Intrusion Response Systems: A Survey

Bingrui Foo, Matthew W. Glause, Gaspar M. Howard, Yu-Sung Wu, Saurabh Bagchi, Eugene H. Spafford [+]

Center for Education and Research in Information Assurance and Security (CERIAS)

Dependable Computing Systems Laboratory
School of Electrical and Computer Engineering
Purdue University

(+) School of Computer Science
Purdue University

E mail: *(foob, mglause, gmodeloh, yswu, sbagchi, spaf)@purdue.edu*[1]

Contact author: Saurabh Bagchi

Abstract

Protecting networks from computer security attacks is an important concern of computer security. Within this, intrusion prevention and intrusion detection systems have been the subject of much study and have been covered in several excellent survey papers. However, the actions that need to follow the steps of prevention and detection, namely response, have received less attention from researchers or practitioners. It was traditionally thought of as an offline process, with humans in the loop, such as system administrators performing forensics by going through the system logs and determining which services or components need to be recovered. Our systems today have reached a level of complexity and the attacks directed at them a level of sophistication that manual responses are no longer adequate. So far there has been limited work in autonomous intrusion response systems, especially work that provides rigorous analysis or generalizable system building techniques. The work that exists has not been surveyed previously. In this survey paper, we lay out the design challenges in building autonomous intrusion response systems. Then we provide a classification of existing work on the topic into four categories – response through static decision tables, response through dynamic decision process, intrusion tolerance through diverse replicas, and intrusion response for specific classes of attacks. We

[1] Student authors (the first four authors) are listed in alphabetical order.

describe the existing intrusion response systems after classifying them. We present methods for benchmarking the intrusion response systems. We end with ideas for further work in the field.

1 Introduction

The occurrence of outages due to failures in today's information technology infrastructure is a real problem that still begs a satisfactory solution. The backbone of the ubiquitous information technology infrastructure is formed by distributed systems – distributed middleware, such as CORBA and DCOM, distributed file systems, such as NFS and XFS, distributed coordination based systems, such as publish-subscribe systems, network protocols, and above all, the distributed infrastructure of the world wide web. They support many critical applications in the civilian and in the military domains. Critical civilian applications abound in the private enterprise, such as banking, electronic commerce, industrial control systems, as well as in the public enterprise, such as air traffic control, nuclear power plants, and protection of public infrastructures through SCADA systems. The dependency dramatically magnifies the consequence of failures, even if transient. Little wonder that distributed systems are called upon to provide always-available and trustworthy services. The terminology that we will use in this paper is to consider the distributed systems as composed of multiple services and the services interact with one another through standardized network protocols. Consider for example a distributed e-commerce system with the traditional three tier architecture of a web server, an application server, and a database server. The services are typically located on multiple hosts.

The importance of distributed systems has led to a long interest in securing such systems through prevention and runtime detection of intrusions. The prevention is traditionally achieved by a system for user authentication and identification (e.g., users log in by providing some identifying information such as login and password, biometric information, or smart card),

access control mechanisms (rules to indicate which user has what privileges over what resources in the system), and building a “protective shield” around the computer system (typically a firewall that inspects incoming and optionally outgoing network traffic and allows it if the traffic is determined to be benign). The prevention mechanism by itself is considered inadequate because without being too restrictive, it is impossible to block out all malicious traffic from the outside. Also, if a legitimate user’s password is compromised or an insider launches an attack, then prevention may not be adequate.

Intrusion detection systems seek to detect the behavior of an adversary by observing its manifestations on the system. The detection is done at runtime when the attack has been launched. There are many intrusion detection systems (IDSs) that have been developed in research and as commercial products. They fundamentally operate by analyzing the signatures of incoming packets and either matching them against known attack patterns (misuse based signatures), or against patterns of expected system behavior (anomaly based signatures). The two metrics for evaluating IDSs are rate of false alarms (legitimate traffic being flagged as malicious) and rate of missed alarms (malicious traffic not flagged by the IDS).

However, in order to meet the challenges of continuously available trustworthy services from today’s distributed systems, intrusion detection needs to be followed by response actions. This has typically been considered the domain of system administrators who manually "patch" the system in response to detected attacks. The traditional mode of performing response was the system administrator would get an alert from the IDS. She would then consult logs and run various system commands on the different machines comprising the entire system in an effort to determine if the attack is currently active and what damage has been caused by it. There were several sophisticated but ad hoc tools that system administrator would execute to aid in the

determination process, such as, a script to log into all the machines in the system to determine if .rhosts files have been tampered with and if so, set them to some overly restrictive privileges. Clearly, this process, still the dominant method today, is ad hoc and very human intensive. By the nature of the actions, this process cannot reasonably hope to respond in real time and is therefore considered offline. However, as distributed systems become larger, more complex and ubiquitous, the number of users increases, and sophisticated automated script-based attacks gain ground, automated tools for intrusion response become vitally important[2].

The autonomous intrusion response systems (IRSs) are designed to respond at runtime to the attack in progress. The goals of an IRS may be a combination of the following – to contain the effect of the current attack if the underlying model is that it is a multi-stage attack, to recover the affected services, and to take longer term actions of reconfiguration of the system to make future attacks of a similar kind less likely to succeed. There are several challenges in the design of an IRS. First, the attacks through automated scripts are fast moving through the different services in the system. Second, the nature of the distributed applications enables the spread of the attack, since under normal behavior the services have interactions among them and a compromised service can infect another. Third, the owner of the distributed system does not have knowledge of or access to the internals of the different services. For example, the source code may not be available or even if available, the expertise to understand the internals may not be available. Hence, an IRS should ideally work at the interfaces rather than in the internals. Fourth, it may not be possible to deploy detectors at each service for performance reasons (say, the performance overhead imposed by the packet matching at a network-based detector is excessive for a host) or

[2] We will discuss response systems in the context of distributed systems since their application to standalone systems is simpler and does not have many of the challenges that makes this topic intellectually challenging. To distinguish between the system that the IRS protects from the IRS itself, we will call the former the *payload system*.

deployment conditions (say, no host-based detector is available for the particular platform). Additionally, the detectors, if installed, may be faulty and produce false alarms or missed alarms. The IRS therefore has to suppress inaccurate detections and extrapolate from the available detectors to determine the appropriate services at which to take the response action. Finally, the distributed systems are often complex enough that the universe of attacks possible against such systems is not enumerable and therefore the IRS has to work with possibly unanticipated attacks.

The current IRSs meet only a subset of the above challenges, and none that we are aware of addresses all of them. The general principles followed in the development of the IRS naturally classify them into four categories.

1. *Static decision making*. This class of IRS provides a static mapping of the alert from the detector to the response that is to be deployed. The IRS includes basically a look-up table where the administrator has anticipated all alerts possible in the system and an expert indicated responses to take for each. In some cases, the response site is the same as the site from which the alarm was flagged, as with the responses often bundled with anti-virus products (disallow access to the file that was detected to be infected) or network-based IDS (terminate a network connection which matched a signature for anomalous behavior). The systems presented in [1-4] fall in this category.
2. *Dynamic decision making*. This class of IRS reasons about an ongoing attack based on the observed alerts and determines an appropriate response to take. The first step in the reasoning process is to determine which services in the system are likely affected, taking into account the characteristics of the detector, the network topology, etc. The actual choice of the response is then taken dependent on a host of factors, such as, the amount of evidence about the attack, the severity of the response, etc. The third step is to determine the effectiveness of

the deployed response to decide if further responses are required for the current attack or to modify the measure of effectiveness of the deployed response to guide future choices. Not all IRSs in this class include all the three steps. A wide variety is discernible in this class based on the sophistication of the algorithms. The systems presented in [5-13] fall in this category.

3. *Intrusion tolerance through diverse replicas*. This class of IRS implicitly provides the response to an attack by masking the effect of the response and allowing the computer system to continue uninterrupted operation. The basic approach is to employ a diverse set of replicas to implement any given service. The fault model is the replicas are unlikely to share the same vulnerabilities and therefore not all will be compromised by any given attack. A voting process on the outputs or the state of the replicas can mask the compromised replicas provided less than half are compromised. An advantage of this approach is the system can continue operation without a disruption. This approach is reminiscent of active replication in the fault tolerance field. The systems presented in [14-18] fall in this category.
4. *Responses to specific kinds of attacks*. This class of IRS is customized to respond to specific kinds of attacks, most commonly, distributed denial of service (DDoS) attacks. The approach is to trace back as close to the source of the attack as possible and then limit the amount of resources available to the potentially adversarial network flows. A characteristic of this approach is cooperation is required from entities outside the computer system being protected for an accurate trace back. The systems reported in [19-21] fall in this category.

In this paper, we will describe the primary IRSs that have been reported in the literature and label each in one of these four categories.

Next, we consider the metrics that are relevant for evaluating an IRS. Both low-level metrics and high-level metrics need to be considered for this purpose. Low level metrics are those that

look at specific activities within an IRS, such as, the latency in deploying a response, and the fraction of times a given response is successful in terminating an attack. However, these metrics need to be combined in application and domain specific ways to evaluate the impact of the IRS on the distributed system being protected. An example of the high level metric is the net value of transactions that could not be completed in an e-commerce system due to an attack, when an IRS is deployed. We provide a scientific basis for the high level metrics in this paper.

The rest of the paper is organized as follows. Sections 2-5 present the IRSs that belong to each class introduced above. Section 6 presents a discussion of metrics used to evaluate an IRS and gives an example with the ADEPTS IRS applied to protect a distributed e-commerce system. Section 7 describes our thoughts for future evolution of the work on IRSs. Section 8 concludes the paper.

2 Static Decision Making Systems

The characteristic that defines this class of IRSs is that they respond to attacks defined exactly, prior to deployment and using responses that are enumerated and completely configured. They are in general simple to understand and deploy and work well for a large class of systems that have determinism in the kinds of workload and where the attack modes are enumerable a priori. However, they are not very effective for dynamic systems with changing workloads, new kinds of services installed and new vulnerabilities introduced due to hardware or software changes.

