of the proxies is hardened, the nonessential OS services are turned off, and the proxies execute runtime verifiers based on formal specifications. The leader only receives http requests and does not process them. It simply analyzes the requests, filters out those detected as suspicious, and forwards the others to the Web servers. The adjudicator is not directly accessible from the outside but only from the Web servers (mediators) and the leader. Moreover, each proxy monitors the behavior of the other ones and periodically checks their integrity. As a consequence,

any corrupted proxy can be rebooted, and a safe one can replace it. Injections of false messages or modifications of messages on the networks are immediately detected by the network IDS.

If the assaulter targets the database server, it must either launch its assault by sending a malicious SQL request (through an http request) or by assaulting first the adjudicator (see discussion above). An assault that attempts to exploit specific SQL weaknesses [3] could try to generate the same malicious SQL requests on all the servers. To detect such assaults, we have implemented in the mediator some request filtering mechanisms that reject suspicious SQL requests (SQL injections for example). The SQL assaults based on stack buffer overflow like the Slammer worm [2] also fail, because there is no way of accessing the database server directly from the outside.

To be successful on our manner, an assault should be one of the following:

- a "silver bullet" (an assault that would be successful on all the servers) that would stay invisible enough to assault all the servers without being detected (if the current rule is the full rule, that assault would succeed immediately, but for other rules, the assault would have to evade detection by any of the different detection mechanisms during enough time for the assaulter to corrupt all Web servers),
- a massive and very fast assault that would try to exploit vulnerabilities of all our diversified servers (a lot of malicious requests, each one trying to exploit a vulnerability specific to a particular platform: this assault would be detected but would probably provoke a denial of service of the whole system, because all the servers would reboot nearly at the same moment)
- An assault that would succeed in corrupting the leader (changing its behavior) without being detected by the other proxies or by the runtime verifier.

Taking into account the mechanisms of our manner, we consider these assaults as very difficult to build and launch and, thus, as very unlikely.

TABLE 1
Global Processing Time According to the Rule and Size of the File

	Direct	Simplex	Duplex	Triplex
0 byte	0.0037 sec	0.0074 sec	0.0087 sec	0.0096 sec
44Kb	0.0115 sec	0.0145 sec	0.0167 sec	0.0170 sec
1Mb	0.14 sec	0.316 sec	0.321 sec	0.322 sec

In [5], Chinchani et al. claim that using foible-lenient techniques to secure a system gives a false sense of protection, because the assumption of a "correct" majority of servers is unreasonable. The assumption that an assault exploiting a specific vulnerability can succeed on a particular platform (hardware, OS, and application software) but fails on others is clear for some vulnerabilities such as buffer overflow. For other kinds of vulnerabilities, for example, CGI assaults on Web servers, [7] report several experiments and conclude in the same way.

IV. PRESENTATION DIMENSIONS

A prototype of the manner presented in this paper has been implemented. The Web server implemented as an example in the prototype is the Web server of a travel agency. Hardware diversification was realized with Pentium, PowerPC, and Sparc processors. In the same way, diversified OS's was used: Linux, Windows XP, MacOSX, and Solaris. Three proxies, three Web servers (two versions of Apache and IIS), and one MySQL server were used.

4.1 Presentations According to the rule

To test the presentation of the prototype according to the rule, identical client requests (without access to the database) have been executed in a conventional Web server and in the manner in simplex, duplex, and triplex rules.

Processing an http request in the manner includes the following steps:

1. The request is received by the leader.
2. The leader sends to the adjudicator the RID and the servers that are chosen to execute the request.
3. The leader sends the request to the mediators associated with the chosen servers.
4. The leader receives the MD5 hash codes from the servers and starts the API (the servers do not send the whole response to the leader but an MD5 hash code of the response).
5. The leader sends a full response request to a particular server among the majority.
6. The leader receives the full response and recomposes the MD5 hash code.

