A Multi-Layer Architecture for Intrusion Tolerant Web Services

Zahra Aghajani Kalkhoran[1] and Mohammad Abdollahi Azgomi[2]

[1] ICT Group, E-Learning Center, Iran University of Science and Technology, Tehran, Iran
E-mail: z_aghajani@vu.iust.ac.ir
[2] Performance and Dependability Engineering Lab., Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
E-mail: azgomi@iust.ac.ir

Abstract. Web services as fundamental building blocks for next generation distributed systems, play an important role in today enterprise application architectures. The flexibility and openness of the web services computing model can expose corporate data and business processes to security risks. To support critical applications, the existing web service models need to be extended to assure survivability. In this paper we present a multi-layer architecture for intrusion tolerant web services. The specific goal of the architecture is to use single version software fault tolerance concepts in case of malicious failures. We will also present a coloured Petri net model, which is used for the formal analysis of the proposed architecture.

Keywords: Security, intrusion tolerance, web service, coloured Petri nets.

1 Introduction

An intrusion is a malicious operational fault resulting from a successful attack on system vulnerability [2]. Fault tolerance techniques usually are designed to work against faults that are modeled as rare events occurring independently. The faults that pertain specifically to computer and network security have different characteristics. A programming mistake produces a latent error, which can be exploited by an attacker. An attacker can try many exploits against a system to find its vulnerabilities, and he can repeat successful exploits against the same system or other systems with the same vulnerability [1].

Intrusion tolerance is similar to fault tolerance in that both disciplines aim to have the system continue providing acceptable service in the presence of anomalies. Some of the key fault tolerance concepts and approaches can be used as a basis for developing corresponding intrusion tolerance solutions. However, a fundamental premise of fault tolerance research is that the errors that may give rise to failures occur randomly. Attacks on systems on the other hand are intentional, systematic and repeatable [2].

A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any operating system or programming

language. Interoperability, self-containing and self-describing are the main features of these services which make them suitable to implement complex and distributed service-oriented applications over the Internet [10]. Service-oriented architecture can help to prevent companies to rebuild existing applications, improving the reusability and maintenance cost. However, service-oriented architectures are often unstable due to the unreliability of web services, which can be moved, deleted and subjected to various sources of failures and attacks. Therefore the ability to withstand intrusions and faults becomes highly important when considering the availability and survivability of these services.

Web services as an interface for accessing system functionalities, are placed on top the hierarchical structure of web servers, operating systems, hardware and database systems and operating in the open environment of the Internet, exposes all parts of this hierarchical structure to a wide range of malicious attacks. Due to the works done in the lower levels of this structure [11, 12, 13, 14], in the proposed architecture we only focus on the standard structure of these services.

The rest of this paper is organized as follows. In section 2 we introduce the proposed architecture and its assumptions. Section 3 describes the main detection, isolation and recovery mechanisms in the architecture. Section 4 describes a simple coloured Petri net model of the architecture and section 5 concludes the paper.

2 The Proposed Architecture

The specific goal of the proposed architecture is to use software fault tolerance techniques in case of malicious failures. Software fault tolerance techniques can be divided into two main groups of single-version and multi-version techniques [8] and selecting the best design approach depends on the application specifications and its security requirements such as the availability, integrity and confidentiality.

With due attention to application functionalities, we can classify web services into two groups of public and special services. In public services such as electronic payments and ticket reservations, it is simple to use the diversity of the completely independent implementations in different organizations but N-version programming techniques, due to their disadvantages and high cost, are not applicable in case of special and complex services. Therefore, we try to add some functional capabilities to a single service implementation to build an intrusion tolerant web service.

2.1 Architecture Components

There are four fundamental building blocks in this architecture that form different layers of security in key parts of the system to detect, prevent, confine and tolerate attacks. This architecture uses the simple primary and backup scheme. The primary and backup services have the same implementation and therefore they have the same vulnerabilities. But if we could prevent previously detected attacks from occurring continually on the system, we can use the benefits of single version fault tolerance techniques even in case of malicious failures.

