Indian Journal of Fundamental and Applied Life Sciences ISSN: 2231– 6345 (Online)
An Open Access, Online International Journal Available at www.cibtech.org/sp.ed/jls/2015/03/jls.htm
2015 Vol. 5 (S3), pp. 295-305/Khoshbakht and Veisi
Research Article

A SURVEY OF SELF-PROTECTING SOFTWARE SYSTEMS

***Fakhreddin Khoshbakht[1] and Kobra Veisi[2]**
[1]*Department of Science, College of Mathematic, Osmania University, Hyderabad, India*
[2]*Department of Marketing, College of Commerce and Business Management, Osmania University, Hyderabad, India*
**Author for Correspondence*

ABSTRACT
The complexity of today's distributed computing environment is such that the presence of bugs and security holes is statistically unavoidable. A very promising approach to this issue is to implement a self-protected system, similarly to a natural immune system which has the ability to detect the intrusion of foreign elements within the system. We designed and implemented an autonomic system called Jade, which relies on software component architectures to reconfigure applications according to observed events. The knowledge of the application architecture can be used to detect foreign activities and to trigger counter-measures. We described how this approach can be applied the provide self-protection for a clustered J2EE application. While existing research has made significant progress towards autonomic and adaptive security, gaps and challenges remain. This paper presents a significant extension of our preliminary study in this area. In particular, unlike our preliminary study, here we have followed a systematic literature review process, which has broadened the scope of our study and strengthened the validity of our conclusions. By proposing and applying a comprehensive taxonomy to classify and characterize the state-of-the-art research in this area, we have identified key patterns, trends and challenges in the existing approaches, which reveals a number of opportunities that will shape the focus of future research efforts.

Keywords: *Self-Protection, Self-Adaptive Systems, Self- Properties, Autonomic Computing, Adaptive Security, J2EE Applications*

INTRUDUCTION
Enforcing the security of a computing system lies on some key capacities. First, as preventive measures, it is important to define tight access control policies, so that hackers can hardly break into the system and hide their tracks. Second, one should be able to distinguish suspicious activities from the "normal" operation of the system. Third, once detected, the malicious processes must be stopped in a comprehensive an efficient1 way. In addition, it is desirable to log the system's activity with a good amount of details (and protection/redundancy to prevent attackers from completely destroying the logs), so that the full sequence and scope of malicious acts can be determined a posteriori, to launch the appropriate recovery procedures and take new measures against future attacks. Unfortunately, these goals are very hard to meet in practice, for several reasons.

• It is notoriously complex to specify and maintain access control policies that are effective, globally consistent (across different programs and computers) and not overly restrictive for users.

• The complexity of today's software components (and their interactions) is such that the presence of bugs and security holes is statistically unavoidable. This leaves the opportunity for attackers to develop new hijacking techniques ("exploits") at a very high pace. Keeping up with the appropriate security patches requires a continuous vigilance.

• Detecting malicious activities within the system is, in general, far from trivial and relies almost exclusively on human expertise. For this reason, most intrusions are only noticed once much damage has been done.

• Careful logging of the system's activity under normal circumstances often leads to unacceptable performance and tremendous need in terms of storage. Besides, extracting crucial hints from the verbose logs is not obvious at all.

Overall, most problems stem from the fact that (human) administrators are unable to cope with the

amount of work required to properly secure a computing infrastructure at the age of the Internet.
We propose to address the above problems through the design and implementation of a selfprotection system. Our two main goals are to: (i) simplify the configuration (and reconfiguration) of security components according to the knowledge of the system's structure and operation and (ii) ease the development of automated counter measures to various classes of attacks.
We have designed and implemented a prototype of autonomic management system (called Jade) which has been successfully used to provide self-healing (Bouchenak *et al.,* 2005) and self-optimizing (Taton *et al.,* 2005) capacities to a clustered J2EE architecture. We are currently investigating the use of Jade to provide self-protection capacities for the same application class.
The rest of the paper is structured as follows. Section 2 presents the self-protection approach. Section 3 provides an overview of Jade, the autonomic management system that we implemented, and it application to clustered J2EE applications. Section 4 describes a scenario which illustrates the use of Jade to implement self-protection in a J2EE application. We overview related works in Section 5 and then conclude the paper. Security is increasingly a principal concern for the design and construction of most modern software systems. In spite of the significant progress over the past few decades, the challenges posed by security are more prevalent than ever before. As the awareness grows of the limitations of traditional, often static and rigid, security models, research shifts to dynamic models, where security threats are detected and mitigated at runtime, i.e., self-protection. This paper has significantly extended our preliminary study of self-protecting software systems (Yuan and Malek, 2012). In particular, unlike our preliminary study, here we have followed a systematic literature review process proposed by Kitchenham (2004). This has broadened the scope of our study and strengthened the validity of our conclusions. In particular, we expanded our preliminary study of 32 publications to a systematic study of more than 1030 papers, from which 107 publications were deemed relevant (including a few that were published after the previous study). Our taxonomy and observations have been refined, enriched with more in-depth analysis, and in some cases altogether revised. To the best of our knowledge, this study is the most comprehensive and elaborate investigation of the literature in this area of research.

