A Taxonomy and Survey of Self-Protecting Software Systems

Eric Yuan
Department of Computer Science
George Mason University
Fairfax, VA 22030
eyuan@gmu.edu

Sam Malek
Department of Computer Science
George Mason University
Fairfax, VA 22030
smalek@gmu.edu

***Abstract*—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 shown inadequate for the challenges posed by modern software systems. While existing research has made significant progress towards autonomic and adaptive security, gaps and challenges remain. In this paper, we report on an extensive study and analysis of the literature in this area. The crux of our contribution is a comprehensive taxonomy to classify and characterize research efforts in this arena. We also describe our experiences with applying the taxonomy to numerous existing approaches. This has shed light on several challenging issues and resulted in interesting observations that could guide the future research.**

Keywords – self-protection; autonomic systems; taxonomy; self-management; adaptive security

I. INTRODUCTION

Security is the Achilles heel of most modern software systems. In spite of the significant progress over the past few decades, the challenges posed by security are more prevalent than ever before. As the awareness grows of the limitations of traditional, often static and rigid, security models, research shifts to dynamic models, where security threats are detected and mitigated at runtime, i.e., *self-protection*.

Self-protection is closely related to the other self-* properties, 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 integrity 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 adaptation.

Self-protection has been identified as one of the essential traits of self-management for autonomic computing systems. Kephart and Chess characterized self-protection from two perspectives [16]: 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.

Chess et al. also provided valuable guidance on the key research directions for securing autonomic systems, such as ways to represent and reason about security policies and state, methods for effectively differentiating between normal system failures and malicious attacks, policies and algorithms for resisting fraud and persuasion, and common languages and taxonomies for communicating and negotiating security policies and states [7].

The past decade has seen extensive and systematic research being conducted around self-adaptive and self-managing systems. Research that focuses 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 such as information survivability and tactical systems.

This paper surveys recent research on self-protecting software systems with a dual focus – to evaluate relevant research approaches, but at the same time identify the taxonomies and schemes against which such research efforts may be classified and organized. The contributions of the paper include: (1) A proposed taxonomy for consistently and comprehensively classifying self-protection mechanisms and research approaches; (2) An extensive 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

The rest of the paper is organized as follows. Section 2 details the need for self-protection in today's software systems through a motivating example. Section 3 surveys the existing taxonomies and classification schemes related to system self-protection and adaptive security. Section 4 proposes a coherent and comprehensive taxonomy that builds on top of existing taxonomies, and Section 5 attempts to classify current and past self-protection research initiatives against the proposed taxonomy. Section 6 presents the analysis on the survey results, offering observations on patterns, trends, gaps, and opportunities. Based on this analysis, Section 7 concludes the paper with a set of recommendations for future self-protecting system research.

II. MOTIVATION

There is an unprecedented need for autonomic and adaptive security 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.

A. From Outside: Ever-Increasing Cyber Threats

As software systems become more distributed, interactive and ubiquitous, networking services become an integral part of 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 [52].
- 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 attack demonstrates its ability to target multiple architecture layers of the target system — exploiting the network and host-level vulnerabilities is only a stepping stone for malicious actions at the application level [55].
- The Duqu worm, lately 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, coordinated and persistent nature of today's cyber threats [54].

What becomes 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 at different layers.

B. *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 [56]). Similarly, a web service orchestrator could be constructed to dynamically discover and access different service providers (e.g., using Business Process Execution Language (BPEL) and a 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.

Thus, 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.

C. *A Simple Motivating Example*

Self-protecting 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 application and starts to exfiltrate a large amount of confidential user information. The much simplified architecture of the application is shown in Figure 1.

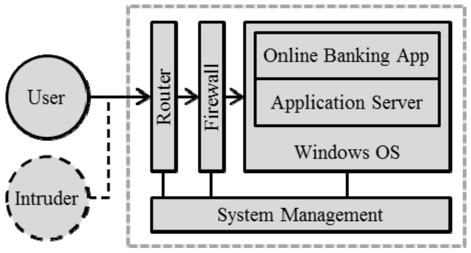

Figure 1: Simple Online Banking System Example

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 intrusion detection capability of the network router detects this illegal access and automatically disabled the connection from the source IP address;
- The firewall detects unusually large data transfer which exceeds predefined policy threshold and accordingly disables the HTTP connection;
- An overall system transaction monitor (not shown), sensing an unusual data retrieval pattern from the Windows server, shuts down the server and redirects all request to a backup server;
- Multiple application server instances are deployed on the Windows machine. By comparing the behavior from all server instances, the anomaly from the compromised application server instance is detected and the instance consequently is shut down;
- The built-in access control policies within the online banking application detects abnormal usage pattern, and therefore disables the user account in question.

As the paper will show later, many other self-protecting mechanisms are possible. How do these different approaches compare against one another? What approaches apply to what threats? Are some more effective than others? If so, under what conditions? To better answer these questions, one must take a methodical approach in evaluating 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 paper seeks to take a step further 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 to self-protection.