TABLE 2
Relative Evolution of Global Processing Time According to the Rule and Size of the File

	Direct to Simplex	Simplex to Duplex	Duplex to Triplex
0 byte	100%	18%	10%
44Kb	26%	15%	2.2%
1Mb	131%	1.49%	0.34%

In this experiment, the Global Processing Time for an http request was measured: GPT_{HTTP} = t7 -t1:

4.2 Presentation of Database entrées

In this section, the measure realized corresponds to a request that needs an access to the database server. The objective here is to evaluate the cost induced by the operations executed by the adjudicator. The test was realized in the duplex rule with a PHP script that consists of the following operations:

1. Connection to the database,
2. Sending "select * from flights" request,
3. Receiving the 2-Kbit data response, and
4. Closing the connection to the database server.

This experiment includes the following steps:

1. The request is received by the leader.
2. The leader sends to the adjudicator the RID and the servers that are chosen to execute the request.
3. The leader sends the request to the mediators associated with the chosen servers.
4. The adjudicator receives the connection requests to the database from the mediators, executes an AP, and establishes the connection to the database. The database response is returned to the mediators.
5. The adjudicator receives the selection requests to the database from the mediators, executes an AP, and transmits the request to the database. The database response is returned to the mediators.

TABLE 3
Comparison of Duration Using Our Library and MySQL Standard Library

	GPT_{DB}	APT (leader)	GPT_{HTTP}
using our database library	0.028 sec	0.038 sec	0.045 sec
using a standard MySQL library	-	0.015 sec	0.020 sec

6. The adjudicator receives the select requests to the database from the mediators, executes an AP, and transmits the request to the database. The database response is returned to the mediators.
7. The adjudicator receives the disconnection requests from the mediators, executes an AP, and transmits the disconnection request to the database. The database response is returned to the mediators.
8. The servers calculate their response, generate a MD5 hash code on this response, and send it to the leader.
9. The leader executes an AP on these hash codes and then asks one proxy to send the whole response. It then forwards this response to the client.

Table 3 summarizes the measures. The GPT_{HTTP} is the global processing time of the request. The APT represents the time spent from the sending of the requests by the leader to the mediators until the end of the AP on the responses. The GPT_{DB} represents the global processing time of the database request.

As shown in Table 3, the average GPT_{HTTP} for this experiment is 0.045 second, which represents twice the GPT_{HTTP} using a standard MySQL library (0.02 second). The latter case corresponds to a direct access to the database without the adjudicator, mediator, and AP. This result seems acceptable but could be improved: during the development, we focused on providing the essential functionalities and not on improving the library presentation.

4.3 Presentation of Isolation and Reboot of a Corrupted Server

In this experiment, we focused on the attentive reaction time in case of the corruption of one Web server. We have considered a CRP attentive. The experiment begins with the reception of the attentive by one of the proxies and ends with the reboot of the corrupted machine. The first proxy that receives the attentive begins a vote among the proxies about the suspected component.

This experiment includes the following steps:
1. A CRP monitor receives an attentive "server S is corrupted" and sends a vote request to the AP module.
2. The vote finishes with a consensus on the corruption of S.
3. The vote result is broadcast to all proxies.
4. The AP module sends a request to the ITP supervisor to generate the list of countermeasures to react to this interruption.
5. The countermeasures list is broadcast to all proxies.
6. The Executor module sends to the specific module the orders to isolate S and to change the current rule. The Executor module also changes the CRP frequency and sends a reboot order to the corrupted machine.
7. The attentive supervisor receives the reboot notification sent by S after its reboot.

V. RELATED WORK

Interruption tolerance is not a new concept. The Delta-4 Project [9], [4] was one of the earliest works on interruption tolerance, in contrast with the traditional protection approaches aiming at avoiding interruptions. It proposed an original approach for protection in open distributed systems. One of the interesting features of the Delta-4 manner is the fragmentation scattering technique that has been applied to file storage, protection management, and data processing. For protection management, the principle resides in the distribution of the authentication and authorization functions between a set of sites administered by different people so that failure of a few sites or abuse of privilege by a small

number of administrators do not endanger the protection functions. Finally, for data processing, two data kinds are considered: 1) numerical and logical data, whose semantics are defined by the application, and 2) contextual data (for example, character strings) that is subjected only to simple operations (input, display, concatenation, etc.). In this scheme, contextual data is ciphered and deciphered only on a user site during input and display.

