A Systematic Survey of Self-Protecting Software Systems

ERIC YUAN, NAEEM ESFAHANI, and SAM MALEK, George Mason University

Self-protecting software systems are a class of autonomic systems capable of detecting and mitigating security threats at runtime. They are growing in importance, as the stovepipe static methods of securing software systems have been shown to be inadequate for the challenges posed by modern software systems. Self-protection, like other self-* properties, allows the system to adapt to the changing environment through autonomic means without much human intervention, and can thereby be responsive, agile, and cost effective. While existing research has made significant progress towards autonomic and adaptive security, gaps and challenges remain. This article 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.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures

General Terms: Algorithms, Design, Reliability, Security

Additional Key Words and Phrases: Self-protection, self-adaptive systems, self-* properties, autonomic computing, adaptive security

ACM Reference Format:
Yuan, E., Esfahani, N., and Malek, S. 2014. A systematic survey of self-protecting software systems. *ACM Trans. Auton. Adapt. Syst.* 8, 4, Article 17 (January 2014), 41 pages.
DOI:http://dx.doi.org/10.1145/2555611

1. INTRODUCTION

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, that is, self-protection.

Self-protection has been identified as one of the essential traits of self-management for autonomic computing systems. Kephart and Chess [2003] characterized self-protection from two perspectives: From a "reactive" perspective, the system automatically defends against malicious attacks or cascading failures, while from a "proactive" perspective, the system anticipates security problems in the future and takes steps to mitigate them. Self-protection is closely related to the other self-* properties,

This work was supported in part by awards CCF-1252644 and CCF-1217503 from the National Science Foundation, D11AP00282 from the Defense Advanced Research Projects Agency, and W911NF-09-1-0273 from the Army Research Office.
Authors' address: E. Yuan (corresponding author), N. Esfahani, and S. Malek, Department of Computer Science, George Mason University, Fairfax, VA 22030; email: eyuan@gmu.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 1556-4665/2014/01-ART17 $15.00
DOI:http://dx.doi.org/10.1145/2555611

TOYOTA EXHIBIT 1026
TOYOTA v. EMERGING AUTO
IPR2024-00814

such as self-configuration and self-optimization. On one hand, a self-configuring and self-optimizing system relies on self-protection functions to ensure the system security remains intact during dynamic changes. On the other hand, the implementation of self-protection functions may also leverage the same techniques used for system reconfiguration and optimization.

The past decade has seen extensive and systematic research being conducted around self-adaptive and self-managing systems. Research on self-protecting capabilities, however, has been relatively speaking less abundant. Scattered efforts can be found in various application domains such as autonomic computing, mobile and ad-hoc networks, sensor networks, fault-tolerant systems, trust management, and military domains like information survivability and tactical systems.

The contributions of this article 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 article 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 (1.1), illustrating it using a motivating example (1.2), and laying out the organization of the entire article (1.3).

1.1. Problem Description and Motivation

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. Over the years the frequency, complexity, and sophistication of attacks are rapidly increasing, causing severe disruptions of online systems with sometimes catastrophic consequences. From some of the well-publicized recent incidents, we can get a glimpse of the characteristics of such threats. The Conficker worm, first detected in 2008, caused the largest known computer infection in history and was able to assemble a botnet of several million hosts—an attack network that, if activated, would be capable of large-scale Distributed Denial of Service (DDoS) attacks. What is unique about Conficker is not just the scale it achieved, but also its use of sophisticated software techniques including peer-to-peer networking, self-defense through adaptation, and advanced cryptography [Dittmann et al. 2010]. The Stuxnet worm, discovered in 2010, is the first known malware to target and subvert industrial control systems. In addition to being credited with damaging the Iranian nuclear program, the malware demonstrates its ability to attack multiple architecture layers of the target system—exploiting the network and host-level vulnerabilities is only a stepping stone for malicious actions at

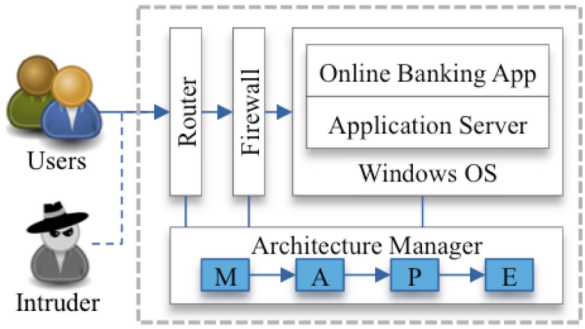


Fig. 1. Simple online banking system example.

the application level [Langner 2011]. The Duqu worm, discovered in September 2011, is a reconnaissance worm that does no harm to the infected systems but is tasked to collect and exfiltrate information such as valid digital certificates that may be used in future attacks. It further illustrates the deliberate and persistent nature of today's cyber threats [Bencsáth et al. 2012].

What has become increasingly clear from examples like these is that to protect today's software systems, especially those that are mission critical, applying static point security solutions (e.g., firewall and one-time password authentication) is no longer sufficient. Rather, there is a need for dynamic approaches that actively evaluate and reassess the overall security posture of the entire system architecture.

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, for example, be reassembled and provisioned with different components (e.g., using Service Component Architecture [Marino and Rowley 2010]). Similarly, a web service orchestrator could be constructed to dynamically discover and access different service providers (e.g., using a Business Process Execution Language (BPEL) engine). Runtime architectural changes like these tend to be security-relevant. For example, if a BPEL orchestrator switches a Partner Link from a non-responsive local service provider to an alternative external provider, the new SOAP connection becomes an additional source of vulnerability.

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.

1.2. 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 system is shown in Figure 1.

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.

While the first two examples merely execute predetermined actions using a particular component, the latter two clearly exhibit self-adaptive and self-protecting behavior at the system level. As this article will show later, many other self-protecting mechanisms are possible. How do these different approaches compare against one another? Are some more effective than others? If so, under what conditions? To better answer these questions, one must methodically evaluate the state of the art of the self-protection approaches, architectures, and techniques, and assess how they address the externally-driven and internally-driven security needs mentioned above.

This article seeks to take a step toward this goal by proposing a comprehensive taxonomy for self-protecting systems. The next section starts with a survey of existing taxonomies and classification schemes that are relevant.

1.3. Organization of This Article

The rest of this article 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.

2. SELF-PROTECTION DEFINED

Before we delve into the study, it is important to establish a working definition of self-protection property, given that our experience shows the term has been used rather loosely in the literature. The goal of this definition is to clarify what we have considered to be a self-protecting software system, which in turn has defined the scope of this study. Our understanding of the self-protection property is consistent with that laid out in FORMS [Weyns et al. 2012], a formal language for specifying the properties and architectures of self-* (i.e., self-management, self-healing, self-configuration, and self-protection) software systems. According to FORMS, a software system exhibiting a self-* property is comprised of two subsystems: a meta-level subsystem concerned with the self-* property that manages a base-level subsystem concerned with the domain functionality.

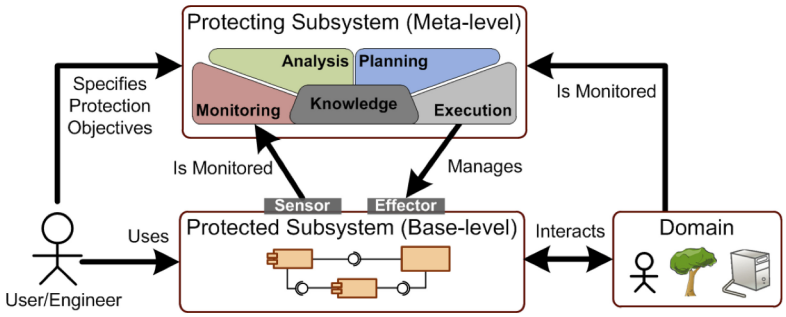


Fig. 2. Self-Protection in light of FORMS reference architecture.

Figure 2 shows what we consider to be a self-protecting software system in light of FORMS' concepts. The meta-level subsystem is part of the software that is responsible for protecting (i.e., securing) the base-level subsystem. The meta-level subsystem would be organized in the form of feedback control loop, such as the MAPE-K architecture depicted in the figure [Kephart and Chess 2003]. One should not interpret this reference architecture to mean that the base level subsystem is agnostic to security concerns. In fact, the base-level subsystem may incorporate various security mechanisms, such as authentication, encryption, etc. It is only that the decision of when and how those security mechanisms are employed that rests with the meta-level subsystem. In the case of the online banking system introduced in Section 1.2, the banking application logic corresponds to the base-level subsystem, while the logic used for detecting an intruder and mitigating the threat through changes in the system corresponds to the meta-level subsystem.

In addition to the intricate relationship between the meta-level and base-level subsystems, we make two additional observations. First, we underline the role of humans in such systems. Security objectives often have to be specified by human stakeholders, which are either the system's users or engineers. As we will see in the remainder of this paper, the objectives can take on many different forms (e.g., access control policies, anomaly thresholds). Second, we observe that for self-protection to be effective, it needs to be able to observe the domain environment within which the software executes. The domain environment is comprised of elements that could have an impact on the base-level software, but are outside the realm of control exercised by the meta-level subsystem. For instance, in the case of the online banking system, the domain could be other banking systems, which could impact the security of the protected system, but the meta-level subsystem has no control over them.

These concepts, although intuitive, have allowed us to define the scope of our study. For instance, we were able to distinguish between an authentication algorithm that periodically changes the key it uses for verifying the identities of users, and a system that periodically changes the authentication algorithm it uses at runtime. The former we classified to be an adaptive security algorithm, as the software used in provisioning security does not change, and therefore outside the scope of this article. While the latter we classified to be a self-protecting software system, as it changes the software elements used in provisioning security, and therefore within the scope of our study. Though other reference frameworks exist (such as control theory-based DYNAMICO [Villegas et al. 2013]), we will use the basic concepts introduced in this section throughout the paper to illustrate the differences between the self-protection approaches surveyed.

3. RELATED SURVEYS

Because self-protection mechanisms fall into the intersection of self-adaptive systems and software security, we have sought survey papers from both research domains.

First, even though the research field of self-adaptive and self-managing systems is a fertile research ground with rapidly advancing state of the art [Cheng et al. 2009; Lemos et al. 2013], little endeavor has been devoted to security as an adaptation property. Nonetheless, a number of related surveys are worth noting. Villegas et al. [2011] developed a control theory-based framework for surveying and evaluating self-adaptive software systems, in which security is included as one of the observable adaptation properties. None of their surveyed papers, however, covered security. A taxonomy of compositional adaptation [McKinley et al. 2004] focuses on composition as a key paradigm for adaptation, and describes a taxonomy based on how, when, and where software composition takes place. A related survey can be found in Sadjadi [2003] with a stronger focus on adaptive middleware. Even though these two surveys are not directly related to self-protection systems, our article draws certain taxonomy attributes for our purposes. Salehie and Tahvildari [2009] presented a comprehensive survey on self-adaptive software in general. It offers a taxonomy of self-adaptation that covers a variety of dimensions, some of which are security-relevant such as adaptation layers (OS, middleware, etc.), realization approach (such as static vs. dynamic decision making), and temporal characteristics (such as reactive vs. proactive adaptation). Even though many of these dimensions are relevant for self-protection, they need to be further defined in the specific context of security before they become useful. A comprehensive survey of self-healing systems [Psaier and Dustdar 2011] provides a taxonomy of system failure classes (security being one of them) and catalogs self-healing approaches such as architecture-based, agent-based, middleware-based, etc. Albeit not security-focused, the paper identified approaches and techniques that overlap with the self-protection research especially around the security goal of availability, as will be seen later in this article.

Across these surveys, the profound influence of the IBM Autonomic Computing (AC) vision as presented in Kephart and Chess [2003] is clearly visible, specifically around the adopted definitions of self-* properties and the MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) loop. A recent survey, for instance, further expanded the MAPE-K concept with a "Degree of Autonomicity" dimension with four progressive levels of maturity: Support, Core, Autonomous, and Autonomic [Huebscher and McCann 2008].

Second, we have found quite a number of relevant surveys in the software security domain. We recognize that computer security is a vast research domain, and that our objective is not to advance the state of the art of security techniques but rather to apply them to self-protecting systems. Consequently, we have limited our search to high-level surveys and review papers from which we can draw useful attributes for our self-protection taxonomy (described in Section 5). To that end, we have found good sources that cover various security concepts.

— 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 (encryption, access control policies, behavior policies, agent-oriented software engineering, and language-based security). Other related threat taxonomies include the top 25 software vulnerabilities [MITRE 2011] and dependability threats and failure types [Avižienis et al. 2004] that are a superset of security threats and failures.

— 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].

— 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.

Even though these generic surveys on security attacks, objectives, techniques and patterns are helpful, they do not specifically apply to software self-protection. Four other surveys, however, offer more pertinent insight into how software systems adapt to security threats: First, Elkhodary and Whittle [2007] provided a good survey on adaptive security mechanisms. It builds on top of the taxonomy of computational paradigms defined in Sadjadi [2003], and adds additional dimensions such as reconfiguration scale and conflict handling. These dimensions are certainly applicable to self-protection systems in general; however, this article's focus is primarily on the application layer. Second, Nguyen and Sood [2011] offered an up-to-date survey on *Intrusion Tolerant Systems* (ITS), a class of self-protecting systems that focus on continued system operations even in the presence of intrusion attacks. ITS architectures are often based on fault tolerance techniques. Some research efforts identified in this article are also covered in our analysis in Section 6. As correctly pointed out by the authors, these approaches are by no means mutually exclusive and may be used together. Third, Stakhanova et al. [2007a] and Shameli-Sendi et al. [2012] surveyed a different class of systems called *Intrusion Response Systems* (IRS) that focus on dynamic response mechanisms once an intrusion has been detected. Both surveys proposed an IRS taxonomy that included dimensions such as adjustment ability (adaptive vs. nonadaptive), response selection (static, dynamic, or cost-sensitive), and response type (proactive vs. reactive), which overlap to some extent with our self-protection taxonomy. Cost-sensitive response selection, in particular, corroborated with a similar trend we have identified in our survey.

Even though ITS and IRS have moved beyond traditional static and human-driven security mechanisms, they are still intrusion-centric and perimeter based and as such do not yet constitute true self-protection. In fact, none of the four surveys focused specifically on self-protection research in the AC context. Nor did any of them follow the systematic literature review methodology.

4. RESEARCH METHOD

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 with respect to 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.

5. TAXONOMY

To define a self-protection taxonomy for RQ1, 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]. It consists of 14 dimensions that fall into three groups: Approach Positioning, Approach Characterization, and Approach Quality. They are defined and illustrated in Figure 3 and Figure 4, and explained in the following sections.

5.1. Approach Positioning

The first part of the taxonomy, Approach Positioning, helps characterize the "WHAT" aspects, that is, the objectives and intent of self-protection research. It includes five dimensions, as depicted in the left part of Figure 3.

(T1) Self-Protection Levels. This dimension classifies self-protection research based on the level of sophistication of its meta-level subsystem (as defined in Section 2).