2.1 Generic Authorization and Access Control API (GAA-API)

2.1.1 Introduction

The Generic Authorization and Access Control - Application Programming Interface (GAA-API) [3] is a signature-based intrusion detection and response system that provides a dynamic authorization mechanism at the application layer of a computer system. The basic idea is integrate access control policy with intrusion detection and some countermeasure according to policy, such as generating audit records. GAA-API, developed by the Information Sciences Institute supports access control policies and

conditions defined by a BNF-syntax language. It is a generic tool that has been integrated with many applications, including Apache, SSH, SOCKS5, and FreeS/WAN (IPSec VPN), running on Linux and Sun Solaris platforms. It is designed as a generic interface based on standard C language APIs, so it can be easily ported to other platforms and applications.

2.1.2 Details

GAA-API extends the access control capabilities from an application, while providing the opportunity to identify application-level attacks and specify different types of real-time responses to intrusions (Figure 1). A key component of the API is its Extended Access Control List (EACL) language, which allows the formulation of policies for the API to evaluate, decide, and respond to possible attack scenarios. Each object in the application is associated with an EACL, where the access rights are defined along with a set of conditions necessary for the rights to be matched. The conditions can state the requirements necessary to grant or deny access (pre-conditions), determine what to do when an access request arrives (request-result), what must hold while the access is granted (mid-conditions), and what to do after the access ends (post-conditions).

The conditions allow for the API to interact with intrusion detection systems, to modify existing policy rules, and to trigger a response. As an example of the interaction with an IDS, the API can report attack information such as violation of threshold conditions and access requests with parameters that do not comply with a site's policy. The API can also request an IDS for network-based attack information, such as spoofed addresses. The API can deploy responses according to the conditions previously defined. The API might, for example, limit the consumption of resources, increase the auditing level, or request user authentication to access a certain application. Nevertheless, it is unclear as to which type of language or protocol is used for the framework to exchange messages with an IDS.

GAA-API defines two types of policies: system-wide policies, which can be applied to all the objects in an application; and local policies that are selectively applied to individual objects. The final policy application to an object, for which both system-wide and local policies exist, depends on the composition mode selected. There are three alternatives: expand, which provides access to an object if either system or

local policy allows it; narrow, where mandatory access control rules defined by system-wide policy overrule any discretionary rule defined at the local policy level; and stop, where local policies are ignored if a corresponding system-wide policy exists.

The policies defined and implemented allow for the GAA-API framework to also interact with system administrators. An administrator can receive messages and validate the impact and effectiveness of the response actions taken by the framework. An example would be a rule defined for an Apache web server that states updating the list of malicious IP addresses after a potential attack is detected and sending an e-mail with the IP address of the potential attacker, the URL attempted, and the reported time of the attack. The administrator would later validate the effectiveness of the response.

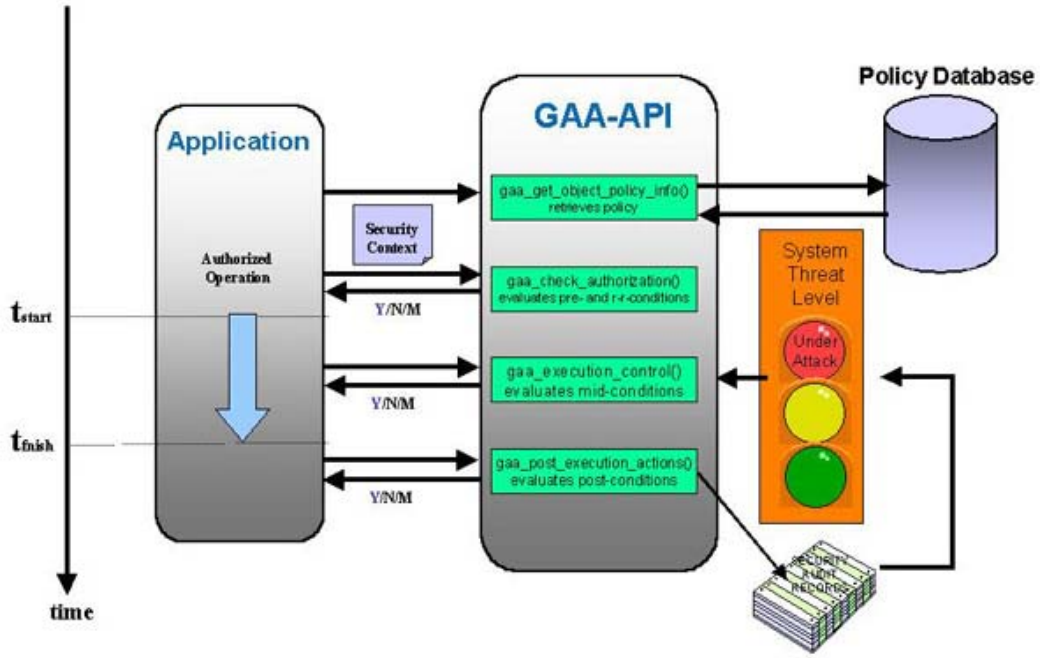

Figure 1. Application interacting through the GAA-API to enforce policies at different stages of interaction (pre-conditions, request-result, mid-conditions, and post-conditions). The policy in effect is dependent on the threat level as communicated by the IDS. [22]

The authors report that GAA-API functions introduce a 30% overhead for an Apache Web Server function call, when email notification to administrators was disabled. If the email notification is enabled, the overhead rises to 80%.

2.1.3 Significance

The authors present an extended approach to the regular access control model found in popular Unix-based applications. The access control policies interact with other important security mechanisms such as IDS and firewalls, allowing for a richer set of potential responses in the presence of attacks. More recently, the authors further developed the concepts presented in GAA-API with the introduction of dynamic detection and response mechanisms during the trust negotiation phase between two parties, usually client and server, and the support it can provide for stronger access control. A potential drawback to this model could be the complexity introduced by such policies, with many variables and the interaction among them, making it hard to administer in a large environment.

2.2 Snort Inline

2.2.1 Introduction

Snort Inline is a mode of operation for Snort, the popular open-source intrusion detection system. Originally developed as an independent, modified version of Snort, it was integrated in version 2.3.0 RC1 of the Snort project to provide intrusion prevention capabilities. It requires the Netfilters/IPtables software developed by the same project. Snort Inline provides detection at the application layer to the IPtables firewall so it can respond dynamically to real time attacks that take advantage of vulnerabilities at the application level.

2.2.2 Details

Snort Inline is the intrusion prevention component of Snort, a popular network intrusion detection and prevention system capable of real-time IP network traffic analysis. Snort was originally developed by Martin Roesch and is currently owned and developed by Sourcefire, a company founded by Roesch. Snort Inline started as a separate project that used Snort for its packet logging and traffic analysis capabilities but has since been included in the Snort distribution, providing the intrusion response capabilities that the popular IDS had hitherto lacked.

The Netfilter/IPtables software allows for the implementation of the response mechanism while Snort Inline provides the policies based on which IPtables makes the decision to allow or to deny packets. After

an incoming packet to the network is provided by IPtables, Snort performs the rule matching against the packet. There are three new rule types, included in Snort, for Snort Inline to define the actions that IPtables might take after receiving an incoming packet. All three rule types drop the packet, if it matches a predefined rule. The second type of rule also logs the packet and the third type sends a control message back. The rules are applied before any alert or log rule is applied. The current version of Snort, also allows the system to replace sections of a packet payload when using Snort Inline. The only limitation is that the payload selected must be replaced by a string of the same length. For example, an adversary that is looking to propagate malicious code through the PUT command could have it replaced by the TRACE command, thus halting further propagation of the code.

In order for Snort Inline to interface with IPtables, two C libraries are needed: libipq and libnet. Libipq [23] is a library for IPtables packet queuing that allows Snort Inline to exchange messages with IPtables. LibNet is the popular networking interface to construct, handle, and inject packets into a network.

2.2.3 Significance

The inclusion of Snort Inline to the popular Snort project is a good example of the evolution of intrusion detection systems, as more proactive, dynamic capabilities are necessary to assist systems against today's attacks. However, the rule matching is against a statically created rule base and thus needs prior estimate of the kinds of attacks that will be seen and the action is taken at the site of detection.

2.3 McAfee Internet Security Suite (ISS)

2.3.1 Introduction

The McAfee Internet Security Suite (ISS) is a commercial product developed for the Windows operating system platform that integrates many security technologies to protect desktop computers from malicious code, spam and unwanted or unauthorized access. The suite also includes monitoring and logging capabilities as well as backup, file and printing sharing, privacy, spam filtering, and file wiping utilities. The interaction between several of these technologies allows for prevention, detection and response of various types of attacks, chief among them being attacks related to malicious code. However,

for this system, it is impossible to find detailed technical material while there is an overabundance of documents listing the features of the solution.

2.3.2 Details

The two main components of ISS are an anti-virus sub-system and a firewall sub-system. The anti-virus sub-system allows for detection of viruses, worms, and other types of malicious code by using a signature-based approach along with a heuristic engine for unknown attacks. The firewall sub-system can be configured to scan multiple points of data entry such as electronic email, storage devices, instant messaging, and web browser. An intrusion detection module allows the firewall to interact with the anti-virus, providing a limited set of automatic responses to ongoing attacks. Another component of the ISS that is relevant to intrusion response is a system monitor. The system monitor detects and blocks changes on important components of the operating system such as configuration files, browser settings, startup configuration, and active protocols and applications.

2.3.3 Significance

The evolution from an anti-virus product to an all-in-one security solution is a natural transformation that vendors such as McAfee and Symantec have experimented with in the last few years. The increase in complexity, speed and variety for malicious code, along with the requirement to respond to attacks in real time, have led these vendors to integrate multiple security mechanisms. The response mechanisms implemented are still static and limited but one could expect more dynamic responses in future versions of these suites..

