2011 First International Conference on Data Compression, Communications and Processing

Intrusion Tolerant Approach for Denial of Service Attacks to Web Services

Massimo Ficco and Massimiliano Rak
Department of Information Engineering
Seconda Universita' di Napoli (SUN)
via Roma 29, 81031 Aversa, Italy
Email: ficco@unina.it, maxrak@gmail.com

***Abstract*—Intrusion Detection Systems are the major technology used for protecting information systems. However, they do not directly detect intrusion, but they only monitor the attack symptoms. Therefore, no assumption can be made on the outcome of the attack, no assurance can be assumed once the system is compromised. The intrusion tolerance techniques focus on providing minimal level of services, even when the system has been partially compromised. This paper presents an intrusion tolerant approach for Denial of Service attacks to Web Services. It focuses on the detection of attack symptoms as well as the diagnosis of intrusion effects in order to perform a proper reaction only if the attack succeeds. In particular, this work focuses on a specific Denial of Service attack, called Deeply-Nested XML. Preliminary experimental results show that the proposed approach results in a better performance of the Intrusion Detection Systems, in terms of increasing diagnosis capacity as well as reducing the service unavailability during an intrusion.**

***Keywords*-intrusion tolerance; denial of service; web services; diagnosis; intrusion recovery;**

I. Introduction

Intrusion tolerance is an emerging paradigm for developing systems that are able to provide an acceptable level of service even after that intruders have broken in. In the context of an Intrusion Tolerant System (ITS), intrusion detection is the key activity to perform intrusion recovery. It is characterized by two sub-activities: *monitoring* and *diagnosis*. Monitoring recognizes that something unexpected has occurred in the system. It observes manifestation of attack symptoms, $i.e$, that the attack is occurred. Diagnosis monitors manifestation of one or more system errors (intrusion symptoms), $i.e.$, that the attack is successful. It identifies the causes of the intrusion ($e.g.$, the attack type), and in particular, the intrusion effects (the target of the intrusion and the extent of the damages).

Although, Denial of Service (DoS) problem is not new, DoS attacks against Web Services (WSs) still remains a research challenge. Many systems are exposed through WS interfaces to DoS [1], [2]. The recent tide of DoS attacks against high-profile WSs demonstrate how devastating DoS attacks are [3].

Nowadays, the technologies available to protect WSs from attacks provide only a support for attack detection or intrusion prevention [4]. On the other hand, for many mission and business critical systems, such as government agencies, Google, eBay, Amazon [5], it is preferable to provide mechanisms that prevent intrusions from leading to a system security failure [6], $i.e.$, it is preferable to support system operations after an intrusion, rather then shutting the system down and losing the services it provides [7]. The ability to diagnosis the intrusion effects on WS applications is becoming increasingly important to infer the least costly and most effective reaction in order to avoid intrusions from leading a system failure. On the other hand, Intrusion Detection Systems (IDSs), which are the main solution to protect networked systems, do not directly detect intrusions, but only the attacks symptoms, $i.e.$, the erroneous or anomalous behaviors of the system [8] [9]. They very much lack intrusion diagnosis capabilities, which would make it possible to figure out causes/consequences of the detected attacks.

In this paper, we present an intrusion tolerant approach for DoS attacks to WSs that exploit XML vulnerabilities. In this work, we consider an attack as a malicious intentional fault through which an attacker aims to exploit a system vulnerability. Intrusion is a malicious externally induced fault resulting from an attack that has been successful in exploiting a vulnerability. The proposed solution emphasizes the relation among attack detection, intrusion diagnosis and intrusion recovery. It consists of monitoring both the attack and intrusion symptoms. In particular, if an attack is detected ($e.g.$, a malicious request to the WS is observed), it is verified whether the intrusion symptoms are also present on the target system ($e.g.$, the CPU consumption exceeds a fixed threshold). Only if the attack succeeds, an alert is triggered and a reaction is performed in order to mitigate the intrusion from generating a WS unavailability.

Our preliminary experiments focus on the Deeply-Nested XML DoS attacks (XDoS), which exhaust the computational resources of the host system by processing numerous deeply-nested XML tags. The experimental tests have shown that the proposed approach results in a better performance of the IDS, in terms of increasing diagnosis capacity as well as reducing the service unavailability during an intrusion.

The rest of the paper is organized as follows: a review of relevant research in the field is presented in Section II; Section III presents some of more common DoS attacks

978-0-7695-4528-8/11 $26.00 © 2011 IEEE
DOI 10.1109/CCP.2011.44

on Web Services; in Section V, we analyze the effects of Deeply-Nested XML attacks on the considered Web server; Section VI describes the proposed approach; the approach evaluation and the experimental results are presented in Section VII; finally, some conclusions and future work are presented in Section VIII.