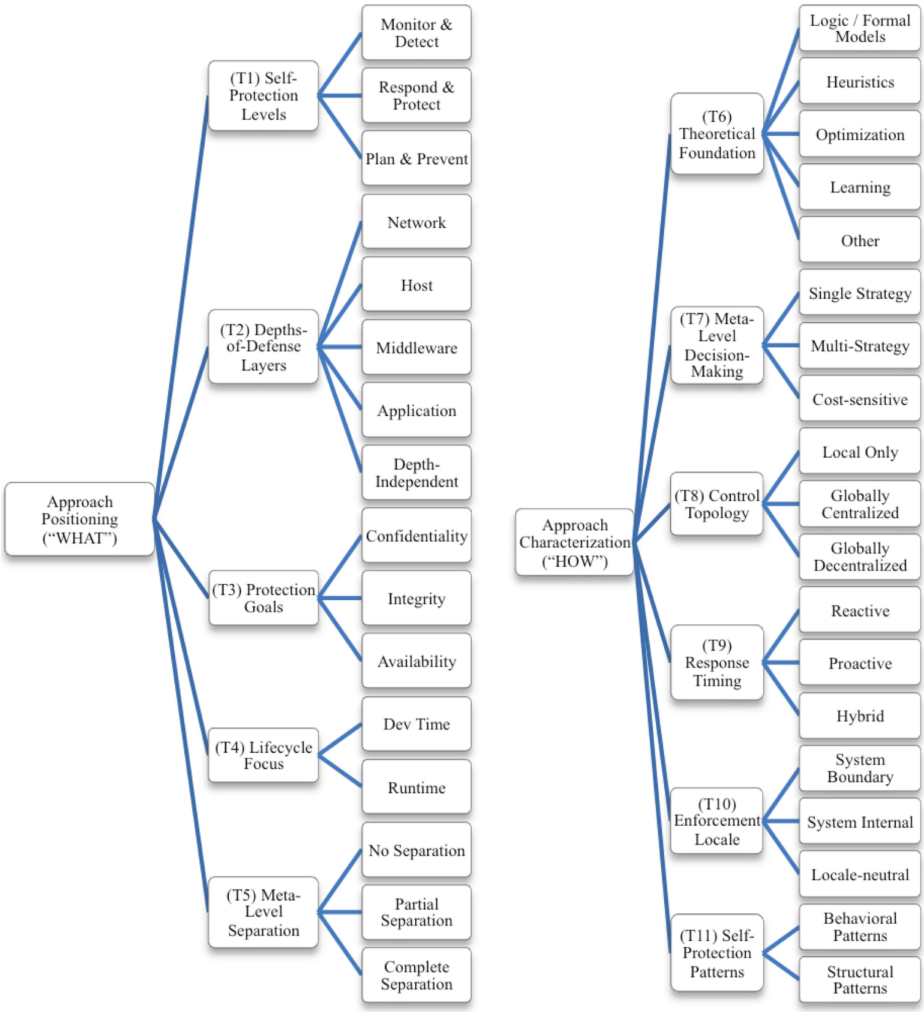


Fig. 3. Proposed taxonomy for self-protection.

"Monitor & Detect" is the most basic level, indicating the protecting subsystem is equipped with the capability to constantly monitor for security threats and detect anomalous or harmful activities from normal system operations. The next level is "Respond & Protect", which indicates the subsystem's ability to take action against the detected attack or anomaly. This implies the protecting subsystem can, ideally in an autonomous fashion, (a) characterize and understand the nature/type of the attacks, and (b) deploy the proper countermeasures to mitigate the security threat and maintain normal system operations to the extent possible – a property often called "graceful degradation". The third level, "Plan & Prevent", represents the highest level of sophistication; a security approach reaching this level allows a system to adapt and strengthen its security posture based on past events so that known security faults are prevented. We illustrate this dimension using the motivating example of Section 1.2.

— The online banking system is at the *Monitor & Detect* level if it is equipped with a network based IDS device connected to the router, which can detect an intrusion

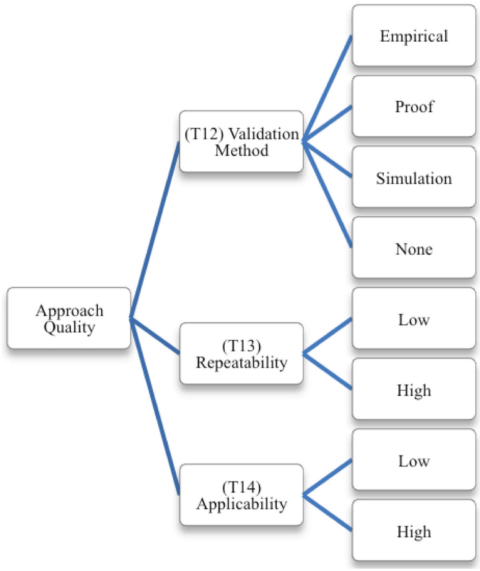


Fig. 4. Taxonomy for self-protection.

attempt based on known attack signatures (such as a DoS attack to the banking server), and generate an appropriate alert to the ARM, which acts as the "meta-level subsystem" for self-protection;

— The system is at the *Respond & Protect* level if, in addition to the previous level, the ARM component responds to the router alert and changes the firewall policy to block all traffic from the source domain;

— The system is at the *Plan & Prevent* level if, in addition to the previous level, the ARM also reviews the history of such attacks and moves the web server to a different IP address, so that future DoS attacks are rendered ineffective.

The three levels are consistent with (and in fact inspired by) Kramer and Magee's [2007] three-level reference architecture for self-managed systems. It is easy to see the mapping from the self-protection levels to Component Management, Change Management, and Goal Management, respectively. It may also be envisioned that each self-protection level may have its own MAPE-K loop; therefore this dimension is not in conflict with the IBM reference architecture.

(T2) Depths-of-Defense Layers. This dimension captures the renowned security principle of Defense in Depth, which simply acknowledges the fact that no single security countermeasure is perfectly effective and multiple layers of security protections should be placed throughout the system. For self-protecting systems, the following defensive layers are possible, starting with the outmost layer (please see Section 6.1 for examples of security countermeasures at each layer).

— *Network*. Focuses on communication links, networking protocols, and data packets.

— *Host*. Involves the host environment on a machine, involving hardware/firmware, OS, and in some occasions hypervisors that support virtual machines.

— *Middleware*. With the prevalence of component-based and service-oriented systems, use of middleware such as application servers (e.g., JEE), object brokers (e.g., CORBA) and service buses (e.g., JBoss ESB) are becoming a common practice and as such may be used as an additional layer of defense.

— *Applications*. As the last line of defense, application level security is usually concerned with programming language security and application-specific measures.
— *Depth-Independent*. This fifth layer is a special value to indicate any self-protection research that is not specific to any architecture layers. One example may be an approach that addresses self-protection in terms of software architecture abstractions such as software components, connectors, configurations, and architecture styles. A software architecture-based approach enjoys many benefits such as generality, abstraction, and potential for scalability, as pointed out by Kramer and Magee [2007].

The counter-intrusion example given earlier for dimension T1 is clearly a network layer defense. The online banking system can also choose to have a host-level defense such as a periodic patching mechanism to install OS patches that remediate Windows OS vulnerabilities, a middleware level defense that configures a cluster of redundant application servers under a Byzantine agreement protocol (as described by Castro and Liskov [2002]), and an application-level defense where the ARM component dynamically directs the web-based banking application to adopt different levels of security policies.

(T3) Protection Goals. This dimension classifies research approaches according to the security goal(s) they intend to achieve. Here we follow the traditional CIA model for its simplicity:

— *Confidentiality* protects against illegal access, spoofing, impersonation, etc.
— *Integrity* protects against system tampering, hijacking, defacing, and subversion.
— *Availability* protects against degradation or denial of service.

Other goals such as Accountability, Authenticity, and Non-Repudiation may also be considered as implicit subgoals that fit under this model.

In some cases, a security countermeasure may help meet multiple protection goals. Suppose in the online banking example, the banking application is compiled using the StackGuard compiler [Cowan et al. 1998] to safeguard against buffer overflow attacks. This technique stops the intruder from obtaining user financial data stored in memory (confidentiality) *and* from illegally gaining control of the banking application (integrity) through buffer overflows. Note that in this case the technique does not help with the availability goal; we will return to this point later in this article.

(T4) Lifecycle Focus. This dimension indicates what part of the software development lifecycle (SDLC) a self-protection approach is concerned with. For the purposes of this article we simply use two phases, Development Time and Runtime, with the former encompassing also the design, testing, and deployment activities. Security at runtime is undoubtedly the primary concern of self-protection. Nonetheless, from the software engineering viewpoint it is also necessary to take into account how to better design, develop, test, and deploy software systems for self-protection.

As a concrete example, suppose all runtime system auditing data is made available to the development team of the online banking system. By feeding data into the automated testing process, the team can make sure all new code is regression-tested against previously known vulnerabilities, or use the system logs to train the meta-level self-protection mechanisms.

(T5) Meta-Level Separation. This dimension indicates how Separation of Concerns as an architectural principle is applied in a self-protection approach. Specifically, the FORMS reference architecture [Weyns et al. 2012] calls for the separation between the meta-level subsystem and the base-level subsystem, logically and/or physically. The degree of separation is useful as a telltale sign of the degree of autonomicity of

the system. In the security context, it also takes on an added significance, as the meta-level self-protection logic often becomes a high value target for the adversary and thus needs special fortification. Three values are used here – No Separation, Partial Separation, and Complete Separation. Continuing with the online banking system example, complete separation of security concerns is achieved when all self-protection logic is contained in the ARM component as illustrated in Figure 1, and the component (along with communication channels to/from it) is deployed in dedicated, trusted hardware. On the other hand, if the ARM runs in the same application server as the banking application itself, or the security policy decisions are embedded in the banking application code, the degree of separation is low.

5.2. Approach Characterization

The second group of the taxonomy dimensions are concerned with classifying the "HOW" aspects of self-protection research. It includes five dimensions shown in the right half of Figure 3.

(T6) Theoretical Foundation. As a self-protecting software system takes autonomic and adaptive actions against malicious attacks, it often needs to consider many factors from the runtime environment and choose the optimal or near-optimal course of action out of a vast problem space. The theoretical foundation of the approach, as captured in this dimension, is therefore critical and deserves close examination. The following subcategories are defined.

— *Logic/formal models*, which involve logic or other mathematically-based techniques for defining security related properties, as well as the implementation and verification of these properties. The design of the online banking system, for example, may include security policies formulated by finite state automata (such as those defined in Schneider [2003]), and formal proof of policy enforceability;
— *Heuristics*-based, which include knowledge-based, policy-based, or rule-based models whose parameters may change at runtime. For example, the online banking system may implement a policy that disables a user account when suspicious fund withdrawal patterns arise. In this case, these patterns are based on heuristic rules such as a maximum daily withdrawal threshold. The system may further lower or increase the threshold according to security threat levels;
— *Optimization*, which employs analytical techniques that model security-related system behavior through quantitative metrics that is used to select the optimal adaptation strategy. For example, the banking system may use a utility function to model a user's preference between convenience/user-friendliness and strengths of protection, and set security policies accordingly (e.g., username/password vs. multifactor authentication);
— *Learning*-based models, including a rich variety of techniques that use historical or simulated data sets to train the system's autonomic defences. The learning process could be based on cognitive, data mining, stochastic/probabilistic models, etc. The banking system's router, for instance, may use a neural net algorithm to differentiate intrusions from normal network behavior [Kumar et al. 2010].

Note that these models are not meant to be mutually exclusive. In fact, as we will see in the survey results, many approaches leverage more than one model.

(T7) Meta-Level Decision-Making. This dimension attempts to further characterize self-protection research by examining its decision-making strategy and "thought

process". Here we adopt the following rather coarsely grained values (again, illustrated using the online banking system example).

— *Single strategy* is the simplest approach with a single objective, a single decision model, or a single type of attacks/vulnerabilities in mind (many examples given earlier fall into this category).
— *Multistrategy* involves multiple levels of decisions, metrics, and tactics. For instance, consider a situation in which the banking system deploys two intrusion detection sensors, one network-based and the other host-based on the application server. Simple intrusions such as port scanning and buffer overflows are deterred at the device level, but the ARM component also correlates the network and host alerts to look for higher-level attack sequences.
— *Cost-sensitive modeling* is a special case in which security decisions involve tradeoffs with other non-security related factors, such as costs or Quality of Service (QoS) requirements. For example, under certain situations, the banking application may not shut itself down to cope with a user account breach because many other legitimate users will be impacted, resulting in big loss of revenue.

(T8) Control Topology. More often than not, modern software-intensive systems are logically decomposed into separate self-contained components and physically deployed in distributed and networked environments. Self-protection functionality, therefore, needs to be implemented and coordinated among different components and machines. This dimension looks at whether a self-protection approach focuses on controlling the local (i.e., a single host or node) or global scale of the system. For those approaches at the global scale, this dimension also specifies whether they use centralized or decentralized coordination and planning. Under a centralized topology, system self-protection is controlled by a single component that acts as the "brain", whereas under a decentralized topology, the nodes often "federate" with one another in a peer-to-peer fashion without relying on a central node. In the online banking system, for instance, self-protection is *globally centralized* if the ARM component is hosted on a dedicated server that monitors, controls, and adapts security countermeasures on all other devices and servers. Alternatively, if the banking system consists of multiple interconnected servers (possibly at different locations) and each server hosts its own architecture manager component, the topology is *globally decentralized*. In a more trivial situation, the topology would be "local only" if the self-protection technique is used within a single server.

(T9) Response Timing. This dimension indicates when and how often self-protecting actions are executed, which in turn is dependent on whether the approach is reactive or proactive. In reactive mode, these actions occur in response to detected threats. In proactive mode, they may occur according to a predefined schedule, with or without detected threats. Some systems may include both modes. The security countermeasures illustrated earlier using the online banking example, such as intrusion prevention or controlling access to a user account, all fall into the reactive category. Alternatively, the banking system could use software rejuvenation techniques (introduced by Huang et al. [1995]) to periodically restart the web application instances to a pristine state, to limit damage from undetected attacks.

(T10) Enforcement Locale. This dimension indicates where in the entire system self-protection is enforced. Here we adopt a metric from [Hafiz et al. 2007] and define the values as *System Boundary* or *System Internal*. In the former case, self-protection is enforced at the outside perimeter of the system (such as firewalls, network devices, or hosts accessible from external IP addresses). In the latter case, self-protection

mechanisms cover internal system components. The distinction may be easily seen in the online banking example: The router and the firewall represent the system boundary that needs to be protected against intrusions, whereas the web application and the middleware components represent system internals that may also be protected by access control policies issued from the ARM component. Self-protection approaches independent of enforcement locations are categorized as *locale-neutral*.

(T11) Self-Protection Patterns. This dimension indicates any recurring architectural patterns that rise from the self-protection approaches. Many architecture and design patterns exist, but as we can see in the next section several interesting patterns have emerged in our research as being especially effective in establishing self-protecting behavior. Here we simply mention them in two groupings and describe their details in Section 6.7.

— *Structural patterns* use certain architectural layouts to situate, connect, and possibly reconnect system components to achieve better integrity, robustness, and resiliency against attacks. These patterns include Protective Containment, Agreement-based Redundancy, Implementation Diversity, Countermeasure Broker, and Aspect-Orientation.
— *Behavioral patterns* seek to reconfigure and adapt the runtime behavior of existing system components and their connections without necessarily changing the system architecture. These patterns include Protective Recomposition, Attack Containment, Software Rejuvenation, Reconfiguration on Reflex, and Artificial Immunization.

Please note that these patterns are not mutually exclusive. It is conceivable that a system may use a combination any number of them to provide more vigorous and flexible self-protection behavior.

5.3. Approach Quality

The third and last section of the taxonomy is concerned with the evaluation of self-protection research. Dimensions in this group, as depicted in Figure 4, provide the means to assess the quality of research efforts included in the survey.

(T12) Validation Method. This dimension captures how a paper validates the effectiveness of its proposed approach, such as empirical experimentation, formal proof, computer simulation, or other methods. The selected sub-category for the validation method is closely related to the selected subcategory for the theoretical foundation (T6) of the proposed approach. When the approach is based on logic/formal methods, validation is expected to be in the form of formal proof. Approaches that are based on heuristics and optimization demand empirical validation. Finally, simulation is a perfect fit for learning-based models.

(T13) Repeatability. This dimension captures how a third party may reproduce the validation results from a surveyed paper. This dimension classifies repeatability of research approaches using a simplified measure:

— High repeatability, when the approach's underlying platform, tools and/or case studies are publicly available;
— Low repeatability, otherwise.