2.4 Other Systems

2.4.1 McAfee IntruShield IPS

This forms part of the Network Intrusion Prevention product offering from McAfee. There is no technically rigorous publication describing the product. Our discussion is based on the documents put on the specific McAfee web page [24]. This system can be described as a network intrusion prevention system. It provides real-time prevention of encrypted attacks, while its ASIC-based architecture provides deep packet inspection, and shell-code detection leading to zero-day protection. It employs purpose-built

appliances, i.e., specialized hardware. The hardware is of different types depending on deployment—at the core of the network or the perimeter of the corporate network. It claims to prevent a wide variety of attacks, such as botnets, VoIP vulnerability based attacks, and encrypted attacks.

In terms of response, it provides hints for creating some offline response in the manner of forensics. It delivers unique forensic features to analyze key characteristics of known and zero-day threats and intrusions. IntruShield's forensic capabilities provide highly actionable and accurate information and reporting related to intrusion identification, relevancy, direction, impact, and analysis. There is a host based intrusion prevention system also from McAfee [25].

3 Dynamic Decision Making Systems

3.1 Broad Research Issues

Dynamic decision making based IRS involves the process of reasoning about an ongoing attack based on the observed alerts and determining an appropriate response to take. There have been various designs and architectures proposed for this kind of dynamic decision making based IRS systems. However, the core issue underlying all these systems is how the decision making should be achieved. Many factors can contribute to and complicate the decision making process. For instance, a response can come with a certain cost such as the computation resource required for executing the response and the negative impact on the system after the execution of this response. Also, a response can fail with some probability. So, at the highest level of abstraction, for each applicable response option, an IRS has to consider both the outcome from deploying the specific response and not deploying it and makes a decision between these two choices based on some metric. From this point, we can see three potential research issues regarding dynamic decision making based IRSs. One is modeling the effect of an attack on the system, and this is directly related to the outcome from a decision on not using any response. The second issue is modeling the effect of the responses, and this is related to the outcome from a decision on using

responses. Finally, there's the issue of how to decide the *set of responses* for deployment for a given attack, considering that responses are deployed on different hosts or services in a distributed environment and that they are not all independent.

There has been some work done on modeling the effect from responses and incidents. For example, Balepin *et al.* propose the "gain matrix" in [5], which formulates the effect of using response A_k in a system with M potential states $S_1,S_2..,S_M$ as:

$$\left(q_1 a_{k1} + q_2 a_{k2} + \ldots + q_m a_{km}\right)$$

where q_i is the probability of the system being in state S_i, and a_{ki} is the benefit from using response A_k in state S_i. The benefit is derived from the response cost (say, in terms of negative impact on functional services), the likelihood of success of the response, and the potential damage from the system remaining in that system state. Following this formulation, the "optimal" response A_i from a set of response alternatives $\{A_1,A_2,\ldots,A_N\}$ is determined by:

$$\mathrm{i} = \arg\max_{1 \le k \le N} \left(q_1 a_{k1} + q_2 a_{k2} + \ldots + q_m a_{km}\right)$$

The gain matrix brings out two challenging facts in the design and implementation of a dynamic decision based IRS. One is the number of system states to be considered. There are likely to be a vast number of states for a real production system, and this would preclude any approach that relies on statically enumerating the states and creating the responses in each state. This underpins the desirability of dynamic intrusion response approaches. An example is the work by Toth *et al.* in [10], in which they use a dependency tree structure to dynamically calculate the impact on the system from a response.

The second challenge is about selecting the optimal set of responses in real time. A *response plan* is composed of multiple response operations which will be carried out in a certain

sequence and at specific times. For example, {tightening the firewall rules for the entry point to the network at time *x*, rebooting the web server at time *y*, and resetting the firewall rules at time *z*} is a response plan composed of three response operations. Now, consider that the IRS has a choice of N response operations, say one each at a service site. There can be at least 2^N-1 possible response plans even without considering the timings in the sequence of response operations. This imposes a challenge on the response decision process, which is to pick the best choice from all potential plans. A naïve dynamic approach of scanning through the gain matrix and evaluating the expected gain for the large number of response plans will not work well in general since an IRS usually has to respond to incidents in a timely manner. Existing work such as ADEPTS [26, 27] relies on heuristics for limiting the size of the set of response plans by considering only the response operations that are applicable near the sites where an incident was detected. ADEPTS also evaluates the responses with respect to a local optimality criterion (say, effect on the specific service, rather than on the system as a whole). While this is certainly an improvement over static decision making based IRS systems, much work needs to be done to determine how good a given heuristic is for a specific payload system.

Now we provide the details of some representative dynamic IRSs.

3.2 ADEPTS

3.2.1 Design Approach

ADEPTS [26, 27] makes use of the characteristics of a distributed application in guiding its response choiuces. It considers the interaction effects among the multiple services both to accurately identify patterns of the intrusions relevant to the response process (e.g., cascading failures due to service interactions) and to identify the effectiveness of the deployed response mechanism. In designing an IRS, a possible approach is to consider different attacks and provide customized sequence of response actions for each step in an attack. A second approach, subtly yet significantly different, is to consider the constituent

services in the system and the different levels of degradation of each individual service due to a successful attack. For easier understanding, one may visualize a malicious adversary who is trying to impact the constituent services (the sub-goals) with the overall goal of either degrading some system functionality (e.g., no new orders may be placed to the e-store) or violating some system guarantee (e.g., credit card records of the e-store customers will be made public). In ADEPTS, the authors take the latter approach. This is motivated by the fact that the set of services and their service levels are finite and reasonably well understood, while the possible universe of attack sequences is potentially unbounded. They focus on the manifestations of the different attacks as they pertain to the services rather than the attack sequence itself. This leads them to use a representation called an Intrusion Graph (I-GRAPH), where the nodes represent sub-goals for the intrusion and the edges represent pre-conditions/post-conditions between the goals. Thus, an edge may be OR/AND/Quorum indicating any, all, or a subset of the goals of the nodes at the head of the edge need to be achieved before the goal at the tail can be achieved.

In ADEPTS, the response choice is determined by a combination of three factors – static information about the response, such as how disruptive the response is to normal users; dynamic information, essentially history of how effective the response has been for a specific class of intrusion; and out-of-band parameters of the response, such as expert system knowledge of an effective response for a specific intrusion or policy determined response when a specific manifestation occurs. Importantly and distinct from other work, ADEPTS points out the need for the IRS to provide its service in the face of unanticipated attacks. Thus, it does not assume that the I-GRAPH is complete nor that there is a detector to flag whenever an I-GRAPH node is achieved. However, it assumes that the intrusion will ultimately have a manifested goal which is detectable. ADEPTS also considers the imperfections of the detection system that inputs alerts to it. The detectors would have both type I and type II errors, i.e., false alarms and missed alarms. If false alarms are not handled, this can cause the IRS to take unnecessary responses, potentially degrading the system functionality below that of an unsecured system. If missed alarms (or delayed alarms) are not compensated for, the system functionality may be severely degraded despite the IRS.

ADEPTS can co-exist with off-the-shelf detectors and estimates the likelihood that an alarm from the detection system is false or there is a missing alarm. The algorithm is based on following the pattern of nodes being achieved in the I-GRAPH with the intuition that a lower level sub-goal is achieved with the intention of achieving a higher level sub-goal.

The design of ADEPTS is realized in an implementation which provides intrusion response service to a distributed e-commerce system. The e-commerce system mimics an online book store system and two auxiliary systems for the warehouse and the bank. Real attack scenarios are injected into the system with each scenario being realized through a sequence of steps. The sequence may be non-linear and have control flow, such as trying out a different step if one fails. ADEPTS' responses are deployed for different runs of the attack scenarios with different speeds of propagation, which bring out the latency of the response action and the adaptive nature of ADEPTS. The survivability of the system is shown to improve over a baseline system, with a larger number of runs leading to greater improvement.

3.2.2 Contributions and Further Work

ADEPTS presents a worthy framework for reasoning about and responding to multi-stage attacks in systems that have the non-determinism and imperfections of real-world distributed systems. It provides fundamental algorithms for diagnosis of the affected service, taking proactive response, and evaluating the effect of a response by observing further alerts in the system.

However, the responses in ADEPTS only achieve a local optima and are deployed in sites close to where the detector flagged the alarm. It is unclear how close ADEPTS can get to the theoretically best achievable response. Also, ADEPTS needs to consider variants of previously observed attack scenarios and completely unanticipated attack scenarios.

3.3 ALPHATECH Light Autonomic Defense System (αLADS)

3.3.1 Design Approach

This is a host-based Autonomic Defense System (ADS) using a Partially-Observable Markov Decision Process that is developed by a company called Alphatech, which has since been acquired by BAE systems [28-30]. The system αLADS is a prototype ADS constructed around a Partially Observable-Markov

Decision Process (PO-MDP) stochastic controller. The main thrust of the work has been the development, analysis and experimental evaluation of the controller. At the high level, the authors have two goals for their ADS— it must select the correct response in the face of an attack, and it must not take actions to attacks that are not there, notwithstanding noisy signals from the intrusion detection system.

The overall framework is that the system has a stochastic feedback controller based on POMDP that takes its input from a commercially-available anomaly sensor (CylantSecure, from Software Systems International, Cyland Division, http://www.cylant.com/), calculates the probability that the system may be in an attack state, and invokes actuators to respond to a perceived attack. The system is partially observable because the sensors (the intrusion detectors) can give imperfect alerts; the system is also partially controllable since the effect of an action by the ADS will not deterministically bring the system back to a functional state.

The authors set up PO-MDP formulas to determine for each $x \in X$, $b_k(x) = Pr(x_k = x \mid I_k)$, where I_k denotes the set of the first k observations received and all controls selected through the $(k\text{-}1)^{st}$ decision stage. Let $B_k = \{b_k(x) : x \in X\}$ be the set of all individual state estimates after the k^{th} observation. The objective is to choose a response policy μ that outputs the selected control $u_k = \mu(B_k)$ (as a function of B_k).

