

An Agent-based, Application-Layer Security Framework

Hicham Tout
Internet Technology Group
Cisco Systems
htout@cisco.com

William Hafner
School of Computer and Information Science
Nova Southeastern University
hafnerw@nova.edu

Abstract

Although numerous defense mechanisms against application-level attacks have been implemented, application and software vulnerabilities still persist and are actively exploited. Typical exploits include buffer overflow, SQL injection, information theft, and execution of malicious code. Of more concern is the notion that the emergence of Web services as standard means of accessing and publishing data on the Web has made them a primary target for attackers. Web services attacks are particularly dangerous as they may allow attackers to bypass traditional perimeter defenses and enable other protocols, such as SOAP to be tunneled over HTTP. These protocols may embed contents and instructions that can compromise or crash production services. This paper introduces an application-layer detection mechanism that employs agents in a sandboxed environment to intercept and trap application or technology-specific attacks before they reach production Web services. The proposed solution employs two types of agents: The first type simulates the execution process of production services in order to trap application or technology-specific attacks; the second uses a response classification schemes in order to detect possible information theft.

1. Introduction

As organizations become increasingly dependant on information systems to conduct day to day operations, keeping such systems and associated data secure has taken on growing importance [1]. Although perimeter-based security has been the subject of many research efforts, most proposed solutions and implementations have often focused on hardening network and host based security. As a result, attacks against applications and Web services have been on the rise. The

emergence and standardization of Web services as the preferred means of accessing and publishing business critical data on the Web has naturally made them a primary target for attackers. These attacks may allow attackers to bypass traditional perimeter defenses making them particularly dangerous. Two well known perimeter defense mechanisms have been implemented to protect Web applications and services: firewalls and intrusion detection systems [2]. However, these systems mostly recognize known signatures and thus are limited to preventing known malicious attacks. To address these shortcomings, algorithms which use anomaly detection techniques to detect attacks against Web services and applications, have also been investigated [3, 4, & 5]. These techniques rely on building a profile and a set of expected behaviors for each service. Actions embedded in Web requests are then evaluated against expected behavioral patterns. However, the constant influx of new strains and mutations of attacks has rendered many of these detection techniques ineffective against novel, also referred to as Day-zero threats [4]. In certain cases, these attacks were able to successfully crash production services even after being detected by the above techniques. According to [2], Day-zero attacks exploit vulnerabilities which have not been discovered, published to the world community, or yet had a patch provided by the responsible vendor.

This research proposes an application-layer detection mechanism that employs agents in a sandboxed environment to intercept and trap malicious code attacks before they reach production Web applications or services. Instrumented agents simulate Web services by dynamically loading business process execution flows that model the production version of the particular service. Agents are then monitored for any deviations from expected behavior. This paper argues that application and technology-specific attacks, which may exploit weaknesses in application logic or the

technology being used, cannot always be detected by generic defense mechanisms. Such mechanisms are not able to detect improper application behavior, nor are they able to recognize many application or technology-specific attacks. A more effective mechanism would be to implement application-specific defenses based on application profiles, execution models, and flows. In addition, by moving the above detection mechanisms to a sandboxed environment, the risk of attacks crashing production services is minimized.

This paper is organized as follows. Section 2 provides an overview of application-layer attacks & SOA. Section 3 introduces related work and contrasts it with the approach in this paper. Section 4 discusses the proposed framework. Finally, conclusions and future work are summarized in section 5.

2. Application Layer Attacks & SOA

Web applications and services attacks are particularly dangerous for a number of reasons as they:

- May allow an attacker to bypass traditional perimeter defenses, such as firewalls and intrusion detection systems.
- May provide an attacker access to confidential information without having to compromise any particular system or network.
- May allow an attacker to execute malicious code on a target system.