Lerature Review

To have a better understanding of computer security threats and vulnerabilities, we turned to (Igure and Williams, 2008), which provides a state-of-the-art "taxonomy of taxonomies" on types of attacks (general attacks, intrusion detection system (IDS) signatures and anomalies, Denial of Service (DoS) related attacks, web attacks and other specialized taxonomies) and vulnerabilities (software flaws, network vulnerabilities). Similarly, Swiderski and Snyder (2004) presented Microsoft's threat model which classifies attacks along the STRIDE model (spoofing, tampering, repudiation, information disclosure, DoS, and elevation of privilege). A different attack taxonomy was introduced in (Bijani and Robertson, 2012), which defined high-level categories, including Disclosure, Modification, DoS, and Fake Identity. The same paper also organized the countermeasures in terms of detection techniques (peer monitoring, information monitoring, policy monitoring, activity monitoring, and attack modeling) and prevention approaches.
— In addition to understanding the attacks, it is equally important to understand the objectives we would like to achieve when it comes to software self-protection. A common "CIA" model from the security community defines Confidentiality, Integrity, and Availability as the main security objectives for information systems, as used in (Perrin, 2008), (Hafiz *et al.,* 2007), and (Cavalcante *et al.,* 2012).

Related Works

— Software systems use a variety of techniques to mitigate security threats to achieve the CIA objectives. In addition to those countermeasures catalogued in (Bijani and Robertson, 2012), Sundaram (1996) provided a good introduction and categorization on intrusion detection techniques, an important research area related to self-protection. Kumar *et al.,* (2010) provided a good survey of Artificial Intelligence (AI) techniques for intrusion detection.

— A number of surveys focused on organizing and classifying security patterns (Konrad *et al.*, 2003), for example, uses metrics such as purpose (creational, structural, and behavioral) and abstraction level (network, host, application). A similar effort (Hafiz *et al.*, 2007) proposed other ways to organize security patterns, many of which are applicable to classifying self-protection approaches.

Defining computer immune systems is a major trend in self-protected system. This approach has been described by Forrest *et al.*, (1997) and Kephart (A biologically inspired, 1994). The basic idea behind immune systems is to distinguish legal (self) behaviors from illegal (nonself) ones (typically virus, worm, sql injection). Self-protected system: an experiment Forest describes a sense of self in the case of Unix processes by identifying sequences of system call which provide a compact signature for self, distinguishing self from non-self behaviors. This system requires to build up a database of normal behavior for each program of interest. Kephart describes an anti-virus system where detectors are based on the immune system analogy and are able to find unknown virus. Furthermore, when a new virus is found, its signature is spread across the network to all other protected computers. Our work is in the same vein, but we propose to exploit the knowledge of the architecture of the application to detect non-self activities. Self-cleansing system (Huang and Arun, 2002) is another solution to build self-protected software. This pessimist approach makes the assumption that all intrusions cannot be detected and blocked. Therefore, the system is considered to be compromised after a certain time. The control loop used by this system periodically re-installs a part of the system from a secure storage. An important property for self-protected system is the ability to mask important knowledge such as the system's structure, software versions, user's data files... The Secure Distributed Storage (Garay *et al.*, 2000) (SDS) is such a system that secures data by spreading and crypting them across multiple computers. A file is sliced in multiple crypted data chunks. Thus, if a computer is compromised, the hacker can only get the incomplete data stored on the computer. In our scenario, the connection ports which are used to implement bindings between components are randomly chosen, thus making it more difficult for an attacker to exploit legal bindings. In our approach, hiding (as much as possible) the architecture of the application is also a crucial issue. When a system is compromised, another important function is the ability to restore the system in a trusted state. File system snapshots can successfully restore system's data when an intrusion is detected. However this solution also rollbacks legal data modifications induced by users. The Taser system (Goel *et al.*, 2005) provides the file system with a selective self-recovery capability. Taser logs all file system access for each process. If a process is compromised, Taser computes illegal access for each file and is able to rollback illegal modification. However if a dependency is found between an illegal and a legal access (e.g: a legal read operation after a compromised write operation), Taser requires a human intervention. A similar approach could be followed to restore the database tier in J2EE application whenever a nonself activity performed database modifications.