(T14) Applicability. A self-protection approach or technique, though effective, may or may not be easily applied in a broader problem setting. Similar to repeatability, we use a simple "low" vs. "high" measure:

— Low applicability, when the approach is specific to a particular problem domain (suppose the online banking system uses users' income and spending patterns to calculate their risk profile), is dependent upon a proprietary framework or implementation, or requires extensive infrastructure support that may not be generally available;
— High applicability, otherwise.

6. SURVEY RESULTS AND ANALYSIS

A large number of research efforts related to self-protecting systems and adaptive security have been identified in this survey, and are then evaluated against the proposed taxonomy. The detailed evaluation results are included in Appendix B.

Note that the classifications are meant to indicate the primary focus of a research paper. For example, if a certain approach does not have a checkmark in the "Availability" column under Protection Goals, it does not necessarily indicate that it absolutely cannot help address availability issues. Rather, it simply means availability is not its primary focus.

By using the proposed taxonomy as a consistent point of reference, many insightful observations surface from the survey results. The number of the research papers surveyed will not allow elaboration on each one of them in this paper. Rather, we highlight some of them as examples in the observations and analysis below.

6.1. Correlating Self-Protection Levels and Depths of Defense

Starting with the Self-Protection Levels (T1) dimension, we see that abundant research approaches focus on the "Monitor & Detect" level, such as detecting security-relevant events and enforcing security policies that respond to these events. For example, Spanoudakis et al. [2007] used Event Calculus to specify security monitoring patterns for detecting breaches in confidentiality, integrity and availability. Liang and Sekar [2005] used forensic analysis of victim server's memory to generate attack message signatures. At the "Respond & Protect" level, research efforts attempt to characterize and understand the nature of security events and select the appropriate countermeasures. For example, He et al. [2010b] used policy-aware OS kernels that can dynamically change device protection levels. Taddeo and Ferrante [2009] used a multi-attribute utility function to rank the suitability of cryptographic algorithms with respect to the runtime environment and then used a knapsack problem solver to select optimal algorithm based on resource constraints. At the highest "Plan & Prevent" level, research efforts are relatively speaking not as abundant; such efforts seek to tackle the harder problem of planning for security adaptation to counter existing and future threats. To that end, many approaches offer a higher degree of autonomicity. Uribe and Cheung [2004], for instance, used a formal network description language as the basis for modeling, reasoning, and auto-generating Network-based Intrusion Detection System (NIDS) configurations. The Self-Architecting Software SYstems (SASSY) framework, by contrast, achieves architecture regeneration through the use of Quality of Service scenarios and service activity schemas [Malek et al. 2009; Menasce et al. 2011].

Along the Depths-of-Defense Layers (T2) dimension, we see many self-adaptive security approaches focusing on the "traditional" architecture layers, such as network, host, middleware and application code. At the network level, abundant research can be found in the field of intrusion-detection and intrusion-prevention. Examples include Yu et al. [2007, 2008] who used fuzzy reasoning for predicting network intrusions, and the Wireless Self-Protection System (WSPS) [Fayssal et al. 2008] which uses both standard and training based anomaly behavior analysis that can detect and deter wide range of network attack types. Because network vulnerabilities are closely linked to the

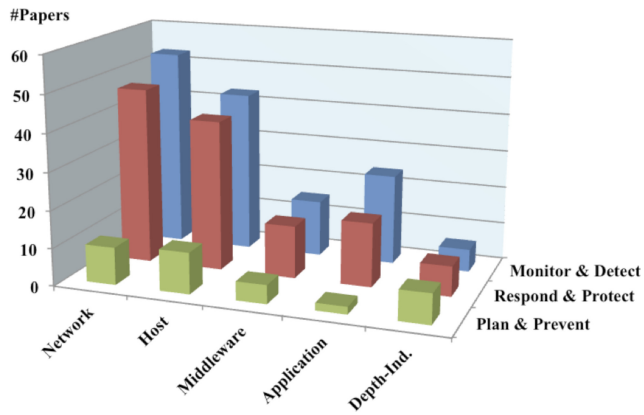


Fig. 5. Correlating self-protection levels and depths of defense.

network topology and equipment configurations, devoted research can also be found on adapting network security policies based on such network characteristics [Burns et al. 2001]. At the host/node level, antivirus and malware detection/prevention have been receiving a lot of attention from the research community (a latest example on adaptive rule-based malware detection can be found in Blount et al. [2011]).

When we shift our focus to defense at the middleware level, self-protection approaches start to focus and/or leverage distributed middleware platforms such as Java Enterprise Edition or JEE (as in De Palma et al. [2012]), object request brokers (as in Yau et al. [2006]), and message-oriented middleware (as in Abie et al. [2008] and Abie [2009]). More importantly, researchers started to recognize the benefit of a robust middleware layer as an extra line of defense against host and application level attacks (as seen in the QuO adaptive middleware example [Atighetchi et al. 2003, 2004]). More recent research has started to focus on adaptive security for web services middleware in a SOA. Such research can be found, for example, around service trust [Maximilien and Singh 2004] and service-level assurance [Casola et al. 2008]. Research around the security behavior of a collection of services (such as a BPEL orchestration or a composite service), however, seems to be lacking.

As we move up to the application level, self-adaptive security research is more concerned with programming language level vulnerabilities such as those concerning pointers, memory buffers, and program execution points. Lorenzoli et al. [2007], for example, presented a technique, called From Failures to Vaccine (FFTV), which detects faults using code-level assertions and then analyzes the application to identify relevant programming points that can mitigate the failures.

Research seems to be sparse on the adaptation of the software architecture as a whole in dealing with security concerns. Nonetheless, the "Depth-Independent" subcategory in this dimension does capture some interesting and sophisticated approaches. The RAINBOW [Cheng et al. 2006; Garlan et al. 2004] and SASSY frameworks are two examples that fit into this category, even though they are not specifically focused on self-protection alone. Additionally, work by Morin et al. [2010] and Mouelhi et al. [2008] represent a key example of applying "models@runtime" thinking to security adaptation, which can be applied to all architecture layers.

To take a further look at the research trends, we use Self-Protection Levels and Depths of Defense as two crosscutting dimensions to map out the existing self-protection research approaches, as shown in Figure 5. In the plot, the height of each column represents the number of papers per each self-protection level and each line of

defense. We clearly see that abundant research exist at the network and host levels for attack detection and response, fueled by decades of research in such fields as Intrusion Detection/Intrusion Prevention (ID/IP), Antivirus/Malware, and Mobile Ad-hoc Networks (MANET) security. It becomes apparent, however, that existing research start to "thin out" as we move up the two respective dimensions. Autonomic and adaptive security approaches that apply to attack prediction and prevention, especially at middleware, application, or abstract architecture levels, appear to be a research gap to be filled.

6.2. Runtime vs. Development-Time

Along the Lifecycle Focus (T4) dimension, the vast majority of self-protection research (96% to be exact) focuses on runtime. Indeed, it is a general consensus that software components are never completely fault-free and vulnerability-free no matter how carefully they are designed and coded. Nonetheless, 18% of the papers also involve development time activities. They generally fall under three cases.

— Runtime techniques that happen to need development time support. The FFTV approach [Lorenzoli et al. 2007], for instance, complements runtime healing/protection strategies with design-time construction of "oracles" and analysis of relevant program points, and also with test-time generation of reference data on successful executions. In Hashii et al. [2000], the dynamically reconfigurable security policies for mobile Java programs also rely on supporting mechanisms put in at deployment time (such as policy class loaders).
— Programming language level protection approaches that focus primarily at development time. They employ novel techniques such as fuzz testing [Abie et al. 2008], whitebox "data perturbation" techniques that involve static analysis [Ghosh et al. 1998; Ghosh and Voas 1999], or software fault injection which merges security enforcement code with the target code at compile time [Erlingsson and Schneider 2000].
— Model-driven approaches that essentially blur the line between development time and runtime. They achieve self-protection through incorporating security requirements into architecture and design models, and relying on Model-Driven Engineering (MDE) tool sets to instrument security related model changes at runtime. In addition to Morin et al. [2010], the Agent-oriented Model-Driven Architecture (AMDA) effort [Xiao 2008; Xiao et al. 2007] also falls into this category. Such approaches may hold some promise for future self-protection research, although empirical results so far are far from convincing.

Because the philosophy, structure, and process through which software components are constructed could have a significant impact on their quality of protection at runtime, we believe that full lifecycle approaches combining development-time and runtime techniques will result in the best self-protection of software systems – another research opportunity.

6.3. Balancing the Protection Goals

Along the Protection Goals (T3) dimension, the survey results revealed that most research efforts seem to focus on either Confidentiality and Integrity or Availability, but not all three goals. As shown in the Venn diagram in Figure 6(a), a large portion of the survey papers focus on Confidentiality (68%) and Integrity (81%), but only 40% of the papers address availability, and even fewer (20%) deal with all three goals. The dichotomy between confidentiality and availability objectives is not surprising: the former seeks mainly to protect the information within the system, but is not so much concerned with keeping the system always available; the opposite is true for the latter. For

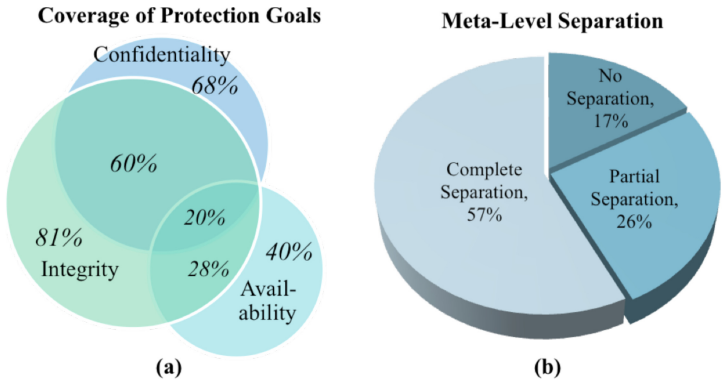


Fig. 6. (a) Coverage of protection goals; (b) Meta-level separation.

example, when a host-based intrusion is detected, the typical system responses involve stopping/restarting a service, rebooting the server, disable user logins, etc. [Strasburg et al. 2009] – system confidentiality and integrity are preserved, whereas availability suffers.

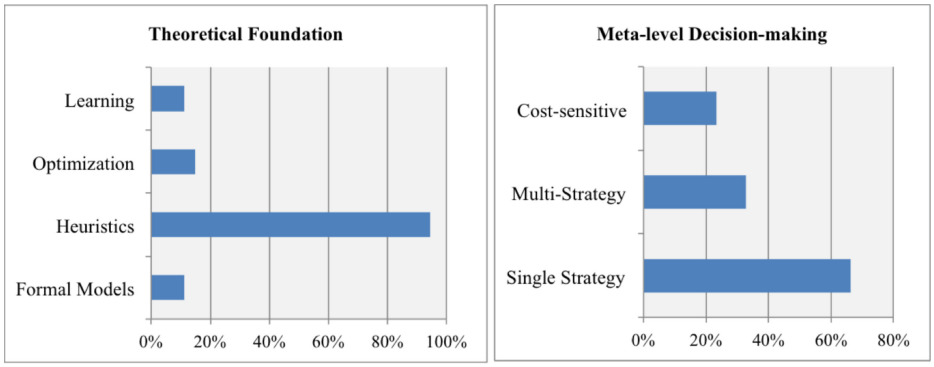
Preserving system availability, on the other hand, goes beyond the security realm and is closely related to system QoS, thus requiring different treatments. Intrusion Tolerant Systems (e.g., Sousa et al. [2007] and Reiser and Kapitza [2007b]) address availability especially well by leveraging fault tolerance mechanisms, though they tend to focus on the network and host levels rather than taking a broader architectural approach.

This observation, though a bit subtle, shows that a selfprotecting system may need to include a "best of breed" combination of adaptive security techniques rather than relying on a single mechanism, to meet all protection goals.

6.4. Separation of Meta-Level and Base-Level Subsystems

As introduced in Section 5.1, the Meta-level Separation dimension (T5) intends to show self-protection research separates the meta-level ("protecting") components from the base-level ("protected") components. The survey results summarized in Figure 6(b) indicate that self-protection architectures from 83% of the papers show at least partial separation, which serves as strong evidence that the meta-level separation proposed in Section 2 has been indeed widely practiced in the research community. Instantiations of the meta-level subsystem take on many interesting forms among the surveyed papers, such as managerial nodes [Abie 2009], Security Manager [Ben Mahmoud et al. 2010], guardians [Montangero and Semini 2004], Out-of-Band (OOB) server [Reynolds et al. 2002], or control centers [Portokalidis and Bos 2007]. Those approaches that have been put under "partial separation" either rely on certain enforcement mechanisms that are an intrinsic part of the base-level subsystem (e.g., through library interposition with the protected application [Liang and Sekar 2005]), or do not provide a clear architecture that depicts the separation boundary.

The remaining 17% of papers that do not exhibit meta-level separation deserve special attention. Closer examination reveals two contributing factors are the primary "culprits", with the first having to do with the domain environment and the second pertaining to the nature of the research technique. First, for MANETs, wireless sensor networks, or agent-based networks of a self-organizing nature, because no central control typically exists within the system, self-protecting mechanisms would have to be implemented within each network node or agent. This is the case with Adnane et al.



(a) (b)

Fig. 7. (a) Theoretical foundation (b) Meta-level decision-making.

[2008], Alampalayam and Kumar [2003], Chigan et al. [2005], and Jean et al. [2007]. It is no coincidence that these papers also fall into the "Global – Decentralized" subcategory of the "Control Topology" dimension (T8); see Section 6.6 for more details. Since each node/agent is just as susceptible to attack and subversion as any other node/agent, protecting the security mechanism itself becomes a real challenge. Costa et al. [2008] used a mechanism called Self-Certifying Alerts (SCA) that are broadcasted over an overlay network to overcome this problem, but the challenge of "protecting the meta-level" in a globally decentralized topology is largely unanswered in the surveyed papers.

The second contributing factor arises from those approaches that use code generation and code injection techniques at the application level, because the protection mechanism becomes part of the codebase, meta-level separation is obviously lacking. It is quite revealing that most of the papers cited in Section 6.2 as having a development time focus – such as FFTV [Lorenzoli et al. 2007], SASI [Erlingsson and Schneider 2000], and AMDA [Xiao et al. 2007] – belong to this case! Here, we see another research challenge, that is, to find ways to employ valuable techniques (e.g., programming language analysis and model-driven engineering) while staying true to the self-protection reference architecture with clear meta-level vs. base-level separation.

6.5. Foundations and Strategies for Self-Protection Decision-Making

When it comes to the "HOW" part of the taxonomy, we see the surveyed papers employ a large variety of models, schemes, algorithms, and processes. First of all, a simple analysis along the Theoretical Foundation dimension (T6) shows a predominant use of heuristics-based methods, as shown in Figure 7(a), in such forms as expert systems [Neumann and Porras 1999; Porras and Neumann 1997], policy specification languages [Burns et al. 2001], event-condition-action rules [English et al. 2006], directed graphs [Balepin et al. 2003], genetic/evolutionary algorithms [Raissi 2006], structured decision analysis (such as Analytic Hierarchy Process or AHP, as in [Ben Mahmoud et al. 2010]), or human input as a last resort [White et al. 1999]. Even when nonheuristics-based methods are used, whether it is using formal semantics [Dragoni et al. 2009] or utility function based optimization [Taddeo and Ferrante 2009] or stochastic modeling [Sousa et al. 2006], they are more often than not complemented by heuristics. Our analysis along this dimension has revealed the following insights.