Typical exploits include buffer overflow, SQL injection, information theft, and execution of malicious code such as XML eXternal Entity (XXE). Web-based application vulnerability represents a substantial risk to the security of computer networks [6]. Web servers and Web-based applications have become popular attack targets as they may expose many application and system vulnerabilities. During the period between April 2001 and March 2002, 23% of reported vulnerabilities were Web related [6]. In addition, the vector of attacks has dramatically increased with the introduction of Service Oriented Architecture (SOA), especially Web Services.

The concept of service-oriented architecture has been around for several years. The main premise behind SOA has been to provide a common service layer that can be invoked by other services, applications, or portals. The surge of Web services has renewed the interest in SOA due to a number of similarities between the two approaches. Today the name SOA is synonymous with the use of Web services. This paper focuses on the technologies and standards that make up an SOA built on top of Web services.

Web services are self-describing software components that can employ other services to complete business tasks using standard protocols. They make up the building blocks for constructing distributed Web-based applications across disparate platforms, object models, and implementation languages. Web services are built on top of existing and emerging standard communication protocols that include XML [7], Simple Object Access Protocol (SOAP) [8], Universal Description Discovery and Integration (UDDI) [9], and Web Services Description Language (WSDL) [10]. One of the advantages of Web services is their platform independence and development environments. The above protocols allow services to communicate with other Web services irrespective of the runtime platform.

Within SOA, an application may be composed of one or more services distributed over multiple platforms and across various tiers and administrative domains [11]. These services may communicate using one or more standard transport and messaging protocols such as HTTP & SOAP. While SOAP has been adopted as the standard messaging protocol for Web services due to a number of advantages, it has also exposed a number of attack vectors that may not be detected or prevented by traditional defenses [12]. As a result enhanced Web-based security mechanisms must be implemented to protect services that may reside within disparate platforms, layers, and across administrative domains.

3. Related Work

This section enumerates some of the solutions built around Web-based attacks detection and prevention. It's meant to be a brief overview of some of the best known efforts in this area of research. [3] proposed an intrusion detection system that uses a number of anomaly detection techniques to detect attacks against Web and application servers. The main algorithm correlates the scripts referenced by client queries with the parameters included in those queries. The system derives parameter profiles from the analyzed data. While the above mechanism is effective in preventing code or SQL injection attacks, it does not address attacks that do not alter the length of the query parameters. Neither does it address response analysis in the cases of data theft. [11] proposed a processor extension as an alternative to only allow the execution of trusted instructions. Signature verification of each instruction set is done simultaneously with program execution. While this approach is capable of preventing the execution of most unauthorized code, it introduces

a fairly high level of complexity and mostly addresses malicious code injection attacks. However it does not address technology or application-specific attacks. While the above frameworks use anomaly detection-based strategies to detect code injection attacks, they do not address other types of attacks—XXE, information theft—neither do they address a number of application or technology-specific attacks, which exploit vulnerabilities in the business logic or the technology used.

4. Agent-based Security Framework

The proposed security framework is based on the following assertions:

1. To effectively trap application-specific attacks, which exploit weaknesses in technologies or application business logic, application-specific defenses must be built.
2. Application-layer attacks, which exploit vulnerabilities in the technology/language used, may cause changes in application behavior and possibly violate one or more application or system security policies.
3. Application-layer attacks, which exploit weaknesses in application logic may or may not cause changes in application behavior, neither would they cause an application to violate any generic security policies.

The first assertion borrows from one of the most effective detection systems, i.e. the Human Immune System (“HIS”). In HIS, specialized T-Cells are created for each family of viruses. Instead of directly attacking viruses or bacteria, these T-Cells identify and destroy cells infected with a virus or bacteria [13]. The identification mechanism relies on the fact that infected cells display fragments of viral proteins in their surface class I molecules that can be recognized by specialized T-Cells. The above technique relies on a number of important assumptions:

- Viruses & bacteria will periodically penetrate external body defenses and infect internal cells.
- Specialized T-Cells are produced by the human body to combat specific viruses. In most cases, the body is able to reproduce lost cells.
- Class I molecules surface of infected cells can be recognized by specialized T-Cells.