Organisation of the Study

The rest of the paper is organized as follows. Section 2 provides a detailed definition of the self-protection property, which serves to bound the scope of this survey. Section 3 lists the existing surveys that are directly or indirectly related to self-protection.

Section 4 summarizes the research method and underlying protocol of the survey while leaving the process details to Appendix A. Section 5 surveys the existing taxonomies and classification schemes related to system self-protection and adaptive security, before proposing a coherent and comprehensive taxonomy that builds on top of existing taxonomies. Section 6 classifies current and past self-protection research initiatives against the proposed taxonomy. We present the analysis on the survey results, offering observations on patterns, trends, gaps, and opportunities. Threats to validity of the results are addressed in Section 7. Based on this analysis, Section 8 outlines a set of recommendations for future self-protecting system research. Section 9 presents the conclusions.

The contributions of the paper include: (1) A proposed taxonomy for consistently and comprehensively classifying self-protection mechanisms and research approaches;

(2) A systematic survey of the state of the art of self-protecting software systems using the proposed taxonomy; (3) Observations and comparative analysis across these self-protecting systems, to identify

trends, patterns, and gaps; and (4) A set of recommendations for future research directions for self-protecting systems.

This paper has significantly extended our preliminary study of self-protecting software systems (Yuan and Malek, 2012). In particular, unlike our preliminary study, here we have followed a systematic literature review process proposed by Kitchenham (2004). This has broadened the scope of our study and strengthened the validity of our conclusions. In particular, we expanded our preliminary study of 32 publications to a systematic study of more than 1030 papers, from which 107 publications were deemed relevant (including a few that were published after the previous study). Our taxonomy and observations have been refined, enriched with more in-depth analysis, and in some cases altogether revised. To the best of our knowledge, this study is the most comprehensive and elaborate investigation of the literature in this area of research. We begin by introducing our research problem, illustrating it using a motivating example, and laying out the organization of the entire paper .

Statement of the Problem & Significance of the Study

The Self-protection Approach Research on self-protection systems is a recent initiative, still in its prospective stage, and has been emphasized by the more global calls for "Autonomic Computing" (AC), which also encompass concerns about other dimensions such as (self-) configuration, optimization and repair (after failures). This approach is notably inspired by the operation of the human body and has lead to the concept of computer immune system (CIS), in the mid 1990s. The main goal of natural immune systems is to protect a live being from dangerous foreign pathogens. This mission relies on a key ability, the "sense of self" (SoS), that is, the capacity to detect the intrusion of foreign elements within the "system" (in this case, the body), though the distinction of self from nonself. Once an intruder is properly detected, measures can be taken to destroy it (or at least contain its damages and progression). In the context of a computing system, nonself may correspond to the activity of a malicious program or an unauthorized user.

Based on this analogy, Forrest *et al.*, (1997) determined the main design principles required to build computer immune systems, which are summarized below.

• Autonomy: The immune system does not require (much) outside management or maintenance. It autonomously classifies and eliminates attacks, i.e. it is able to recognize previously seen attacks as well as new types of intrusions.

• Distributability: There is no central coordination, and, as a result, no single point of failure within the immune system. This implies that no single component is essential and that the incorrect behavior or death of some security components can be compensated by repairing or creation of new components.

• Multi-layered: multiple layers with different mechanisms are combined to provide robust and flexible facilities for security.

Inspired by these principles, we propose architectural patterns to improve the coordination between the multiple elements which compose a security infrastructure. Our focus is not on the development of new specific techniques for access control, intrusion detection or backtracking but rather on the mechanisms that allow an efficient and flexible integration of these various tools within a global, automated control process. We begin by introducing our research problem (1), illustrating it using a motivating example (2), and laying out the organization of the entire paper (3).

There is an unprecedented need for self-protection in today's software systems, driven by both external factors such as cyber threats as well as internal factors that lie within the system architecture.

From Outside: Ever-Increasing Cyber Threats

As software systems become more distributed, interactive and ubiquitous, networking services become an integral part of the system architecture, making these systems more prone to malicious attacks.