The choice of the optimal response is given by the equation

$$\mu^*(B_k) = \arg\min_{u \in U} \left[\sum_{x \in X} \alpha^*(u, x) b_k(x) \right]$$

where $\alpha^*(u, x)$, for each $u \in U$ and $x \in X$, is proportional to the optimal *cost-to-go*, given current state $x_k = x$ and current decision $u_k = u$. That is, $\alpha^*(u, x)$ is the expected cost obtained through an optimal selection of controls at future decision stages, given current state $x_k = x$ and current decision $u_k = u$. However, determining the optimal response policy that minimizes the infinite horizon cost function is intractable and heuristics must be applied to find near-optimal policies. The heuristics the authors apply is to consider the single-step combination of current state and control.

For the evaluation the authors build a Markov state model for the worm attack on a host. The prototype ADS receives observations from two intrusion detector sensors. One intrusion detector sensor monitors activities on the IP port and the other sensor monitors processes operating on the host computer. These two sensors are calibrated against activity that is determined to be representative of how the computer system will typically be used. For the experiments, the training data was a combination of stochastic HTTP and FTP accesses plus random issuances of commands that are commonly used by operators of Linux. The first experiment demonstrates that an ADS built on a feedback controller is less likely to respond inappropriately to authorized system activity than a static controller, i.e., is less susceptible to noises from the detection system, and is thus able to effectively use a more sensitive anomaly detector than a static controller. The second experiment demonstrates the ability to respond to attacks not seen before—αLADS was trained with a worm attack on the ftp server and able to thwart similar worm attacks to the named and rpcd servers. The surprising result is αLADS is able to thwart every single instance of the not seen before attacks. To interpret the results, a crucial piece of information is the degree of similarity between the different worms, which is not available in the published papers.

3.3.2 Contributions and Further Work

The work is significant in its use of a formal modeling technology (partially observable Markov decision process) in intrusion response. The design is rigorous and while the modeling technique has the challenge of determining the correct transition matrix from suitable training data, this challenge is not unique to the αLADS system. It is expected that the work will mature and use more sophisticated techniques for creation of the matrices that are available in related literature.

What has gotten short shrift in this work is development of the actual responses that would be effective in a distributed environment. In fact their experiments only use the ability to kill a process or shut a computer down (apart from just observation or human notification). The system has to be made hierarchical and distributed so that it can respond to attacks in different parts of a distributed infrastructure.

3.4 Cooperating Security Managers (CSM) and Adaptive, Agent-based Intrusion Response System (AAIRS)

3.4.1 Design Approach

Both systems come from the same research group with CSM preceding AAIRS in chronology. CSM is designed to be used as an intrusion detection tool in a large network environment. CSM follows an approach in which individual intrusion detection monitors can operate in a cooperative manner, without relying on a centralized director to perform the network intrusion detection. To stress the role of the individual components in managing intrusions, not just monitoring them, the term used is Security *Managers*. CSM employs no centralized director. Instead, each of the individual managers assumes this role for its own users when that manager suspects suspicious activity. Each CSM reports all significant activity to the CSM for the host from which the connection originated. This enables CSM to track a user as she travels from one host to another in a distributed environment.

If an intruder is suspected or detected, it is up to the intruder-handling (IH) component to determine which action to take. This is where the intrusion response capability is embedded in CSM. The responsibility of the IH module is to take appropriate actions when intrusive activity is detected. Performing a specific action in response to an abuse will depend on the perceived severity of the detected abuse. Simple notification of the system manager on the detecting system is still the first step. The second step is to also notify all other CSMs in the trail for this user. This information is obtained from the user-tracking module. Beyond this, several other activities may be deemed appropriate. Two actions would be to kill the current session of the suspected intruder and to lock the account that was used to gain access so the intruder cannot simply return. However, they have to be done with care only when the evidence is strong and the disruption due to lack of response is severe.

A later work coming from the same group is the Adaptive, Agent-based Intrusion Response System (AAIRS) ([20][21]). In AAIRS, multiple IDSs monitor a computer system and generate intrusion alarms. The Interface Agents receive the alerts and use an iteratively built model of false alerts and missed alerts from the detectors to generate an attack confidence metric. It passes this metric along with the intrusion

report to the Master Analysis agent. The Master Analysis agent classifies whether the incident is a continuation of an existing incident or is a new attack using several different parameters, such as the target application and target port. Their decision algorithm for determining if an alarm corresponds to a new attack or an existing attack is adopted by other systems, such as ADEPTS.

If the Master Analysis Agent determines this is a new attack, it creates a new Analysis agent for handling this attack. The Analysis agent analyzes an incident and generates an abstract course of action to resolve the incident, using the Response Taxonomy agent from [18] to classify the attack and to determine a response goal. The Analysis agent passes the selected course of action to the Tactics agent, which decomposes the abstract course of action into very specific actions and then invokes the appropriate components of the Response Toolkit.

The proposed methodology provides response adaptation through three components: the Interface, Analysis, and Tactics agents. The Interface agent adapts by modifying the confidence metric associated with each IDS. As the Analysis components receive additional incident reports, these reports may lead to reclassification of the type of attacker and/or type of attack. This reclassification may lead to the formulation of a new plan or a change in how the response goal is accomplished. The Analysis component may change the plan steps being used to accomplish the goal if alternative steps are available and can be substituted into the plan. Alternatively, the Tactics components may have multiple techniques for implementing the plan step and adapt by choosing alternate steps. These components maintain success metrics on their plans and actions respectively and weight the successful ones so that they are more likely to be taken in subsequent instances of an attack.

The work provides a good framework on which the IRS can be built. However, it does not provide any of the system-level techniques and algorithms that will be required for the AAIRS to work in practice. It leaves many unanswered questions, most important of which are what is the algorithm to determine a sequence of response actions to an incident, how does the system measure the success of previous responses, or how are multiple concurrent attacks handled.

3.4.2 Contributions and Further Work

CSM highlights the tradeoffs to be made in any autonomous response system. Its module for tracking a user and the architecture for distributed detection are valuable in an IRS for a distributed system. However, the work is lacking in system level details and actual design decisions made for a specific application context. The evaluation does not shed any light on the IH component of their system.

AAIRS presents a compelling architecture with different modules that make up an IRS. The modules are at two basic levels of abstraction – application system neutral and application system specific. These levels are important to the extensibility of an IRS to new applications. AAIRS also raises important concerns for any IRS – the imperfections of any IDS both for false alarms and missed alarms have to be accounted for in the IRS and there should be feedback about the success or failure of a deployed response. However, the work is lacking in specific algorithms for any of the steps of an IRS. There are no system level details provided and this is especially critical for IRS since many tradeoffs in algorithms are brought out by actual implementations and deployments. The system description indicates competent system administrators may still need to be involved in the loop, e.g., in manually determining if an alert from the IDS was a false one.

3.5 Emerald

3.5.1 Design Approach

Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) developed an architecture that inherits well-developed analytical techniques for detecting intrusions, and cast them in a framework that is highly reusable, interoperable, and scalable in large network infrastructures [8, 31]. Its primary goal is not to do automated intrusion response; however, its modular structure and tools can enable effective response mechanisms.

The primary entity within EMERALD is the monitor, with multiple monitors deployed within each administrative domain. The monitors may interact with the environment passively (reading activity logs or network packets) or actively (via probing that supplements normal event gathering). The monitors may interact with one another. An EMERALD monitor has a well-defined interface for sending and receiving

event data and analytical results from third-party security services. An EMERALD monitor is capable of performing both signature analysis and statistical profile-based anomaly detection on a target event stream. The work on these components represent state-of-the-art development in the intrusion detection literature within each domain. In addition, each monitor includes an instance of the EMERALD resolver, a countermeasure decision engine capable of fusing the alerts from its associated analysis engines and invoking response handlers to counter malicious activity.

A feature that makes EMERALD well-suited to intrusion response in a distributed environment is its capability for alert aggregation. This is achieved through a tiered arrangement of monitors and exchange of CIDF-based [32] alert information. Thus resolvers are able to request and receive intrusion reports from other resolvers at lower layers of the analysis hierarchy enabling the monitoring and response to global malicious activity. Each resolver is capable of invoking real-time countermeasures in response to malicious or anomalous activity reports produced by the analysis engines. The countermeasures are defined in a field specific to the resource object corresponding to the resource in which the monitor is deployed. Included with each valid response method are evaluation metrics for determining the circumstances under which the method should be dispatched. These criteria are the confidence of the analysis engine that the attack is real and the severity of the attack. The resolver combines the metrics to formulate its monitor's response policy.

3.5.2 Contributions and Further Work

An important lesson from the design of EMERALD is the separation of generic and target-specific parts of the system. Target-specific refers to the service (FTP, SSH, etc.) and the hardware resource (router, etc.) which EMERALD is deployed on. This design approach simplifies reusability of components and extensibility and enhances integration with other data sources, analysis engines, and response capabilities. While we see the great potential in EMERALD to build automatic responses in the resolver, we did not find any detailed description of its capabilities or its application. The infrastructure provides the common EMERALD API, event-queue management, error-reporting services, secondary storage management (primarily for the statistical component), and internal configuration control. The

statistical and P-BEST components are integrated as libraries and provide powerful intrusion detection capabilities. The EMERALD API can likely be used to build a powerful intrusion response engine. However, this has not been reported in the project.

3.6 Other Dynamic Intrusion Response Systems

There are some other systems that employ dynamic decision making for intrusion response. In the interest of space, we will limit the discussion of these systems to their key contributions.

