An Observation Model to Detect Security Violations in Web Services Environment

K. Aldrawiesh
De Montfort University,
Leicester LE1 9BH, UK,
Khalid@dmu.ac.uk

F. Siewe
De Montfort University,
Leicester LE1 9BH, UK,
fsiewe@dmu.ac.uk

H. Zedan
De Montfort University,
Leicester LE1 9BH, UK
hzedan@dmu.ac.uk

ABSTRACT

Growing violation activity makes monitoring of information technology resource systems day by day necessity. As a matter of importance, the popularity of surveillance systems increases with its associated systems. The security of such surveillance systems plays a critical role as their compromise has a technical impact and the need for them is increasing. The complexity of surveillance systems is growing as the system architecture and application must fulfill various requirements of ever demanding project scenarios. The surveillance system is a tool that observes the service behaviour as the e-observer technique works. This paper is proposed an enhanced observer model which maintains a list of its dependents, and then automatically reports any changes in state to an evaluator model, by calling one of their methods. The e-observer is concerned with the state of service behaviour to determine whether it obeys, using its intended behaviour or policy rules; these policies are used to refer to the specific security rules for particular systems. However, web services have become more sophisticated in recent years. WSs are being used successfully for interoperable solutions across various networks.

Categories and Subject Descriptors

D.3.2. [Observe]

D. Web services and Security

Subject descriptor: Observer

General Terms

Security

Keywords

Security policy, requirements, web services and Observation

1. INTRODUCTION

The nature of computing has changed tremendously in the last decade. As heterogeneous computer software, systems, services, visionaries and technologies have become smaller and more powerful; they have also become more complex.

These technologies seemed visible in the field of information revolution. Most of these technologies and systems are connected together over networks. The computer systems and their associated architecture have been developed quickly. Because of these technologies and their resources, security has become coherent and crucial issue to protect these technologies whenever possible. As security is becoming a significant requirement for many organizations, formal methods are being used increasingly to protect and address security issues in the field of development, since confidentiality, integrity and availability of information are paramount, security is the first line of protection of information or resources in a system. Also, it should be addressed against events that can result in loss of availability, unauthorized access, or modification of data.

On the other hand, an observation framework has been used for many purposes. The concept of observation has been developed quickly by many enterprises. The observation framework consists of some components that have been joined to adhere service behaviour whenever interactions have occurred with the expected results of these interactions, by making an aggregation of the results to achieve our system goals. The observers are used in many systems and enterprises in a wide variety of ways with high efficiency. They are simply monitors that used for years in selected industries with complex mathematics. However, Web Services (WS) are identified as computing systems that are used in a wide variety of organizations, for a wide range of reasons. As shown in Fig 1, WSs are a group of emerging and established communication protocols that include Extensible Markup Language (XML), Simple Object Application Protocol (SOAP), Universal Description Discovery and Integration (UDDI) and Web Services Description Language (WSDL) over Hyper Text Transfer Protocol (HTTP). Further, WSs permit a number of solutions/applications to be integrated faster, more easily and cheaply than ever before. They are expressed as a WSDL which is an XML-based language[1,2]

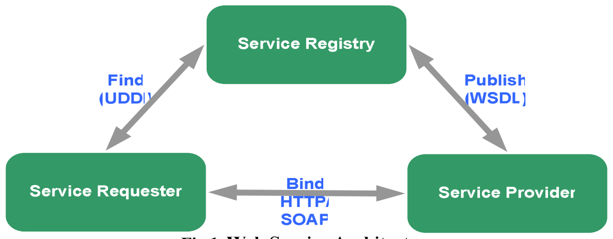


Fig 1. Web Service Architecture

The paper is organized as follows: Section 2 states the problem. Section 3 describes an observation framework. Section 4 shows an enhanced observer implementation. And then, the evaluator and enforcer models have briefly stated. Finally, the concluding remarks and results are given in section 5.