3 Detection and Prevention Tools

3.1 Web Service Firewall

This application level gateway performs the second layer of defense in the proposed architecture. The most common message protocol for web services is SOAP, an XML based message format that contains user request and service response. With its reliance on HTTP as the underlying transport protocol, malicious SOAP messages can potentially cause harm to critical computing assets. Therefore a suitable means to prevent such kinds of attacks is the full grammatical validation [4] of messages before forwarding them to services.

Operations that are available through the service are extracted from the service description file. After parsing the WSDL, an appreciate XML schema is generated by inserting the operation specifications to a common SOAP message template. By hardening this XML schema and validating all messages before forwarding them to web servers, we can protect the services against invalid or malicious requests. Determining the upper and lower bound for the number of elements in the document, restricting the length of inputs, determining valid characters, total size of the request and anything that may be misused by attackers, are what we mean by hardened XML schema.

3.2 Intrusion Detection

We use the combination of two anomaly based and knowledge based detection mechanisms [3] to detect misuse and abnormal behaviors of the components in the proposed architecture. Intrusion masking is typically achieved by replication. Therefore it is not applicable in the case of single version or redundancy based architecture since the same inputs lead to the same outputs in all replicas. In these systems, system activity patterns are followed and compared against the normal and abnormal activity patterns to detect misuses or abnormal behaviors.

Using acceptance test to apply validity checks on the responses and the state transition machines to determine the next accepted state after running each service operation, are the two detection techniques that are used in the proposed architecture. The intrusion detection system in this architecture also does offline auditing on the protected log files to detect abnormal activities.

After failover, the intrusion detection module performs attack diagnosis by analyzing the protected log file that contains recent requests to determine which request(s) caused the system failure. After detecting malicious requests, the containment module tries to extend these requests to attack patterns and updates attack pattern database. In this way, previously unknown attacks and yet unseen variants are automatically and immediately prevented from repeatedly causing failovers.

3.2.1 DoS Attacks. The basic idea in defending against denial of service attacks is to isolate and protect legitimate traffic from huge volumes of DoS traffic when an attack occurs. An attacker can use spoofed IP addresses, zombies with valid IP addresses or request flooding to run DDoS attacks. As a result, we use the combination of three different mechanisms to protect the service against these kinds of attacks. If the rate of incoming requests becomes above a threshold, the system enters to under-attack mode.

In under attack mode and to protect the service against requests from spoofed IP addresses, service requesters need to be authenticated before the requests being processed. On the other hand, we must be sure that the sender is alive and is waiting to receive the response. This is done by using the simple HTTP redirection method [6]. The web service proxy firewall assigns a unique ID to each request and redirects the client to a new location. The new location is same as the firewall's base URI but contains an ID as a HTTP header parameter. This ID will serve as a message authentication code for the client's source IP address and is set by encrypting the (authenticator ID, source IP, timestamp, nonce) string. If the incoming request has an ID in its parameter list, the web service proxy firewall forwards it to a randomly chosen authenticator. The authenticator verifies the validity of these ID and forwards valid requests to the appreciate service and discards invalid requests. The encryptions are done using shared symmetric keys between the web service firewall and authenticators. These keys are updated periodically by the reconfiguration manager. The authenticator verifies the client's IP address with IP in the ID value to ensure that the sender is alive. Authentication is a source consuming mechanism and it is not done by the service itself.

However, attackers may also use their genuine IP addresses to send large volume of requests to the service. Protecting the service against this kind of DDoS attacks is done by binding user information to the service description file. Each requester must acquire a valid WSDL file before accessing to the web service [5]. Accessing to the description files in the proposed architecture is restricted by access control mechanisms. During the description retrieving stage and after authenticating requester, session ID manager allocates a unique session ID to the user. This unique ID will be sent in every user request. During the accessing stage, the web service firewall and the authenticators will verify and filter the service requests according to session information attached to them. As a result, flooding the services by using legitimate IP addresses will be detected by these session IDs.