III. RELATED TAXONOMY WORK

A. *Taxonomy of Compositional Adaptation*

Reference [25] focuses on composition as a key paradigm for adaptation, which includes enabling techniques such as Separation of Concerns, Computational Reflection, and Component-based Design. The paper describes a taxonomy based on how, when, and where software composition takes place:

- The "how" metric consists of criteria around techniques, transparency, granularity, coverage, and support;
- The "when" metric looks at whether the composition is static or dynamic, and whether the adaptation occurs at development time, compile/link time, load time, or runtime;
- The "where" metric is concerned with whether the adaptation occurs in the application layer, the middleware layer, or in the Operating System or hardware layer.

A related survey can be found in [34] with a stronger focus on middleware. It classifies middleware as either Quality of Service (QoS) oriented, dependable, or embedded. It follows the same taxonomy on adaptation type, namely, static (customizable, configurable) and dynamic (tunable and mutable).

B. *Adaptive Security Approaches at the Application Level*

Reference [8] provides a good survey on *application-level* adaptive security mechanisms. It builds on top of the taxonomy of computational paradigms defined in [34], and adds the following additional dimensions: (1) *Reconfiguration scale*, which categorizes whether the reconfiguration happens in a single unit, inter-unit, or architecture wide; (2) *Conflict handling*, which refers to whether the system resolves conflicts among candidate configurations in an autonomic fashion or delegates it to a human user (or a hybrid of the two)

These classification dimensions are certainly applicable to self-protection systems in general; however, the paper's focus is primarily on the application layer.

C. *Taxonomy of Self-Adaptive Software*

Reference [35] is a comprehensive survey on self-adaptive software in general. It offers a taxonomy of self-adaptation that covers a variety of dimensions, such as objects to adapt, realization approach, adaptation type, temporal characteristics, and degree of human interactions. Some security-relevant ones include:

- Adaptation layers, such as middleware or application-layer, as defined by McKinkey et al. [25]
- Realization approach, such as static vs. dynamic decision making, and external vs. internal
- Realization type, such as open vs. close, model-based vs. model-free, or specific vs. generic
- Temporal characteristics, such as reactive vs. proactive adaptation
- Trust. The paper offers different views on the interpretation of trust on self-adaptive systems without offering a definitive taxonomy

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.

D. *Intrusion-Tolerant System (ITS) Architectures*

Reference [29] offers an up-to-date survey on 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. This paper classifies ITS approaches using a taxonomy with four categories: (1) Detection-triggered, (2) Algorithm-driven, (3) Recovery-based, and (4) Hybrid of the three. The paper identified a rich set of research initiatives, some of which are also covered in our analysis in Section V.

This taxonomy is closely related to other classifications identified earlier. For example, detection-triggered approaches are by definition "reactive" approaches, while recovery-based approaches are "proactive" in nature. As correctly pointed out by the authors, these approaches are by no means mutually exclusive and may be used together.

E. *Other related taxonomies*

A number of surveys focused on classifying security patterns. Reference [6], for example, uses metrics such as purpose (creational, structural, and behavioral) and abstraction level (network, host, application). Even though they are used to classify security patterns, metrics such as abstraction level will apply to adaptive security approaches as well.

It is worth noting that software systems today have evolved to include another layer of abstraction beyond middleware and software applications – namely, the *services* layer. SOA as an architecture style promotes loose-coupling, heterogeneity and composability, and deserves security treatment in its own right.

A similar effort [14] proposed other ways to organize security patterns, many of which are applicable to classifying self-protection approaches:

- Along the "CIA" model [51], i.e. Confidentiality, Integrity, and Availability;
- Along the context of where the security patterns are applied: perimeter security, core security, and external security;
- Along Microsoft's classification scheme of Stakeholder, Function, Data, and Test;
- Along the Zachman's Enterprise Architecture Framework [57] which uses the familiar "What, Where, Who, When, Why, How" dimensions;
- Along the "STRIDE" security threat model [40]: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege.

Other related taxonomies include those on software vulnerabilities [53], and fault tolerance schemes [2].

F. *Observations*

The aforementioned taxonomies, 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, or
- focus only on certain layers of self-protection systems (such as middleware), or
- are 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 (with [29] being an exception).

In the next section, we propose our taxonomy to help classify existing self-protection and adaptive security research.

IV. Proposed Taxonomy

The proposed taxonomy builds on top of the existing work surveyed in the previous section. It consists of nine dimensions that fall into two categories: Research Positioning and Technique Characterization. The dimensions and their allowed values are shown in Table 1, along with examples using the simple online banking service scenario given in Section II.C.

The first category, Research Positioning, helps characterize the "WHAT" aspects, that is, the objectives and intent of self-protection research. It includes four dimensions:

1) Self-Protection Levels. This dimension focuses on the sophistication of self-protection mechanisms, and is inspired by the Monitor, Analyze, Plan, Excute (MAPE) value chain proposed in [16]. "Monitor & Detect" is the most basic level, followed by "Analyze & Characterize", which focuses on the characterization and understanding of the nature/type of the attacks. The third level is "Plan & Prevent" that represents the highest level of sophistication; a security approach reaching this level will allow a system to predict security issues and plan for countermeasures in an autonomous fashion. The three levels are also consistent with Kramer and Magee's three level reference architecture for self-managed systems [17], namely, Component Management, Change Management, and Goal Management.