***From Within*:** Dynamic Architectural Behaviors.

An equally pressing need for system self-protection arises from the fact that software systems are increasingly designed to take on more dynamic behaviors at runtime. As dynamic architectural styles (such as service-orientation) become more widely adopted, a system function may,

Therefore, as runtime system architectures become adaptive and dynamic, so must their protection, as manual changes in security policies would simply be too slow and too costly.
A Simple Motivating Example Self-protection mechanisms for a software system can take many diverse forms. As an example, let us suppose an intruder, through attempts such as phishing, has gained access to an online banking system and starts to exfiltrate confidential user information. A much-simplified architecture of the Figure 1.

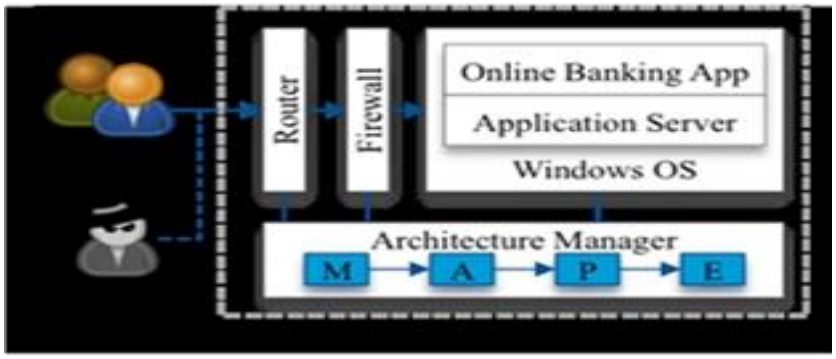

Figure 1: Simple Online Banking mSystem Example system

Suppose shortly after the intruder breaks into the system, his access gets denied and he can no longer gain access. To achieve this effect, the system could have taken any of the following different measures:

— The router's intrusion detection capability detects this intrusion at the network level and automatically disables the connection from the source IP address;

— The firewall detects unusually large data transfer that exceeds the predefined policy threshold and accordingly disables the HTTP connection;

— The ARchitecture Manager (ARM) monitors and protects the system by implementing the Monitor, Analyze, Plan, Execute (MAPE) loop for self-adaptation [Kephart and Chess 2003]. Upon sensing an unusual data retrieval pattern from the Windows server, the ARM shuts down the server and redirects all requests to a backup server accordingly;

— Alternatively, the ARM deploys and manages multiple application server instances on the Windows machine. By comparing the behavior from all server instances (e.g., using a majority voting scheme), the ARM detects the anomaly from the compromised application server instance and consequently shuts it down.

Research Objectives

The contributions of the paper include: (1) A proposed taxonomy for consistently and comprehensively classifying self-protection mechanisms and research approaches;

(2) A systematic survey of the state of the art of self-protecting software systems using the proposed taxonomy;

(3) Observations and comparative analysis across these self-protecting systems, to identify trends, patterns, and gaps; and

(4) A set of recommendations for future research directions for self-protecting systems.

Operational Definitions

1. *Self-protection Define*

Self-protection has been identified as one of the essential traits of self-management for autonomic computing systems. Kephart and Chess characterized self-protection from two perspectives (Kephart and Chess, 2003).

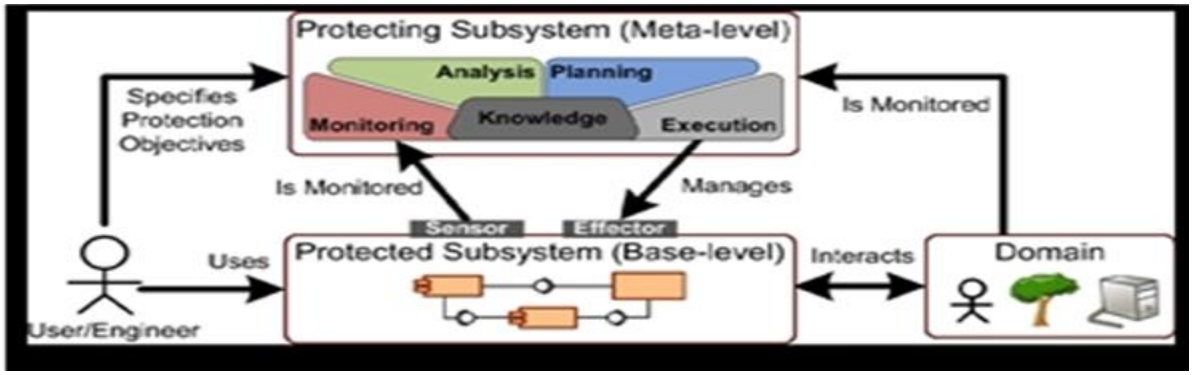

Figure 2: Shows what we consider to be a self-protecting software system in light of FORMS' concepts

2. *Data Safe Operation*

DataSafe operates in four stages – Data Initialization, Setup, Use, Cleanup and Writeback, as explained below.