1. In [10], the authors propose a network model that allows an IRS to evaluate the effect of a response on the network services. There exists dependencies between entities in the system either a direct dependency (user A depends on DNS service) or an indirect dependency that needs to be satisfied for the direct dependencies (if DNS service on a different subnet, then firewall rules must allow access to that subnet.) A dependent entity may become unavailable either because no path exists in the network topology, or firewall rules disallow access. Indirect dependencies are determined automatically by analyzing the network topology (which is encoded in routing tables) as well as firewall rules. Dependencies are represented using AND-OR tree and the degree of dependency is represented by a number between 0 and 1. Capability of an entity is the portion of the entity's functionality that is available under the current response strategy (number between 0 and 1). The capability is computed from the dependency tree. A penalty is assigned for the unavailability of each entity. The net penalty cost of an entity = Capability × Penalty. At each step, the system takes the response that minimizes the penalty. This is a greedy algorithm and does not necessarily lead to a global optima.
2. Security agility [33] is a software flexibility technique that extends the functionality of software components to accommodate the dynamic security properties of their environment. Thus when access control is tightened in the environment, the software does not fail. The Security Agility toolkit provides the means to integrate more sophisticated response capabilities (than simply killing an offending process) into processes to realize more flexible intrusion-tolerant systems. At its heart the response actions are changes to access control rules with activation criteria specified. The chief

contribution of this work is policy reconfiguration techniques at runtime. A secondary contribution is that the reconfiguration capability enables reconfiguration as part of a response.

In general, dynamic decision making based IRS is a promising technology that is still in its nascent phase. There is scarce deployment of them in real world production systems, at least what is reported in open literature. Part of the reason is the many open issues that need to be solved before generalizable design principles can be presented. For example, the heterogeneity among real world systems has been an obstacle for modeling the effect on the system from incidents and responses. In a sense, an IRS has to figure out what are the services in the system, what are their functionality, what are the interactions among them and what are the effects of a response on the system. Each of this is a topic of active research in distinct fields, such as system management. Besides, there are many properties in the response decision making process that need to be quantitatively modeled and analyzed, such as the optimality of a response. There is surprisingly little in the way of comparative evaluation of the different techniques with respect to each other and with respect to an idealized scenario. We believe this is an exciting field of development in IRS technology and we hope to see many worthwhile research efforts in it in the years to come.

4 Intrusion Tolerance through Diverse Replicas

The use of diverse replicas in IRS borrows ideas from the field of natural fault tolerance and from observations of biological systems. By introducing artificial diversity, a common phenomenon in biological systems, an attack specific to a vulnerability in a system cannot affect another system that lacks that vulnerability. Coupled with redundancy, the effect of an attack can be masked, allowing the system to provide continued service in the presence of disruptions. The basic approach is to employ a diverse set of replicas for a given service, such that they provide the same high level functionality with respect to other services, but their internal designs and implementations differ. The fault masking techniques used are

similar to methods in natural fault tolerance, such as voting and agreement protocols. The use of diverse replicas is attractive because provable theoretical improvements to the survivability or security of the system can be obtained, compared to other techniques that are more suitably classified as heuristics. Evaluation techniques from the mature field of natural fault tolerance are more readily adapted to this class of IRSs.

A common assumption is to assume at most a fraction of the servers in a network may fail. This assumption is strengthened through the use of active and periodic recovery. Another common assumption is that failures in the system are independent, which motivates the use of diversity. Extending this argument to vulnerabilities, the assumption states that vulnerabilities do not occur across different operating systems and applications.

4.1 Broad Research Issues

Two main issues that arise are (1) How to introduce diversity into the system; (2) How to achieve redundancy that improves survivability and security of the system. To handle the first issue, most system architects have chosen to manually introduce diversity, such as by installing different operating systems and applications. Taking it a step further, different versions of an application are also used. Introducing diversity automatically is a topic of much ongoing research, such as through the OASIS and SRS programs within DARPA [14, 16, 34, 35]. We survey two of these papers here. The second issue is system specific, and in general, advances in the fields of cryptography and dependability have given us a better idea of how redundancy impacts the survivability of the system.

The following are sample systems in this domain.

4.2 Building Survivable Services using Redundancy and Adaptation

This paper [18] advocates the use of redundancy and adaptation to build survivable services and presents a general approach to do so. The authors introduce various forms of redundancy – redundancy in space and time, and the use of redundant methods. Redundant methods enforce a security attribute, and the attribute remains valid if at least one of the methods remains uncompromised. An example is to perform multiple encryption operations using different cipher systems. They motivate the use of

redundancy to avoid single points of vulnerability and to introduce artificial diversity and unpredictability into the system. They provide a useful characterization of the effectiveness of redundancy as the independence of the redundant elements, and the main goal of the design is to maximize this independence.

As an example, they apply redundancy to a secure communication service called *SecComm*. SecComm provides customizable secure communication by allowing user-specified security attributes and algorithms for implementing these attributes. The traditional approach of selecting a better encryption algorithm or increasing the key size to increase security is not survivable, since they in essence still contain single points of vulnerability. Therefore they propose using two or more techniques to guarantee an attribute rather than a single method. For maximal independence, a different key established using different key distribution methods is used for each method. Fragmentation is also proposed when there are multiple connections. At a basic level, they implement redundancy by applying multiple methods sequentially to the same data. They suggest numerous general ideas to vary the application of the methods, with the main goal being to maximize the independence as mentioned above. The increase in survivability is relatively hard to quantify, therefore their experimental results only measure the cost of redundancy against performance.

They have done a good job motivating the use of redundancy and diversity with respect to the security of computer systems, and have provided many examples on how to use diverse replicas for various purposes. The next two sections illustrate specific architectures that use diverse replicas.

4.3 SITAR (Scalable Intrusion Tolerant Architecture)

4.3.1 Design Approach

SITAR is an intrusion tolerant system that relies heavily on redundancy [14]. The main components of the SITAR architecture are proxy servers that validate incoming and outgoing traffic, and detect failures within the application servers and among themselves. The mitigation of adverse effects due to intrusions is through the use of redundant and diverse internal components. The diversity is achieved manually by choosing different server codes (e.g. Apache, Internet Information Server for web servers) and different

operating systems (e.g. Linux, Solaris, MS). Through this, they assume that only one server can be compromised by a single attack, allowing them to build a simple model for their system.

A specific sub-system that employs redundancy is their ballot monitor sub-system. The monitors receive responses from acceptance monitors, performs validation of responses, and applies a voting algorithm to detect the presence of an attack and respond to the attack by choosing the majority response as the final response.

4.3.2 Contributions and Further Work

SITAR is a good example of using diverse replicas to improve the survivability of the system. The architecture used clearly illustrates the practical benefits of diversity and replication, and how that can be used to detect and respond to attacks. However, the work does not lay down generalizable design principles for diverse replicas.

4.4 Survival by Defense-Enabling

4.4.1 Design Approach

The authors propose an approach to survivability and intrusion tolerance called *survival by defense* [16]. The main idea is to introduce defense mechanisms that enhance the common protection mechanisms with a dynamic strategy for reacting to a partially successful attack. Due to their assumption that they lack control over the environment (e.g. OS, network), they focus on ensuring correct functioning of the critical applications. Therefore defense enabling is performed around the application, with the assumption that it can be modified.

The type of attack considered is the corruption that results from a malicious attack exploiting flaws in an application's environment (they conclude it is most likely). Since the knowledge and actions within the environment is limited, an assumption they make is that administrator privileges will eventually be obtained by an attacker. In this context, defense enabling is divided into two complementary goals: (1) Attacker's privilege escalation is slowed down; (2) Defense responds and adapts to the privileged attacker's abuse of resources. Therefore an application is

defense-enabled if mechanisms are in place to cause most attackers to take significantly longer to corrupt it (these mechanisms tend to be in the middleware).

To prevent the quick spread of privilege, the main idea is to divide the system into distinct security domains consisting of user-defined elements (e.g. host, LAN, router), such that each domain has its own set of privileges. They suggest the use of a heterogeneous environment (various types of hardware, OS) to prevent domain admin privileges from one domain to be converted into domain admin privileges in another domain. Also, applications are distributed across the security domains (i.e. application redundancy) to reduce the effect of privilege escalation within a domain. Not limited to replicating applications, they suggest other forms of replication such as communication redundancy.

A strong assumption made is that attacks proceed sequentially (staged attacks) instead of concurrently, that is, an attack on an application in multiple domains is slower than an attack on one single domain. Their design approach is to design applications intelligently distributed across security domains, so that privilege in a set of domains is needed to compromise the application. However, there is no discussion on how the staging is to be enforced.

4.4.2 Contributions and Future Work

With respect to the use of diverse replicas, the significance of this work is their higher level approach to the use of redundancy. This is illustrated by partitioning the network into various domains, such that an attack on a domain only has a limited affect on another domain.

4.5 Implementing Trustworthy Services using Replicated State Machines

This paper [17] is a comprehensive survey on techniques to implement distributed trust. In essence, the problem of distributing trust is solved by using replicas, and the issues faced are the same as the general theme of this section. By distributing trust, it is possible for the fault-tolerance of the distributed system to exceed the fault-tolerance of any single server. The first emphasis of the authors is the use of

proactive recovery to transform a system that tolerates *t* failures in its lifetime to a system that tolerates *t* failures within a time window. This is useful because it strengthens the assumption that not more than *t* failures will exist at any one time, though additional complexity to the system design is introduced. Next, they discuss service key refresh and scalability, which is achieved through the use of secret sharing protocols. This allows periodic changes to the shared secret keys to be transparent to the clients. With regards to server key refresh, they discuss three solutions, namely the use of trusted hardware, offline keys, or read-only service public key.

The independence assumption of failures is discussed and methods to reduce correlated failures are mentioned. They are (1) developing multiple server implementations, which is generally an expensive endeavor; (2) employing pre-existing diverse components, such as using different operating systems; (3) introducing artificial diversity during compile or runtime, which will not eliminate flaws inherent in the algorithms implemented.

The next important requirement in distributed trust is replica coordination. This is mainly required for consensus, which is impossible to implement deterministically in the asynchronous model. The solutions provided are (1) abandon consensus and use quorum systems or embrace all the sharings of a secret (rather than having to agree on one sharing); (2) employ randomization to solve Byzantine agreement; (3) sacrifice liveness (temporarily) for weaker assumptions of the asynchronous model.