2. Problem statement and contributions

This paper will primary deal on how to support and protect our observation model from threats by controlling service behaviour, and also, how security policies should be enforced against any defection that has different aspects and changeable. Hence our contributions are:

- The proposal and development of an observation model that increased the surveillance by evaluating outcome results via the evaluator model after processing them.
- The taming of the design complexity of the observation model by leaving considerable degrees of freedom for their structure and behaviour and by bestowing upon them certain characteristics, and to learn and adapt with respect to dynamically changing environments.
- Propose an enforcement technique to detect and control any violation activity by addressing security policy system using a systematic approach that can be leveraged by the model requirement within the Web context.

3. OBSERVATION FRAMEWORK

An observation framework is addressed and animated most security problems to enhance protection for resources; thus an idea of security in this paper is to state an insight into the techniques of the model that can be used to mitigate threats. The e-observer technique[3] addresses most problems, and governs and provides a proven solution that is reusable in similar contexts.

As illustrated in Fig. 2, the e-observer is the main component that controls flow data based on system policy. It has the responsibility for appropriate surveillance and feedback. The System Observer Model (SOM) has a sensor and actuator at the heart of the architecture. A control closed loop is created on top of the SOM to rotate the states/services. Furthermore, e-observer [3] observes behaviour through sensors by comparing results with expectations; it then decides what action is necessary and controls the SOM with the best known action through the enforcer but after assessing results by the evaluator model. This framework consists of other components that are used to assist and manage the data flow of observation. These components can typically be found in security systems that are based on surveillance.

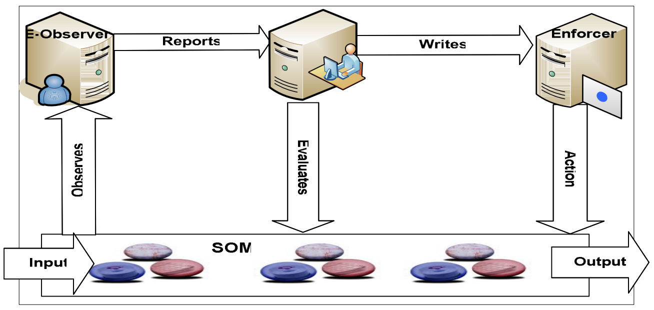


Fig2 ar Observatior Framework

As a consequence, some criteria are assumed to be used when observing, e.g. comparing expectation knowledge of historical, current data and the rules. The e-observer's task is measured to quantify and predict emergent behaviour with basic metrics of QoS, for instance, time, availability and security.

As pointed out in [3], the observer collects and aggregates information of the SOM, and then the aggregation values are reported to the processor; the evaluator model assesses the situation and takes appropriate action on the enforcer model to influence the SOM. Because the observation behaviour itself is variable, our e-observer model influences the observation procedure by selecting certain detectors or certain attributes of interest. The feedback from the e-observer to the evaluator directs attention to certain observables of interest in the current context. Based on the aggregate results by the e-observer model, the evaluator can benchmark the data with an objective function and either knows or learns which actions are best to guide the SOM in the desired direction by informing the enforcer model which acts as a switcher to detect the violation.

3.1 Policy

Policy is a main component that constrains our observation model and its associated parts. Because policy is a big issue, it constrains the behaviour of system components. Policy[4, 5] becomes an increasingly popular approach to the dynamic adjustability of applications and management. In addition, policies [6] which are derived from the goals of management express the desired behaviour of distributed heterogeneous systems and networks. They are often requested to apply to automate network administration tasks, such as configuration, security and QoS. In our model, policies manage and control the system behaviour. They work as a key to permit those who can access the model.

3.2 An Enhanced Observer model

An e-observer works by combining the knowledge and information of services, like behaviour, parameters, attributes and feedback, in order to extract actions that are superior to those which can be obtained and detected. The purpose of an e-observer is to combine measured feedback with knowledge of SOM components to be reported to an evaluator by a processor. The e-observer can be used to enhance system performance and security. It can be more accurate than sensors or can decrease the phase lag inherent in the sensor. E-observer technology is not a panacea, but it can be worked with assistant tools. It adds complexity to the system and requires computational resources, such as high cost setting.