2) Architecture Levels. This dimension denotes which architecture layer is the focus of a particular research effort, namely:

- Network – focusing on communication links, protocols, intrusion detection, and intrusion protection
- Host – dealing with hardware, firmware, OS, and virtualization
- Software/applications – concerning programming languages, middleware, and application platforms
- Services – seeking to protect web services and service interactions, a recent security area attributed to the rise of SOA [49].

This dimension also includes a 5th level, called "abstract architecture", to indicate any self-protection research that focuses on the architecture as a whole, at the abstract level, i.e., dealing with architecture components, connectors, configurations, and architecture styles. An architecture-based approach enjoys many benefits such as generality, abstraction, and potential for scalability, as pointed out in [17].

3) Lifecycle Focus. This dimension indicates what part of the software development lifecycle is a research effort concerned with. For the purposes of this paper we simply use two phases, Development Time and Runtime. Obviously self-protection is for the most part concerned with a system at runtime, however, it is also necessary to consider how to design, develop, test, and deploy software systems for self-protection.

4) Protection Goals. This dimension specifies the security goal(s) of a research approach. Here we follow the same traditional CIA model for its simplicity, as used in [29] and [14].

- Confidentiality – to protect against unauthorized access, spoofing, impersonation, etc.
- Integrity – to protect against system tampering, hijacking, defacing, and subversion
- Availability – to protect against degradation or denial of service

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

The second category, Technique Characterization, is concerned with describing the "HOW" aspects of self-protection research. It includes five dimensions:

5) Adaptation Topology looks at whether a security approach focuses on the local (i.e., a single host or node) or global scale of the system. For those approaches focusing on the global scale, this dimension also specifies whether they use centralized or decentralized coordination and planning. In a decentralized topology, the nodes often "federate" with each other in a peer-to-peer fashion without relying on a central node.

6) Adaptation Decision-Making. This dimension deals with how the adaptive security decisions are made – is it made by a human user? If it is made autonomously by the system, is it driven by an algorithm, or by heuristics such as policies or rule sets? This dimension is important in the sense that it characterizes the extent of the "intelligence" behind an adaptive security approach.

7) Adaptation Tempo. This dimension indicates when and how often adaptations occur, which in turn is dependent on whether the approach is reactive or proactive. In reactive mode, system adaptation occurs in response to detected threats. In proactive mode, system adaptation may occur according to a predefined schedule, with or without detected threats.

8) Adaptation Point represents where in the system security adaptations occur. Here we adopt a metric from [14] and define the values as System Boundary or System Internal. In the former case, the system self-protection relies on peremeter security. For the latter, the system relies on internal security mechanisms.

9) Adaptation Patterns indicate any recurring architectural patterns that rise from the self-protection approaches. Many architecture and design patterns exist, but five key patterns have emerged in our research as being especially effective in establishing self-protecting behavior:

- Containment – use proxies, wrappers, or containers around the protected resource, so that the request to and response from the resource may be monitored and sanitized in a way that is transparent to the resource. The SITAR system [43], for instance, protects COTS servers by deploying an adaptive proxy server in the front, which detects and reacts to intrusions. Invalid requests/responses trigger reconfiguration of the COTS server.
- Redundancy – use replicas in parallel or failover configurations. To safely switch requests from one instance to another, the system may need to use checkpointing to save the current system state. Redundancy is a well-known pattern that is widely used in self-protecting systems (e.g., to mitigate denial of service attacks) and often combined with other patterns.
- Diversity – develop different implementations for the same software specification, in the hope that the attacks to one implementation may not affect others. This may be achieved through the use of different programming languages, OS, or hardware platforms. The HACQIT system [32] combines redundancy and 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. The TALENT system [30] uses OS-level heterogeneity such as deployment C-language programs on different OS kernels to thwart platform-specific attacks.
- Recomposition – Use techniques such as reorganization,

	Dimension	**Value**	**Example (Illustrated using the Simple Online Banking System in Section II.C)**
Research Positioning	Self-Protection Levels	Monitor & Detect	*Network based instruction detection function of the router detects intrusion event*
		Analyze & Characterize	*System Management (SM) component analyzes intrusion event and matches against known attacks*
		Plan & Prevent	*SM reconfigures router policy to prevent similar attack types*
	Architecture Levels	Network	*Firewall adapts policies on ports and protocols based on security threat levels*
		Host	*SM pushes OS patches to the server in response to new Windows vulnerabilities*
		Software/Apps	*SM instructs application server to disable the faulting application module*
		Services	*The online banking web service disables trust for the infected user account*
		Abstract Architecture	*Triggered by slow response time, SM uses adapation strategy to spawn a new app server instance*
	Lifecycle Focus	Development Time	*The intrusion was captured as a new security test case in the system dev/test environments*
		Runtime	*See all examples for the "Self-Protection Level" and "Archtiecture Level" dimensions*
	Protection Goals	Confidentiality	*Prevent intruder from getting user financial or personal data*
		Integrity	*Prevent intruder from altering or erasing user financial data*
		Availability	*Prevent intruder from bringing down the banking app or web service*
Technique Characterization	Adaptation Topology	Local only	*Adaptative security measures within a single app server*
		Centralized	*When the SM monitors and controls multiple host servers*
		Decentralized	*The system in Figure 1 are deployed in multiple sites, interconnected with one another*
	Adaptation Decision-Making	Human-driven	*Intrusion alert is sent to system administrator, who immediately logs in to disable access to server*
		Heuristics-driven	*A policy states "When the data retrieval amount exceeds 100KB threshold, disable user account"*
		Algorithm-driven	*There are N servers running concurrently; SM uses voting algorithm to identify the infected server*
	Adaptation Tempo	Reactive	*System reacts to the intrusion event by disabling old server, connect router to backup server*
		Proactive	*System runs penetration testing every night to ensure system integrity*
	Adaptation Point	System boundary	*System relies on network based instruction detection function of the router*
		System internal	*System relies on access control policies managed within the SM*
	Adaptation Patterns	Containment	*App server has a proxy serve that inspects every request before it is sent to the banking app*
		Redundancy	*Multiple app server instances are running at the same time*
		Diversity	*System runs two implementations of the banking web service, one in Java, the other in .NET*
		Recomposition	*SM uses instrumentation interfaces to adjust the maximum concurrent requests to app server*
		Rejuvenation	*System recycles the existing app server instance every 15 minutes*