II. Related work

Several papers have been published that survey DoS vulnerabilities in WS technologies [10], [11]. In particular, Jensen et al. [4] present a list of top WS security vulnerabilities in WSs and perform several attacks on widespread WS implementations. Moreover, they present general countermeasures for prevention of such attacks. Suriadi et al. [12] present an analysis of the effects (in terms of CPU and memory utilization) of exploiting known WSs DoS vulnerabilities against recent WS platforms. On the other hand, no solution has been proposed to tolerate the effects of such attacks. Desmod et al. [13] purpose a testbed that can be used to study the effectiveness of different DoS mitigation strategies and to allow testing of defense appliances. Our testbed is inspired by this work.

Even if intrusion tolerance notion is not a new concept, real research programs dedicated to this topic are relatively recent. MAFTIA [14], CRUTIAL [15] and INTERSECTION [16] are projects that uniformly applied the tolerance paradigm to the dependability of complex critical systems. Their major innovation is a comprehensive approach for tolerating both accidental faults and malicious attacks. In accordance with the their results, we agree that intrusion tolerance and intrusion detection are two tightly linked topics, and we highlight that also on-line intrusion diagnosis is a key building block of intrusion tolerance. Therefore, in order to operate the correct reaction to current intrusion, we propose an approach that presents both diagnosis and detection properties.

Saidome et al. [17] and Valdes et al. [18] propose generic ITS architectures for WSs based on adaptive redundancy and diversification principles. The redundancy level is selected according to the current alert level. Authors claim that attacks, in general, take advantage of specific vulnerabilities either in the operating system or in the application software. Therefore, diversification is a technique to give the system the possibility of continuing its mission, even if some of its components are compromised. On the other hand, the attack considered in this paper is independent of the kind of operating system or application implementation. It exploits a vulnerability presents in the XML standard.

Majorczyk et al. [19] propose an ITS based on COTS diversification applied to WSs. The authors adopt an approach for anomaly detection using redundancy and diversification techniques. Although, the proposed solution provides a high coverage of detection, and a low level of false positives, no diagnostic functionality is defined in the system.

III. XDoS in Web Services

Web Services are vulnerable to a variety of security threats. DoS is a serious and growing problem for corporate and government services doing business on Internet. Targets for DoS attacks include the computational resources, the memory buffers, the application processing logic, the communications bandwidth, and the network protocol, whereas, if it is successful, its effect is the denial or degradation of provided service.

As described in [12], WSs are vulnerable to several DoS attacks. The main factors that make WSs vulnerable to DoS attacks are related to: (i) the use of XML technology, which itself contains DoS vulnerabilities, and (ii) the lack of mechanisms for authentication requests. On the other hand, the authentication of every request could itself be exploited by attackers in order to consume computational resources of the WS provider.

In the following we present some DoS attacks to WSs, which exploit XML vulnerabilities.

- *Oversize Payload*: It is a resource exhaustion attack, which is performed by querying a service using a very large request message. It exploits the high memory consumption of XML processing implemented by most Web Service frameworks that use a tree-based XML processing model, such as the Document Order Model (DOM) [29]. A possible countermeasure against such an attack consists in restriction of the total buffer size for incoming SOAP messages.
- *Coercive Parsing*: It is also called *Deeply-Nested XML*. It exploits the SOAP message format by means of inserting of a large number of nested XML tags in the message body. The goal is to force the XML parser within the server application to exhaust the computational resources of the host system by processing numerous deeply-nested tags. A countermeasure could consist in limit the number of namespace declarations per XML elements. On the other hand, XML specification does neither limit the number of nested XML elements, nor the length of the namespace URIs. Any restriction could lead to unpredictable rejection of legitimate messages.
- *Malformed External Schema Referencing*: The syntax of the XML schema specification allows a document to reference an external XML namespace. A XML parser may then attempt to contact the referenced location to obtain the schema. This functionality may be used by attacker to provide a malformed schema as well as a malicious provider to point to a bogus schema location that provides a large or malicious payload.
- *Heavy Cryptographic Processing*: Another vulnerabilities introduced by WS-Security is that the encryption may be used almost anywhere within a SOAP message, and no strict schema validation is fixed. The flexible and