— Given the multitude of decision factors such as objectives, system properties, resource constraints and domain-specific environment characteristics, the problem

space for self-protection decision making is usually too large for classic problem solving methods (though they may still prove effective in solving a small, narrowly focused subproblem, such as formalisms and proofs around software rejuvenation [Ostrovsky and Yung 1991] or reinforcement-based learning for malware detection [Blount et al. 2011]);

— Because the entire system is at stake with regard to self-adaptive security decisions, a wrong move may lead to severe consequences. As such, few approaches in this survey leave such decisions (such as threat containment or deploying countermeasures) solely to an algorithm without any heuristic input. Indeed, as pointed out in a number of papers [Al-Nashif et al. 2008; Crosbie and Spafford 1995], autonomic responses often require near 100% accuracy in threat detection and characterization (i.e., the rate of false positives at near zero). Many papers went to great lengths to analyse and reduce the rate of false positives while maintaining a high detection rate (i.e., low false negatives), with varying results.

— The lack of nonheuristics-based methods may also be explained by the daunting challenge of quantitatively assessing the overall security posture of a complex software system. Several papers proposed various metrics as attempts to this goal – the Security Health Index comprised of a weighted basket of security metrics from Savola and Heinonen [2010] and the Compromise Confidence Index as a measure to pinpoint attack location from Foo et al. [2005a] are representative examples. Empirical validation of these metrics, however, is far from sufficient and convincing from the surveyed papers. This is definitely a pressing research need, especially in today's heated domain of cyber warfare.

The meta-level decision-making dimension (T7) of our taxonomy offers an even more interesting perspective on self-protection decision making. From Figure 7(b), we can see two important opportunities in self-protection research.

From Single-Strategy to Multi-Strategy. Some researchers have come to the realization that a single technique or a point solution is no longer enough to match the ever-increasing sophistication of today's cyberattacks, as described in Section 1.1. Rather, self-protecting systems should be able to (1) detect higher-level attack patterns and sequences from low-level events, (2) have an arsenal of countermeasures and response mechanisms that can be selectively activated depending on the type of attack, and (3) have a concerted strategy to guide the selection and execution of the responses at multiple defense depths and resolve conflicts if necessary. A number of research papers have started down this path. The APOD initiative [Atighetchi et al. 2003, 2004], for example, uses higher level strategies (e.g., attack containment, continuous unpredictable changes, etc.) to derive/direct lower-level substrategies and local tactics in responding to attacks. Similarly, the AVPS approach [Sibai and Menasce 2011, 2012] generates signatures (low-level rules) based on high-level rules; Tang and Yu [2008] showed that high-level goal management can optimize the lower level policy execution at the network security level. This is an encouraging trend, although the survey shows the multistrategy-based papers are still a minority (at 33%).

From Security-at-Any-Cost to Cost-Sensitive Protection. Though earlier attempts exist in quantifying the cost of intrusion detection and prevention (such as Lee et al. [2002]), an increasing number of recent research papers start to consciously balance the cost (i.e., penalization of other quality attributes) and benefits of autonomic responses to security attacks. Stakhanova et al. [2007b] and Strasburg et al. [2009], for example, defined a set of cost metrics and performed quantitative tradeoff analyses between cost of response and cost of damage, and between the importance of early pre-emptive responses (when there is a high cost of missed or late detections)

vs. the importance of detection accuracy (when there is a high cost of false positives). Similarly, Nagarajan et al. [2011] developed cost models involving operational costs, damage costs, and response costs, and implemented the model using Receiver Operating Characteristic (ROC) curves. At 23%, the cost-sensitive strategies are a minority but we believe they represent a promising and sensible direction for self-protection research, especially in the larger picture of self-* systems.

6.6. Spatial and Temporal Characteristics

Together, the three taxonomy dimensions Control Topology (T8), Response Timing (T9), and Enforcement Locale (T10) expose interesting characteristics and trends about the spatial and temporal aspects of self-protection approaches – that is, where the "brain" of the self-protection is within the system and where the "action" takes place, as well as when the adaptive actions are carried out.

First, as shown in Figure 8, survey results along the Control Topology dimension clearly shows that adaptive security approaches functioning at the global level are predominantly centralized – about 57% of the papers. For example, many research efforts (e.g., He et al. [2010a] and Abie et al. [2008]) recognize the need for coordination between local and global security policies. In most cases, the coordination is through a central controller (e.g., Huang et al. [2006]). One of the main reasons behind widespread adoption of centralized topology may be the fact that using a central controller makes coordination and global optimization easier. However, central controller runs the risk of becoming the single point of failure of the system, prone to denial of service and subversion attacks. Some approaches put more robust protection around the central controller, such as using hardened and trusted hardware/software (as in the case of the Malicious-and Accidental-Fault Tolerance for Internet Applications (MAFTIA) system [Verissimo et al. 2006]) or putting the controller in dedicated network zones [Chong et al. 2005]. Another potential disadvantage for the centralized approach is scalability. For pervasive systems with highly distributed computing resources, it may be inefficient and costly to have all of the resources communicate with a central controller. Accounting for only 8% of the total papers, globally decentralized approaches appear to be an exception rather than norm. As pointed out in Section 6.4, self-protection efforts from the MANET and self-organizing agent domains tend to fall into this category because the system topology does not allow for a centralized component. The decentralized control topology is not limited to these domain environments however. MAFTIA, for example, also uses local middleware controllers (called "wormholes") at each server that are interconnected yet do not appear to require a central controller. Decentralized security approaches hold more promise in their resilience and scalability. The fact that coordination and global optimization is harder in a decentralized setting indicates the need for more research attention. Indeed, decentralized control has been highlighted as a key research topic on the roadmap of self-adaptive systems [Lemos et al. 2013].

Secondly, survey results along the Response Timing dimension indicate reactive adaptation based on the "sense and respond" paradigm still seems to be the norm for self-protection (79% of total papers). That being said, the survey results also show an interesting trend that proactive security architectures are gaining ground in the past decade, with 21% of papers claiming some proactive tactics. By proactively "reviving" the system to its "known good" state, one can limit the damage of undetected attacks, though with a cost. The TALENT system [Okhravi et al. 2010, 2012], for example, addresses software security and survivability using a "cyber moving target" approach, which proactively migrates running applications across different platforms on a periodic basis while preserving application state. The Self-Cleansing Intrusion Tolerance (SCIT) architecture [Nagarajan et al. 2011] uses redundant and diverse

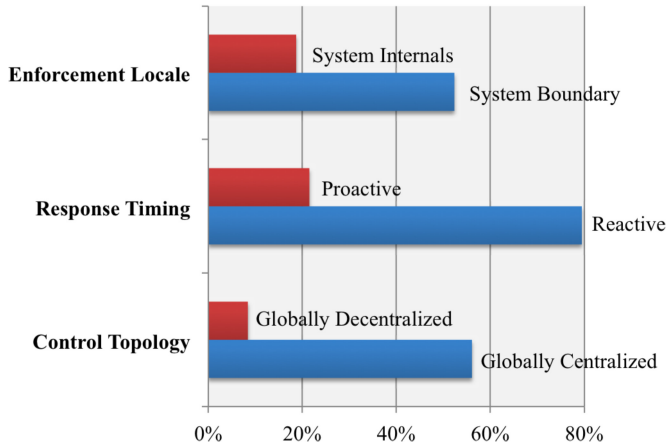


Fig. 8. Temporal and spatial characteristics.

servers to periodically "self-cleanse" the system to pristine state. The aforementioned R-Xen framework [Jansen et al. 2008] proactively instantiates new VM instances to ensure system reliability, a technique much faster than rebooting hardware servers thanks to hypervisor-based virtualization technology.

Third, the Enforcement Locale dimension shows that over 52% of self-protection approaches still rely on perimeter security, especially those that focus on intrusion detection and intrusion prevention. Systems relying solely on perimeter security, however, are often rendered helpless when the perimeter is breached; nor can they effectively deal with threats that originate from inside of the system. To compensate for this weakness, some approaches follow the "defense-in-depth" principle and establish multiple layers of perimeters or security zones [Pal et al. 2007], but the disadvantage still exists. In light of this, we feel there is a need to shift focus from perimeter security to overall system protection, especially from monitoring the system boundary to monitoring overall system behavior. For example, recent research has started to focus on detecting and responding to insider threats based on monitoring user-system interactions [Sibai and Menasce 2011, 2012]. Another possible approach is to shift the focus from delimiting system boundaries to identifying system assets under protection, as developed by Salehie et al. [2012] and Pasquale et al. [2012].

Figure 8 summarizes the statistics around the spatial and temporal traits of surveyed approaches, highlighting the research gaps around (1) global self-protection architectures that do not require a central controller, (2) combining reactive protection tactics with proactive ones, and (3) protecting the overall system and not just the perimeter.

6.7. Repeatable Patterns and Tactics for Self-Protection

One of the most revealing findings from our survey is the emergence of repeatable architectural patterns and design tactics that software systems employ specifically for self-protection purposes (T11 of the taxonomy). Even though some of these patterns bear similarity to the generic software architecture and design patterns, their usage and semantics are quite different. As mentioned in Section 5.2, they can be loosely categorized as structural and behavioral patterns. Their description, examples, and perceived pros/cons are summarized in Table I.

These patterns cover 84% of the surveyed papers; therefore, their use is quite pervasive. Note that the patterns are by no means mutually exclusive. A system may

Table I. Catalog of Self-Protection Patterns

Pattern Definition	Examples	Evaluation
	Structural Patterns	
Protective Wrapper—Place a security enforcement proxy, wrapper, or container around the protected resource, so that request to/response from the resource may be monitored and sanitized in a manner transparent to the resource.	The SITAR system [Wang et al. 2003] protects COTS servers by deploying an adaptive proxy server in the front, which detects and reacts to intrusions. Invalid requests trigger reconfiguration of the COTS server. Virtualization techniques are increasingly being used as an effective protective wrapper platform. VASP [Zhu et al. 2011], for example, is a hypervisor-based monitor that provides a trusted execution environment to monitor various malicious behaviors in the OS.	*Pros*—Security adaptation is transparent to the protected resource; easy to implement. *Cons*—Since the wrapper is inherently intended for outside threats, this pattern cannot address security vulnerabilities inside the protected component. The wrapper, esp. when externally visible, may itself become an exploitation target.
Agreement-based Redundancy—In addition to reliability and availability benefits provided by the common redundancy mechanism, this pattern uses agreement-based protocols among the replicas to detect anomalies and ensure correct-ness of results.	The seminal work by Castro and Liskov [2002] described a Byzantine Fault Tolerance (BFT) algorithm that can effectively tolerate f faulty nodes with $3f$ replicas. Similar agreement-based voting protocols have been used in many other systems such as SITAR and [Valdes et al. 2004]. The Ripley system [Vikram et al. 2009] implements a special kind of agreement based technique by executing a "known good" replica of a client-side program on the server side.	*Pros*—Robust mechanism that can meet both system integrity and availability goals; effective against unknown attacks. *Cons*—Due to required number of replicas, needs significant hardware and software investments, which can be costly. Further, by compromising enough replicas, the system will be essentially shut down, resulting in denial of service.
Implementation Diversity—Deploy different implementations for the same software specification, in the hope that attacks to one impl. may not affect others. This may be achieved through the use of diverse prog. languages, OS, or H/W platforms. To safely switch requests from one instance to another, checkpointing is necessary to save the current system state.	The HACQIT system [Reynolds et al. 2002, 2003] achieves diversity by using two software components with identical functional specifications (such as a Microsoft IIS web server and an Apache web server) for error detection and failure recovery. Similarly, the DPASA system [Chong et al. 2005; Pal et al. 2007] included controlled use of hardware and OS level diversity among redundant environments as part of a comprehensive survivable architecture. A similar approach is dubbed Architecture Hybridization in the MAFTIA system [Verissimo et al. 2006].	*Pros*—Effective defense against attacks based on platform-specific vulnerabilities. Increases system resilience since an exploited weakness in one impl. is less likely to kill the entire system. *Cons*—Significant efforts required to develop, test, and deploy diverse program implementations. Diverse languages/platforms may give rise to more software defects. Further, checkpoint-ing to preserve program state at runtime may prove technically challenging.
Countermeasure Broker—The self-protecting system includes a brokering function that, based on the type of an attack, performs dynamic matching or tradeoff analysis to select the most appropriate response or countermea-sure, often from a predefined repository.	Case-based Reasoning (CBR) techniques are sometimes used to detect intrusions and select responses. The SoSMART system [Musman and Flesher 2000] describes an agent-based approach where the CBR "brain" picks the suitable response agent. Alternatively, the ADEPTS effort [Foo et al. 2005a; Wu et al. 2007] uses attack graphs to identify possible attack targets and consequently suitable responses.	*Pros*—Flexibility and dynamic nature of the response, when implemented correctly, makes it harder for an adversary to predict and exploit the security defense. *Cons*—Not effective against unknown attacks. Static, "knee-jerk" like responses are likely to be predictable, thus lose

Table I. Catalog of Self-Protection Patterns

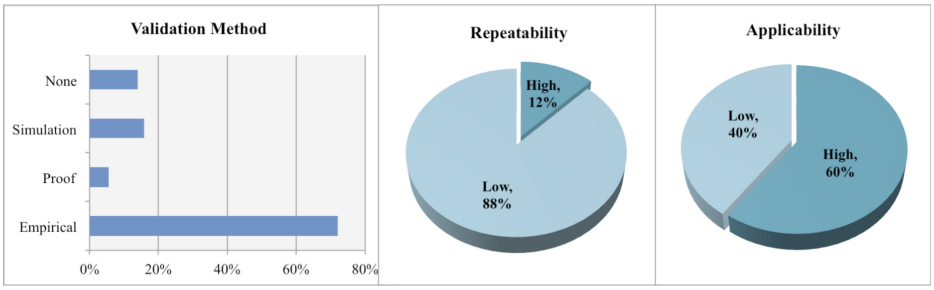
Pattern Definition	Examples	Evaluation
		effectiveness in the long run. The broker component may become a high value target for adversaries.
Aspect-Orientation—Following the Aspect Oriented Programming (AOP) principle, this pattern deploys self-protection mechanisms as a separate aspect in the system, transparent to application logic. This pattern is often assisted by Model Driven Engineering (MDE) techniques, as mentioned in Section 6.2.	Morin et al. [2010], for example, showed how a security access control metamodel is defined and combined with the business architecture metamodel; the security aspect is "woven" into the overall model using MDE tools such as the Kermata language. A related effort [Mouelhi et al. 2008] shows how the security policies from the model are generated in XACML and integrated into the application using AOP. Xiao et al. [2007] and Xiao [2008] also used a similar model driven approach to model security policy rules and dynamically weave them into the runtime application in an agent-based environment.	*Pros*—Bringing AOP benefits to self-protection, such as reuse, separation of concerns, and improved application quality. Assisted with MDE techniques, it also provides a way of expressing security policies as models. *Cons*—It is not yet known if self-protection as a cross-cutting concern can be adequately expressed in today's modeling languages and tools. Weaving in the security aspect into the very application it is protecting makes security logic just as vulnerable.
Behavioral Patterns		
Protective Recomposition—Dynamically adapt security behavior of a system through altering how security-enforcing components are connected and orchestrated. This may include tuning of security parameters, changing authentication/authorization methods, switching to a different crypto algorithm, or regeneration of access control policies.	The E2R Autonomic Security Framework [He and Lacoste 2008a; Saxena et al. 2007], for example, allows nodes in a wireless network to collect and derive security context information from neighboring nodes and reorganize upon node failures. Other examples include changes in contract negotiations between security components [Feiertag et al. 2000], altered security service sequences based on QoS objective changes [Malek et al. 2009], or regenerating new concrete policy instances from a generic policy based on dynamic security context [Debar et al. 2007].	*Pros*—A key pattern that touches the very essence of self-adaptive security: the ability to adapt security posture based on changing threats and real time security contexts. Makes the system more defendable and harder to exploit. *Cons*—Dynamic and sometimes even non-deterministic security behavior is difficult to test and even harder to evaluate its correctness and effectiveness.
Attack Containment—A simple pattern that seeks to isolate a compromised component from the rest of the system to minimize the damage. Typical techniques include blocking access, denying request, deactivating user logins, and shutting down the system component.	De Palma et al. [2012] developed an approach for clustered distributed systems that involves isolating a compromised machine from the network. Solitude uses file-system level isolation and application sandboxing to limit attack propagation [Jain et al. 2008]. SASI [Erlingsson and Schneider 2000], a code-level containment approach, uses a compile-time Software Fault Isolation (SFI) method to enforce security policies.	*Pros*—Simple, fast, and effective way to mitigate and contain security compromises. *Cons*—Often carried out at the opportunity cost of (at least temporary) unavailability of system resources to legitimate users.
Software Rejuvenation—As defined by Huang et al. [1995], this pattern involves gracefully terminating an application and immediately restarting it at a clean internal state. Often	In addition to the rejuvenation-based SCIT system [Huang et al. 2006; Nagarajan et al. 2011] and the aforementioned HACQIT system, the Proactive Resilience Wormhole (PRW) effort [Sousa et al. 2006, 2007, 2010] also employs proactive rejuvenation for intrusion tolerance and high	*Pros*—Effective technique that addresses both security and high availability. Periodic software rejuvenation limits the damage of undetected attacks. *Cons*—The rejuvenation