Table 1: Proposed Taxonomy for Self-Protection Approaches, with Examples.

parameterization, orchestration, or aspect-orientation to dynamically change the behavior of a software component at runtime. The E2R Autonomic Security Framework [18], for example, allows each node to collect and derive security context information from neighboring nodes. When a node fails, other nodes can use their security contexts and pre-defined policies to reorganize the network and prevent service disruption. The approach is especially suited for protecting pervasive distributed systems, such as wireless sensor networks.

- Rejuvenation – periodically regenerates or recycles system to valid state. The R-Xen framework [21], for example, used hypervisor-based software rejuvenation techniques to proactively regenerate new Virtual Machines (VM) that can seamlessly take over control from potentially compromised VMs.

These patterns are not mutually exclusive. It is conceivable that a system may use a combination of them to provide more robust and flexible self-protection behavior.

V. Applying the Proposed Taxonomy

A 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 results are summarized in Table 2.

Note that the survey is meant to be representative not exhaustive, and the check marks in the table are meant to indicate the *primary* focus of the research effort. For example, if the "Availability" under Protection Goals is not checked for a certain research effort, it does not necessarily mean that this approach cannot help address availability issues. Rather, it simply means availability is not its primary focus.

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 following observations section.

VI. Observations and Analysis

By using the proposed taxonomy as a consistent point of reference, many insightful observations may be drawn from the survey results of Table 2.

1) Correlating Self-Protection Levels and Architecture Levels

Here 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, reference [38] uses Event Calculus to specify security monitoring patterns for detecting breaches in confidentiality, integrity and availability, respectively. Reference [20] uses policy-aware OS kernels that can dynamically change device protection levels. At the "Analyze & Characterize" level, research efforts attempt to characterize and understand the nature of security events. For example, reference [22] uses forensic analysis of victim server's memory to generate attack message signatures. 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. Techniques and approaches vary greatly, from fuzzy reasoning [46] for predicting network intrusions to architecture regeneration based on Quality of Service scenarios [26].

Along the Architecture Levels dimension, we see many adaptive security approaches focusing on the "traditional" architecture layers, such as network, host, and application code. At the network level, abundant research can be found in the field of intrusion-detection and intrusion-prevention, as represented by [46] and [11]. Because network vulnerabilities are closely linked to the network topology and equipment configurations, devoted research can also be found on adapting network security policies based on such network characteristics [5]. 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 [4]). As we move up to the application level, adaptive security research is more concerned with programming language level vulnerabilities such as those concerning pointers, memory buffers, and program execution points. Reference [23], for example, presents a technique, called From Failures to Vaccine (FFTV), that detects faults using code-level assertions and analyzes the application to identify relevant programming points that can mitigate the failures.

More recent research has started to focus on adaptive security for web services in a SOA. Such research is generally around service trust [27], service-level monitoring [39], and service middleware [1]. Research around the security behavior of a collection of services (such as a BPEL orchestration or a composite service), however, seems to be lacking.

Even less research seems to be focusing on the adaptation of the abstract software architecture as a whole, let alone from the adaptive security perspective. The RAINBOW [12] and SASSY [26] frameworks are two examples that fit into this category, even though they are not specifically focused on self-protection alone.

To take a further look at the research trends, we use Self-Protection Levels and Architecture Levels as two crosscutting dimensions to map out the existing adaptive security research approaches, as shown in Figure 2. In the plot, a dot represents a research effort presented in the referenced paper. A cloud represents a cluster of rich research that exists but not covered in detail here in this paper. From this plot it becomes more apparent that existing research starts to "thin out" as we move up the two respective levels. Autonomic and adaptive security approaches that apply to service-based architectures or abstract architectures seem like a research gap to be filled.

2) Lifecycle Focus

As expected, a vast majority of self-protection research focuses on runtime not development time, as it is generally assumed that software components will never be completely fault-free and vulnerability-free.

Still, a few research efforts can be found to also involve development time activities. The FFTV approach [23], 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 [15], the dynamically reconfigurable security policies for mobile Java programs also rely on supporting mechanisms put in at development time (such as policy class loaders).

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 feel that combining development-time and run-time approaches will result in the best self-protection of software systems – another research opportunity.