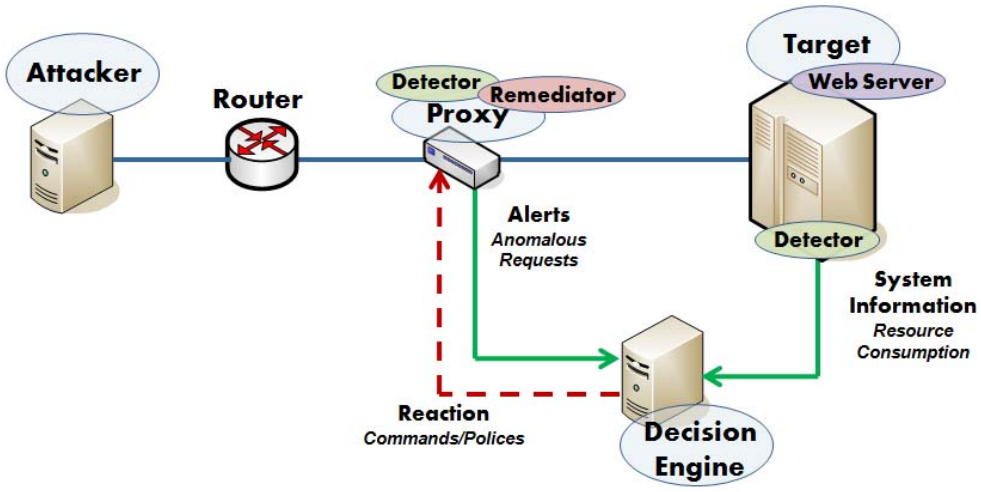

Figure 1. The experimental testbed

expandable nature of WS standard [20] allows using of different security elements in the SOAP header block. This aspect limits the possibility to define robust validation schema of each single element. This flexibility can be used by attacks to relies an over sized security header, which consists of a large number of nested encrypted header blocks within a SOAP message. If the target system processes the whole security header, it may be effected by a CPU overload due to the large number of cryptographic operations. Moreover, if the decrypted content is buffered before further processing, the memory consumption is a multiple of the original message size.

In this work, we focus on Deeply-Nested XML attacks. However, the proposed intrusion tolerant approach is generic enough to be used for diagnosis and recovery of other kinds of DoS attacks.

IV. The Experimental Testbed

The fundamental building blocks in our the proposed ITS are the *Attacker*, the *Target*, the *Detector*, the *Decision Engine*, and the *Remediator*:

- `Target`: It represents the victim node.
- `Attacker`: It is the attacker node running an application that generates malicious SOAP requests.
- `Detector`: It is a software probe used to observe relevant attack symptoms or intrusion effects at a specific architecture level (*e.g.*, application level, operating system level), by using an anomaly-based detection method.
- *Decision Engine* : It is a component that performs the control and intrusion diagnosis activities. In order to determine the most effective intrusion reaction, the Decision Engine uses information inferred by the Detectors.
- `Remediator`: It reacts to the intrusion on the basis of the information received by the Decision Engine.

During the experimental campaign, the implemented testbed on which the experiments are carried out is shown schematically in Figure 1. The testbed consist of:

- a dual-core 2.8 GHz CPU, with 4 GB of memory and Ubuntu 9.10 Linux running a simple WS application built on Apache Axis (the victim server), a well-known open source XML based Web service framework;
- a desktop PC single core AMD 1.7GHz processor with 512MB of memory running the Attacker;
- a desktop PC single core AMD 1.8GHz processor with 2 GB of memory running the Decision Engine;
- a Host-based Detector on the Target machine tracks CPU usage before and during the attack;
- an Application-based Detector monitors the requests to the WS application;
- a Remediator filters malicious requests for intrusion mitigation/blocking.

The Application-based Detector and the Remediator are virtual machines, which are physical hosted on the same PC with single core 3GHz processor and 2GB of memory, running Squid web proxy [21]. The proxy is responsible for receiving, filtering, and forwarding client requests to the Web server. The Application-based Detector and the Remediator are implemented by Java software components based on Greasy Spoon [22], which allows to manipulate Web traffic on the fly. Based on a specific detection model, the Application-based Detector explores SOAP requests, identifies anomalous requests, and alerts the Decision Engine. The Remediator filters out malformed requests according to the recovery policies defined by the Decision Engine. Depending on the policy, the Remediator can also reconfigure the proxy in order to block future requests from the machine suspected of launching the attack.