In the meantime, the observation as a technique has become an art, and has been adopted by several enterprises and organizations to enhance performance and control their ability to examine, in close detail, service behaviour. In addition, design an e-observer[7] model is an extremely powerful tool for developers and designers actively engaged in any development project. The e-observer model ensures that common problems are considered and addressed via well-known and accepted highly efficient solutions. The primary strength of paradigms rests with the fact that it is likely that most technical problems have been encountered and solved by other individuals or development teams. Therefore, paradigms provide a mechanism for sharing workable solutions among developers, designers and enterprises. In spite of where these paradigms find their genesis, paradigms leverage this collective knowledge and experience. This ensures that correct codes are designed more rapidly, reducing the possibility that a mistake will occur during design or implementation. Furthermore, the presenting of common semantics during design paradigms takes place between members of an engineering team, since, as anyone who has been included in a large-scale development project knows, knowing and having a general set of design terms and principles is important for achieving completion of the project.

3.2.1 Definition of e-observer

Logically, an e-observer system is not a good enough solution: it needs technical tools to work perfectly. It is used to regulate an enormous variety of services, software and processes. It controls and manages the attributes, parameters and interactions of service behaviour. Moreover, it is common to use the concept of the e-observer to address certain problems relating to application design and architecture. The definition of the e-observer model is often difficult to convey with any level of accuracy warranted by the concept, so a proposal of observation are conventional means.

The e-observer can be a monitor that catches and checks the interaction and execution of communication of the service behaviour, and meets properties in accordance of its policy. Observing allows a tool or user to monitor and analyze, to recover detected faults, which may lead to catastrophic failure to an evaluator. Thus, the e-observer is a component of software or hardware that observes interactions and events of the service behaviour, or any interactions between services, but does not participate in these interactions. In addition, it is a piece of the system that supervises and watches the communication between any events of the services, and then reports to the evaluator by processor, based on its regulation but with the requirements of the model. Also, it is a technique which is used to confirm that the security practices and controls in place are being adhered to and are effective.

E-observer monitors the interception of communications, interacting in the checking of our systems the logging, recording, inspecting and auditing of data. Hence, the e-observers are digital algorithms that observe interactions of service behaviour with knowledge of the observation policy to provide results and information superior from SOM to traditional structures, which rely wholly on the processor and then the evaluator model.

However, as a technical definition, the e-observer model is used to maintain consistency between related objects while minimizing their coupling and maximizing reusability of the objects. Usually this type of paradigm is used to implement a distributed event handling system.

3.2.2 Aggregator:

An aggregation part is a main component of the e-observer model of the observation framework, which aggregates existing information with complex calculations of QoS metrics, and then reports to the evaluator model by the processor. The e-observer model has some components that can support and control the data flow, and sequence of information, such as an aggregator. The aggregator [3] is the main outcome part of e-observer that receives some results and feedback and then makes aggregation. Next, it sends the outcomes to the evaluator model to be evaluated, after processing them. It has a memory to store current values; their history is also stored, forming a set of data vectors. These vectors are needed to perform filtering, as, for example, a smoothing of the results to remove the effects of noise. Furthermore, it delivers and sends a set of filtered current and previous values to the evaluator model via the processor. In some sense of a technical system, the e-observer plays the key roles of the limbic system.

It monitors the external environment via the sensory input and also the internal behaviour of the low level execution unit, and manipulates it in various ways. The major aim of the e-observer is to perform an aggregation of obtainable information and data about the SOM in the form of indicators to give a global description of the state and the dynamics of the underlying system.

4. E-Observer Implementation

As programmatically, the observer paradigm has been provided and support by several computer programming languages such as C#, PHP, C++ and Java. Observer paradigms have been supported by Sun Microsystems. Java has adopted observer paradigms directly. Also, it supports observer paradigms by providing a concrete observable class and an observer interface in the package java.util. This implementation offers a loosely coupled relationship where observers are implemented as an interface and observables must inherit from a base class. So, observables are coupled to an interface rather than directly to observer instances. The observer paradigm is a behavioural OO design that has been defined as the broadcaster-listener subscriber paradigm. This OO design scheme ensures that an observer object or a set of observer objects automatically performs appropriate actions when required to do so by an observable object. Observers register their interest with one or more observable objects, and are notified when an event or interaction that satisfies this interest is recognized by an observation's policy. The observer paradigm also supports a fundamental OO design heuristic by facilitating loose coupling between communicating objects[8, 9].