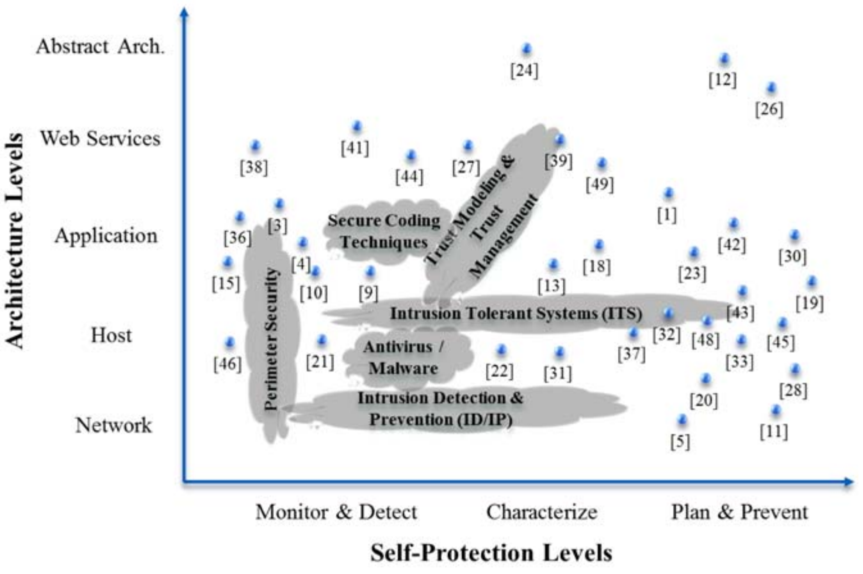

Figure 2: Correlating Self-Protection Levels with Architecture Levels.

3) *Balancing the Protection Goals*

Along the Protection Goals dimension (see Table 2), the survey results revealed that research efforts seem to focus on either Confidentiality+Integrity or Availability+Integrity, but not 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 example, when a network intrusion is detected, reconfiguration of the network settings (such as VPN settings [11]) oftentimes involves cutting off the affected host/node – system confidentiality and integrity are preserved, whereas availability suffers.

In fact, preserving system availability goes beyond the security realm and is closely related to system QoS, thus requiring different treatments. Intrusion Tolerant Systems (e.g., [37], [42], [43]), for example, addresses 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 self-protecting 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.

4) *Topology Tradeoffs*

Survey results along the Adaptation Topology metric (see Table 2) clearly shows that adaptive security approaches functioning at the global level are predominantly centralized. For example, many research efforts (e.g., [19] and [24]) recognize the need for coordination between local and global security policies. In most cases, the coordination is through a central controller (e.g., [28]). One of the few exceptions seems to be the Malicious-and Accidental-Fault Tolerance for Internet Applications (MAFTIA) effort [23], which uses local middleware controllers (called "wormholes") at each node that are interconnected, but do not require a central controller.

A central controller makes coordination and global optimization easier, yet runs the risk of becoming the single point of failure of the system, prone to denial of service and subversion attacks. Therefore, some approaches put more robust protection around the central controller, such as using hardened and trusted hardware/software [42], or putting the controller in dedicated network zones [31].

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. Decentralized security approaches in such case hold more promise in their resilience and scalability, and may need more research attention.

5) *Basis of Adaptation Decision-Making*

Security threats are diverse and often unpredictable; wrong decisions usually will lead to severe consequences. As such, few approaches in this survey leave adaptive security decisions (such as reconfiguration or conflict resolution) solely to an algorithm. Instead, most approaches use heuristic rules/policies, in such forms as expert system rule sets [4], policy specification languages [5], event-condition-action rules [10], or human input as a last resort [45].

The limited few algorithm-based approaches are only used against narrowly focused problems, such as anomaly detection (e.g., using security automata [9]), event pattern recognition (e.g., using event calculus [38]), or fault isolation (e.g., using Byzantine agreement or threshold cryptography [33] [37]). The algorithms are often used in conjunction with heuristic policies.

The lack of algorithm-based methods may be explained by the daunting challenge of quantitatively assessing the overall security posture of a complex software system.

Reference [36] proposes the concept of a Security Health Index comprised of a weighted basket of security metrics as an attempt at this goal, but it is not clear whether the approach has been empirically validated. This is definitely a pressing research need, especially in today's heated domain of cyber warfare.

6) Proactive Defense

From Table 2, we can see that survey results along the Adaptation Tempo dimension indicate *reactive* adaptation based on the "sense and respond" paradigm still seems to be the norm for self-protection. That being said, the survey results also show an interesting trend that proactive security architectures are gaining ground. The TALENT system [30], for example, addresses software security and survivability using a "cyber moving target" approach, which periodically migrates running applications across different platforms while preserving application state. The SCIT system [28] uses redundant and diverse servers to periodically "self-cleanse" the system to pristine state. The aforementioned R-Xen framework [21] proactively instantiate new VM instances to ensure system reliability, a technique much faster than rebooting hardware servers thanks to hypervisor-based virtualization technology [21].

Table 2: Applying the Proposed Taxonomy to Self-Protection Research Approaches.