Finally, they discuss the problems and solutions that arise when confidential data is involved. The main problems of confidential data are that they cannot be changed by proactive recovery, and at any time they are unencrypted their confidentiality may be lost. By using either re-encryption or blinding, it is possible for clients to retrieve encrypted data in the case where services implement access control.

This paper is a well written introduction to the issues and solutions of distributing trust, which are the same issues one would face when using diverse replicas in a system. The next section presents a specific approach to this problem.

4.6 Distributing Trust on the Internet

4.6.1 Design Approach

In [15] Christian Cachin presents an architecture for secure state machine replication in an asynchronous and adversarial network. This is achieved through recent advances in threshold cryptography and protocols for atomic broadcast. Depending on the level of diversity in the replicated servers, guarantees of liveness and safety can be made under certain assumptions (e.g. there are n static servers and at most t may fail). The designer of the distributed system can easily define meaningful attributes, such as the location and the type of operating system, which represent the different measures of diversity within the system. From these attributes, one can produce a description of how the servers may be compromised simultaneously (which is formalized as a General Adversary Structure in the paper) and design a secret sharing scheme that ensures the guarantees of the distributed system are kept.

The use of diversity enables them to avoid making the standard independence assumption of faults, allowing the system to tolerate malicious acts and better approximate reality. The consensus protocol described within the paper is part of the Secure Intrusion-Tolerant Replication Architecture (Sintra) toolkit.

4.6.2 Contributions and Further Work

The paper presents specific techniques for distributing trust in an untrusted environment. This work is significant in that a clear approach is presented that allows a system designer or administrator to easily incorporate this architecture into a network with no existing use of diverse replicas and obtain an improvement in the survivability of the system.

Extensions to their scheme are discussed, such as using proactive recovery, dynamic grouping of servers, hybrid failure structures that distinguish between natural and malicious failures, and optimistic protocols that adapt their speeds depending on the presence of adversaries (due to the significant overhead of the atomic broadcast protocols).

5 Responses to Specific Kinds of Attacks

This class of IRS is customized to respond to specific kinds of attacks, most commonly, distributed denial of service (DDoS) attacks. There are scarce efforts at responses to other kinds of specialized attacks. One example is responding to internal attacks, where approaches proposed include changing the access control rules or dropping some connections when a high threat level is perceived [36]. However, such techniques so far have been human intensive and little exists in the literature in terms of rigorous validation of an automated system against different internal attacks. Hence, we focus on the DDoS attack response technology here.

Distributed denial of service attacks present a growing problem for network administrators around the world. On the Internet, a DDoS attack is one in which a multitude of compromised systems attack a single target, thereby causing denial of service for users of the targeted system. The flood of incoming messages to the target system essentially forces it to shut down, thereby denying service to the system to legitimate users. An adversary begins a DDoS attack by exploiting a vulnerability in one computer system and making it the DDoS "master." It is from the master system that the intruder identifies and communicates with other systems that can be compromised. The intruder loads cracking tools available on the Internet, onto multiple—sometimes thousands of—compromised systems. With a single command, the intruder instructs the controlled machines to launch one of many flood attacks against a specified target. The inundation of packets to the target causes a denial of service.

Increasingly powerful DDoS toolkits are readily available to potential attackers and essential systems are ill prepared to defend themselves. Both their ease of use and effectiveness make them the perfect tool for malicious individuals attempting to disrupt networks and web services. Accordingly corporations and academia are working overtime to solve this complex problem and several systems exist that work to address DDoS. Unfortunately none of the solutions fully handles the ever evolving DDoS toolkits being used today and fail to present an all encompassing solution to the problem. Despite this failing of existing DDoS handling systems, systems such as Cooperative Intrusion Traceback and Response Architecture (CITRA) and the cooperative architecture hold much promise for the future.

5.1 Primitives for responding to DDoS

DDoS attacks typically require four components: the attacker, master hosts, zombie hosts and a victim host. Using exploits in a remote system, the attacker installs the attack program that can be remote controlled by the master host. When the attack begins it usually falls in to one of two classes: *bandwidth depletion* and *resource depletion*. Attackers can perform these attacks directly or through reflection. Reflection makes it more difficult to track down the source of the problem and offers a greater challenge to DDoS handling systems by bouncing packets off other hosts. The first line of defense against DDoS is of course intrusion prevention. Rate limiting filters are commonly used for preventing DDoS attacks [37, 38]. The reason why intrusion prevention and intrusion detection are unlikely to solve all kinds of DDoS attacks is that it is often difficult to tell the two kinds of traffic apart. Although some DDoS traffic can be easily distinguished from legitimate traffic, this is not true in the general case. More sophisticated DDoS toolkits generate traffic that "blends in" with legitimate traffic and therefore cannot be blocked. Hence autonomous intrusion response is called for. Responses when a DDoS attack is detected usually involve some type of traceback or packet marking procedure to locate the source of the attack and block it.

Fundamentally response mechanisms for DDoS attacks have to be distributed in nature as pointed out in [21]. This is due to several factors (a) attackers most of the time spoof packet source IPs address; (b) the possibility of the attack initiating from a wide range of networks worldwide; and (c) the inability of a domain to enforce incoming traffic shaping; detected malicious flows can be blocked locally but the assistance of the upstream network is still needed in order to free the resources occupied on the incoming link.

5.2 CITRA

5.2.1 Design Approach

The Cooperative Intrusion Traceback and Response Architecture (CITRA) is one of these systems currently in development working to handle bandwidth depletion attacks [19, 39]. Originally, the Cooperative Intrusion Traceback and Response Architecture (CITRA) and the Intruder Detection and Isolation Protocol (IDIP) on which it is based, did not have DDoS response as their goal. They were

developed to provide an infrastructure enabling intrusion detection systems (IDS), firewalls, routers, and other components to cooperatively trace and block network intrusions as close to their sources as possible. Later it was adapted for responding to DDoS attacks. CITRA is used by creating a cooperative network of nodes each installed with the CITRA software. A node registers itself and coordinates efforts with the rest of the nodes through the Discovery Coordinator (DC). When an attack is detected, the CITRA nodes traceback toward the source through the use of network audit data. Along the path of the traceback, temporary action lasting only 2 minutes is taken to decrease the network flooding. During this two minute window, the DC formulates a more reasoned plan of how to handle the attack.

At each CITRA component along the path of attack, responses are taken in accordance with the CITRA policy mechanisms. Traffic rate limiting is used rather than packet filtering because of the difficulty of telling a legitimate packet from one that is part of the adversarial stream of packets. This response strategy is approximate since some DoS traffic may get through while some legitimate traffic may get blocked out. But with well chosen parameters, enough bandwidth should be available for legitimate traffic even though it may be at a reduced speed. So the authors integrated a rate limiter function using token bucket rate limiting service available with netfilter. Experiments using RealPlayer were carried out on a testbed composed of several subnets each with their own CITRA enabled router. Results showed that when the system was active it allowed uninterrupted viewing but at reduced quality. It took 10 seconds to slow the attack even on their small-scale testbed, bringing up a possible issue of scalability. On more powerful hardware, however, the delay was reduced to 2s and the quality was not reduced.

5.2.2 Contributions and Further Work

The architecture presented is appealing and points in the direction of further development for DDoS mitigation. It does not require universal deployment to be meaningful, which is a big positive factor for any DDoS response mechanism.

Possible problems with the system involve slower then optimal traceback and scalability limitations. The system currently needs more effective means of dealing with attacks then simply limiting the

bandwidth due to more sophisticated, multi-pronged attacks. Also, monitoring all packets coming in from all other networks is not scalable without smart algorithms. There is considerable ongoing work in the area of making routers fast by developing high-speed algorithms for packet lookups and classification (see work by George Varghese *et al.* from UCSD and Nick McKeown *et al.* at Stanford).

5.3 Cooperative Counter-DDoS Entity

5.3.1 Design Approach

Similar to CITRA, the cooperative architecture [21] attempts to locate the source of the attack but through the use of cooperative domains which internally check if they are sending a DDoS attack and if so alert the other networks that may be affected. To deal with the scalability of this system and increased network congestion from the message it uses multicast transmission of alerts. Multicast allows a source host to send a message to multiple hosts through an optimal spanning tree by only sending the message once and replicating it only when the path along the spanning tree splits. Within each domain there are entities which determine the probability of an attack internally by looking at the alerts coming in from other entities and domains and the results of local intrusion detection systems. If one entity fails another entity has the ability to take over. Once the number of alerts exceeds a threshold the entities take action through use of a reaction table. This reaction table provides the action that should be taken given the current state. This system lacks any experimental evidence to support its claims and offers a different approach then CITRA. By not using a traditional traceback mechanism, the system can react faster and more efficiently. However, it relies on the multicast backbone heavily and should an attacker target the backbone the system may be rendered ineffective.

5.3.2 Contributions and Further Work

The work lays out an impressive architecture for quickly reacting to DDoS attacks. It moves away from the reliance on traceback that underlies the vast majority of approaches in this domain. However, problems facing the systems in existence today are how to detect legitimate packets sent to a network and packets intended to perform a DDoS attack without disrupting the legitimate users. This is no easy task when one considers that a network may simply be undergoing an increase in legitimate traffic or a DDoS

attack may even use legitimate requests. Another problem deals with determining the source of an attack. How does one find the source of the attack if it is constantly switching sources or it passes through a network with a less than helpful network administrator? Lastly, what is the universe of responses that should be included with any DDoS mitigation system—are the current ones of bandwidth throttling or packet filtering sufficient.

DDoS handling systems today have several weaknesses that can be exploited. They still need a better way of determining the source of the attack and better ways of responding to a detected attack. Blocking the remote host at the source would be optimal but traceback procedures are too slow and resource intensive. Also, they rely on the cooperation of other networks and a limited set of responses. The aforementioned actions either fail to stop the attack and only slow it down or stop the attack but in the process block legitimate users. Better response, detection and traceback technology will have to be developed if these systems are to be deployable in real-world systems.