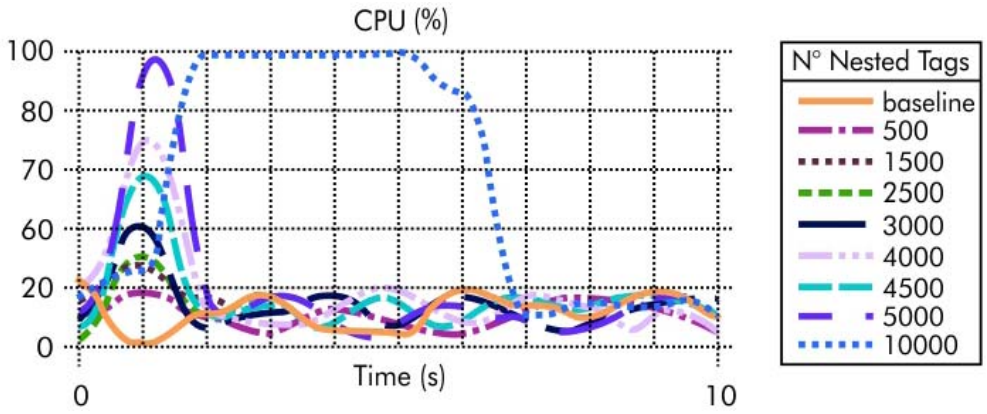

Figure 2. CPU consumption with different nested XML tags

Finally, the implemented Host-based Detector is a software Java component based on Ganglia-Gmond, which is a real-time monitoring system [23].

V. Intrusion Model

The purpose of intrusion tolerance is to prevent errors ($i.e.$, intrusion effects) from resulting in system failures. Obviously, intrusion tolerance cannot be absolute, but must be defined with respect to an intrusion model, that is, the specification of effects that a successful attack has on the target system. Therefore, in this section we present the effects of Deeply-Nested XML attacks against a simple WS application built on Apache Axis2.

During the experimental campaign, we analyze the CPU consumption depending on the number of nested XML tags and the frequency with which the malicious SOAP messages are injected. Each nested XML tag is a string of sixteen bytes.

In the first set of experiments, we analyze the CPU consumption of the target machine to parser a single message. In particular, we injected several invalid SOAP messages containing XML tags with different nested depths. Figure 2 represents the effects of the attacks on the Web server. The solid line represents the CPU load without the attacks, whereas the dotted lines represent the CPU consumption for each injected malicious message. The experiments show that a message of 5000 nested tags is sufficient to produce a peak of CPU load of about 97%, whereas with 10000 tags the CPU is fully committed to process the malicious message for about 3 seconds.

During the second set of experiments, in order to study the CPU consumption in presence of a sustained Deeply-Nested XML DoS attack, we perform different attack scenarios, which consists of a sequence of messages injected with a fixed frequency and a fixed number of nested tags. An attack scenario takes about 30 seconds. Figure 3 represents the average value of CPU for the different scenarios that are performed using different message frequencies and with a

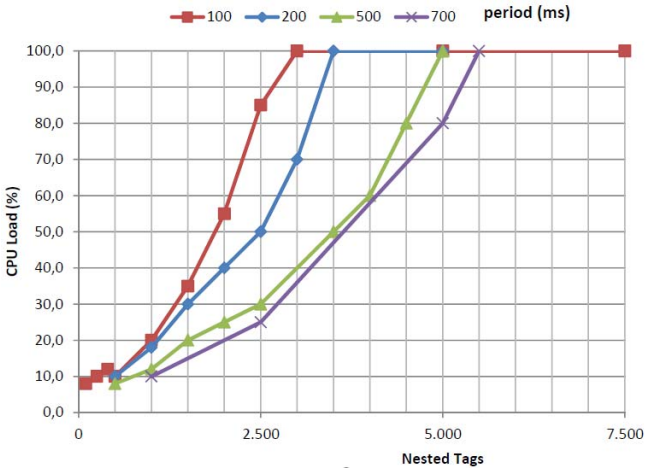

Figure 3. CPU consumption with different message frequencies

number of tags nested to different depths. The experiments show that it is sufficient to inject SOAP messages with about 3000 nested tags every 100 ms, to make unavailable the Target machine ($i.e.$, to exhaust the CPU resource).

VI. The Intrusion Tolerant Approach

The proposed intrusion tolerant approach emphasizes the relation among attack monitoring, intrusion diagnosis and recovery [24]. In particular, the defined solution aims to trigger a specific reaction in order to mitigate the effects of the detected attacks and improve the availability of the WS in presence of a real intrusion.

The adopted approach consists of three phases:

- *Monitoring*: In order to infer symptoms of attacks and intrusion, an anomaly-based monitoring approach is adopted [25]. It computes a probability value, which reflects the likelihood that the observed symptoms represent a malicious/anomalous behavior. If this probability does not exceed a threshold estimated during

a training phase, the symptom is discarded ($i.e.$, it is considered as a normal behavior), otherwise an alarm is triggered [26].