The second assertion relies on the fact that most technology or language-specific attacks may trigger a change in application behavior causing it to violate one

or more security rules. Changes in behavior could be in the form of access to privileged information or unauthorized connections to other systems. Those changes in behavior can best be detected by running the request thru a simulation process that models the exact behavior of the production application. The simulation process can be in the form of replicating the exact flow execution, which can be accomplished by running an instrumented copy of the service in a sandboxed environment. Simulation can also be performed by modeling the service execution flow using languages such as Business Process Execution Language for Web Services (WSBPEL) [14].

The third assertion is the most elusive as it may require detailed understanding of authentication and authorization rules for each application in addition to the application business flow. In this scenario even a simulation process that models the exact behavior of the production application may fail to intercept or defend against such attacks. A mechanism that may prevent some of those attacks without requiring detailed understanding of authorization rules or business flow could be in the form of comparing expected responses against actual responses for each type of request. Comparison would be based on an elaborate response classification scheme. This mechanism may detect attacks related to data theft.

The framework is broken into seven components.

1. Profile Generator.
2. Interceptor.
3. Gatekeeper.
4. Agent.
5. Response Classifier.
6. Threat Qualifier.
7. Threat Advisor.

Figure 1 below depicts the interaction between various components of the proposed framework.

Profile Generator:

The profile generator builds a profile of each application based on observed and expected behavior over a specified period of time. The profile includes the types of activities conducted by the application such as access to external systems and resource consumption. The profile for observed behavior is built based on activities performed by the application over a specified period of time. This profile may contain types and instances of requests and actions conducted by the application within a certain time windows. It may also contain resource consumption trends. On the other hand, the profile for expected behavior can be initially built by the application and security subject matter experts. This profile may contain the expected types and instances of actions performed by the application

including pre-defined access to various systems and data stores.

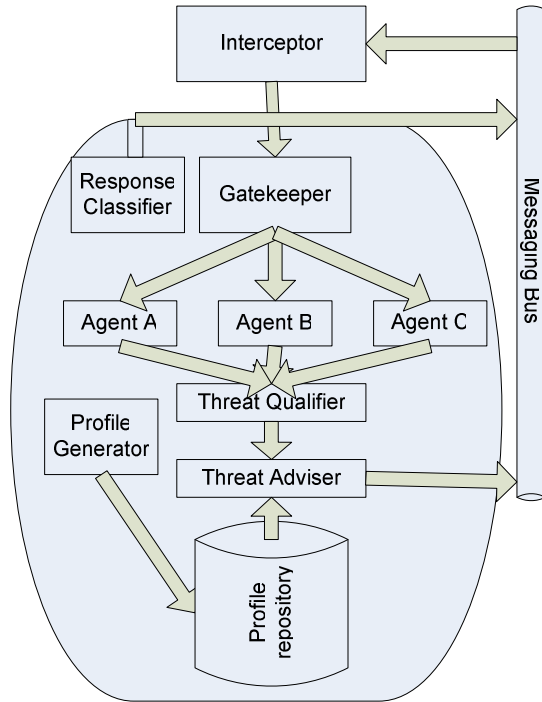


Figure 1: Framework Components

Interceptor:

The role of the interceptor is to intercept and detain certain requests based on pre-determined rules. Requests are detained until a clearance is received from the Threat Advisor component. Rules for intercepting & detaining requests can be implemented at multiple layers of the OSI stack:

- Layer 3: Rules implemented at layer 3/4 of the OSI stack are restricted to the combination of source IP, port and/or destination IP, port. Despite being primitive, rules at layer 3/4 can be used to intercept and hold traffic based on origin or destination, which in this case could be an application listening on a specific port or port range.
- Layer 7: Rules implemented at layer 7 of the OSI stack provide additional flexibility. They could be based on any of the following: Uniform Resource Locator (URL); Uniform Resource Identifier (URI); Contents embedded in the protocol header (HTTP); Contents embedded in the cookie. This is in addition to Layer 3 rules.
- Beyond Layer 7: Rules implemented at this layer enable the interceptor to apply complex logic based on deep content parsing that may

include the protocol and message header and body.