6 Benchmarking Intrusion Response Systems

It is important to benchmark any IRS using quantifiable metrics. This is a nascent field within IRS design and development and one which needs significant work to get to maturity. Hence this section is based around suggested course of action for future development and an example from an existing IRS, ADEPTS. The metrics should capture the two essential goals of IRSs – to provide gracefully degraded functionality in the presence of attacks, and to make the system more robust to future attacks. These two notions are addressed respectively by the metrics *survivability* and *vulnerability*.

One commonly accepted definition of *survivability* is the capacity of a system to provide essential services in the face of intrusions [40, 41]. The challenge with this definition is how to define essential services – is this by the different categories of users for the different services, or by business criticality, or by some other measure. Also, the question arises if there exists a

minimum essential service level that can be guaranteed. In [42], the authors inject errors into a network specification and visualize effects in the form of scenario graphs. Model checking is used to verify if states that violate certain temporal properties can be reached. Cactus [18] presents a framework for constructing highly customizable and dynamically adaptable middleware services for networked systems. The fine-grained customization allows customized tradeoffs between QoS attributes, including performance, reliability, and survivability, while the dynamic adaptation allows services to change behavior at runtime as a reaction to incoming intrusions.

To start, consider a simple combinatorial model for survivability. Let us define G as the overall goal of the system (e.g., sell products or services on the internet), which is accomplished through several sub-goals G_i (e.g., the different transactions which are possible on the system), $i=1,...,N$. Each sub-goal G_i is given a weight W_i indicating its importance in the achievement of the system goal G, ($\sum_i W_i=1$). This can be estimated by the number of users who reach the sub-goal as a fraction of the total number of users, fraction of the usage of the sub-goal, or a quantity defined by the system owner. Each sub-goal G_i is decomposed into conjunction of sets of services $\overrightarrow{S_{ij}}$, $j=1,...,N_i$, such that each set of services must be functional for goal G_i to be reached. Each such set can be further decomposed to be disjunction of basic services $\overrightarrow{S_{ijk}}$, $k=1,...,N_{ij}$, such that any service of this set being functional causes the set to be functional. Let p_x denote the probability that a service X is affected by a disruption and cannot be used and P_Y denote the probability that a goal Y cannot be reached. Then, P_{G_i} = Max (Min ($p_{S_{ijk}}$), over all $k=1,...,N_{ij}$), over all $j=1,...,N_i$. $P_G = \sum_i W_i P_{G_i}$. The survivability is given by $1- P_G$.

To apply this formulation, we will have to decompose a goal into the services and estimate the probability of a service being not functional. The former can be deduced from a Service Net (network indicating interactions between services during normal operation) through a training phase when the transaction corresponding to a particular sub-goal G_i is executed and the service interactions observed. We may use techniques from software reliability engineering of path testing [43] to determine the conjunction and disjunction of services.

To illustrate the concept, let us consider an example of its application as shown by the authors in ADEPTS [26]. Figure 2 depicts the testbed that is used for experiments on ADEPTS. The payload system mimics an e-Commerce webstore, which has two Apache web servers running webstore applications, which are based on Cubecart [44] and are written in the PHP scripting language. In the backend, there's a MySQL database which stores all the store's information, which includes products inventory, products description, customer accounts, and order history. There are two other organizations with which the webstore interacts – a Bank and a Warehouse. The Bank is a home-grown application which verifies credit card requests from the webstore. The Warehouse is also a home-grown application, which takes shipping requests from the webstore, checks inventory, applies charges on the customer's credit card account, and ships the product. The clients submit transactions to the webstore through a browser. Some important transactions are given in Table 1.

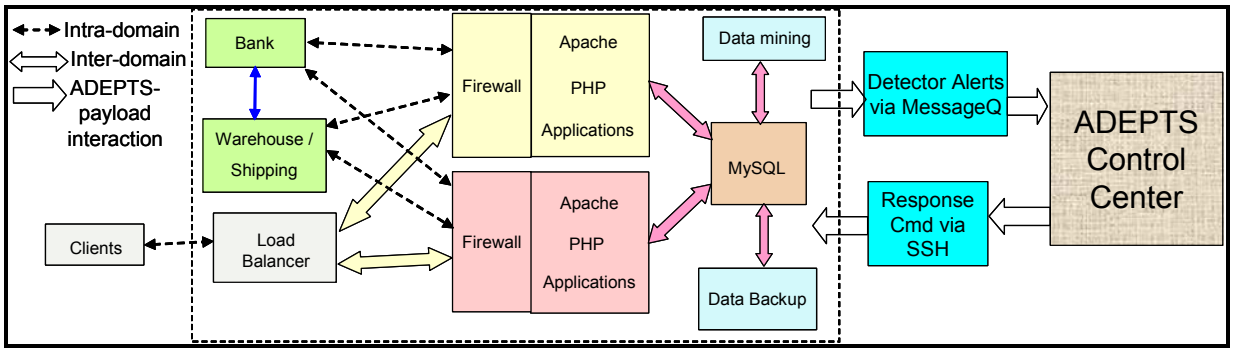

Figure 2: Layout of e-commerce testbed for the experiments on ADEPTS

Name	Description	Services involved	Weight

Browse webstore	Customer uses web browser to access webstore and browse the products available	Apache, MySQL	10
Add merchandise to shopping cart	Customer adds products to shopping cart	Apache, MySQL	10
Place order	Customer can input credit card information, submit orders, and webstore will authenticate credit card with bank	Apache, MySQL, bank	10
Charge credit card	Warehouse charges credit card through bank when order is shipped	Warehouse, bank	5
Admin work	Admins/webmasters can modify various source codes	Variable	10

Table 1: List of important transactions in e-commerce system

There are certain security goals for the system, the complement of which are specified in Table 2, along with the weights. Thus adding the word "prevent" before each gives the goal. The attached weights to the transactions and security goals are used for survivability computation as discussed below.

Illegal read of file (20)	Corruption of MySQL database (70)	Unauthorized credit card charges (80)
Illegal write to file (30)	Confidentiality leak of customer information stored in MySQL database (100)	Cracked administrator password (90)
Illegal process being run (50)	Unauthorized orders created or shipped (80)	

Table 2: List of security goals for e-commerce testbed

The authors define survivability based on the high level transactions and security goals. The metric thus shows the effect of ADEPTS on the high level functioning of the e-commerce system.

$$\text{Survivability} = 1000 - \sum \text{unavailable transactions} - \sum \text{failed security goals}.$$

When a transaction became unavailable or the security goal is violated, the survivability drops by its corresponding weight, which was given in Table 1 and Table 2. Transactions become unavailable due to ADEPTS responses, such as rebooting a host, or due to attacks. Security goals may be violated due to the successful execution of an attack step or an erroneous response action. If a security goal is violated multiple times during an attack, then each violation causes a decrease in the survivability.

The survivability metric considers the state of the system at the present time and does not consider the resilience of the system to future disruptions. This is an important measure and is captured by the *vulnerability* metric. The basic idea of this metric is to fit a temporal distribution for the probability that a given goal in a multi-stage attack is reached. This curve is analogous to unreliability curves seen in traditional fault-tolerant systems. To get the vulnerability at a point in time *T*, we aggregate the individual unreliability curves using some structure as an attack graph and map the nodes to the services which are

affected. Then an analysis similar to the survivability analysis above is performed. Note that the different curves are not independent. Given the Service Net with times for interaction between services, the edges in the attack graph will also have time for the delay between a lower level goal and a higher level goal being achieved. We believe the dependence introduces substantial complexity in the analysis and requires further investigation.

7 Thoughts on Evolution of IRS Technology

We anticipate that for IRSs to be widely deployed they will have to evolve in several directions over the coming years. These include the following.

1. *Ability to withstand unpredictable attack scenarios*. It is inconceivable that all attack scenarios would be "programmed in" the IRS. The IRS should therefore be able to extrapolate strategies available in its knowledge base and take responses to hitherto unseen attacks. This will be an important requirement since polymorphic worms, viruses, and other forms of attacks are rampant in today's security landscape. In this matter, there is a delicate balancing game between learning from the past and being agile to respond to future attacks. It is possible to build up large knowledge bases and do exact matches with them to choose appropriate response from the history. However, this may affect the ability of the system to respond quickly. Also, in taking lessons from the past, the IRS should take into account the fact that the impact of the attack may be different even though the attack steps may be the same. Thus a more drastic or quicker response may be called for.

2. *Dynamic responses with changing network configurations*. The IRS will have to deal with topology and configuration changes in the distributed system. It may take inputs from change notification software systems, such as Tripwire, and modify its response strategies accordingly. In any medium to large sized distributed system, there are multiple administrators responsible for maintaining the system. The tools are often not standardized or uniform across different administrators. Thus modifying the tools to send notification to the IRS seems daunting. A more feasible approach appears to be software to observe the resultant changes and notify the IRS. A change in the configuration may

render some responses unnecessary (such as, a critical service being made accessible from only inside the corporate network) or some responses more critical (such as, a service being made web accessible).

3. *Interaction with other components of the security framework*. The response strategy decided on by the IRS is predicated on confidence placed on other components of the security framework, such as, IDS, change notification software, firewalls, etc. The confidence placed on these components should not be pre-defined constant values. The confidence should change as new software is installed, rules updated, or configurations change. This also indicates why a probabilistic framework for the IRS seems the promising avenue, rather than deterministic response decisions. On another point, the IRS may depend on various basic functionalities in the system, such as, firewalls or access control system to deploy the computed responses.
4. *Separation of policy and mechanism*. It is important for the IRS to provide mechanisms for determining the appropriate response based on security policy settings. As far as practicable the two aspects should be clearly delineated. This will enable a system administrator to set the policy, which can be at various levels of abstraction, such as a paranoid versus *laissez faire* policy at the system-wide level, to policy levels for individual services. In the absence of this, an IRS will not have buy-in for production systems.
5. *User interface design*. Visualizing the different effects of an attack and its responses in a distributed environment is inherently challenging. The speed of the processes (attacks as well as responses) makes this a particularly daunting task. However, for critical functions, all the stake holders (system administrators to CIOs of the organization) will like to have a human digestible form of the information available to them. This should include online tools which lets them visualize the network while an attack or its responses are being deployed as well as offline tools which will aid in forensics action.