3.3 Intrusion Containment

To make critical systems tolerant to failures, it is necessary to confine the spread of the occurrence automatically once it is detected. The containment module ensures that the damage caused directly or indirectly by malicious requests will be restricted to a subset of the entire system, which may allow users access to degraded functionality [7]. The containment module uses an intrusion graph and a service vulnerability dependency matrix to confine the spread of failures.

The intrusion graph represents the paths for the intrusion to spread from one service to its neighbors. In this graph each intrusion goal is represented by a node.

The intrusion goals have dependency relationships between each other and the edges are used to model this kind of dependency. Service vulnerabilities are determined by querying the common vulnerability databases. The effects of successful attacks on any vulnerability are manifested into five groups as information leakage, execution of arbitrary code, incorrect behavior of service, denial of service and service termination.

The service vulnerability dependency matrix is a pre-defined matrix that is set by the system administrator. This matrix is a comprehensive model that shows the effects of all known and unknown attacks on the system. The dependency matrix has four dimensions: 1) the source of intrusion, 2) the effects of intrusion on the source, 3) the affected services and 4) the effect of intrusion on the affected services. The value of each element is chosen from the set {0, 1, 2}. The value *0* means that the effect E_1 on the service S_1 does not have the effect E_2 on the service S_2. The value *1* means that the unknown vulnerability V with the Effect E_1 on the service S_1 potentially has the effect E_2 on the service S_2. The value *2* means that the vulnerability V with the effect E_1 of the service S_1 being actively exploited now and has the effect E_2 on the service S_2.

By announcing any new vulnerability, the related *1* value is changed to *2* which mean that the possible vulnerability is reported and activated now. Also this may lead the dependency matrix to be changed more than once, because effects may be cascaded on the services. The intrusion graph is constructed by this matrix.

For each alert that is received from the detection systems, the alert is mapped to a node on the intrusion graph. A mathematical function [7] measures the likelihood that each node has been achieved. After detecting compromised nodes, the intrusion spread channels will be cut to localize the effects of intrusions.

3.4 Recovery and Reconfiguration

3.4.1 Intrusion Recovery. Recovery from intrusions is done by the recovery method which is introduced in [1]. Unauthorized accesses to the system resources which try to violate the system environment integrity policies are recovered by both wrapping mechanism and security event auditing. The wrapper on web service host mediates every attempt to access the file system through web service host allowing only policy authorized operations. Thus, every file access by a wrapped process should be legitimate. Even if the process is taken over by a buffer overflow attack, the wrapper continues to mediate file access and prevent unauthorized access.

In case of wrapper failures, the host monitor can detect any process creation and every access to file systems almost immediately by using security event auditing. Also there are uncompromised copies of all files on a separate read only disk and continual recovery will replace any corrupted, deleted or modified files by unwrapped processes. The tripwire is a security utility that generates cryptographic checksum or hashed of files on a system and periodically checks to ensure that they have not changed. Intrusion detection systems can only detect active intrusions. Passive intrusions are not detected until they become active. Thus, the reconfiguration manager performs random rejuvenations periodically to minimize the effectiveness of stealth attacks that sleep before they cause errors we can identify [1].

To cope with intrusions which cause crash on the primary, we use a hot standby copy as backup. After detecting and confining each successful intrusion,

compromised services are disabled and the reconfiguration manager checks whether the service level is satisfiable or not. If not, the online server must be cleansed to restore to a clean state and the hot standby copy will promote to the online server. The effectiveness of this method is depends on fast self-cleansing mechanism on servers. If service level is not satisfiable on the online server and the hot standby copy is not cleansed too, the service will work in a degraded level until the standby copy becomes clean. Note that we do assume that the system can learn from previous attacks and can prevent the backup copy against the last learned attacks.