Data Initialization: During the Data Initialization stage, represented by Step 0, a DataSafe package containing the (encrypted) data to be protected, along with its associated policy, is brought into a DataSafe enabled machine

Setup: In the Setup stage, a secure data compartment (SDC) is dynamically created for the data file. An SDC consists of hardware enforceable tags defined over a memory region that contains decrypted data. Hardware tags are generated from the policy associated with the data. Once an SDC is created for a file, users can subsequently use th data file via potentially untrusted applications, while the hardware ensures that the data is used in accordance with the associated policy.

In Step **1**, a user starts a new session by providing his/her credentials, and is authenticated by the policy/domain handler. A session with an authenticated user, data properties and other system or environment properties sets up the context within which the data item is to be used.

During the session, the user requests file interaction using a third-party application, as shown in Step *2*. The third-party application's request is forwarded to the file management module in Step *3* by the modified file access library of the runtime.

In step **4**, the file management module requests the policy/domain handler to provide the hardware tags to be set for the file. The policy/domain handler validates the policy associated with the data file taking into consideration the current context (i.e. the user/session properties, data properties and system/environment properties), andgenerates appropriate hardware tags for the data file. In Step 5, the file management module requests the hypervisorto create an SDC for the data file with the correspondinghardware tags. In Step 6, the hypervisor decryptsthe data file, and creates an SDC for the data file associatingthe appropriate tags with each word in the SDC. In Step 7, the file handle of the SDC is returned back to thepolicy/domain handler and the execution is returned backto the application.Use. In the Use stage, the DataSafe hardware tags eachword of the protected data in each SDC and persistentlytracks and propagates these tags, as shown by Step 8. Oncean SDC is set up for a data file, in accordance with thesession properties, any third-party application can operateon the protected data as it would on any regular machine.The DataSafe hardware will ensure that only those actions.

3. *Taxonomy*

Define a self-protection taxonomy, we started with selecting suitable dimensions and properties found in existing surveys. The aforementioned taxonomies described in Section 3, though relevant and useful, are not sufficiently specific and systematic enough for classifying self-protection approaches in that they either focus on adaptive systems in general, but not specifically on security, or focus on software security in general, but not on autonomic and adaptive security. Many focus on only certain architectural layers of software systems (such as middleware). Even when a taxonomy dimension is

appropriate for our purposes here, it is oftentimes too generic (e.g., open vs. closed) and need to be further qualified in the self-protection context. Furthermore, many of the taxonomies and classification schemes lean heavily towards implementation tactics and techniques (such as those for implementation patterns) but perhaps fall short on covering architectural strategies or styles (though some exceptions do exist, such as (Nguyen and Sood, 2011).

For such reasons, we have defined our own taxonomy to help classify existing self-protection and adaptive security research. The proposed taxonomy builds upon existing work surveyed in Section 3, and is a refinement and substantial extension of what we proposed in earlier work (Yuan and Malek, 2012).

4. Self-protection with Jade for J2EE Applications

4.1. Architecture-based sense of self" (SoS) is the capacity to detect the intrusion of foreign elements within the administrated system though the distinction of self from nonself. As we have seen in the overview of the Jade system, we aim at providing a component-based architectural representation of the administrated environment in order to enable observations and reconfigurations. The architecture of the application provides a notion of sense of self, as it defines the components which are supposed to be running on the machines and the communication channels which may be used by these components. Any execution which does not take place within these components or communication channels is considered nonself. The design of Jade also follows the design principles of immune systems .

*Autonomy***:** The self-protection policy which may be defined with Jade does not have to know much about the attacks it may have to face. It can detect abnormal behaviors, that is those which are nonself(which don't comply with the architecture of the application). In our scenario, we detect intrusions as nonselfbehaviors. For instance, if an application attempts to use a not-declared communication channel (binding), an alarm event is raised, which will triggers a counter-measure.