A typical scenario utilizing Web Content Caching Protocol (WCCP) can be illustrated in the following steps:

- Upon receiving a packet marked with a source and destination IP, a WCCP enabled router determines whether to route the packet to the intended destination or to send it to a WCCP client.
- A WCCP client, which could be a special purpose system such as a blade within a switch, a blade server, or an appliance, determines where to route the message by applying pre-defined rules. Those rules could be as simple as matching a URI against a pre-defined list or as complex as parsing the protocol and message header and body for specific contents or patterns that may decide whether or not to route the request to the sandboxed environment.
- Should a match occur, the request is held in a queue, while a copy of it is sent to the Gatekeeper for further analysis. The request is detained until the interceptor receives a response by the Threat Advisor indicating whether or not to allow the request to proceed.

An alternative to the above WCCP-based interception mechanism could be in the form of a custom plug-in or add-on module that is implemented in the Web listener. In this case, all applications must implement a separate Web layer that resides in the DMZ. In addition, in order to ensure consistency, rules may have to be simultaneously deployed to multiple destinations.

Gatekeeper:

A fairly lightweight process with the primary role of providing traffic control services to agents running in the sandboxed environment. Among other services it provides are: Admissions Control, Bandwidth Control, Flow Control, Authorization, and Request Management. Using a combination of firewalls, network access control (ACL), and digital certificates, the gatekeeper may only accept requests originating from and signed by the Interceptor.

Depending on agents' response times, the gatekeeper may apply flow control in order to reduce the impact of impedance mismatch should one occur. It may also invoke a series of commands that may instantiate or fork additional agents in order to horizontally scale agent capacity.

Agent:

The most basic, yet critical role an agent plays is a general purpose container where modeled applications

flows are executed. Agents may dynamically load one or more flows but can only execute a single application flow at a time in response to a request by the gatekeeper. In order to manage both agents and flows, a number of configuration parameters are required:

- **Queue Size:** This parameter specifies the number of flows that can be loaded into an agent memory space at one time. Should the number of flows requested by the gatekeeper exceed queue size, a combination of Least Recently Used (LRU) algorithm and flow priority are used to determine the candidate flow to be unloaded.
- **Time-to-live:** This parameter specifies the flow's expiration date. An agent must check the flow expiration date prior to loading it.
- **Flow Priority:** Each flow is assigned a priority level, which may determine whether a flow can be unloaded despite being ranked last based on an LRU algorithm. A high priority flow may be kept in memory to ensure optimal performance.

An agent may either run an instrumented copy of the production application or an application flow that models it.

The instrumentation process plays a key role in enabling the agent log, report, and categorizes actions resulting from the execution of flows. Among actions that must be logged are:

- Access to external data stores such as entitlement and databases.
- Access to other applications or services.
- Resource consumption such as memory, CPU, disk, and threads spawned.

In addition, a typical log may include the following: action taken; action category or type; matching request; request category or type; subject that initiated the request; end-point accessed.

In the case where the agent loads an application execution flow that models the production application, traps or hooks between process steps can be implemented to achieve the same results provided by the instrumentation process.

Response Classifier:

The job of the Response Classifier is to classify and rank the validity of an actual response against the expected one. Expected responses are created using mappings of known types of requests to the expected types of responses built by either application subject matter experts or by extracting request/response mappings from application logs.

The combination of dependencies, mapping levels, and classification makes the process of producing

response classifiers highly complex. One way to simplify it is to use request and response classification hierarchies with mappings between nodes, branches, or leaves may be necessary. A typical request classification may include:

- **Action:** identified by URL/URI and parameter types.
- **Subject:** identified by user or user role requesting the action.
- **Resource:** identified by the resource or resources being requested.
- **Context:** identified by a canonical representation of a request including the values of one or more environment variables such as application identifier, date/time, originating address, server address, etc...