- *Diagnosis*: During the diagnosis phase, the monitored attack and intrusion symptoms are correlated on the basis of the temporal proximity. The diagnosis is characterized by two sub-phases: (i) for each attack reporting by the monitoring phase, the diagnosis determines whether the attack was either a successful attack or a no relevant attack, $i.e.$, an attack that did not lead to intrusion; and (ii) in the case of intrusion, it has to identify the intrusion effects, $i.e.$, the extent of the damages and the faulty components. In particular, it is verified whether intrusion symptoms are present on the considered target ($e.g.$, the CPU or memory consumption exceeds a fixed threshold)
- *Recovery*: Finally, it is determined the kind of intrusion reaction on the basis of the degree of success of the intruder in terms of damage, corruption or consumption. In general, the choice of reaction depends on the current system state and the nature of the alert. Alerts can be evaluated according to the severity of the compromise they indicate ($e.g.$, the degree of success of the intruder in terms of resource consumption), the detail they provide, and the reliability of the alert itself.

A. Attack and intrusion monitoring

In order to monitor the attack and intrusion symptoms, we adopt anomaly-based detection methods that assign a weight to the triggered events, which reflect the anomaly levels with regard to an established profile. The anomaly-based detection approach can be perform in one of two phases: training and detection. In the training phase, a data set is used to parameterize the detection methods (necessary to determine the characteristics of 'normal' behavior), as well as to establish the thresholds that are used to distinguish between regular and anomalous behaviors during the detection phase. In the detection phase, anomaly detection models are used to monitor anomalous behaviors with respect to normal profile computed during the training phase. The threshold choice is the main problem. It is a trade-off process between the number of false positives and the expected detection accuracy. A low threshold can result in many false positives, whereas a high threshold can result in many false negatives. Once the profiles and thresholds have been derived, the detection phase is operated. If the computed anomaly score exceeds the fixed threshold an alert is reported to the Decision Engine. We briefly describe the two models adopted for monitoring the attack and intrusion symptoms observed.

In order to detect anomalous SOAP messages, we adopt an anomaly-based model that estimates an approximation of the actual distribution of nested XML tags in a SOAP message, and detects anomalous number of nested XML tags with respect to normal profile. In particular, during the training phase we estimated an empirical mean μ and variance σ of the real XML tags distribution. In order to analyze the number of observed nested tags n, we used a *Chebyshev* function described by Equation 1, where x is a random variable.

$$p(|x-\mu| \geq k*\sigma) \leq \frac{1}{k^2} \quad with\ k = \frac{|n-\mu|}{\sigma} \tag{1}$$

Replacing $k*\sigma$ with the distance $|n$ - $\mu|$ is possible to obtain an upper bound $p(l)$ on the probability that the number of tags deviates more from the mean than the current instance. Assuming that the training data set is attack free traffic, during the detection phase, the function 2 can be used to determine the deviation of the observed nested tags from the normal behavior.

$$p(l) = \frac{\sigma^2}{(l-\mu)^2} \tag{2}$$

The degree of abnormality associated with each SOAP message is computed as one minus $1/p(l)$.

In order to detect anomalous CPU consumption in the Target machine, different diagnosis mechanisms could be adopted. In this work, a threshold-based over-time diagnostic mechanisms using heuristic approaches is adopted. Heuristic approaches are typically simple mechanisms suggested by intuitive reasoning and then validated by experiments or modeling. We adopted a mechanism that count anomalous behaviors ($e.g.$, excessive CPU consumption), and when the count exceeds a preset threshold an intrusion is diagnosed. This mechanism relies on a simple $\alpha(L)$ filtering function, where L is a discrete time variable, and α is a score that keeps track of the number of consecutive times that the CPU load is greater than a threshold T_{CPU}. An intrusion is detected if $\alpha(L)$ exceeds a certain threshold α_T, that represents the minimum number of consecutive CPU readings greater than T_{CPU}, necessary for triggering an alarm.

The values to assign to α_T and T_{CPU} depend on the accuracy and the precision required. For example, higher values of α_T and T_{CPU} can increase the probability of detecting the attack and reducing the false positives; on the other hand, it can increase the number of false negatives as well as the time required to active a reaction ($i.e.$, it can extend the period of unavailability of the service). Moreover, the values of α_T and T_{CPU} must be chosen on the base of the number of CPU measured per second ($nCPU$) and the intrusion model.