Within OASIS, the first version of our architecture was designed by SRI International and LAAS-CNRS in the Dependable Intrusion Tolerance (DIT) Project [13]. This version was developed for servers with static content only. The architecture that is proposed in this paper takes into account also the dynamic content issue and the problems related to online updating.

Among all the other projects included in the OASIS Program, the SITAR Project has many similarities with our work. The SITAR [12] architecture addresses highly available distributed applications. It is based on redundancy and diversification, adaptive reconfiguration, and vote. This architecture is composed of the following:

1. COTS servers,
2. Proxy servers
3. Acceptance monitors, in charge of validating the responses from the COTS servers
4. Ballot monitors, applying a chosen voting mechanism to solve conflicts and deciding the final response
5. An adaptive reconfiguration module, and
6. The audit control.

Chameleon [6] is an adaptive infrastructure that allows different levels of availability requirements to be simultaneously supported in a distributed system. Chameleon provides dependability through the use of Adaptive, Reconfigurable, and Mobile Objects for Reliability (ARMORs) that control all operations in the Chameleon environment. Employing ARMORs, Chameleon makes different foible-lenient configurations available and maintains runtime adaptation to changes in the availability requirements of an application. ARMORs can be reconfigured in order to provide an adaptive software infrastructure. A prototype has been implemented, and the experimental results show that overheads in application execution and recovery times are acceptable. The detection mechanisms rely on a local observation of a node and not a global analysis of the whole system behavior. The main similarity between Chameleon and our work is that both manners can adapt dynamically their level of idleness and, thus, the foible tolerance capability, according to variations of the runtime context. Moreover, some components of our manner play a role similar to some Chameleon components: our attentive supervisors can be considered as ARMOR Supervisors, and our mediator, proxy, and error detection mechanisms can be considered as common ARMORs.

VI. CONCLUSIONS

In this paper, we propose a standard interruption lenient manner based on idleness and diversification. The manner was conational to be standard, even if we have focused in this paper on Web servers. The proposed manner matches our original goals: 1) to build secure systems using COTS components that are likely to include vulnerabilities, 2) to achieve high integrity and availability requirements, which are the main needs of the critical systems targeted by our manner, and 3) to find a good trade-off between protection and presentation.

The manner is based on three levels of diversification— software, OS, and hardware—so that it would be difficult to corrupt a majority of the servers. The diversification increases the independence of the replicated servers: vulnerability is specific to one server platform, and it is highly unlikely that it will also affect the others. The efficiency of interruption tolerance is strongly dependent on the deployed detection mechanisms. Our manner includes a set of diversified complementary detection mechanisms: we deploy HIDS and NIDS, a CRP to check the integrity of remote machines, runtime verification techniques to check the trustworthiness of the proxies, and an AP to detect any corrupted behavior among a minority of servers or proxies. We propose an attentive supervisor to process and correlate the attentive generated by these mechanisms. It implements the ITP decided by the administrators, which stipulates the countermeasures to be executed to face different situations.

A thorough protection analysis has yielded a qualitative evaluation of the deployed interruption-lenient mechanisms. It would be interesting to carry out a quantitative and experimental study to evaluate the robustness of our manner under “real” assaults. Moreover, we plan to make some measures to evaluate the capacity of the manner to be deployed for large-scale systems and to determine its limits from both robustness and presentation points of view.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the contributions of the reviewers. Their comments have considerably improved the quality of this article. We, authors express gratitude to all the anonymous reviewers for their affirmative annotations among our paper.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” IEEE Trans. Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004.

[2] CERT Advisory ca-2003-04, http://cert.org/advisories/CA-2003- 04.html, 2008.