Nevertheless, as required in our model to state the policy when implementing the observation on state at SOM, the policy in Fig 3 is illustrated as:

```
POLICY        observation   {
        SUBJECT      o= observer;
        TARGET       s= state;
        ACTION       get state observation;
}
```

Fig 3. The Policy of State Observation

As shown in fig 4 and fig 5, the e-observer, evaluator and enforcer have run by using a simulation tool to control and manage our model. The simulation tool is the Attributed Graph Grammar (AGG). AGG is directed mean graph of conveying the structure and operation of many types of systems. AGG is a technical tool that has been used by number of developers and designers for attributed graph transformation systems to support an algebraic approach to graph transformation. It particularly, purposes for rapid prototyping applications with complex, graph structured data. Thus, as shown this tool to illustrate our approach by adapting the requirements on system rules to state the view of the capabilities of the system [11].

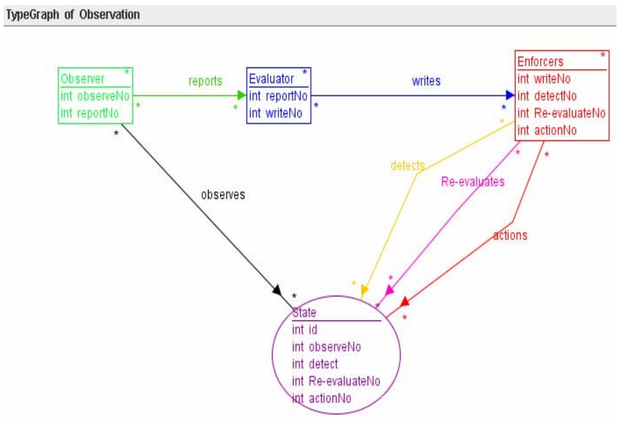


Fig.4 states the relationship of observation components

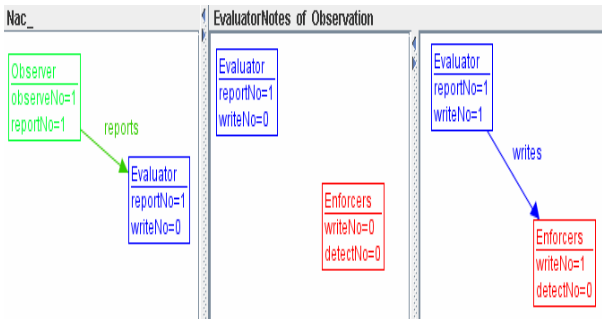


Fig.5 states the simulation of observation

4.1 Observer Manager (OM)

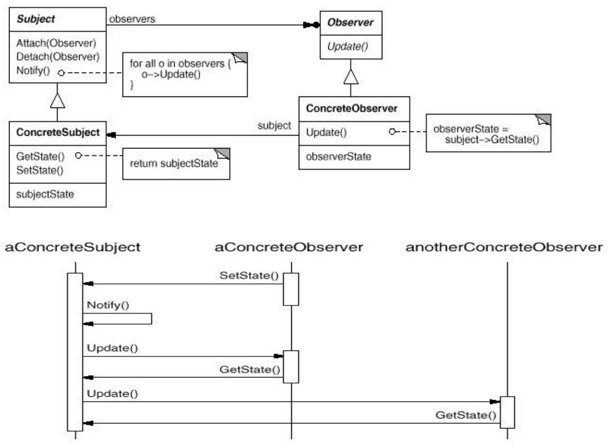
An observer manager [9] is the chief of the e-observer model that can allow and control the surveillance. The proposed solution to the implementation challenges created by the OM includes a class that manages the lifecycles of observer and observable instances. Observers are added and removed to observables by OM, which also ensures that updates are performed correctly. This OM class recursively updates registered observers while avoiding multiple updates and cyclical behaviour. The OM updates, controls and notifies all registered observers of a particular observable when an update is required. Where an observer is also an observable, the method recursively processes all of its observables and their dependencies. Multiple updates and cycles are controlled and avoided by viewing the update process as a graph traversal which sustains and maintains a list of visited objects. An argument parameter and a reference to the observable implement both the push and pull models of event notification. In the following section, the observer manager runs and updates all registered observers in fig 6.

```
Void notifyManager (Object anObservable, Object arg)
{
if ( anObservable s exists )
{ for ( all observers of anObservable )  // process in reverse order of
registration
{ currentObserver notify (anObservable, arg)  // combine push and
pull models  place the current observer on the visited list
if ( currentObserver is also an Observable )
call notifyManager (currentObserver, arg)
                    }       }     }
```

Fig. 6 shows an updating by observer manager.