B. Intrusion diagnosis and recovery

During the on-line diagnosis activities, in order to identify the intrusion effects, $i.e.$, CPU overloading due to malicious SOAP messages, we adopted an active monitoring approach. It consists in monitoring anomalous CPU consumptions and correlating this information with alerts generated by the Application-based Detector. In particular, when a XDoS alert

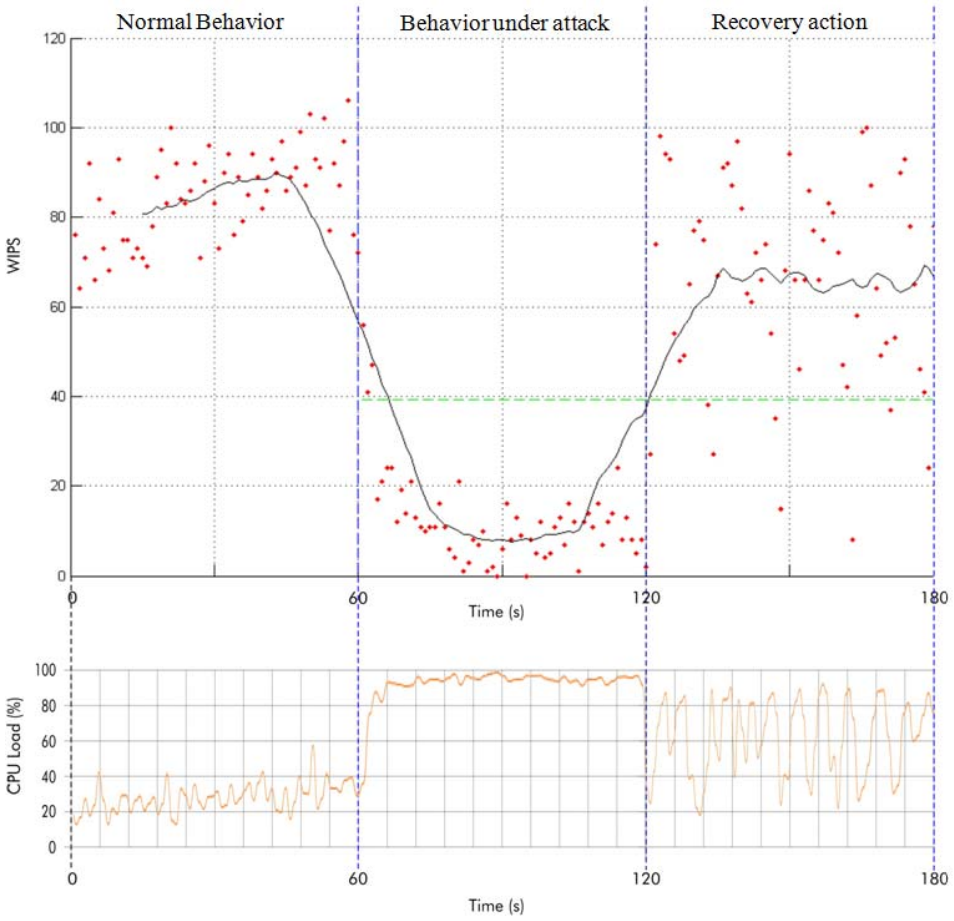

Figure 4. WIPS evaluation during a intrusion recovery process

is received, the Decision Engine triggers an reaction on the base of the Target CPU load. If the Host-based Detector observes that the CPU consumption exceeds the fixed severity level, the Decision Engine alerts the Remediator, which filters each SOAP request that contains a number of nested tags greater than a specific threshold. The purpose of this action is to reduce the CPU load on the Target, thus reducing the period in which the system is unavailable.

VII. Experiments and results

In this section we present preliminary results achieved by applying the proposed intrusion tolerant approach in the experimental testbed described in Section IV. The experiment results show that an accurate diagnosis allows to improve the availability of the WS. The experiment consists of the following steps:

1) tuning of the detection methods;
2) launch attacks from Attacker machine, while maintaining a simulated stress load on the Web server;
3) trigger the reaction only when an intrusion is detected.

The previously steps consists of performing the following activities:

- As for the first detection method, during the training phase, in order to define the normal profile (in terms of mean μ and variance σ of the real XML tags distribution) and the threshold $T_P = 1/p_T(l)$ for the adopted model, we used real traffic collected from production server at the university in which the considered application runs, during an interval time of fifteen days. In particular, we choose a threshold T_P that does not produce false alerts with the training data set. As for the second detection method, assuming two CPU readings per second, and considering the intrusion model represented in Figure 2, we fixed T_{CPU} to 90%, and $nCPU$ to 6 ($i.e.$, an alert is triggered if six consecutive CPU readings greater than 90 have been observed). Regarding the recovery mechanism, according to the results shown in Figure 3, we fixed the threshould to 2500 nested tags. For the considered Target machine, this value can be considered as the maximum number of tags that does not cause the

complete consumption of the CPU resource.