Table I. Catalog of Self-Protection Patterns

Pattern Definition	Examples	Evaluation
done proactively and periodically.	availability. Wang et al. [2009] presented a special case of rejuvenation involving software hotswapping, i.e., swapping out infected components at runtime, replaced with a valid/more strongly protected equivalent.	process, short as it may be, temporarily reduces system reliability. Extra care is needed to preserve application state and transition applications to a rejuvenated replica. Extra investment is needed for maintaining redundant replicas.
Reconfiguration on Reflex—A bio-inspired pattern that reconfigures the system to a higher level of protection (which may be more resource consuming), and when attack passes, returns to a less restrictive mode.	The Security Adaptation Manager (SAM) [Hinton et al. 1999] dynamically operates the system at 3 levels of implementations (calm, nervous, panic) depending on the threat level. The DASAC approach [Jean et al. 2007] uses a boosting-based algorithm and heuristically defined security levels that allow the agent network to react to agent trustworthiness.	*Pros*—A technique for balancing competing goals of security and performance, depending on the ongoing threat level. *Cons*—Ineffective in the case of undetected threats. An attacker can trick the system to always run at heightened security levels to sacrifice performance, therefore not suitable for persistent threats.
Artificial Immunization—Inspired by adaptive immune systems in vertebrates, this pattern seeks to capture samples of worms or viruses; analyze the virus to derive a signature that can be used to detect and remove it from infected resources; and disseminate the "antidote" to all vulnerable systems.	Kephart et al. [1997] and White et al. [1999] designed one of the first Digital Immune Systems in response to early virus epidemics, when it was realized that automation was needed to spread the cure faster than the virus itself. Later approaches such as SweetBait [Portokalidis and Bos 2007] use more sophisticated "honeypot" techniques to capture suspicious traffic and generate worm signatures. A variant of this pattern uses the so-called Danger Theory to as the basis for autonomic attack detection and defense [Rawat and Saxena 2009; Swimmer 2007].	*Pros*— "Detect once, defend anywhere"; the pattern proved to be a successful approach for defending against computer viruses and helped creation of the anti-virus industry. *Cons*—Centralized and top-down architecture is a challenge to scalability and agility, esp. as attacks become more localized and targeted. Further, just like all signature-based techniques, it is not effective against unknown/zero-day attacks.

combine a number of complementary patterns to add to its depths of defense and boost survivability. The TALENT system, for instance, employs OS-level and programming language-level diversity and periodically moves running applications to a clean platform. Though technical challenges remain, this "cyber moving target" approach holds promise for defense against advanced, zero-day attacks. As another example, the R-Xen framework [Jansen et al. 2008] used hypervisor-based Protective Wrapper to provide application monitoring and protection. Fearing the protective wrapper itself may become a weak link for the system, it also used software rejuvenation to protect the hypervisor core.

We also found, not surprisingly, that the positioning and techniques employed by a self-protection approach will to some extent determine the architectural patterns being used. This observation, however, does point to a critical research opportunity, that is, to further identify and catalogue such correlations, to codify them into machine-readable forms, so that a system may dynamically re-architect itself using repeatable patterns



(a) (b) (c)

Fig. 9. (a) Validation method; (b) Repeatability; (c) Applicability.

as requirements and environments change. This is a higher level of self-protection and may only be enabled through an architecture-based approach.

6.8. Quality Assessment of Surveyed Papers

We used reputable sites in our review protocol (see Appendix A). This resulted in the discovery of high-quality refereed research papers from respectable venues. We use Validation Method (T12), Repeatability (T13), and Applicability (T14), which are the three Approach Quality dimensions in the taxonomy, to develop better insights into the quality of the research papers surveyed. Figure 9 summarizes some of our findings. Figure 9(a) depicts the share of different Validation Methods in assessing the quality of self-protection approaches. Most (70%) of the approaches have used Empirical techniques to assess the validity of their ideas. The Empirical techniques range from detailed assessment of full implementation of the approach (e.g., Castro and Liskov [2002]) to a proof-of-concept experimentation of a prototype (e.g., Raissi [2006]). A limited number (6%) of approaches (e.g., Ostrovsky and Yung [1991]) have provided mathematical Proof to validate their ideas. Some approaches (16%) have relied on Simulation to validate their claims (e.g., Taddeo and Ferrante [2009]). Finally, there are approaches (14%) that have either not validated their ideas (e.g., Valdes et al. [2004]) or validated aspects (such as performance) of their work other than security (e.g., Swimmer [2007]).

The evaluation of security research is generally known to be difficult. Making the results of experiments repeatable is even more difficult. We can see this in Figure 9(b), where only limited portions (12%) of approaches are highly repeatable. The rest have not made their implementations, prototypes, tools, and experiments available to other researchers. This has hampered their adoption in other relevant domains. As we can see in Figure 9(c), many approaches (60%) are portable and have high potential for being applicable to broad range of situations and domains. However, the fact that their artifacts are not accessible outside the boundary of the same team/organization, has limited their usage and prevented their potential applicability from becoming actual applicability.

7. THREATS TO VALIDITY

As detailed in Appendix A, by carefully following the SLR process in conducting this study, we have tried to minimize the threats to the validity of our results and conclusions made in this article. Nevertheless, there are three possible threats that deserve additional discussion.

One important threat is the completeness of this study, that is, whether all of the appropriate papers in the literature were identified and included. This threat could be due to two reasons: (1) some relevant papers were not picked up by the search engines

or did not match our keyword search, (2) some relevant papers that were mistakenly omitted, and vice-versa, some irrelevant papers that were mistakenly included. To address these threats, we adopted multiple strategies. First, we used multiple search engines of different types, including those targeted specifically at scientific publications as well as general-purpose search engines. Second, we adopted an iterative approach to keyword-list construction. As the study progressed, we noticed different research communities refer to the same concepts using different words. The iterative process allowed us to ensure proper list of keywords were used in our search process. Nonetheless, the limited number of keyword phrases we used (see Section A.2 in Appendix A) in order to keep the study manageable has prevented the coverage of some interesting areas, such as:

— research pertaining to *security assurance* and security Service Level Agreements (SLA) during the autonomic adaptation of services/components, such as the ongoing work at EU projects ASSERT4SOA and Aniketos (see Anisetti et al. [2012], Foster et al. [2012] and other related publications);
— risk-based self-protecting solutions, such as an approach by Cheng et al. [2007];
— adaptive solutions addressing privacy as introduced by Schaub et al. [2012].

It is our hope that these areas will be covered in our future research.

Another threat is the objectiveness of the study, that is, whether the included papers were classified without any bias. To avoid misclassification of papers, the authors crosschecked papers reviewed by one another, such that no paper received only a single reviewer. This allowed us to reduce the effect of bias in our review process.

Yet another threat is the validity of the proposed taxonomy, that is, whether the taxonomy is sufficiently rich to enable proper classification and analysis of the literature in this area. To mitigate this threat, we adopted an iterative content analysis method, whereby the taxonomy was continuously evolved to account for every new concept encountered in the papers. While the changes to the taxonomy were drastic toward the beginning of our study (roughly the first 30 papers that were classified), toward the end (roughly the last 30 papers) resulted in almost no changes to the taxonomy. This gives us confidence that the taxonomy provides a good coverage for the variations and concepts that are encountered in this area of research.

Finally, the last threat is that synthesis of the results and conclusions made in Section 6 could be biased or flawed. As mentioned earlier, we have tackled the individual reviewer's bias by crosschecking the papers, such that no paper received a single reviewer. We have also strived to base the conclusions on the collective numbers obtained from the classification of papers, rather than individual reviewer's interpretation or general observations, thus minimizing the individual reviewer's bias. Lastly, we have made the classification of the papers available to the public, which can be accessed at http://goo.gl/Ksy1u.

8. RECOMMENDATIONS FOR FUTURE RESEARCH

From systematically reviewing the self-protection-related literature, we see some important trends in software security research. Starting in the 1990s, dynamic and automated security mechanisms started to emerge in the antivirus and anti-spam communities. The late 1990s and early 2000s saw a research boom in the Intrusion Detection (ID) / firewall communities, as online systems faced the onslaught of network and host based attacks. We then see two important trends in the past decade, as reflected by the observations in Section 6:

(a) from Intrusion Detection to Intrusion Response (IR), as the increasing scale and speed of attacks showed the acute need for dynamic and autonomic response

mechanisms, as confirmed in recent surveys [Shameli-Sendi et al. 2012; Stakhanova et al. 2007a];

(b) from Intrusion Detection to Intrusion Tolerance (IT), when both the industry and academia came to the realization that security threats will persist and prevention mechanisms will likely never be adequate, and began to give additional consideration to maintaining system performance even in the presence of an intrusion [Nguyen and Sood 2011].

Only in recent few years did we see an explicit focus on self-protection as a system property in the autonomic computing context. Frincke et al. [2007] and Atighetchi and Pal [2009], for example, see the pressing need for autonomic security approaches and true self-protecting systems. From the break-down of papers across ID, IR, IT, and Self-Protection (SP) communities, we see an encouraging sign of growing SP research, yet we also see the continued influence of the intrusion-centric mindset.

Therefore, our first and foremost recommendation is to increase attention, convergence, and collaboration on self-protection research, and to leverage this community for integrating a diverse set of strategies, technologies, and techniques from ID, IR, IT and other communities toward achieving a common goal. More specifically, the survey using our proposed taxonomy has revealed some gaps and needs for future research. To summarize, self-protection research needs to focus on the following to stay ahead of today's advancing cyberthreats.

— Advance the sophistication at each self-protection level, that is, from automatically monitoring and detecting threats and vulnerabilities to autonomously predict and prevent attacks.
— Move beyond defending the network and host layers, towards developing approaches that are independent of specific system architectures and that can select suitable strategies and tactics at different architecture layers.
— Pursue integrated, "full lifecycle" approaches that span both development-time and runtime.
— "Protect the protector", that is, safeguard the meta-level self-protection module, especially in a globally decentralized topology.
— Explore models@runtime and model-driven engineering techniques while maintaining clear meta-level and base-level separation.
— Explore the definition and application of qualitative and quantitative metrics that can be used to dynamically assess overall system security posture and make autonomic response decisions.
— Continue to explore multilevel, multiobjective, as well as cost-sensitive security decision-making strategies based on stakeholder requirements.
— Continue the paradigm shift from perimeter security to overall system protection and monitoring, and from merely reactive responses to a combined use of both reactive and proactive mechanisms.
— Catalog, implement, and evaluate self-protection patterns at the abstract architecture level.
— Promote collaboration in the community by making research repeatable (such as providing tools and case studies) and developing common evaluation platforms/benchmarks.

9. CONCLUSION

Self-protection of software systems is becoming increasingly important as these systems face increasing external threats from the outside and adopt more dynamic architecture behavior from within. Self-protection, like other self-* properties, allows

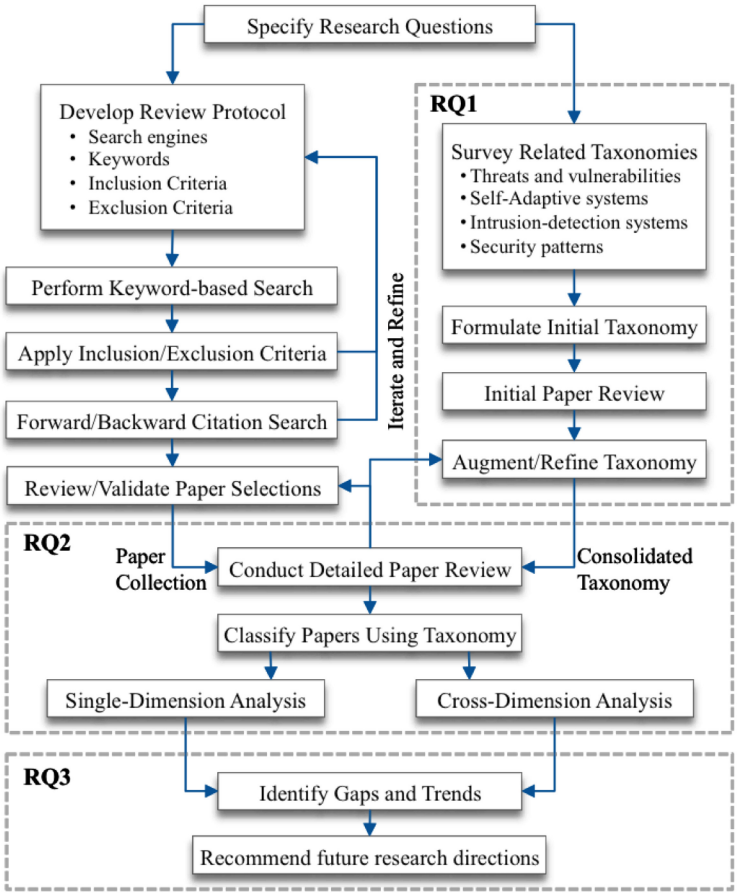


Fig. A.1. Research process flow and tasks.

the system to adapt to the changing environment through autonomic means without much human intervention, and can thereby be responsive, agile, and cost effective. Existing research has made significant progress towards software self-protection, such as in intrusion tolerance systems and adaptive security mechanisms at the application level. This article proposes a comprehensive taxonomy to classify and characterize research efforts in this arena. We have carefully followed the systematic literature review process, resulting in the most comprehensive and elaborate investigation of the literature in this area of research, comprised of 107 papers published from 1991 to 2013. The research has revealed patterns, trends, and gaps in the existing literature and underlined key challenges and opportunities that will shape the focus of future research efforts. In particular, the survey shows self-protection research should advance from focusing primarily on the network and host layers to layer-independent and architecture-based approaches; from single-mechanism and single-objective to multi-strategy and cost-sensitive decision-making, and from perimeter security to overall system protection. We believe the results of our review will help to advance the much needed research in this area and hope the taxonomy itself will become useful in the development and assessment of new research.

APPENDIX A

A.1 Research Tasks

To answer the three research questions introduced in Section 4, we organized our tasks into a process flow tailored to our specific objectives, yet still adhering to the three-phase SLR process. The overall process flow is outlined in Figure A.1 and briefly described here.

First, in the planning phase, we defined the review protocol that includes selection of the search engines, the initial selection of the keywords pertaining to self-protecting software systems, and the inclusion/exclusion criteria for the candidate papers. The protocol is described in detail in Section A.2.

The initial keyword-based selection of the papers is an iterative process that involves exporting the candidate papers to a "research catalog" and applying the pre-defined inclusion/exclusion criteria on them. In the process, the keyword search expressions and the inclusion/exclusion criteria themselves may also need to be fine-tuned, which would in turn trigger new searches. Once the review protocol and the resulting paper collection were stabilized, our research team also conducted peer-reviews to validate the selections.