8 Conclusion

In this survey paper, we have presented the motivation for designing Intrusion Response Systems (IRSs) for distributed systems. We lay out the design challenges in designing and implementing IRSs. Then we present existing work in the field, classified into four classes. The first category of IRSs called *static decision making* provides a static mapping of the alert from the detector to the response that is to be deployed. The second class called *dynamic decision making* reasons about an ongoing attack based on the observed alerts and determines an appropriate response to take. The third class called *intrusion tolerance through diverse replicas* provides masking of security failures through the use of diverse replicas concurrently for performing security critical functions. The fourth class includes IRSs meant to target specific kinds of attacks, with our focus being Denial of Service (DDoS) attacks. Then, we presented a discussion on the nascent field of benchmarking of IRSs. Finally, we presented five key areas in which IRSs need to evolve for a widespread adoption. In summary, we find that the design and development of IRS has been gaining in research attention and we expect that they will become mainstream in the computer security landscape in the near future.

References

[1] W. Metcalf et al., "Snort-inline."

[2] Symantec Corp., "Norton Antivirus," http://www.symantec.com/home_homeoffice/products/overview.jsp?pcid=is&pvid=nav2007, Last Accessed:

[3] T. Ryutov, C. Neuman, K. Dongho, and Z. Li, "Integrated access control and intrusion detection for Web Servers," in *23rd International Conference on Distributed Computing Systems (ICDCS)*, pp. 394-401, 2003.

[4] McAfee Inc., "Internet Security Suite," http://us.mcafee.com/root/package.asp?pkgid=272, Last Accessed: Nov 24, 2006.

[5] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using specification-based intrusion detection for automated response," in *6th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 136–154, 2003.

[6] S. M. Lewandowski, D. J. Van Hook, G. C. O'Leary, J. W. Haines, and L. M. Rossey, "SARA: Survivable Autonomic Response Architecture," in *DARPA Information Survivability Conference & Exposition II (DISCEX)*, pp. 77-88 vol.1, 2001.

[7] G. B. White, E. A. Fisch, and U. W. Pooch, "Cooperating security managers: a peer-based intrusion detection system," *Network, IEEE,* vol. 10, pp. 20-23, 1996.

[8] P. G. Neumann and P. A. Porras, "Experience with EMERALD to Date," in *Workshop on Intrusion Detection and Network Monitoring*, pp. 73-80, 1999.

[9] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch, "Adaptation Techniques for Intrusion Detection and Intrusion Response Systems," in *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2344-2349, 2000.

[10] T. Toth and C. Kruegel, "Evaluating the impact of automated intrusion response mechanisms," in *18th Annual Computer Security Applications Conference (ACSAC)*, pp. 301-310, 2002.

[11] M. Atighetchi, P. Pal, F. Webber, R. Schantz, C. Jones, and J. Loyall, "Adaptive cyberdefense for survival and intrusion tolerance," *Internet Computing, IEEE,* vol. 8, pp. 25-33, 2004.

[12] M. Tylutki, "Optimal Intrusion Recovery and Response Through Resource and Attack Modeling," Ph. D. Thesis, Davis, CA., 2003.

[13] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok, "Toward cost-sensitive modeling for intrusion detection and response," *Journal of Computer Security,* vol. 10, pp. 5-22, 2002.

[14] D. Wang, B. B. Madan, and K. S. Trivedi, "Security analysis of SITAR intrusion tolerance system," in *ACM workshop on Survivable and self-regenerative systems*, pp. 23-32, 2003.

[15] C. Cachin, "Distributing trust on the Internet," in *International Conference on Dependable Systems and Networks (DSN)*, pp. 183-192, 2001.

[16] P. Pal, F. Webber, and R. Schantz, "Survival by defense-enabling," in *Foundations of Intrusion Tolerant Systems (Organically Assured and Survivable Information Systems)*, pp. 261-269, 2003.

[17] F. B. Schneider and L. Zhou, "Implementing trustworthy services using replicated state machines," *Security & Privacy Magazine, IEEE,* vol. 3, pp. 34-43, 2005.

[18] M. A. Hiltunen, R. D. Schlichting, and C. A. Ugarte, "Building survivable services using redundancy and adaptation," *Computers, IEEE Transactions on,* vol. 52, pp. 181-194, 2003.

[19] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, and T. Reid, "Autonomic Response to Distributed Denial of Service Attacks," in *4th International Symposium on Rapid Advances in Intrusion Detection, RAID*, 2001.

[20] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks,* vol. 44, pp. 643-666, 2004.

[21] G. Koutepas, F. Stamatelopoulos, and B. Maglaris, "Distributed Management Architecture for Cooperative Detection and Reaction to DDoS Attacks," *Journal of Network and Systems Management,* vol. 12, pp. 73-94, 2004.

[22] U. of Southern California-Information Sciences Institute, "Generic Authorization and Access-control API (GAA-API)," http://gost.isi.edu/info/gaaapi/, Last Accessed: Nov 24, 2006.

[23] Netfilter Core Team, "Libipq - Iptables Userspace Packet Queuing Library," http://www.cs.princeton.edu/~nakao/libipq.htm, Last Accessed: Nov 24, 2006.

[24] McAfee Inc., "Network intrusion prevention," http://www.mcafee.com/us/smb/products/network_intrusion_prevention/index.html, Last Accessed: Nov 24, 2006.

[25] McAfee Inc., "McAfee Host Intrusion Prevention," http://www.mcafee.com/us/local_content/datasheets/partners/ds_hips.pdf, Last Accessed: Nov 24, 2006.

[26] B. Foo, Y. S. Wu, Y. C. Mao, S. Bagchi, and E. Spafford, "ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment," in *International Conference on Dependable Systems and Networks (DSN)*, pp. 508-517, 2005.

[27] Y.-S. Wu, B. Foo, Y.-C. Mao, S. Bagchi, and E. Spafford, "Automated Adaptive Intrusion Containment in Systems of Interacting Services," *Elsevier Journal on Computer Networks, Special Issue on "Security through Self-Protecting and Self-Healing Systems",* vol. (In press), to appear in Spring 2007; available as Purdue ECE TR05-14., 2005.

[28] D. Armstrong, S. Carter, G. Frazier, and T. Frazier, "Autonomic defense: Thwarting automated attacks via real-time feedback control," *Wiley Complexity, Special Issue on "Resilient and Adaptive Defense of Computing Networks",* vol. 9, pp. 41-48, 2003.

[29] D. Armstrong, G. Frazier, S. Carter, T. Frazier, and I. Alphatech, "A controller-based autonomic defense system," *DARPA Information Survivability Conference and Exposition, 2003. Proceedings,* vol. 2, pp. 21-23, 2003.

[30] O. P. Kreidl and T. M. Frazier, "Feedback control applied to survivability: a host-based autonomic defense system," *Reliability, IEEE Transactions on,* vol. 53, pp. 148-166, 2004.

[31] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *National Information Systems Security Conference,* pp. 353-365, 1997.

[32] P. Porras, D. Schnackenberg, S. Staniford-Chen, M. Stillman, and F. Wu, "The Common Intrusion Detection Framework," CIDF working group document,

http://www.gidos.org.

[33] M. Petkac and L. Badger, "Security agility in response to intrusion detection," in *16th Annual Computer Security Applications Conference (ACSAC)*, pp. 11-20, 2000.

[34] P. P. Pal, F. Webber, R. E. Schantz, and J. P. Loyall, "Intrusion Tolerant Systems," *Proceedings of the IEEE Information Survivability Workshop (ISW-2000),* pp. 24-26, 2000.

[35] V. Stavridou, B. Dutertre, R. A. Riemenschneider, and H. Saidi, "Intrusion Tolerant Software Architectures," *Proceedings of the 2001 DARPA Information Survivability Conference & Exposition,* 2001.

[36] S. M. Khattab, C. Sangpachatanaruk, D. Mosse, R. Melhem, and T. Znati, "Roaming honeypots for mitigating service-level denial-of-service attacks," in *the 24th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328-337, 2004.

[37] W. J. Blackert, D. M. Gregg, A. K. Castner, E. M. Kyle, R. L. Hom, and R. M. Jokerst, "Analyzing interaction between distributed denial of service attacks and mitigation

technologies," in *the DARPA Information Survivability Conference and Exposition (DISCEX)*, pp. 26-36 vol.1, 2003.

[38] D. K. Y. Yau, J. C. S. Lui, L. Feng, and Y. Yeung, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *Networking, IEEE/ACM Transactions on,* vol. 13, pp. 29-42, 2005.

[39] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for intrusion detection and response," in *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX)*, pp. 3-11 vol.2, 2000.

[40] Carnegie Mellon - Software Engineering Institute, "Survivable Network Technology," http://www.sei.cmu.edu/organization/programs/nss/surv-net-tech.html, Last Accessed: Nov 24, 2006.

[41] R. J. Ellison, R. C. Linger, T. Longstaff, and N. R. Mead, "Survivable network system analysis: a case study," *Software, IEEE,* vol. 16, pp. 70-77, 1999.

[42] S. Jha, J. Wing, R. Linger, and T. Longstaff, "Survivability analysis of network specifications," in *In Dependable Systems and Networks (DSN)*, pp. 613-622, 2000.

[43] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Computer,* vol. 27, pp. 60-69, 1994.

[44] Devellion Limited, "CubeCart: PHP and MySQL Shopping Cart," http://www.cubecart.com/, Last Accessed: Nov 24, 2006.