- During the experimental campaign, in order to evaluate the proposed solution, a workload based on TPC Benchmark W (TPC-W) is adopted [28]. It is a transactional web benchmark. The workload is performed in a controlled environment that simulates the activities of a business oriented transactional Web server. It simulates the execution of multiple transaction types that span a breadth of complexity. Multiple web interactions are used to simulate the activity of a retail store, and each interaction is subject to a response time constraint. The performance metric reported by TPC-W is the number of web interactions processed per second (WIPS). It is used to simulated stress load and to assess the effectiveness of the proposed solution by the WIPS measurements. Therefore, using results represented in Figure 3, during the detection phase, we performed several experiments, in which we injected malicious SOAP messages with 3000 nested tags every 200 ms. Moreover, we added TPC-W background traffic on top of XDoS attacks traffic.
- An example of the intrusion and recovery effects is shown in Figure 4. It represents the WIPS and CPU variations with respect to the time, during an interval time of tree minutes. The experiment consists of tree temporal windows. During the first two windows the Decision Engine is disabled. In particular, the first 60 seconds show the values of the WIPS and the CPU load in absence of the attack. The second 60 seconds show the intrusion effects ($i.e.$, the CPU load is about 100% and the number of TPC-W interactions processed is very low). Finally, during the last 60 seconds, the Decision Engine is enabled. When the Decision Engine detects the condition of a reaction ($i.e.$, the attack is in progress and the CPU consumption exceed the 90%), it alerts the Remediator, which filters all the suspicious messages. The reaction is applied until the CPU load falls below the 90%. On the other hand, since the attack is already in progress, the system's CPU resources are again exhausted by processing of new malevolous SOAP messages ($i.e.$, as shown in Fig. 3, the CPU consumption after a few seconds increasing again), therefore, the Decision Engine triggers a new reaction. Alternativly, the Decision Engine could enable the reaction when it receives the first alert, and disable it only if it does not receive new alerts within a certain time window, $i.e.$, if the Target is taken over by a continued XDoS attack, the Remediator continues to filter malevolus SOAP messages to prevent CPU exhaustion.

  This kind of reaction allows to counter/mitigate the effects of the intrusion in order to ensure service provision, although in a degraded manner. Meanwhile, a recovery action that requires more time can be undertaken. For example, by using the malicious detected IP sources provided by the Application-based Detector, an upstream Access Control Lists or a routing reconfiguration can be performed to isolate the attacker.

VIII. Conclusion and future work

In this paper we presented an intrusion tolerant approach for DoS attacks to Web services. In particular, we focused on a specific DoS attack called Deeply-Nested XML. The objectives of our future work will be: (i) to estimate the number of false positives and false negatives generated by the adopted detection and diagnosis models, and (ii) to use the proposed solution to tolerate a larger set of DoS attacks.

However, in general, attackers are aware of the presence of such protection mechanisms: they thus attempt to perform their activities in a stealthy fashion in order to elude local security mechanisms. From an attacker point of view, one of the most effective way of circumventing these security countermeasures consists in distributing the attacks both in 'form' and 'time' executing. For example, in a distributed XDoS attack, an attacker could used SOAP messages with a different number of nested tags in order to circumvent configured thresholds (distribution in form), and delays single messages so as to bypass time window controls (distribution in time). Therefore, another objective will be design a solution that have to be able to prevent repeated attacks from succeeding, even when the attacks can be varied, with a combination of attack learning and generalization as part of a control loop that filters out bad requests.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 256910 (mOSAIC Project) as well as the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest DOTS-LCCI (PRIN-2008-LWRBHF). Moreover, the authors wish to thank Dario Pinzi for his significant contribution in this work.

References

[1] M. Jensen and J. Schwenk. The accountability problem of flooding attacks in service-oriented architectures. In Proc. of *the The Fourth International Conference on Availability, Reliability and Security (ARES'09)*, Mar. 2009, pp. 25-32. IEEE CS.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. In *IEEE Computer*, vol. 40, no. 11, 2007, pp. 38-45.

[3] Witter, Facebook fend off DoS attacks, at: http://www.securityfocus.com/brief/992. Published: Aug. 2009.

[4] M. Jensen, N. Gruschka, R. Herkenh, and N. Luttenberger. SOA and Web Services: New Technologies, New Standards - New Attacks. In Proc. of *the Fifth European Conference on Web Services*, 2007, pp. 35-44, IEEE CS.

[5] W. Iverson. Real World Web Services - Integrating EBay, Google, Amazon, FedEx and more. OReilly, 2004, no. ISBN 10: 0-596-00642-X.

[6] J. Xu and W. Le. Sustaining availability of Web services under distributed denial of service attacks. In *IEEE Transactions on Computers*, vol. 52, no. 2, Feb. 2003, pp. 195-208.