Distributability: There is no single point of failure to security. Any node can detect an abnormal execution, on the local machine or as an incoming request from another node. In our scenario, each node is responsible for the detection of nonself incoming communications. **Multi-layered**: Jade allows for the combination of many security techniques. Many detection and counter-measure mechanisms can me added by wrapping existing tools, deploying and configuring them. In our scenario, we wrap a firewall to detect nonself communications.

4.2. Scenario

We describe in this Section a simple scenario which aims at illustrating the implementation of sense of self protection policies on top of the Jade system. This scenario is currently under development. We consider a security flaw which allows attackers to execute arbitray code in one tier of the deployed J2EE architecture. An example of such a flaw is the Apache Chunked Encoding Overflow as defined in (Apache Chunked Encoding Overflow, 2002): Apache Web Server contains a flaw that allows a remote attacker to execute arbitrary code. The issue is due to the mechanism that calculates the size of "chunked" encoding not properly interpreting the buffer size of data being transferred. By sending a specially crafted chunk of data, an attacker can possibly execute arbitrary code or crash the server. Therefore, exploiting this security flaw (or anyone such), an attacker can gain control on a machine running this Apache server and subsequently attempt to attack other machines. Notice that Noel Depalma et al. this flaw can be exploited even if the Apache server is placed behing a firewall. Our assumption is that attackers will always find a way to bypass statically defined protection barriers. In order to detect nonself execution, we place a firewall on each machine involved in the J2EE architecture, as illustrated in Figure 3.

The firewall software is wrapped in a component, so that it can be deployed and configured by Jade (similarly as the other software resources). Each firewall is configured so that it will only accept requests issued by machines which have a declared binding pointing to it. For instance, firewall on Node4 will only accept requests from Node2 and Node3. The firewalls are automatically configured according to the deployed J2EE architecture. Moreover, each firewall is configured to behave as a intrusion detector. When the J2EE architecture is deployed, the communication ports used to connect the J2EE tiers are chosen randomly, which makes it more difficult for an attacker to exploit legal bindings. And an attempt

to use/scan an unbound port is detected by the firewall to raise an alarm event. Therefore, the configuration of the firewalls only enable requests which follows legal bindings and illegal requests raise an alarm.

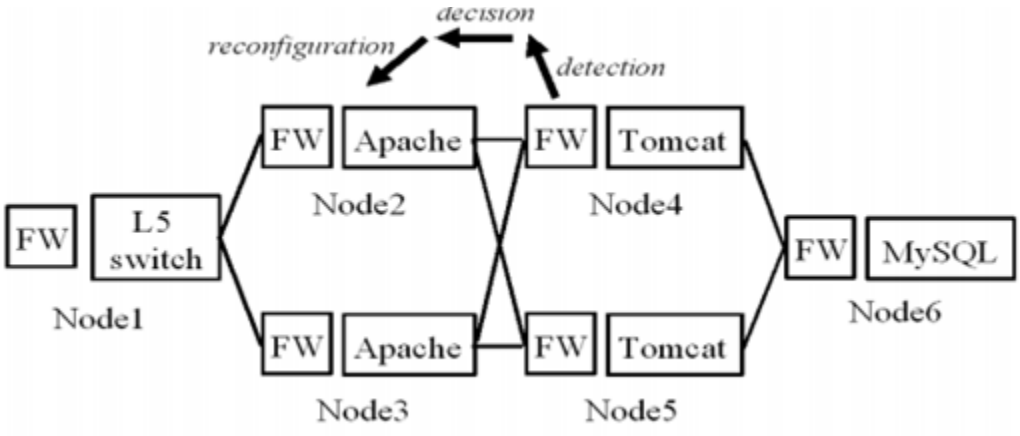

Figure 3: Self-protection scenario

Therefore, the deployment and configuration of firewalls allows detecting (some of) the nonselfbehaviors. Whenever an alarm is raised, different counter-measures can be executed: • the machine (in the cluster) which issued the request can be isolated from the J2EE architecture and replaced by another one (as in the repair algorithm). The reconfiguration of the architecture also reconfigures the firewalls accordingly. • a message can be sent to the human administrator to take any additional measure. • ananalyze of logs on the different tiers can eventually allow to find the remote machine which issued the web request (on the global J2EE web server), thus allows a reconfiguration of the firewall placed in front of the overall J2EE architecture (on Node1) to deny access from this remote machine.

MTERIALS AND METHODS

Research Methodology

This survey follows the general guidelines for systematic literature review (SLR) process proposed by Kitchenham (2004). We have also taken into account the lessons from (Brereton *et al.*, 2007) on applying SLR to the software engineering domain. The process includes three main phases: planning, conducting, and reporting the review. Based on the guidelines, we have formulated the following research questions, which serve as the basis for the systematic literature review:

— RQ1: How can existing research on self-protecting software systems be classified?