For RQ1, in order to define a comprehensive taxonomy suitable for classifying self-protection research, we first started with a quick "survey of surveys" on related taxonomies. Since our research topic straddles both the autonomic/adaptive systems and computer security domains, we identified some classification schemes and taxonomies from both domains, as described in Section 3. After an initial taxonomy was formulated, we then used the initial paper review process (focusing on abstracts, introduction, contribution, and conclusions sections) to identify new concepts and approaches to augment and refine our taxonomy. The resulting taxonomy is presented in Section 5.

For the second research question (RQ2), we used the validated paper collection and the consolidated taxonomy to conduct a more detailed review on the papers. Each paper was classified using every dimension in the taxonomy, and the results were again captured in the research catalog. The catalog, consisting of a set of spreadsheets, allowed us to perform qualitative and quantitative analysis not only in a single dimension, but also across different dimensions in the taxonomy. The analysis and findings are documented in Section 6.

To answer the third research question (RQ3), we analyzed the results from RQ2 and attempted to identify the gaps and trends, again using the taxonomy as a critical aid. The possible research directions are henceforth identified and presented in Section 8.

A.2 Literature Review Protocol

The first part of our review protocol was concerned with the selection of search engines. As correctly pointed out in Brereton et al. [2007], no single search engine in the software engineering domain is sufficient to find all of the primary studies, therefore multiple search engines are needed. We selected the following search sites to have a broad coverage: IEEE Explore, ACM Digital Library, Springer Digital Library, Elsevier ScienceDirect (Computer Science collection), Google Scholar.

For these search engines we used a targeted set of keywords, including: *Software Self-Protection*, *Self-Protecting Software*, *Self-Securing Software*, *Adaptive Security*, and *Autonomous Security*. It is worth noting that the exact search expression had to be fine-tuned for each search engine due to its unique search interface (e.g. basic search vs. advanced search screens, the use of double quotes, and the AND/OR expressions). In each case, we tried to broaden the search as much as possible while maintaining a manageable result set. For example, because Google Scholar invariably returns

thousands of hits, we limited our search to the first 200 results. We also used the following inclusion and exclusion criteria to further filter the candidate papers.

— Only refereed journal and conference publications were included.
— Based on our definition of self-protection in Section 2, we included autonomic and adaptive software research that is directly relevant to the self-protecting and self-securing properties. Other properties such as self-healing and self-optimization are out of the scope of this survey.
— Our review focuses on software systems only, therefore does not include self-protection pertaining to hardware systems.
— Software security research that doesn't exhibit any self-adaptive/autonomic traits is excluded.
— Our definition of self-protection pertains to protecting the software system against malicious threats and attacks. Sometimes other connotations of self-protection may be possible. For example, protecting a system from entering into an inconsistent state (from a version consistency perspective), or protecting a wireless sensor network (from the field-of-view perspective) may also be viewed as self-protection. Such papers are excluded in this review.
— Position papers or research proposals not yet implemented or evaluated are excluded.

When reviewing a candidate paper, we have in many occasions further extended the collection with additional papers that appear in its citations or those that are citing it (backward and forward citation search).

APPENDIX B

The following evaluation matrix contains the detailed survey results.

Source	Self-Prot. Levels			Depths of Defense				Life cycle Focus		Security Goals			Theoretical Foundation							Meta-Level Sep.			Meta Level DM			Control Topology			Response Timing			Enforce. Locale			Self-Protection Patterns									Validation Method				App.		Rep.	
	Monitor & Detect	Respond & Protect	Plan & Prevent	Network	Host	Middleware	Application	Depth-Ind.	Dev Time	Runtime	Confidentiality	Integrity	Availability	Formal Models	Heuristic Rules	Optimization	Prob. Models	Machine Learning	Single Strategy	Partial Sep.	Complete Sep.	No Separation	Multi-Strategy	Cost-sensitive	Local Only	Globally Centr.	Globally Decentr.	Reactive	Proactive	Timing-Neutral	System Boundary	System Internals	Locale-neutral	Protective Wrapper	Agmt-based Redund.	Impl. Diversity	S/W Rejuvenation	Protective Recomp.	Attack Containment	Reconfig on Reflex	Ctr-measure Broker	Aspect-Orientation	Artificial Immun.	Empirical	Proof	Simulation	None	Low	High	Low	High
[Abie et al. 2008]	×	×	×	×	×	×			×	×	×	×	×		×				×		×		×				×	×	×			×																			×	×
[Abie 2009]	×	×	×	×	×	×			×	×	×	×	×		×				×		×		×				×	×	×			×																			×	×
[Savola and Heinonen 2010]	×	×	×	×	×	×			×	×	×	×	×		×				×		×		×				×	×	×			×																			×	×
[Adnane et al. 2008]	×	×		×	×				×	×	×	×	×	×	×				×							×						×							×	×										×			
[Alampalayam and Kumar 2003]	×	×	×	×	×	×			×	×	×	×	×	×	×		×		×				×			×						×							×	×										×			
[Alia and Lacoste 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Alia et al. 2010]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Al-Nashif et al. 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Atighetchi et al. 2003]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Atighetchi et al. 2004]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Balepin et al. 2003]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Ben Mahmoud et al. 2010]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Benjamin et al. 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Blount et al. 2011]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Burns et al. 2001]	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×		×		×		×			×			×			×							×	×										×			
[Casola et al. 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Castro and Liskov 2002]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Chigan et al. 2005]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Chong et al. 2005]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Pal et al. 2007]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Costa et al. 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Cox et al. 2007]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Crosbie and Spafford 1995]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[de Oliveira et al. 2011]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[De Palma et al. 2012]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Debar et al. 2007]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Dragoni et al. 2009]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[English et al. 2006]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Erlingsson and Schneider 2000]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Fayssal et al. 2008]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Feiertag et al. 2000]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Foo et al. 2005]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Wu et al. 2007]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Garlan et al. 2004]	×	×	×	×	×	×	×	×	×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Cheng et al. 2006]	×	×	×	×	×	×	×	×	×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Gelenbe and Loucas 2007]	×	×	×	×	×	×	×		×	×	×	×	×		×		×		×		×		×			×			×			×							×	×										×			
[Ghosh et al. 1998]	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×		×		×		×		×	×	×			×							×	×										×			
[Ghosh and Voas 1999]	×	×	×	×	×	×	×	×	×	×	×	×	×		×		×		×		×		×		×		×	×	×			×							×	×										×			

Source	Self-Prot. Levels			Depths of Defense					Life cycle Focus		Security Goals			Meta-Level Sep.		
	Monitor & Detect	Respond & Protect	Plan & Prevent	Network	Host	Middleware	Application	Depth-Ind.	Dev Time	Runtime	Confidentiality	Integrity	Availability	No Separation	Partial Sep.	Complete Sep.
[Goel et al. 2005]	x	x			x					x		x				x
[Jain et al. 2008]	x	x			x					x		x				x
[Guttman and Herzog 2005]	x	x	x	x						x	x	x			x	
[Hashii et al. 2000]	x						x		x	x	x	x			x	
[He et al. 2010a]	x	x	x	x	x					x	x	x	x			x
[He et al. 2010b]	x	x	x	x	x					x	x	x	x			x
[Hinton et al. 1999]	x	x					x			x		x			x	
[Huang et al. 2006]	x	x			x					x			x			x
[Nagarajan et al. 2011]	x	x			x					x			x			x
[Jabbour and Menasce 2008]	x	x					x			x	x			x		
[Jabbour and Menasce 2009]	x	x					x			x	x			x		
[Jansen et al. 2008]	x	x			x					x	x	x	x			x
[Jean et al. 2007]	x					x				x		x		x		
[Kephart et al. 1997]	x	x	x		x		x		x	x	x	x				x
[White et al. 1999]	x	x	x		x		x		x	x	x	x				x
[Labraoui et al. 2010]	x			x						x		x		x		
[Lamprecht and van Moorsel 2007]	x	x		x						x	x	x			x	
[Lamprecht and van Moorsel 2008]	x	x		x						x	x	x			x	
[Laureano et al. 2007]	x	x			x					x			x			x
[Lee et al. 2002]	x			x	x					x	x	x	x		x	
[Liang and Sekar 2005]	x						x		x	x	x	x			x	
[Lim et al. 2009]	x	x					x			x	x	x			x	
[Lorenzoli et al. 2007]	x	x					x		x	x		x		x		
[Malek et al. 2009]	x	x	x					x	x	x	x	x	x			x
[Menasce et al. 2011]	x	x	x					x	x	x	x	x	x			x
[Maximilien and Singh 2004]	x	x					x			x		x	x		x	
[Montangero and Semini 2004]	x	x		x						x	x	x				x
[Mouelhi et al. 2008]		x	x					x	x	x	x	x		x		
[Morin et al. 2010]		x	x					x	x	x	x	x		x		
[Musman and Flesher 2000]	x	x		x	x					x	x	x			x	
[Okhravi et al. 2010]	x	x		x	x		x		x	x			x			x
[Okhravi et al. 2012]	x	x		x	x		x		x	x			x			x
[Ostrovsky and Yung 1991]		x				x				x		x				x
[Porras and Neumann 1997]	x	x		x						x	x	x				x
[Neumann and Porras 1999]	x	x		x						x	x	x				x
[Portokalidis and Bos 2006]	x	x		x						x		x				x
[Raissi 2006]	x	x		x						x	x					x
[Rawat and Saxena 2009]	x	x		x						x			x	x		

Source	Theoretical Foundation					Meta Level DM			Control Topology			Response Timing			Enforce. Locale		
	Formal Models	Heuristic Rules	Optimization	Prob. Models	Machine Learning	Single Strategy	Multi-Strategy	Cost-sensitive	Local Only	Globally Centr.	Globally Decentr.	Reactive	Proactive	Timing-Neutral	System Boundary	System Internals	Locale-neutral
[Goel et al. 2005]		x				x			x			x				x	
[Jain et al. 2008]		x				x			x			x				x	
[Guttman and Herzog 2005]	x	x				x				x		x					x
[Hashii et al. 2000]		x				x			x					x			x
[He et al. 2010a]		x					x			x		x					x
[He et al. 2010b]		x					x			x		x					x
[Hinton et al. 1999]		x				x		x	x			x				x	
[Huang et al. 2006]		x		x		x		x		x			x		x		
[Nagarajan et al. 2011]		x		x		x		x		x			x		x		
[Jabbour and Menasce 2008]		x				x			x			x				x	
[Jabbour and Menasce 2009]		x				x			x			x				x	
[Jansen et al. 2008]		x				x			x			x			x		
[Jean et al. 2007]		x			x	x					x	x			x		
[Kephart et al. 1997]		x				x				x		x	x		x		
[White et al. 1999]		x				x				x		x	x		x		
[Labraoui et al. 2010]		x				x					x	x					x
[Lamprecht and van Moorsel 2007]		x				x		x	x			x			x		
[Lamprecht and van Moorsel 2008]		x				x		x	x			x			x		
[Laureano et al. 2007]		x			x	x			x			x			x		
[Lee et al. 2002]		x			x		x	x	x			x			x		
[Liang and Sekar 2005]		x				x			x					x		x	
[Lim et al. 2009]		x	x		x		x		x			x					x
[Lorenzoli et al. 2007]		x				x			x					x		x	
[Malek et al. 2009]		x	x				x			x				x			x
[Menasce et al. 2011]		x	x				x			x				x			x
[Maximilien and Singh 2004]		x				x				x		x					x
[Montangero and Semini 2004]	x	x				x				x		x			x		
[Mouelhi et al. 2008]		x				x			x					x		x	
[Morin et al. 2010]		x				x			x					x		x	
[Musman and Flesher 2000]		x				x				x		x			x		
[Okhravi et al. 2010]		x				x			x				x				x
[Okhravi et al. 2012]		x				x			x				x				x
[Ostrovsky and Yung 1991]	x	x				x			x			x	x			x	
[Porras and Neumann 1997]		x					x			x		x					x
[Neumann and Porras 1999]		x					x			x		x					x
[Portokalidis and Bos 2006]		x					x			x		x			x		
[Raissi 2006]		x	x			x		x	x			x			x		
[Rawat and Saxena 2009]		x				x				x		x			x		

Source	Self-Protection Patterns										Validation Method				App.		Rep.	
	Protective Wrapper	Agmt-based Redund.	Impl. Diversity	S/W Rejuvenation	Protective Recomp.	Attack Containment	Reconfig on Reflex	Ctr-measure Broker	Aspect-Orientation	Artificial Immun.	Empirical	Proof	Simulation	None	High	Low	High	Low
[Goel et al. 2005]	x					x					x					x		x
[Jain et al. 2008]	x					x					x					x		x
[Guttman and Herzog 2005]											x	x				x		x
[Hashii et al. 2000]	x				x						x					x		x
[He et al. 2010a]					x						x					x		x
[He et al. 2010b]					x						x					x		x
[Hinton et al. 1999]							x							x		x		x
[Huang et al. 2006]				x							x	x	x		x			x
[Nagarajan et al. 2011]				x							x	x	x		x			x
[Jabbour and Menasce 2008]														x	x			x
[Jabbour and Menasce 2009]														x	x			x
[Jansen et al. 2008]	x			x							x				x			x
[Jean et al. 2007]							x				x				x			x
[Kephart et al. 1997]	x									x	x					x		x
[White et al. 1999]	x									x	x					x		x
[Labraoui et al. 2010]		x											x			x		x
[Lamprecht and van Moorsel 2007]					x						x					x		x
[Lamprecht and van Moorsel 2008]					x						x					x		x
[Laureano et al. 2007]	x					x					x				x			x
[Lee et al. 2002]											x				x		x	
[Liang and Sekar 2005]											x					x		x
[Lim et al. 2009]													x			x	x	
[Lorenzoli et al. 2007]	x									x	x				x			x
[Malek et al. 2009]					x						x				x			x
[Menasce et al. 2011]					x						x				x			x
[Maximilien and Singh 2004]	x				x								x		x			x
[Montangero and Semini 2004]					x							x			x		x	
[Mouelhi et al. 2008]									x		x				x			x
[Morin et al. 2010]									x		x				x			x
[Musman and Flesher 2000]						x		x			x				x			x
[Okhravi et al. 2010]			x	x							x				x			x
[Okhravi et al. 2012]			x	x							x				x			x
[Ostrovsky and Yung 1991]				x	x							x				x	x	
[Porras and Neumann 1997]								x						x		x		x
[Neumann and Porras 1999]								x						x		x		x
[Portokalidis and Bos 2006]						x					x					x		x
[Raissi 2006]					x						x					x		x
[Rawat and Saxena 2009]										x			x			x		x