[7] V. Stavridou, B. Dutertre, R. A. Riemenschneider, and H. Saidi. Intrusion tolerant software architectures. In Proc. of DARPA Information Survivability Conference & Exposition II (DISCEX '01), vol. 2, Jun. 2001, pp. 230-241.

[8] Snort: an open source network intrusion prevention and detection system (IDS/IPS). Available at: http://www.snort.org/, 2010.

[9] Bro: an open source Unix based Network intrusion detection system (NIDS). Available at: http://www.bro-ids.org/, 2010.

[10] M. Jensen, N. Gruschka, and R. Herkenh. A survey of attacks on web services. In *Computer Science - R&D*, vol. 24, no. 4, 2009, pp. 185197.

[11] S. Padmanabhuni, V. Singh, K. M. S. Kumar, and A. Chatterjee. Preventing service oriented denial of service (presodos): A proposed approach. In Proc. of *the IEEE International Conference on Web Services (ICWS'06)*, Sept. 2006, pp. 577-584. IEEE CS.

[12] S. Suriadi, C. Andrew, and S. Desmond. Validating denial of service vulnerabilities in web services. In Proc. of *the IEEE Computer Society Proceedings of the Fourth International Conference on Network and System Security*, 2010, pp. 175-182, IEEE CS.

[13] S. Desmond, S. Suriadi, T. Alan, C. Andrew, M. George, A. Ejaz, and M. James. A distributed denial of service testbed. In Proc. of *the 1st IFIP International Conference on Advances in Information and Communication Technology*, LNCS 328, 2010, pp. 338-349, Springer-Verlag.

[14] P. Verissimo, N. F. Neves, and M. Correia. Intrusion-tolerant architectures: concepts and design. In *Architecting Dependable Systems*, LNCS 2677, May 2003, pp 23-36, Springer-Verlag.

[15] P. Verissimo, N. F. Neves, and M. Correia. The CRUTIAL Reference Critical Information Infrastructure Architecture: A Blueprint. In Proc. of *the 1st Inter. Workshop on Critical Information Infrastructures Security (CRITIS)*, Aug. 2006, pp. 78-95.

[16] INfrastructure for heTErogeneous, Resilient, SEcure, Complex, Tightly Inter Operating Networks, at http://www.intersection-project.eu/

[17] A. Saidane, V. Nicomette, and Y. Deswarte. The design of a generic intrusion-tolerant architecture for web servers. In *IEEE Transactions on Dependable and Secure Computing*, vol. 6. no. 1, Jan. 2009, pp. 45-58.

[18] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Sadi, V. Stravidou, and T. E. Uribe. An adaptive intrusion-tolerant server architecture. In Proc. of *the 10th International Workshop on Security Protocols*, Apr. 2002, pp. 158-178.

[19] F. Majorczyk, L. Totel, and L. Me. COTS Diversity Based Intrusion Detection and Application to Web Servers. In Proc. of *the 8th International Symposium on Recent Advances in Intrusion Detection (RAID '05)*, Sept. 2005, pp. 43-62.

[20] OASIS, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS, Nov. 2006, at http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[21] Squid: an open source fully-featured HTTP/1.0 proxy, at http://www.squid-cache.org/

[22] GreasySpoon, a real-time solution allowing you to manipulate Web traffic, at http://greasyspoon.sourceforge.net/index.html

[23] Ganglia, a scalable distributed monitoring system for high-performance computing systems, at http://ganglia.sourceforge.net/

[24] M. Ficco. Achieving Security by Intrusion-Tolerance Based on Event Correlation. In *Special Issue on Data Dissemination for Large scale Complex Critical Infrastructures, International Journal of Network Protocols and Algorithms (NPA)*, vol. 2, no. 3, Oct. 2010, pp. 70-84.

[25] F. Palmieri and U. Fiore. Network anomaly detection through nonlinear analysis. In *Computers & Security*, vol. 29, no. 7, 2010, pp. 737-755.

[26] M. Ficco, L. Coppolino, and L. Romano. A Weight-Based Symptom Correlation Approach to SQL Injection Attacks. In Proc. of *4th ACM Latin-American Symposium on Dependable Computing (LADC2009)*, Set. 2009, ACM Press.

[27] M. Serafini, A. Bondavalli, and N. Suri. Online Diagnosis and Recovery: On the Choice and Impact of Tuning Parameters. In *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, 2007, pp. 295-312.

[28] TPC Benchmark W (TPC-W), a transactional web benchmark, at http://www.tpc.org/tpcw/

[29] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - SOAP Binding. W3C Recommendation, May 2006.