3.4.2 Fault Recovery. The faulty services are detected by periodic and random heartbeat signals from the pre-defined sets of <*request, response*> pairs which are sent by the reconfiguration manager. These signals are random requests. By comparing service responses with the desired responses, we can detect if a service is compromised. When failure is detected on primary, continued service to the end user is provided by promoting the backup to the primary. However, this may lead attackers to run denial of service attacks by exploiting the same vulnerability repeatedly to cause continuous reconfiguration on the primary or backup. A simple solution that we use to cope with this problem is to determine and block the causes of compromisation. We consider that all accesses to the web service are done by SOAP messages.

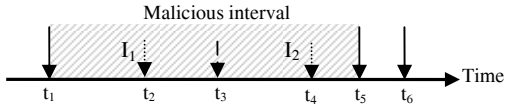

Fig. 1. Defending against DoS attacks exploiting the reconfiguration mechanisms

As shown in Fig.1, the system was cleared on time t_1 by the rejuvenation mechanism. A passive intrusion I_1 occurs on time t_2 and becomes active on time t_3. Another passive intrusion I_2 occurs on time t_4. The detection block alarms for an intrusion on the time t_5. Therefore, we can be sure that the cause of the detected intrusion is a set of requests between t_1 to t_5. By auditing the log file for this malicious interval and eliminating certainly known legitimate requests, the remained requests are considered as suspicious requests. Suspicious requests should have a significant probability that they are an attack and the web service firewall discards these requests temporarily to avoid repeated failure on the system.

4 A Coloured Petri Net Model of the Architecture

Fig. 2shows a coloured Petri net (CP-net) [15, 16, 17] model for the overall processes of proposed architecture. We have used CPN Tools [17] to construct the model. In CP-net model of Fig. 3, the *WSF* transition receives user request and validates it by using the known attack patterns and validation rules. After validation phase, the request is inserted to the system *log* and will be forwarded to the primary web service. The *dispatch to primary* detects and forwards the request to it. The *RCFG* transition

verifies the service level on primary. If the service level is not satisfiable, the backup will be promoted to the primary.

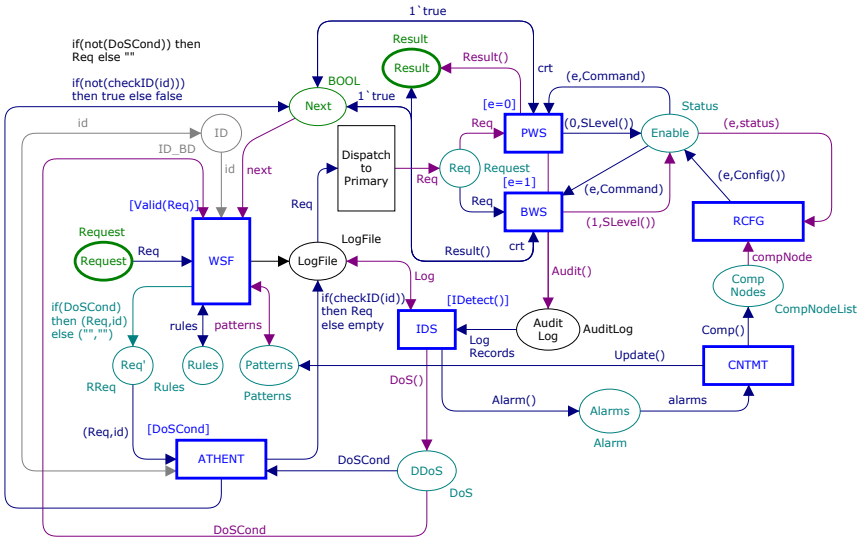

Fig. 2. A CP-net model of the proposed architecture

The *IDS* transition audits the log file to detect and alerts for abnormal behaviors. By announcing any alert, the *CNTMT* transition will be fired to prepare the list of compromised nodes and the *RCFG* transition fires and disables the compromised services on the primary. Also, the *CNTMT* transition updates the *attack pattern* list by extending the malicious requests to new attack pattern. If the system enters to DDoS attack conditions, the *ATHENT* transition will be fired and receives the redirected requests and verifies the validitity of assigned ID. When the system prepares the response of the accepted request, the next state will be set to *true* and thw next request enters to the system.