Source	Self-Prot. Levels			Depths of Defense				Life cycle Focus		Security Goals			Meta-Level Sep.			Theoretical Foundation					Meta Level DM			Control Topology			Response Timing			Enforce. Locale			Self-Protection Patterns									Validation Method				App.		Rep.	
	Plan & Prevent	Respond & Protect	Monitor & Detect	Network	Host	Middleware	Application	Depth-Ind.	Dev Time	Runtime	Confidentiality	Integrity	Availability	No Separation	Partial Sep.	Complete Sep.	Formal Models	Heuristic Rules	Optimization	Prob. Models	Machine Learning	Single Strategy	Multi-Strategy	Cost-sensitive	Local Only	Globally Centr.	Globally Decentr.	Reactive	Proactive	Timing-Neutral	System Boundary	System Internals	Locale-neutral	Protective Wrapper	Agmt-based Redund.	Impl. Diversity	S/W Rejuvenation	Protective Recomp.	Attack Containment	Reconfig on Reflex	Ctr-measure Broker	Aspect-Orientation	Artificial Immun.	Empirical	Proof	Simulation	None	High	Low	High	Low
[Reiser and Kapitza 2007a]			×		×					×			×			×		×				×			×	×		×	×		×			×	×									×				×		×	
[Reiser and Kapitza 2007b]		×	×		×					×			×			×		×				×			×	×		×	×		×			×	×									×				×		×	
[Reynolds et al. 2002]			×	×						×	×	×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Reynolds et al. 2003]			×	×	×					×		×	×			×		×				×			×	×		×	×		×			×	×									×				×		×	
[Saxena et al. 2007]			×				×			×	×	×				×	×	×				×	×		×	×		×	×		×			×	×									×				×		×	
[He and Lacoste 2008]			×				×			×	×	×				×	×	×				×	×		×	×		×	×		×			×	×									×				×		×	
[Sibai and Menasce 2011]		×		×						×						×		×				×			×	×		×	×		×			×	×									×				×		×	
[Sibai and Menasce 2012]		×		×					×	×						×		×				×			×	×		×	×		×			×	×									×				×		×	
[Sousa et al. 2006]			×	×	×					×		×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Sousa et al. 2007]		×	×	×	×					×	×	×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Sousa et al. 2010]		×	×	×	×				×	×	×	×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Stakhanova 2007]		×	×		×					×	×	×			×			×	×			×	×		×	×		×	×		×			×	×									×				×		×	
[Strasburg 2009]		×	×		×					×	×	×			×	×		×	×			×	×		×	×		×	×		×			×	×									×				×		×	
[Swimmer 2006]		×	×	×						×		×				×		×									×						×	×									×				×		×	
[Taddeo and Ferrante 2009]		×	×	×					×	×	×			×				×				×	×		×	×		×	×		×			×	×									×				×		×	
[Tan et al. 2008]			×					×		×	×	×					×	×										×	×		×			×	×									×				×		×	
[Tang and Yu 2008]		×	×	×						×	×	×				×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Uribe and Cheung 2004]	×								×		×						×							×																											
[Valdes et al. 2004]		×	×			×				×		×	×		×			×				×	×		×	×		×	×		×			×	×									×				×		×	
[Verissimo et al. 2006]		×	×		×	×				×		×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Vikram et al. 2009]			×				×			×						×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Wang et al. 2003]	×	×	×		×					×	×	×	×			×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Wang et al. 2009]		×	×				×			×		×	×		×	×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Xiao et al. 2007]		×	×			×	×		×	×	×	×				×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Xiao 2008]		×	×				×		×	×	×	×				×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Yau et al. 2006]		×	×			×	×		×	×	×	×			×	×		×				×	×		×	×		×	×		×			×	×									×				×		×	
[Yu et al. 2007]			×	×					×	×	×	×			×	×	×	×		×		×	×		×	×		×	×		×			×	×									×				×		×	
[Yu et al. 2008]			×	×					×	×	×	×			×	×	×	×		×		×	×		×	×		×	×		×			×	×									×				×		×	
[Zhang and Shen 2009]		×	×	×	×				×	×	×	×	×			×		×		×	×	×	×		×	×		×	×		×			×	×									×				×		×	
[Zhang et al. 2011]	×	×	×	×	×	×			×	×	×	×				×		×		×	×	×	×		×	×		×	×		×			×	×									×				×		×	
[Zou et al. 2002]		×	×	×					×	×	×	×			×	×	×	×		×		×	×		×	×		×	×		×			×	×									×				×		×	

REFERENCES

Abie, H. 2009. Adaptive security and trust management for autonomic message-oriented middleware. In *Proceedings of the IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS'09)*. 810–817.

Abie, H., Dattani, I., Novkovic, M., Bigham, J., Topham, S., and Savola, R. 2008. GEMOM - Significant and measurable progress beyond the state of the art. In *Proceedings of the 3rd International Conference on Systems and Networks Communications (ICSNC'08)*. 191–196.

Adnane, A., de Sousa, Jr., R. T., Bidan, C., and Mé, L. 2008. Autonomic trust reasoning enables misbehavior detection in OLSR. In *Proceedings of the 2008 ACM Symposium on Applied Computing*. ACM, 2006–2013.

Alampalayam, S. P. and Kumar, A. 2003. Security model for routing attacks in mobile ad hoc networks. In *Proceedings of the 58th Vehicular Technology Conference (VTC'03)*. Vol.3. IEEE, 2122–2126.

Alia, M. and Lacoste, M. 2008. A QoS and security adaptation model for autonomic pervasive systems. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications (COMPSAC'08)*. 943–948.

Alia, M., Lacoste, M., He, R., and Eliassen, F. 2010. Putting together QoS and security in autonomic pervasive systems. In *Proceedings of the 6th ACM Workshop on Qos and Security for Wireless and Mobile Networks*. ACM, 19–28.

Al-Nashif, Y., Kumar, A. A., Hariri, S., Qu, G., Luo, Y., and Szidarovsky, F. 2008. Multi-level intrusion detection system (ML-IDS). In *Proceedings of the International Conference on Autonomic Computing (ICAC'08)*. 131–140.

Anisetti, M., Ardagna, C. A., Damiani, E., Frati, F., Müller, H. A., and Pahlevan, A. 2012. Web service assurance: The notion and the issues. *Future Internet 4*, 1, 92–109.

Atighetchi, M. and Pal, P. 2009. From Auto-adaptive to survivable and self-regenerative systems successes, challenges, and future. In *Proceedings of the 8th IEEE International Symposium on Network Computing and Applications (NCA'09)*. 98–101.

Atighetchi, M., Pal, P., Webber, F., Schantz, R., Jones, C., and Loyall, J. 2004. Adaptive cyberdefense for survival and intrusion tolerance. *IEEE Internet Comput. 8*, 6, 25–33.

Atighetchi, M., Pal, P. P., Jones, C. C. et al. 2003. Building auto-adaptive distributed applications: The QuO-APOD experience. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*. 104–109.

Avižienis, A., Laprie, J.-C., and Randell, B. 2004. Dependability and its threats: A taxonomy. In *Building the Information Society*, R. Jacquart Ed., Springer, 91–120.

Balepin, I., Maltsev, S., Rowe, J., and Levitt, K. 2003. Using specification-based intrusion detection for automated response. In *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson Eds., Springer, Berlin, 136–154.

Ben Mahmoud, M. S., Larrieu, N., Pirovano, A., and Varet, A. 2010. An adaptive security architecture for future aircraft communications. In *Proceedings of the 29th Digital Avionics Systems Conference (DASC)*. IEEE/AIAA, 3.E.2–1–3.E.2–16.

Bencsáth, B., Pék, G., Buttyán, L., and Félegyházi, M. 2012. DUQU: Analysis, detection, and lessons learned. In *Proceedings of the ACM European Workshop on System Security (EuroSec)*.

Benjamin, D. P., Pal, P., Webber, F., Rubel, P., and Atigetchi, M. 2008. Using a cognitive architecture to automate cyberdefense reasoning. In *Proceedings of the ECSIS Symposium on Bio-inspired Learning and Intelligent Systems for Security (BLISS'08)*. 58–63.

Bijani, S. and Robertson, D. 2012. A review of attacks and security approaches in open multi-agent systems. *Artif. Intell. Rev.*, 1–30.

Blount, J. J., Tauritz, D. R., and Mulder, S. A. 2011. Adaptive rule-based malware detection employing learning classifier systems: A proof of concept. In *Proceedings of the 35th Annual IEEE Computer Software and Applications Conference Workshops (COMPSACW)*. 110 –115.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. 2007. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw. 80*, 4, 571–583.

Burns, J., Cheng, A., Gurung, P. et al. 2001. Automatic management of network security policy. In *Proceedings of the DARPA Information Survivability Conference Exposition II (DISCEX'01)*. Vol. 2. 12–26.

Casola, V., Mancini, E. P., Mazzocca, N., Rak, M., and Villano, U. 2008. Self-optimization of secure web services. *Comput. Commun. 31*, 18, 4312–4323.

Castro, M. and Liskov, B. 2002. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst. 20*, 4, 398–461.

Cavalcante, R. C., Bittencourt, I. I., da Silva, A. P., Silva, M., Costa, E., and Santos, R. 2012. A survey of security in multi-agent systems. *Expert Syst. Appl. 39*, 5, 4835–4846.

Cheng, B. H. C., Lemos, R., Giese, H. et al. 2009. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee Eds., Springer, Berlin, 1–26.

Cheng, P. C., Rohatgi, P., Keser, C., et al. 2007. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'07)*. 222–230.

Cheng, S.-W., Garlan, D., and Schmerl, B. 2006. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the International Workshop on Self-Adaptation and Self-Managing Systems*. ACM, 2–8.

Chess, D. M., Palmer, C. C., and White, S. R. 2003. Security in an autonomic computing environment. *IBM Syst. J. 42*, 1, 107–118.

Chigan, C., Li, L., and Ye, Y. 2005. Resource-aware self-adaptive security provisioning in mobile ad hoc networks. In *Proceedings of the Wireless Communications and Networking Conference*. Vol. 4. IEEE, 2118–2124.

Chong, J., Pal, P., Atigetchi, M., Rubel, P., and Webber, F. 2005. Survivability architecture of a mission critical system: The DPASA example. In *Proceedings of the 21st Annual Computer Security Applications Conference*.

Costa, M., Crowcroft, J., Castro, M. et al. 2008. Vigilante: End-to-end containment of Internet worm epidemics. *ACM Trans. Comput. Syst. 26*, 4, 9:1–9:68.

Cowan, C., Pu, C., Maier, D. et al. 1998. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th Conference on USENIX Security Symposium*. Vol. 7, USENIX Association, 5–5.

Cox, D. P., Al-Nashif, Y., and Hariri, S. 2007. Application of autonomic agents for global information grid management and security. In *Proceedings of the Summer Computer Simulation Conference*. Society for Computer Simulation International, 1147–1154.

Crosbie, M. and Spafford, G. 1995. Active defense of a computer system using autonomous agents. Tech. rep. 95-008, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN.

Debar, H., Thomas, Y., Cuppens, F., and Cuppens-Boulahia, N. 2007. Enabling automated threat response through the use of a dynamic security policy. *J. Comput. Virol. 3*, 3, 195–210.

de Oliveira, T. R., de Oliveira, S., Macedo, D. F., and Nogueira, J. M. 2011. An adaptive security management model for emergency networks. In *Proceedings of the 7th Latin American Network Operations and Management Symposium (LANOMS)*, 1–4.

De Palma, N., Hagimont, D., Boyer, F., and Broto, L. 2012. Self-protection in a clustered distributed system. *IEEE Trans. Parall. Distrib. Syst. 23*, 2, 330–336.

Dittmann, J., Karpuschewski, B., Fruth, J., Petzel, M., and M"under, R. 2010. An exemplary attack scenario: Threats to production engineering inspired by the Conficker worm. In *Proceedings of the 1st International Workshop on Digital Engineering*. ACM, 25–32.

Dragoni, N., Massacci, F., and Saidane, A. 2009. A self-protecting and self-healing framework for negotiating services and trust in autonomic communication systems. *Comput. Netw. 53*, 10, 1628–1648.

Elkhodary, A. and Whittle, J. 2007. A survey of approaches to adaptive application security. In *Proceedings of the Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07)*. 16.

English, C., Terzis, S., and Nixon, P. 2006. Towards self-protecting ubiquitous systems: Monitoring trust-based interactions. *Pers. Ubiq. Comput. 10*, 1, 50–54.

Erlingsson, U. and Schneider, F. B. 2000. SASI enforcement of security policies: A retrospective. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'00)*. Vol. 2. 287–295.

Fayssal, S., Alnashif, Y., Kim, B., and Hariri, S. 2008. A proactive wireless self-protection system. In *Proceedings of the 5th International Conference on Pervasive Services*. ACM, 11–20.

Feiertag, R., Rho, S., Benzinger, L. et al. 2000. Intrusion detection inter-component adaptive negotiation. *Comput. Netw. 34*, 4, 605–621.

Foo, B., Wu, Y. S., Mao, Y. C., Bagchi, S., and Spafford, E. 2005a. ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'05)*. 508–517.

Foo, B., Wu, Y.-S., Mao, Y.-C., Bagchi, S., and Spafford, E. 2005b. ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'05)*. 508–517.

Foster, H., Spanoudakis, G., and Mahbub, K. 2012. Formal certification and compliance for run-time service environments. In *Proceedings of the IEEE 9th International Conference on Services Computing (SCC)*. 17–24.

Frincke, D., Wespi, A., and Zamboni, D. 2007. From intrusion detection to self-protection. *Comput. Netw. 51*, 5, 1233–1238.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer 37*, 10, 46–54.

Gelenbe, E. and Loukas, G. 2007. A self-aware approach to denial of service defence. *Comput. Netw. 51*, 5, 1299–1314.

Ghosh, A. K. and Voas, J. M. 1999. Inoculating software for survivability. *Commun. ACM 42*, 7, 38–44.

Ghosh, A. K., O'Connor, T., and McGraw, G. 1998. An automated approach for identifying potential vulnerabilities in software. In *Proceedings of the IEEE Symposium on Security and Privacy*. 104–114.

Goel, A., Po, K., Farhadi, K., Li, Z., and de Lara, E. 2005. The taser intrusion recovery system. *SIGOPS Oper. Syst. Rev. 39*, 5, 163–176.

Guttman, J. D. and Herzog, A. L. 2005. Rigorous automated network security management. *Int. J. Inf. Sec. 4*, 1, 29–48.

Hafiz, M., Adamczyk, P., and Johnson, R. E. 2007. Organizing security patterns. *IEEE Softw. 24*, 4, 52–60.

Hashii, B., Malabarba, S., Pandey, R., and Bishop, M. 2000. Supporting reconfigurable security policies for mobile programs. *Comput. Netw. 33*, 1–6, 77–93.

He, R. and Lacoste, M. 2008a. Applying component-based design to self-protection of ubiquitous systems. In *Proceedings of the 3rd ACM Workshop on Software Engineering for Pervasive Services*. ACM, 9–14.

He, R. and Lacoste, M. 2008b. Applying component-based design to self-protection of ubiquitous systems. In *Proceedings of the 3rd ACM Workshop on Software Engineering for Pervasive Services*. 9–14.

He, R., Lacoste, M., and Leneutre, J. 2010a. A policy management framework for self-protection of pervasive systems. In *Proceedings of the 6th International Conference on Autonomic and Autonomous Systems (ICAS)*. 104–109.

He, R., Lacoste, M., and Leneutre, J. 2010b. Virtual security kernel: A component-based OS architecture for self-protection. In *Proceedings of the IEEE 10th International Conference on Computer and Information Technology (CIT)*. 851–858.

Hinton, H., Cowan, C., Delcambre, L., and Bowers, S. 1999. SAM: Security adaptation manager. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99)*. 361–370.

Huang, Y., Arsenault, D., and Sood, A. 2006. Closing cluster attack windows through server redundancy and rotations. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. 12–21.

Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. D. 1995. Software rejuvenation: Analysis, module and applications. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25)*. 381–390.

Huebscher, M. C. and McCann, J. A. 2008. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv. 40*, 3, 7:1–7:28.

Igure, V. and Williams, R. 2008. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Commun. Surv. Tutor. 10*, 1, 6–19.

Jabbour, G. G. and Menasee, D. A. 2008. Policy-based enforcement of database security configuration through autonomic capabilities. In *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems (ICAS'08)*. 188–197.

Jabbour, G. and Menasce, D. A. 2009. The insider threat security architecture: A framework for an integrated, inseparable, and uninterrupted self-protection mechanism. In *Proceedings of the International Conference on Computational Science and Engineering (CSE'09)*. 244–251.

Jain, S., Shafique, F., Djeric, V., and Goel, A. 2008. Application-level isolation and recovery with solitude. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*. ACM, 95–107.

Jansen, B., Ramasamy, H., Schunter, M., and Tanner, A. 2008. Architecting dependable and secure systems using virtualization. In *Architecting Dependable Systems V*, R. de Lemos, F. Di Giandomenico, C. Gacek, H. Muccini, and M. Vieira Eds., Springer, Berlin, 124–149.