— RQ2: What is the current state of self-protection research w.r.t. this classification?

— RQ3: What patterns, gaps, and challenges could be inferred from the current research efforts that will inform future research?

We have detailed our review process in Appendix A, including the methodology and tasks that we used to answer the research questions (Section A.1) and the detailed SLR protocol including key words, sources, and selection criteria (Section A.2). As a result, we have included 107 papers published from 1991 to 2013, out of the total of over 1037 papers found. No survey can be entirely comprehensive. Our keywords-based search protocol restricts us to papers that explicitly address the self-protection topic while potentially leaving out relevant papers under different terms. Section 7 lists some of the interesting areas that are not in the scope of the survey.

RESULTS AND DISCUSSION

Today's distributed computing environments are increasingly complex and difficult to administrate. This complexity is such that the presence of bugs and security holes is statistically unavoidable. Therefore, access control policies become very difficult to specify and to enforce. A very promising approach to deal

with this issue is, following the autonomic computing vision, to design a self-protected system which is able to distinguish legal (self) from illegal (nonself) behaviors. The detection of an illegal behavior triggers a counter-measure to limit the exploitation of the intrusion and prevent further intrusions. In this vein, we have designed and implemented a system called Jade which allows the definition of autonomous administration programs. Jade relies on a component model for wrapping the administrated resources and provides support for the definition of autonomic managers which capture significant event and trigger relevant actions. Jade has been successfully used to implement selfoptimization and self-healing autonomic policies for clustered J2EE applications. In this paper, we investigated the application of Jade features to implement self-protection for J2EE applications. We showed how the knowledge of the (component-based) architecture of the administrated application can be exploited to implement a notion of self. In our scenario, firewalls are automatically deployed on every nodes and configured according to the deployed J2EE architecture. These firewalls allow detecting (some) non-self behaviors and taking adequate counter-measures. This work is at a preliminary stage, but opens many perspectives. It only detects some of the potential non-self behaviors. For instance, we plan to investigate the definition of detectors for SQL injection attacks (Boyd and Keromytis, 2004) based on an analysis of the logs generated by the administrated software components.

This paper has illustrated the benefits of evaluating the security properties of a software system using architectural models, and in particular at runtime. In support of our ar-gument, we illustrated (1) how existing ad hoc techniques to self-protection can be formulated as architecture-level pat- terns, thus paving the way for a systematic engineering ap- proach to construction of such systems, and (2) how those patterns could be realized in Rainbow. Our experiences with extending Rainbow to protect web applications have cor- roborated some of the hypothesized benefits of architecture- based self-protection (recall section 4). For instance, by ex- plicitly separating the DoS detection and mitigation capa- bility from the application logic, our solution can be easily reused in any other web application managed by Rainbow. In addition, by representing the self-protection capabilities as architectural elements (e.g., a protective wrapper connec- tor), our solution allows us to reason about the security pos- ture of a system in terms of its architectural configuration. This also enables adaptation of self-protection mechanism itself, e.g., addition and removal of new wrapper connectors while our experiences have been very positive so far, a number of research challenges.

REFERENCES

Barna C *et al.*, (2012). Model-based adaptive dos attack mitigation. *In International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* 119-128.

Bouchenak S, Boyer F, Hagimont D, Krakowiak S, Mos A, Depalma N, Quema V and Stefani JB (2005). Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. *24th IEEE Symposium on Reliable Distributed Systems (SRDS), Orlando, Florida*.

Boyd SW and Keromytis AD (2004). Sqlrand**:** Preventing sql injection attacks. *International Conference on Applied Cryptography and Network Security (ACNS)* 292-302.

Bruneton E, Coupaye T, Leclercq M, Quéma V and Stefani JB (2004). An Open Component Model and its Support in Java. *International Symposium on Component-based Software Engineering, Edinburgh, Scotland, may 2004*.

Casanova P *et al.*, (2011). Architecture-based run-time fault diagnosis. *In Proceedings of the 5th EuropeanConferenceon Software Architecture*.

Casanova P *et al.*, (2013). Diagnosing architectural run-time failures. To appear in SEAMS.

Cheng SW and Garlan D Stitch (2012). A language for architecture-based self-adaptation. *Journal of Systems and Software, Special Issue on State of the Art in Self-Adaptive Systems* **85** 12.