Source (see References)	**Solution Name**	**Research Positioning ("What")**													**Technique Characterization ("How")**																		
		Self-Protection Levels			Architecture Levels					Life Cycle Focus		Protection Goals			Adapt. Topology			Adaptation Decision-making				Adapt. Tempo			Adapt. Point			Adaptation Patterns					
		Monitor & Detect	Analyze & Characterize	Plan & Adapt	Network	Host	App / Software	Services	Abstract Architecture	Development Time	Runtime	Confidentiality	Integrity	Availability	Local Only	Centralized	Decentralized	Human-Driven	Heuristics-Driven	Algorithm-Driven	Other	Reactive	Proactive	Neutral / Hybrid	System Boundary	System Internals	Neutral / Hybrid	Proxy/Containment	Redundancy	Diversity	Recomposition	Rejuvenation	Other
[1]	GEMOM	x	x	x		x	x			x	x	x	x	x		x			x	x		x					x		x		x		
[4]	LCS	x					x				x		x		x				x					x			x						
[5]		x	x		x						x	x	x			x			x			x			x						x		
[9]	SASI	x				x	x				x		x		x					x		x				x		x					
[10]	SECURE	x				x	x				x	x	x			x			x					x			x						
[11]	IDIAN	x	x	x	x						x	x	x			x		x	x			x			x						x		
[12]	RAINBOW	x	x	x					x		x	x	x	x		x		x	x			x					x				x		
[13]		x	x		x	x	x				x	x	x			x			x			x					x						
[15]		x					x			x	x	x	x		x				x					x		x		x			x		
[18]	E2R	x	x		x	x	x				x	x	x				x		x			x			x						x		
[19]	ASPF	x	x	x			x				x	x	x	x		x	x		x			x					x				x		
[20]	VSK	x	x	x		x					x	x	x		x				x			x				x					x		
[21]	X-Spy, R-Xen	x				x					x	x	x	x	x				x			x			x			x				x	
[22]	COVERS	x				x				x	x	x	x			x				x				x	x								
[23]	FFTV	x	x	x			x			x	x		x		x			x	x			x					x	x			x		
[24]	ASI		x						x		x	x	x	x		x	x		x					x			x						
[26]	SASSY	x	x	x			x	x	x	x	x	x	x	x		x			x			x					x				x		
[27]	WSAF	x	x					x						x		x			x	x				x		x					x		
[28]	SCIT	x	x	x		x					x		x	x		x			x	x			x		x					x		x	
[30]	TALENT			x			x			x	x		x	x	x			x	x				x				x		x	x			
[31]	DPASA	x	x		x	x					x		x	x		x			x			x			x			x	x	x			
[32]	HACQIT	x	x	x	x	x					x		x	x		x			x			x			x			x	x				
[33]	VM-FIT	x	x	x		x					x			x	x				x	x			x		x			x	x			x	
[36]	GEMOM	x			x	x	x				x	x	x			x			x	x				x			x						
[37]	PRM	x	x	x	x	x					x		x	x					x	x		x	x				x		x			x	
[38]		x						x			x	x	x	x		x				x				x			x						
[39]		x	x					x			x	x	x	x		x				x				x		x							
[42]	MAFTIA	x	x	x		x	x				x		x	x			x		x	x		x					x		x	x			
[43]	SITAR	x	x	x		x	x			x	x		x	x		x			x	x		x			x			x	x				
[45]		x	x	x		x	x			x	x	x	x			x		x	x			x					x	x					
[46]	ADAT	x			x	x					x	x	x			x				x		x			x								
[48]	WILLOW	x	x	x	x	x					x		x	x		x			x			x					x				x		

7) *From Perimeter Security to Overall Protection*

The Adaptation Point dimension of Table 2 shows that many adaptive security approaches still rely on perimeter security, especially those that focus on intrusion detection and intrusion tolerance. 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 counter this, some approaches follow the “defense-in-depth” principle and establish multiple layers of perimeters or security zones [31], 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. Recent research on service-based systems, for instance, has started to focus on monitoring and analyzing service interaction patterns [39] [41].

8) *Determining Adaptation Patterns*

Another revealing insight from the survey results is that adaptation patterns are often determined by, or strongly correlated with, the other dimensions in the taxonomy. Their relationship is briefly described in Table 3.

It is perhaps not entirely a surprise 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 as requirements and environments change. This is a higher level of self-protection and may only be enabled through an architecture-based approach.

Table 3: Adaptation Patterns Influenced by Other Taxonomy Dimensions.

Dimension	Correlation with Adaptation Patterns
Self-Protection Level	While all patterns are effective for system adaptation, approaches that offer monitoring and detection often uses the Proxy / Containment pattern
Architecture Level	Redundancy, diversity, and rejuvenation are mostly used at the host-level, whereas recomposition is more common at the application and service levels
Lifecycle Focus	All patterns are for runtime security adaptation. More sophisticated patterns such as diversity, recomposition and rejuvenation, create significant testing challenges during the development and deployment process
Protection Goals	Proxy/containment is an effective pattern to address system confidentiality; redundancy, diversity and rejuvenation are particularly effective in ensuring system availability
Adaptation Decision-Making	Algorithms usually go hand-in-hand with the applied pattern(s). For example, voting/Byzantine agreements are usually used in conjunction with redundancy patterns, whereas learning classifiers are used in conjunction with proxies
Adaptation Tempo	Rejuvenation is usually the primary pattern for proactive adaptation
Adaptation Point	Proxy/Containment and Redundancy patterns are usually applied at system boundaries, whereas the rest of the patterns are concerned with system internals