Jean, E., Jiao, Y., Hurson, A. R., and Potok, T. E. 2007. Boosting-based distributed and adaptive security-monitoring through agent collaboration. In *Proceedings of the IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology Workshops*. 516–520.

Kephart, J. O. and Chess, D. M. 2003. The vision of autonomic computing. *Computer 36*, 1, 41–50.

Kephart, J. O., Sorkin, G. B., Swimmer, M., and White, S. R. 1997. Blueprint for a computer immune system. In *Proceedings of the Virus Bulletin International Conference*. San Francisco, CA.

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

Klein, C., Schmid, R., Leuxner, C., Sitou, W., and Spanfelner, B. 2008. A survey of context adaptation in autonomic computing. In *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems (ICAS'08)*. 106–111.

Konrad, S., Cheng, B. H. C., Campbell, L. A., and Wassermann, R. 2003. Using security patterns to model and analyze security requirements. In *Proceedings of the Symposium on Requirements Engineering for High Assurance Systems (RHAS'03)*. 11.

Kramer, J. and Magee, J. 2007. Self-managed systems: An architectural challenge. In *Proceedings of the Symposium on the Future of Software Engineering (FOSE'07)*. 259–268.

Kumar, G., Kumar, K., and Sachdeva, M. 2010. The use of artificial intelligence based techniques for intrusion detection: A review. *Artif. Intell. Rev. 34*, 4, 369–387.

Labraoui, N., Gueroui, M., Aliouat, M., and Petit, J. 2010. Adaptive security level for data aggregation in wireless sensor networks. In *Proceedings of the 5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC)*. 325–330.

Lamprecht, C. J. and van Moorsel, A. P. A. 2007. Adaptive SSL: Design, implementation and overhead analysis. In *Proceedings of the 1st International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*. 289–294.

Lamprecht, C. J. and van Moorsel, A. P. A. 2008. Runtime security adaptation using adaptive SSL. In *Proceedings of the 14th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'08)*. 305–312.

Langner, R. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Secur. Privacy 9*, 3, 49–51.

Laureano, M., Maziero, C., and Jamhour, E. 2007. Protecting host-based intrusion detectors through virtual machines. *Comput. Netw. 51*, 5, 1275–1283.

Lee, W., Miller, M., Stolfo, S. J., Fan, W., and Zadok, E. 2002. Toward cost-sensitive modeling for intrusion detection and response. *J. Comput. Secur. 10*, 2002.

Lemos, R., Giese, H., Müller, H. A., et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, R. Lemos, H. Giese, H.A. Müller, and M. Shaw Eds., Springer, Berlin, Heidelberg, 1–32.

Liang, Z. and Sekar, R. 2005. Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*. 213–222.

Lim, Y. T., Cheng, P.-C., Rohatgi, P., and Clark, J. A. 2009. Dynamic security policy learning. In *Proceedings of the 1st ACM Workshop on Information Security Governance*. ACM, 39–48.

Lorenzoli, D., Mariani, L., and Pezze, M. 2007. Towards self-protecting enterprise applications. In *Proceedings of the 18th IEEE International Symposium on Software Reliability (ISSRE'07)*. 39–48.

Malek, S., Esfahani, N., Menasce, D. A., Sousa, J. P., and Gomaa, H. 2009. Self-Architecting Software SYstems (SASSY) from QoS-annotated activity models. In *Proceedings of the ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS'09)*, 62–69.

Marino, J. and Rowley, M. 2010. *Understanding SCA (Service Component Architecture)*. Pearson Education.

Maximilien, E. M. and Singh, M. P. 2004. Toward autonomic web services trust and selection. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, ACM, 212–221.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. C. 2004. A taxonomy of compositional adaptation. Report MSU-CSE-04-17, Michigan State University.

Menasce, D., Gomaa, H., Malek, S., and Sousa, J. 2011. SASSY: A framework for self-architecting service-oriented systems. *IEEE Softw. 28*, 6, 78–85.

MITRE. 2011. CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors. *2011 CWE/SANS Top 25 Most Dangerous Software Errors*. http://cwe.mitre.org/top25/.

Montangero, C. and Semini, L. 2004. Formalizing an adaptive security infrastructure in mobadtl. In *Proceedings of the FCS'04*, 301.

Morin, B., Mouelhi, T., Fleurey, F., Le Traon, Y., Barais, O., and Jézéquel, J.-M. 2010. Security-driven model-based dynamic adaptation. In *Proceedings of the IEEE/ACM International Conference on Automated software engineering*, ACM, 205–214.

Mouelhi, T., Fleurey, F., Baudry, B., and Le Traon, Y. 2008. A model-based framework for security policy specification, deployment and testing. In *Model Driven Engineering Languages and Systems*, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl and M. Völter Eds., Springer, Berlin, 537–552.

Musman, S. and Flesher, P. 2000. System or security managers adaptive response tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'00)*. Vol. 2, 56–68.

Nagarajan, A., Nguyen, Q., Banks, R., and Sood, A. 2011. Combining intrusion detection and recovery for enhancing system dependability. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 25–30.

Neumann, P. G. and Porras, P. A. 1999. Experience with EMERALD to date. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, 73–80.

Nguyen, Q. L. and Sood, A. 2011. A comparison of intrusion-tolerant system architectures. *IEEE Secur. Priv. 9*, 4, 24–31.

Okhravi, H., Comella, A., Robinson, E., and Haines, J. 2012. Creating a cyber moving target for critical infrastructure applications using platform diversity. *Int. J. Crit. Infrastruct. Protection 5*, 1, 30–39.

Okhravi, H., Robinson, E. I., Yannalfo, S., Michaleas, P. W., Haines, J., and Comella, A. 2010. TALENT: Dynamic platform heterogeneity for cyber survivability of mission critical applications. In *Proceedings of the Secure and Resilient Cyber Architecture Conftrence (SCRA'10)*.

Ostrovsky, R. and Yung, M. 1991. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*. ACM, 51–59.

Pal, P., Webber, F., and Schantz, R. 2007. The DPASA survivable JBI-a high-water mark in intrusion-tolerant systems. In *Proceedings of the WRAITS 2007*.

Pasquale, L., Salehie, M., Ali, R., Omoronyia, I., and Nuseibeh, B. 2012. On the role of primary and secondary assets in adaptive security: An application in smart grids. In *Proceedings of the 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 165–170.

Perrin, C. 2008. The CIA Triad. http://www.techrepublic.com/blog/security/the-cia-triad/488.

Porras, P. A. and Neumann, P. G. 1997. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, 353–365.

Portokalidis, G. and Bos, H. 2007. SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Comput. Netw. 51*, 5, 1256–1274.

Psaier, H. and Dustdar, S. 2011. A survey on self-healing systems: Approaches and systems. *Computing 91*, 1, 43–73.

Raissi, J. 2006. Dynamic selection of optimal cryptographic algorithms in a runtime environment. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'06)*. 184–191.

Rawat, S. and Saxena, A. 2009. Danger theory based SYN flood attack detection in autonomic network. In *Proceedings of the 2nd International Conference on Security of Information and Networks*. ACM, 213–218.

Reiser, H. P. and Kapitza, R. 2007a. VM-FIT: Supporting intrusion tolerance with virtualisation technology. In *Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems (in conjunction with Eurosys 2007)*.

Reiser, H. P. and Kapitza, R. 2007b. Hypervisor-based efficient proactive recovery. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'07)*, 83–92.

Reynolds, J., Just, J., Lawson, E., Clough, L., Maglich, R., and Levitt, K. 2002. The design and implementation of an intrusion tolerant system. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*. 285–290.

Reynolds, J. C., Just, J., Clough, L., and Maglich, R. 2003. On-line intrusion detection and attack prevention using diversity, generate-and-test, and generalization. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*.

Sadjadi, S. M. 2003. A survey of adaptive middleware. Michigan State University Report MSU-CSE-03-35.

Salehie, M., Pasquale, L., Omoronyia, I., Ali, R., and Nuseibeh, B. 2012. Requirements-driven adaptive security: Protecting variable assets at runtime. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*. 111–120. IEEE.

Salehie, M. and Tahvildari, L. 2009. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst. 4*, 2, 14:1–14:42.

Savola, R. M. and Heinonen, P. 2010. Security-measurability-enhancing mechanisms for a distributed adaptive security monitoring system. In *Proceedings of the 4th International Conference on Emerging Security Information Systems and Technologies (SECURWARE)*. 25–34.

Saxena, A., Lacoste, M., Jarboui, T., Lücking, U., and Steinke, B. 2007. A software framework for autonomic security in pervasive environments. In *Information Systems Security*, P. McDaniel and S. Gupta Eds., Springer, Berlin, 91–109.

Schaub, F., Konings, B., Weber, M., and Kargl, F. 2012. Towards context adaptive privacy decisions in ubiquitous computing. In *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 407–410.

Schneider, F. B. 2003. Enforceable security policies. *Foundations of Intrusion Tolerant Systems*, [Organically Assured and Survivable Information Systems], 117–137.

Shameli-Sendi, A., Ezzati-Jivan, N., Jabbarifar, M., and Dagenais, M. 2012. Intrusion response systems: Survey and taxonomy. *Int. J. Comput. Sci. Netw. Secur. 12*, 1, 1–14.

Sibai, F. M. and Menasce, D. A. 2011. Defeating the insider threat via autonomic network capabilities. In *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS)*. 1–10.

Sibai, F. M. and Menasce, D. A. 2012. Countering network-centric insider threats through self-protective autonomic rule generation. In *Proceedings of the IEEE 6th International Conference on Software Security and Reliability (SERE)*. 273–282.

Sousa, P., Bessani, A. N., Correia, M., Neves, N. F., and Verissimo, P. 2007. Resilient intrusion tolerance through proactive and reactive recovery. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing (PRDC'07)*. 373–380.

Sousa, P., Bessani, A. N., Correia, M., Neves, N. F., and Verissimo, P. 2010. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Trans. Parall. Distrib. Syst. 21*, 4, 452–465.

Sousa, P., Neves, N. F., Verissimo, P., and Sanders, W. H. 2006. Proactive resilience revisited: The delicate balance between resisting intrusions and remaining available. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*. 71–82.

Sowa, J. F. and Zachman, J. A. 1992. Extending and formalizing the framework for information systems architecture. *IBM Syst. J. 31*, 3, 590–616.

Spanoudakis, G., Kloukinas, C., and Androutsopoulos, K. 2007. Towards security monitoring patterns. In *Proceedings of the ACM Symposium on Applied Computing*, ACM. 1518–1525.

Stakhanova, N., Basu, S., and Wong, J. 2007a. A taxonomy of intrusion response systems. *Int. J. Inf. Comput. Sec. 1*, 1, 169–184.

Stakhanova, N., Basu, S., and Wong, J. 2007b. A cost-sensitive model for preemptive intrusion response systems. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications (AINA'07)*. 428–435.

Strasburg, C., Stakhanova, N., Basu, S., and Wong, J. S. 2009. A framework for cost sensitive assessment of intrusion response selection. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*. 355–360.

Sundaram, A. 1996. An introduction to intrusion detection. *Crossroads 2*, 4, 3–7.

Swiderski, F. and Snyder, W. 2004. *Threat Modeling*. Microsoft Press, Redmond, WA.

Swimmer, M. 2007. Using the danger model of immune systems for distributed defense in modern data networks. *Comput. Netw. 51*, 5, 1315–1333.

Taddeo, A. V. and Ferrante, A. 2009. Run-time selection of security algorithms for networked devices. In *Proceedings of the 5th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. ACM, 92–96.

Tan, L., Zhang, X., Ma, X., Xiong, W., and Zhou, Y. 2008. AutoISES: Automatically inferring security specifications and detecting violations. In *Proceedings of the 17th Conference on Security Symposium*. 379–394.

Tang, C. and Yu, S. 2008. A dynamic and self-adaptive network security policy realization mechanism. In *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC'08)*. 88–95.

Uribe, T. E. and Cheung, S. 2004. Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*. 66–74.

Valdes, A., Almgren, M., Cheung, S. et al. 2004. An architecture for an adaptive intrusion-tolerant server. In *Security Protocols*, B. Christianson, B. Crispo, J. Malcolm, and M. Roe Eds., Springer, Berlin, 569–574.

Verissimo, P. E., Neves, N. F., Cachin, C. et al. 2006. Intrusion-tolerant middleware: The road to automatic security. *IEEE Secur. Priv. 4*, 4, 54–62.

Vikram, K., Prateek, A., and Livshits, B. 2009. Ripley: Automatically securing web 2.0 applications through replicated execution. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ACM, 173–186.

Villegas, N. M., Müller, H. A., Tamura, G., Duchien, L., and Casallas, R. 2011. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 80–89.

Villegas, N. M., Tamura, G., Müller, H. A., Duchien, L., and Casallas, R. 2013. DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, R. Lemos, H. Giese, H. A. Müller, and M. Shaw Eds., Springer Berlin, 265–293.

Wang, F., Jou, F., Gong, F., Sargor, C., Goseva-Popstojanova, K., and Trivedi, K. 2003. SITAR: A scalable intrusion-tolerant architecture for distributed services. In *Foundations of Intrusion Tolerant Systems* [Organically Assured and Survivable Information Systems], 359–367.

Wang, H., Dong, X., and Wang, H. 2009. A method for software security growth based on the real-time monitor algorithm and software hot-swapping. In *Proceedings of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC'09)*. 137–142.

Weyns, D., Malek, S., and Andersson, J. 2012. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Trans. Autonom. Adapt. Syst. 7*, 1, 1–61.

White, S. R., Swimmer, M., Pring, E. J., Arnold, W. C., Chess, D. M., and Morar, J. F. 1999. Anatomy of a commercial-grade immune system. IBM Research White Paper.

Wu, Y.-S., Foo, B., Mao, Y.-C., Bagchi, S., and Spafford, E. H. 2007. Automated adaptive intrusion containment in systems of interacting services. *Comput. Netw. 51*, 5, 1334–1360.

Xiao, L. 2008. An adaptive security model using agent-oriented MDA. *Inf. Softw. Tech. 51*, 5, 933–955.

Xiao, L., Peet, A., Lewis, P. et al. 2007. An adaptive security model for multi-agent systems and application to a clinical trials environment. In *Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC'07)*. 261–268.

Yau, S. S., Yao, Y., and Yan, M. 2006. Development and runtime support for situation-aware security in autonomic computing. In *Proceedings of the 3rd International Conference on Autonomic and Trusted Computing*, Springer-Verlag, 173–182.

Yu, Z., Tsai, J. J. P., and Weigert, T. 2007. An automatically tuning intrusion detection system. *IEEE Trans. Syst. Man Cybernet., Part B: Cybernetics 37*, 2, 373–384.

Yu, Z., Tsai, J. J. P., and Weigert, T. 2008. An adaptive automatically tuning intrusion detection system. *ACM Trans. Auton. Adapt. Syst. 3*, 3, 10:1–10:25.

Yuan, E. and Malek, S. 2012. A taxonomy and survey of self-protecting software systems. In *Proceedings of the 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 109–118.

Zhang, Z., Naït-Abdesselam, F., Ho, P.-H., and Kadobayashi, Y. 2011. Toward cost-sensitive self-optimizing anomaly detection and response in autonomic networks. *Comput. Secur. 30*, 6–7, 525–537.

Zhang, Z. and Shen, H. 2009. M-AID: An adaptive middleware built upon anomaly detectors for intrusion detection and rational response. *ACM Trans. Auton. Adapt. Syst. 4*, 4, 24:1–24:35.

Zhu, M., Yu, M., Xia, M. et al. 2011. VASP: Virtualization assisted security monitor for cross-platform protection. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, ACM, 554–559.

Zou, J., Lu, K., and Jin, Z. 2002. Architecture and fuzzy adaptive security algorithm in intelligent firewall. In *Proceedings of the MILCOM'02*. Vol. 2, 1145–1149.

Received November 2012; revised December 2012, April 2013; accepted June 2013