Cheng SW *et al.*, (2009). Evaluating the effectiveness of the rainbow self-adaptive system. *In ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*.

Chess DM *et al.*, (2003). Security in an autonomic computing environment. *IBM Systems Journal* **42**(1) 107-118.

Christianson B, Crispo B, Malcolm J and Roe M (2004). Lecture Notes in Computer Science. (Springer Berlin / Heidelberg) **2845** 569-574.

Foo B *et al.*, (2005). ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment. *In International Conference on Dependable Systems and Networks, 2005, DSN 2005, Proceedings (July 2005)* 508-517.

Forrest S, Hofmeyr SA and Somayaji A (1997). Computer immunology. *Communications of the ACM* **40**(10) 88–96.

Garay JA, Gennaro R, Jutla C and Rabin T (2000). Secure distributed storage and retrieval. *Theoretical Computer Sciences* **243**(1-2) 363-389.

Garlan D *et al.*, (No Date). Acme: Architectural Description of Component-Based Systems. In Foundations of Garlan D *et al.*, Rainbow: Architecture-based self-adaptation with reusable infrastructure (IEEE).

Gennari J and Garlan D (2011). Measuring attack surface in software architecture. Tech. Rep. CMU-ISR-11-121, Institute for Software Research, School of ComputerScience, Carnegie Mellon University.

Goel A, Po K, Farhadi K, Li Z and de Lara E (2005). The taser intrusion recovery system. *Twentieth ACM Symposium on Operating Systems Principles*, NewYork, NY, USA 163-176.

Hafiz M *et al.*, (2007). Organizing security patterns. *IEEE Software* **24**(4) 52-60.

Huang Y and Arun S (2002). Self-cleansing systems for intrusion containment. *Workshop on Self-Healing, Adaptive and Self-managed Systems (SHAMAN)*.

Huang Y *et al.*, (1995). Software rejuvenation: analysis, module and applications. *In Twenty-FifthInternational Symposium on Fault-Tolerant Computing*, FTCS-25, Digest of Papers 381-390.

Kephart J and Chess D (2004). The vision of autonomic computing. *Computer* **36**(1) 41-50.

Kephart JO (1994). A biologically inspired immune system for computers. *Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Cambridge,MA, US (MIT Press) 130–139.

Kephart JO and Chess DM (2003). The Vision of Autonomic Computing. *IEEE Computer Magazine* **36**(1).

Kitchenham B (2004). *Procedures for Performing Systematic Reviews*. Keele, UK, Keele University 33.

Leavens G and Sitaraman M (2000). *Component-Based Systems* (Cambridge University Press) Comput. Syst. **20**(4) 398-461, *Computer* **37**(10) 46-54.

Li M and Li M (2010). An adaptive approach for defending against ddos attacks. *Mathematical Problems in Engineering*.

Nagarajan A *et al.*, (2011). Combining intrusion detection and recovery for enhancing system dependability. *In 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)* 25 -30.

Nguyen Q and Sood AA **(2011).** Comparison of intrusion-tolerant system architectures. *IEEE Security Privacy* **9**(4) 24-31.

North DA **(1968).** Tutorial introduction to decision theory. *IEEE Transactions on Systems Science and Cybernetics* **4**(3) 200-210.

Okhravi H *et al.*, (2012). Creating a cyber moving target for critical infrastructure applications using platform diversity. *International Journal of Critical Infrastructure Protection* **5**(1) 30-39.

Sibai F and Menasce D (2011). *Defeating the Insider Threat via Autonomic Network Capabilities*.

Sousa P *et al.*, (2010). Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems* **21**(4) 452-465.

Stakhanova N *et al.*, (2007). A taxonomy of intrusion response systems. *International Journal of Information and Computer Security* **1**(1) 169-184.

Taton C, Bouchenak S, Boyer F, De Palma N, Hagimont D and Mos A (2005). SelfManageable Replicated Servers, VLDB Workshop on Design, Implementation, and Deployment of Databas Replication, Trondheim, Norway.

Wang F *et al.*, (2003). SITAR**:** a scalable intrusion-tolerant architecture for distributed services. *Foundations of Intrusion Tolerant Systems* 359-367.

Yoshioka N *et al.*, (2008). A survey on security patterns. *Progress in Informatics* **5**(5) 35-47.
Yuan E *et al.*, (2013). A survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*.
Zhu M *et al.*, (2011). VASP: virtualization assisted security monitor for cross-platform protection. *In Proceedings of the 2011 ACM Symposium on Applied Computing* 554-559.