[3] C. Cowen, "Software Security for Open Source Systems," IEEE Replace and Privacy, pp. 38-45, Jan./Feb. 2003.

[4] Y. Deswarte, L. Blain, and J.-C. Fabre, "Interruption Tolerance in Distributed Computing Systems," Proc. Int'l Symp. Protection and Privacy (S&P '91), pp. 110-121, May 1991.

[5] R. Chinchani et al., "A Tamper-Resistant Framework for Unambiguous Detection of Assaults in User Space Using Process Monitors," Proc. Second IEEE Int'l Workshop Information Assurance (IWIA), 2003.

[6] Z. Kalbarczyk, R.K. Iyer, S. Bagchi, and K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Foible Tolerance," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 6, pp. 560-579, June 1999.

[7] F. Majorczyk, E. Totel, and L. Me´, "COTS Diversity Based Interruption Detection and Application to Web Servers," Proc. Eighth Int'l Symp. Recent Advances in Interruption Detection (RAID '05), Sept. 2005.

[8] Pax, http://pax.grprotection.org, 2008.

[9] D. Powell, G. Bonn, D. Seaton, P. Verı´ssimo, and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems," Proc. 18th IEEE Int'l Symp. Foible-Lenient Computing Systems (FTCS '88), pp. 46-51, 1988.

[10] "Malicious-and Accidental-Foible Tolerance for Internet Applications: Conceptual Model and Manner," Technical Report 03011, Project IST-1999-11583 MAFTIA, Deliverable D21, D. Powell and R. Stroud, eds., LAAS-CNRS, Jan. 2003.

[11] T.E. Uribe et al., "Interruption Tolerance and Worm Spread," Proc. Int'l Conf. Dependable Systems and Networks (DSN '03), pp. B16-B17, June 2003.

[12] F. Wang, F. Gong, C. Sargor, K. Goseva, K. Trivedi, and F. Jou, "Scalable Intrusion Tolerance Architecture for Distributed Server," Proc. Second IEEE SMC Information Assurance Workshop, 2001

[13] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Sai¨di, V. Stavridou, and T. Uribe, "An Adaptative Intrusion-Tolerant Server Architecture," Proc. 10th Int'l Workshop Security Protocols, pp. 158-178, 2003.

Authors:

Sk. Johny Basha received his Bachelor's Degree (B.Tech) in Computer Science & Engineering from Sri Mittapalli College of Engineering. He is presently working as Asst. Prof. in Universal College of Engineering & Technology, Dokiparru (v), Medikondur (m), and Guntur (DT), India. He is having about one and half year experience in Engineering College. He published 2 international journals in the area of Data Mining and Networking.

B. Bhanu Pratap Reddy received his Bachelor's Degree (B.Tech) in Information Technology from Vignan Engineering College in the year 2008 and Master's degree (M.Tech) in Computer Science from SRM University in the year 2010. He is presently working as an Asst. Prof. in Universal College of Engineering & Technology, Dokiparru (v), Medikondur (m), and Guntur (DT), India. He is having about 3 years of teaching experience in Engineering Colleges and Member in IEEE.

Srinivas Eku completed his B.Tech, M.Tech and now working as Asst. Professor in Jaya Institute of Technology and Science, Hanamkonda. He is having 3 years of teaching experience in Engineering colleges. His areas of interest are Data Mining and Computer Networks. He already published 1 paper in international journal in Data mining area.\

K. Siva Sai Ram Prasad received his Bachelor's Degree (B.Tech) in Computer Science from Koneru Lakshmaiah College of Engineering in the year 2009 and Master's degree (M.E) in Software Systems from Birla Institute of Technology & Science (**BITS**) - **Goa** in the year 2012. He is presently working as an Asst. Prof. in Universal College of Engineering & Technology, Dokiparru (v), Medikondur (m), and Guntur (DT), India. He is having about 1 and half year of industrial experience in reputed Software Company.

◈◈◈