The observer manager is implemented using entries in a Java hash map consisting of an observable key and a corresponding list of observers. An iterator processes all observers for a particular observable, while the recursive call handles all cases where an observer also functions as an observable.

Although Java [8] simplifies memory management for the programmer, it is still possible to create memory leaks via live references. Any reference to a listener that remains within a system after attempting to remove the listener will inhibit the Java garbage collector and create a memory leak. Ensuring that all references are removed where an observer may be referenced from multiple observables can be a non-trivial operation. However, as seen above in Figs. 7 and 8, a typical structure of an e-observer which has presented a view to distinguish sequence by UML. It defines 1: M dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It has one subject and potentially many observers. Observers register with the subject, which notifies the observers whenever events occur.



Figs. 7 and 8 Illustrates sequence diagram for e-observer task.

4.2 Finite State Machine

In this stage we will examine the modelling of the software using the Finite State Machine (FSM) and how this tool supports the model to be implemented. The FSM is a model that defines the required behaviour of an implementation; it is important to verify the implementation against the FSM. It comprises a data structure that is used to show actions with a sequence of events. In Java applications, an FSM is able to activate and deactivate certain behaviours in time. This part will define how to describe a finite state machine using JFLAP and then the coding using Sun Java[10].

4.2.1 JFLAP

JFLAP-Java Formal Languages and Automata Package- are instructional software used to experiment with grammar automata. It also allows experimentation with applications and proofs. The main feature of JFLAP is that it can experiment with grammars and theoretical machines. JFLAP allows the building and running of user-defined input on pushdown automata, finite automata and regular grammars[10].

4.2.2 JFLAP WITH Design FSM

This stage will examine the design of an FSM which has five states. As shown below, the first state is the beginning of the search of services/objects; more than one can be searched at the same time. Afterwards, the search will send all outcomes to a monitor state which is logically divided into 2 sections: e.g. system data and individual data. Then, the modulation state will broadcast to all states that are connected with it. Next, the aggregate state sends the results to the evaluator state for processing.

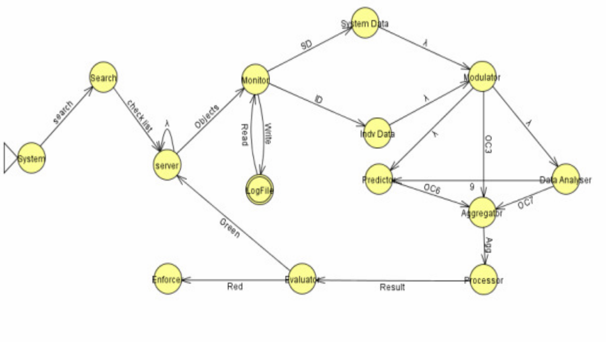


Fig. 9 Test step-by-step of an e-observer

The step by state helps to run our model by using state by state by moving to another state: if green, eventually to a final state of green; if there are any problems between each state, these will be indicated by red.

3.3 Evaluator model

A concept of an evaluator system is to find the risk analysis that can be carried out by detection tools. The intention is to carry out fast evaluation to identify risks and problems. Furthermore, the evaluator is a model that receives the results from the e-observer model to evaluate and process them, to decide whether they adhere to the evaluator policy. The evaluator model has three interfaces, as follows: the aggregated data are obtained from the e-observer. The objectives are imposed on the evaluator by the observation system using the second interface. This is used for the evaluation routine of further actions. It contains all the information needed for the interaction and reconfiguration of the SOM. Every evaluation system provides a number of different parameters and interfaces for manipulation. Then the last decision, called the action selector, and the mechanisms will respond to the enforcer model to take action on the SOM.