The transitions with ticker border lines are *abstract transitions* and we are extending them to model the internal behaviour of the corresponding components. We have used the token-game animation feature of CPN Tools to trace the model and check the correctness of the overal processes of the architecture. We are using this model to analyze the architecture using the powerful features of the CPN Tools.

5 Conclusions

In this paper we presented a multi-layer architecture for intrusion tolerant web services which uses the combination of single version software fault tolerance techniques. We use the combination of traditional security and fault tolerance techniques to provide an effective defense in depth strategy for achieving dependability in face of attacks, failures or accidents. As a future work, we are going to implement a

simple prototype of our architecture to model and evaluate its performance. Also, we will extend the CP-net model of the architecture to use for correctness analysis and model checking of the architecture using CPN Tools. Finally, we intend to construct a stochastic model to evaluate some measures of performance, dependability and security of the proposed architecture.

References

1. Reynolds, J. and et al.: "The Design and Implementation of an Intrusion Tolerant System", *Proceedings of the 2002 International Conference on Dependable Systems and Networks* (2001) 285-290
2. Stavridou, V. and et al.: "Intrusion Tolerant Software Architectures*", DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, Volume II (2001) 230-241
3. Veríssimo, P. and Neves, N. and Pupo Correia, M.: "Intrusion-Tolerant Architectures: Concepts and Design", *Lecture Notes in Computer Science*, Volume 2677 (2003) 3-36
4. Gruschka, N. and Luttenberger, N.: "Protecting Web Services From DoS Attacks by SOAP Message Validation", *IFIP International Federation for Information Processing*, Springer Boston, Volume 201 (2006) 171-182
5. Wang, J.: "Defending Against Denial of Web Services Using Sessions", *IEEE/IST Workshop on Monitoring, Attacking Detection and Mitigation* (2006) 32-37
6. Xu, J. and Lee, W.: "Sustaining Availability of Web Services Under Distributed Denial of Service Attacks", *IEEE Transactions on Computers*, Vol. 52, No. 2 (2003) 195-208
7. Wu, Y. S. and et al.: "Automated Adaptive Intrusion Containment in Systems of Interacting Services", *Computer networks* (2007) 1334-1360
8. Dubrova, E.: *Fault-Tolerant Design: An Introduction*, Kluwer Academic Publishers (2008) (*Draft*)
9. Shi, Y. and Zhang, L. and Shi, B.: "Exception Handling of Workflow for Web Services", *Proceeding of the Fourth International Conference on Computer and information Technology* (2004) 273-277
10. Cerami, E.: *Web Service Essentials*, First Edition, O'Reilly (2002)
11. Liu, P., “Architectures for intrusion tolerant database systems”, *18th Annual Computer Security Applications Conference*, pp. 311-320, 2002
12. Ferraz, R. And et al, “An Intrusion-Tolerant Web Server based on the DISTRACT Architecture”, *Workshop on Dependable Distributed Data Management*, Brazil, pp. 45-50, 2004
13. Herder, J.N. and et al, “Construction of a Highly Dependable Operating System”, *Sixth European Dependable Computing Conference*, pp. 3-12, 2006
14. Huang, Y. and Sood, A., “Incorruptible System Self-Cleansing for Intrusion Tolerance”, *Performance, Computing, and Communications Conference*, pp. 4, 2006
15. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Basic Concepts, EATCS Monographs in Theoretical Computer Science, Springer (1992)
16. K. Jensen: "An Introduction to the Theoretical Aspects of Coloured Petri Nets", Lecture Notes in Computer Science, No. 803, Springer (1994) 230-272.
17. "CPN Tools," CPN Group, University of Aarhus, Denmark, URL: http://wiki.daimi.au.dk/cpntools.