A typical response classification may include:

- **Data Source:** Database identifier.
- **Data Type:** Type of data being accessed as specified by the data owner (business types). Types may include one or more sub-types.
- **Sensitivity Level:** Sensitivity level of data obtained. If data is obtained from multiple sources with varying sensitivity levels then the response would be marked with the highest sensitivity level returned.

Relationships between nodes, branches, or leaves across classification hierarchies are built using connections as depicted in figure 2 below.

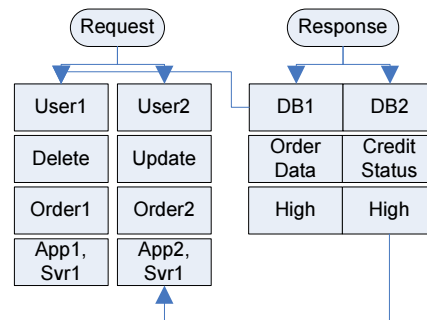


Figure 2: Response Classification

Threat Qualifier:

The primary role of the Threat Qualifier (TQ) is to qualify and rank the potential threat posed by a request. Ranking can be based on one or more of the following measurements:

- Degree of deviation from the observed application behavior.

- Degree of deviation from the expected application behavior.
- If applicable, the severity of the action or task performed.
- If applicable, the sensitivity of the data accessed.

In turn, the above measurements require an application profile to include the following:

- Observed list of actions taken by an application.
- List of actions that can be taken by an application.
- Severity of actions taken by each application as it relates to the privileges granted to it based on its role.
- Sensitivity of the data in relation to the privileges granted to the application based on the role assigned.

Threat Advisor:

The primary role of the Threat Advisor (TA) is to evaluate the overall impact of a potential threat caused by a suspicious request and then communicate in real-time to the interceptor whether or not to allow the request to reach the production application.

In order to communicate back to the interceptor, the TA may use a messaging bus with multicast capabilities to send messages. A message bus is the only accessible production system by components residing within the sandbox environment. Access to the message bus is restricted to specific destinations and actions with no read access allowed, only writes. Figure 2 below depicts a typical TA. Once the level of threat associated with a request has been evaluated, the TA broadcasts a message on a designated topic, to which the interceptor is subscribed. It's worth noting that a consistent vocabulary is imperative to the effectiveness of both the TA and the TQ, as it maps threats, alerts, and rankings generated by various components into a common vocabulary that can in turn be interpreted by the TA. Vocabulary could be based on a number of industry standards that are being created by standards bodies.

Consider the following:

- Suspicious code or logic is allowed to run in an environment that simulates production level targets. Simulation could be in the form of an actual running program which performs some or all logical functions carried by the production application, including connecting to end points, such as databases and directory services. However, end points in this case may not contain production data.

- The behavior of the simulation program where suspicious code is being processed is monitored and analyzed. This could be accomplished by profiling each application, which is done using instrumentation.
- Actions, taken by the simulated application as a result of processing the request that do not conform to expected behavior, would trigger an alert regarding the malicious nature of the request.

One drawback to this approach is that it may incur delays to certain types of requests or applications. However, this additional level of inspection may not be necessary for all types of incoming requests. It may only be used for requests that have been identified as potentially harmful or ones that are made to critical applications or services. An ideal implementation of the above approach would be in a Business-to-Business (B2B) environment.

5. Summary and Conclusion

This paper proposed an application-layer detection mechanism that employs agents in a sandboxed environment to intercept and trap malicious code attacks before they reach production Web services. Instrumented agents simulate Web services by dynamically loading business process execution flows that model the production version of the particular service. Agents are then monitored for any deviations from expected behavior.

The concept is based on the argument that application and technology-specific attacks, which may exploit weaknesses in application logic, cannot always be detected by generic defense mechanisms. Those generic mechanisms are not able to detect improper application behavior, nor are they able to recognize application or technology-specific attacks. A more effective and secure mechanism would be to design application-specific defenses in a sandboxed environment, based on application profiles, execution models and flows.