3.4 Enforcer model

An enforcer executes unpleasant tasks for a superior. It enforces threats that it will not cooperate with the observation policy. It can excel at detecting adversary attempts to violated security on a host. Furthermore, it receives a final evaluated action and aggregation via the evaluator model to perform the best trigger to the states/services that are determined and controlled to influence in accordance with the evaluation policy. However, in technical terms, it is an online monitoring tool that takes action on security violations in the SOM. In addition, it uses the defined evaluation policy as input, creates a separate process as a thread for each state/service and runs parallel activities to detect security violations. However, the major theme behind the evaluator is that the proactive detection and dynamic policy priority features of the evaluator are the additional strengths which then inform the enforcer for effective detection of security violations.

5. Conclusion

Our observation framework has been portrayed. It has shown that the e-observer architectural paradigm is a natural way of implementing surveillance. It has become an active topic in research and development in recent years. Also, it serves the purpose of keeping emergent behaviour within predefined limits, hence enhancing the security of the architecture. It is a highly traditional technique that has been used by different developers and designers for a long time, but it has been enhanced and adopted by many enterprises and organizations, such as Java which designed and supported the observer paradigm directly.

However, the observer manager has facilitated the creation of a software architecture that allows active notification between loosely coupled objects. By forcing inheritance on observable implementations, the native Java implementation highlights the distortions created by OO languages that do not support multiple inheritances. A single inheritance language is enforced to achieve functionality at the expense of semantic clarity. Class Observable in java.util distorts the relationship that defines inheritance. So far, the use of the e-observer has eased the attachment and detachment of monitoring objects to software probes, and decreased the complexity of information distribution. Therefore, this model is responsible for actually changing the internal behaviour and structure of the processes. The e-observer, evaluator and enforcer ensure that the model cycle and flow data are safe by estimating the threats that could affect the model; should this happen, the detection by the enforcer model will prompt appropriate actions to adhere to the SOM. However, using the FSM and JFLAP has supported the implementation that shows how the program will work and declare each state as a class. JFLAP has much strength, but it is still under development by researchers and designers, which will help FSM to be more specific and useful.

6. REFERENCES

[1] Al-ajlan, A., 2008. Service Oriented Computing for Dynamic Virtual Learning Environments (Moodle), in STRL. 2008, De Montfort: Leicester. p. 328.

[2] Newcomer, E.,2002. Understanding Web Services: XML, WSDL, SOAP and UDDI. 2002: Addison-Wesley Professional; 1 edition.

[3] J"rgen, B., et al, 2006. Organic Computing - Addressing Complexity by Controlled Self-Organization, in Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. 2006, IEEE Computer Society.

[4] Andrew, T.C., C. Geoff, and H. David, 1994. A Multimedia Enhanced Transport Service in a Quality of Service Architecture, in Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video. 1994, Springer-Verlag.

[5] Rajan, R., et al., 1999. A policy framework for integrated and differentiated services in the Internet. 1999.

[6] Gorton, S., and, and S. Reiff-Marganiec, 2007 .Policy-driven Business Management over Web Services., in Integrated

Network Management. 2007: IFIP/IEEE International Symposium. 2007, IEEE.

[7] Purdy, D., and, and J. Richter, 2002. Exploring the Observer Design Pattern, 2002, Microsoft Corporation & Wintellect. p. 21.

[8] Aldrawiesh, K. et al., 2009. A comparative study between computer programming languages for developing distributed systems in web environment, in Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. 2009, ACM: Seoul, Korea.

[9] Eales, A., 2005. The Observer Pattern Revisited, in Educating, Innovating & Transforming: Educators in IT: Concise paper. 2005, NACCQ05.

[10] Yazed, A.-S. et al,2009. The Development of Multi-agent System Using Finite State Machine, in Proceedings of the 2009 International Conference on New Trends in Information and Service Science. 2009, IEEE Computer Society.

[11] A. Al-Ajlan, 2010. Devise a Policy-Based Technique for Enforcing the Security Environment of VLEs, in Proceedings of 2010 International Conference on security systems,China.