VII. 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 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 autonomic and adaptive security, but gaps and challenges remain. This paper proposes a comprehensive taxonomy to classify and characterize research efforts in this arena. The analysis of past and ongoing research efforts using this taxonomy has revealed some gaps and needs for future research. Specifically, to stay ahead of today’s advancing cyber threats, adaptive security research needs to:

- Concurrently advance (a) from monitoring and analysis to planning and goal management, and (b) from network, host, and application levels to service-based and holistic architecture-based approaches
- Pursue more “integrated” approaches that span both development-time and runtime
- Explore more decentralized coordination, planning, and optimization approaches
- Explore qualitative and quantitative measures that can be used to dynamically assess overall system security posture
- Continue the paradigm shift from perimeter security to overall system protection and monitoring
- Catalog and automate security adaptation patterns at the abstract architecture level

ACKNOLWEDGEMENTS

This work is partially supported by grants CCF-0820060 and CCF-1217503 from the National Science Foundation and grant N11AP20025 from Defense Advanced Research Projects Agency.

REFERENCES

[1] H. Abie, et al., "GEMOM - Significant and Measurable Progress beyond the State of the Art," International Conference on Systems and Networks Communications, 2008. pp.191-196, Oct. 2008

[2] A. Avižienis, et al., “Dependability and its Threats: A Taxonomy,” proc. IFIP 18th World Computer Congress. 22-27 August 2004

[3] L. Blasi, et al., “Applicability of security metrics for adaptive security management in a universal banking hub system,” European Conference on Software Architecture (ECSA '10), pp.197-204, 2010

[4] J. Blount, D. Tauritz, S. Mulder, "Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept," proc. IEEE 35th Annual Computer Software and Applications Conference Workshops, pp.110-115, July 2011

[5] J. Burns, A. Cheng, et al., "Automatic management of network security policy," DARPA Information Survivability Conference & Exposition II, 2001. pp.12-26 vol.2, 2001

[6] B. Cheng, et al., “Using Security Patterns to Model and Analyze Security Requirements,” High Assurance Systems Workshop (RHAS ’03) of International Conference on Requirements Engineering, 2003.

[7] D. Chess, C. Palmer, S. White, "Security in an autonomic computing environment," IBM Systems Journal , vol.42, no.1, pp.107-118, 2003

[8] A. Elkhodary, J. Whittle, "A Survey of Approaches to Adaptive Application Security," Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp.16, May 2007

[9] U. Erlingsson, F. Schneider, "SASI enforcement of security policies: a retrospective," DARPA Information Survivability Conference and Exposition, 2000. vol.2, pp.287-295, 2000

[10] C. English, S. Terzis, P. Nixon, "Towards Self-Protecting Ubiquitous Systems Monitoring Trust-based Interactions," in UbiSys '04, 2004

[11] R. Feiertag, et al., "Intrusion Detection inter-component adaptive negotiation," Computer Networks, vol. 34, pp.605-621, 2000

[12] S. Cheng, et al., "Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure," IEEE Computer, vol 37, pages 46-54,, 2004

[13] M. Ganna, E. Horlait, "Toward secure autonomic pervasive environments," proc. IEEE GLOBECOM '05. vol.2, pp.6, Nov 2005

[14] M. Hafiz, P. Adamczyk, R. Johnson, "Organizing Security Patterns," in IEEE Software, vol.24, no.4, pp.52-60, July-Aug. 2007

[15] B. Hashii, et al., "Supporting reconfigurable security policies for mobile programs", Computer Networks, vol. 33, pp.77-93, 2000

[16] J. Kephart, D. Chess, "The vision of autonomic computing," in IEEE Computer, vol.36, no.1, pp. 41- 50, Jan 2003

[17] J. Kramer, J. Magee, "Self-Managed Systems: an Architectural Challenge," International Conf. on Software Engineering, pp.259-268, May 2007

[18] Ruan He, M. Lacoste, "Applying component-based design to self-protection of ubiquitous systems," ACM Workshop on Software Engineering for Pervasive Services, pp.9-14, 2008

[19] R. He, et al., "A Policy Management Framework for Self-Protection of Pervasive Systems," International Conference on Autonomic and Autonomous Systems, pp.104-109, March 2010

[20] R. He, et al., "Virtual Security Kernel: A Component-Based OS Architecture for Self-Protection," International Conference on Computer and Information Technology, pp.851-858, June 2010

[21] B. Jansen, H. Ramasay, et al, "Architecting Dependable and Secure Systems Using Virtualization", Architecting Dependable Systems V, LNCS 5135, pp. 124-149, 2008

[22] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: a basis for building self-protecting servers," ACM conf. on Computer and communications security, pp. 213-222, 2005

[23] D. Lorenzoli, L. Mariani, M. Pezze, "Towards Self-Protecting Enterprise Applications," IEEE Int'l Symposium on Software Reliability (ISSRE '07), pp.39-48, Nov. 2007

[24] L. Marcus, "Local and Global Requirements in an Adaptive Security Infrastructure," International Workshop on Requirements for High Assurance Systems, Sep 2003

[25] P. McKinley, S. Sadjadi, E. Kasten, B. Cheng, "A taxonomy of compositional adaptation", http://users.cs.fiu.edu/~sadjadi/Publications/CompositionalAdaptationTaxonomy-TechRep.pdf, May 2004