The proposed framework contains seven main components: Profile Generator; Interceptor; Gatekeeper; Simulation Agent; Response Classifier; Threat Qualifier; & Threat Advisor. The framework uses a detection mechanism that borrows from one of the most effective detection systems—the Human Immune System. Specifically, it attempts to emulate the process T-Cells use to detect and eliminate antigens. It intercepts certain requests at the Web layer for processing by a sandboxed application environment which mimics the production environment. The

simulated applications, within a sandboxed environment, have no access to critical data. However they could be connected to simulated or real data stores, which may contain sample data. This framework should be considered as complimentary to existing detection technologies as it provides a deeper level of inspection and detection that may not be feasible using existing techniques. The detailed design and implementation of a number of components that make up the framework is the subject of our ongoing research.

6. References

- [1] Kuper, P. (2005). "The State of Security". *IEEE Privacy and Security*. September/October 2005. Vol. 3, No. 5. pp. 51-53
- [2] Marin, G. (2005). "Network Security Basics". *IEEE Security and Privacy*. November 2005. pp. 68-72
- [3] Kruegel, C. & Vigna, G. (2003). "Anomaly detection of Web-based attacks". In *Proceedings of the 10th ACM Conference on Computer and Communications Security*. Washington D.C., USA. pp. 251-261.
- [4] Inoue, H. & Forrest, S. (2002). "Anomaly intrusion detection in dynamic execution environment". In *Proceedings of the 2002 ACM Workshop on new Security Paradigms*. Virginia Beach., Virginia, USA. pp. 52-60.
- [5] Anderson, D., Lunt, T., Javitz, H., Tamaru, A., & Valdes, A. (1995). "Next-generation Intrusion Detection Expert System (NIDES)". Retrieved May 10, 2005, from Pennsylvania State University, CiteSeer Web site: <http://citeseer.ist.psu.edu/420493.html>.
- [6] Milenkovic, M., Milenkovic, A., & Javanov, E. (2005). Using instruction block signatures to counter code injection attacks. *Special Issue on Workshop on Architectural Support for Security and Anti-Virus (WASSA)*, ACM SIGARCH Computer Architecture News. pp. 108-117.
- [7] Bray, T., Paoli, C. M., Sperberg-McQueen, and Maler, E. Extensible Markup language (XML) 1.0. Second Edition. W3C Recommendation 2000. <http://www.w3c.org/TR/REC-xml>.
- [8] Mitra, N. Simple Object Access Protocol (SOAP). Version 1.2. W3C Recommendation (June 2003). <http://www.w3c.org/TR/2003/REC-soap12-part0-20030624>.
- [9] Bellwood, T., Clement, L., and Von Riegen, C. UDDI Specification Version 3.0.1. UDDI spec Technical Committee Specifications (Oct 2003). <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>.
- [10] Chnnici, R., Gudgin, M., Moreau, J. J., Schlimmer, J., and Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft (Nov 2003). <http://www.w3c.org/TR/2003/WD-wsdl20-20031110>.
- [11] Tout, H., & Schoenfield, B. (2004). "The Role of an Access Control Policy Sandbox in a Service Oriented Architecture". In *Proceedings of the 2004 International MultiConference in Computer Science & Computer Engineering*. Last Vegas, NV, USA, June 2004.
- [12] Wonohoesodo, R., and Tari, Z. (2004). "A Role based Access Control for Web Services". In *Proceedings of the IEEE International Conference on Services Computing*, Shanghai, China, September 2004. IEEE Digital Library.
- [13] American Cancer Society (2005, April 11). How the Immune System works. Retrieved June 28, 2005, from http://www.cancer.org/docroot/ETO/content/ETO_1_4_X_How_Your_Immune_System_Works.asp?sitearea=ETO&viewmode=print&
- [14] Arkin, A., Zackary, S., Bloch, B., Curbera, F., Goland, Y., and Kartha, N. Web Services Business Execution Language (WSBPEL) 2.0. Committee Draft 21. OASIS Specification 2005. <http://www.oasis-open.org/committees/download.php/16024/wsbpel-specification-draft-Dec-22-2005.htm>.