[26] D. Menasce, H. Gomaa, S. Malek, J. Sousa, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," IEEE Software, vol.28, no.6, pp.78-85, Nov.-Dec. 2011

[27] E. Maximilien, M. Singh, "Toward autonomic web services trust and selection," Int'l Conf. on Service Oriented Computing, pp.212-221, 2004

[28] Q. Nguyen, A. Sood, "Designing SCIT architecture pattern in a Cloud-based environment,", International Conference on Dependable Systems and Networks Workshops (DSN-W), pp.123-128, June 2011

[29] Q. Nguyen, A. Sood, "A Comparison of Intrusion-Tolerant System Architectures," IEEE Security & Privacy, vol.9, no.4, pp.24-31, July-Aug. 2011

[30] H. Okhravi, et al., "TALENT: Dynamic Platform Heterogeneity for Cyber Survivability of Mission Critical Applications", Secure and Resilient Cyber Architecture Conference (SRCA'10), Oct 2010

[31] P. Pal, F. Webber, R. Schantz,"The DPASA Survivable JBI - A High-Water Mark in Intrusion Tolerant Systems", First Workshop on Recent Advances on Intrusion-Tolerant Systems, pp.33-37, 2007

[32] J. Reynolds, J. Just, et al, "The design and implementation of an intrusion tolerant system," Foundations of Intrusion Tolerant Systems, 2003

[33] H. Reiser, R. Kapitza, "Hypervisor-Based Efficient Proactive Recovery," IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), pp. 83–92, 2007

[34] S. Sadjadi, "A Survey of Adaptive Middleware", http://users.cis.fiu.edu/~sadjadi/Publications/AdaptiveMiddlewareSurvey.pdf, Dec 2003

[35] M. Salehie, L. Tahvildari, "Self-adaptive software: Landscape and research challenges," in ACM Trans. on Autonomic and Adaptive Systems (TAAS), vol.4, no.2, Article 14, May 2009

[36] R. Savola, P. Heinonen, "Security-Measurability-Enhancing Mechanisms for a Distributed Adaptive Security Monitoring System," Int'l Conf. on Emerging Security Information Systems and Technologies (SECURWARE), pp.25-34, 2010

[37] P. Sousa, et al., "Resilient Intrusion Tolerance through Proactive and Reactive Recovery," Pacific Rim International Symposium on Dependable Computing (PRDC 2007), pp.373-380, Dec. 2007

[38] G. Spanoudakis, et al., "Towards security monitoring patterns," ACM symposium on Applied computing (SAC '07), pp.1518-1525

[39] G. Spanoudakis, S. LoPresti, "Web Service Trust: Towards a Dynamic Assessment Framework," International Conference on Availability, Reliability and Security (ARES '09), pp.33-40, 2009

[40] F. Swiderski, W. Snyder, "Threat Modeling," Microsoft Press, 2004.

[41] M. Uddin, M. Zulkernine, "ATM: an automatic trust monitoring algorithm for service software," ACM symposium on Applied Computing (SAC '09), pp.1040-1044

[42] P. Verissimo, et al., "Intrusion-tolerant middleware: the road to automatic security," IEEE Security & Privacy, vol.4, no.4, pp.54-62, July-Aug. 2006

[43] F. Wang, F. Jou, F. Gong, et al., "SITAR: a scalable intrusion-tolerant architecture for distributed services," Foundations of Intrusion Tolerant Systems, 2003

[44] Y. Wang, M. Singh, "Evidence-based trust: A mathematical model geared for multiagent systems," ACM Trans. Auton. Adapt. Syst. 5, 4, Article 14, Nov. 2010

[45] S. White, M. Swimmer, et al., "Anatomy of a Commercial-Grade Immune System," International Virus Bulletin Conference, 1999

[46] Z. Yu, et al., "An adaptive automatically tuning intrusion detection system," ACM Trans. Autonom. Adapt. Syst. Vol 3, No 3, Aug.2008

[47] N. Yoshioka, H. Washizaki, K. Maruyama, "A survey on security patterns," Progress in Informatics, 5:35–47, 2008

[48] J. Knight, D. Heimbigner, A. Wolf, "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications," Intrusion Tolerance Systems Workshop of Int'l Conf. on Dependable Systems and Networks, pp.C.7.1–C.7.8, 2002

[49] E. Yuan, J. Tong, "Attributed Based Access Control (ABAC) for Web services," IEEE Int'l Conf. on Web Services (ICWS), July 2005

[50] Aspect Oriented Programming (AOP), http://en.wikipedia.org/wiki/Aspect-oriented_programming

[51] The CIA Triad for Information Security, http://en.wikipedia.org/wiki/Information_security

[52] Conficker worm, http://en.wikipedia.org/wiki/Conficker

[53] The MITRE Corporation, The 2011 CWE/SANS Top 25 Most Dangerous Software Errors, http://cwe.mitre.org/top25/

[54] Duqu computer worm, http://en.wikipedia.org/wiki/Duqu

[55] Stuxnet, http://en.wikipedia.org/wiki/Stuxnet

[56] Service Component Architecture (SCA) specifications, http://www.oasis-opencsa.org/sca

[57] Zachman's Architecture Framework, http://www.zachman.com/