Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 51 (2007) 1334–1360

Computer Networks

www.elsevier.com/locate/comnet

Automated adaptive intrusion containment in systems of interacting services

Yu-Sung Wu [a], Bingrui Foo [a], Yu-Chun Mao [a], Saurabh Bagchi [a,*], Eugene H. Spafford [a,b]

[a] *Dependable Computing System Lab (DCSL) and CERIAS, School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907, United States*

[b] *School of Computer Science, Purdue University, 250 N. University Street, West Lafayette, IN 47907-2066, United States*

Available online 13 October 2006

Abstract

Large scale distributed systems typically have interactions among different services that create an avenue for propagation of a failure from one service to another. The failures being considered may be the result of natural failures or malicious activity, collectively called disruptions. To make these systems tolerant to failures it is necessary to contain the spread of the occurrence automatically once it is detected. The objective is to allow certain parts of the system to continue to provide partial functionality in the system in the face of failures. Real world situations impose several constraints on the design of such a disruption tolerant system of which we consider the following – the alarms may have type I or type II errors; it may not be possible to change the service itself even though the interaction may be changed; attacks may use steps that are not anticipated a priori; and there may be bursts of concurrent alarms. We present the design and implementation of a system named Adepts as the realization of such a disruption tolerant system. Adepts uses a directed graph representation to model the spread of the failure through the system, presents algorithms for determining appropriate responses and monitoring their effectiveness, and quantifies the effect of disruptions through a high level survivability metric. Adepts is demonstrated on a real e-commerce testbed with actual attack patterns injected into it.

Keywords: Automated adaptive intrusion response; Intrusion containment; E-commerce system; Survivability; Attack graphs

1. Introduction

Distributed systems comprising multiple services interacting among themselves to provide end-user functions are an increasingly important platform. Examples abound in the commercial domain and in the critical infrastructure domain, such as, e-commerce systems and Supervisory Control And Data Acquisition (SCADA), systems [1,2], respectively. A fundamental nature of the distributed systems is that they are built of multiple services, such as web service and authentication service, running on individual hosts communicating with each other through standardized protocols, such as SOAP.

* Corresponding author. Tel.: +1 765 494 3362; fax: +1 765 494 2706.

E-mail addresses: yswu@purdue.edu (Y.-S. Wu), foob@purdue.edu (B. Foo), maoy@ purdue.edu (Y.-C. Mao), sbagchi@purdue.edu (S. Bagchi), spaf@purdue.edu (E.H. Spafford).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.09.006

The huge financial stakes involved or the importance to homeland security make them prime candidates for computer security attacks. In addition, as the complexity of these systems increases in exponential order due to the burgeoning number of services and the increasing complexity of the individual services, the occurrences of accidental failures are also on an upswing. Henceforth in the paper, we will refer to deliberate (or malicious) failures, commonly called intrusions, while realizing that some of the discussion carries over to natural failures as well.

The motivations outlined above have led to substantial interest in securing distributed systems through prevention of failures. Prevention of failures is achieved through careful design and development of the components to eliminate most faults, i.e., bugs or vulnerabilities, followed by detection of impending failures achieved through exploitation of residual faults. The detection can be done by matching the patterns of events observed with known patterns – either patterns of expected system behavior (anomaly based signatures) or known failure patterns (misuse based signatures). However, it is widely believed that prevention cannot be the ultimate solution because despite the heroic efforts of developers, testers, and deployment specialists, few systems, if any, can claim to prevent *all* failures. This is especially pertinent when the systems have interfaces to external users, as all e-commerce systems and many SCADA systems do. For example, the 1998–1999 evaluation of several commercial and research intrusion detection systems (IDS) by MIT Lincoln Lab [21] identified that unknown attacks present a considerable challenge to these systems.

This reasoning has led to a focus on efforts to build *survivable systems* that can provide sustained operation of mission critical functions in the face of anticipated and unanticipated intrusions. This has spurred interest in the defense community as evidenced by DARPA’s OASIS program [22] and the follow-on Self Regenerative Systems (SRS) program [23]. Within the area of focus on Intrusion Response Systems (IRSs), two levels of sophistication of systems may be discerned. The first kind of IRSs is able to diagnose the components affected by the intrusion and alert the system administrator for long term remedial measures. However, as distributed systems become ubiquitous and they are often placed in environments difficult to reach for human intervention, automated tools for intrusion response gain importance. The next generation of IRS is able to automatically reconfigure, regenerate, and rejuvenate the system once an intrusion occurs. A common form of IRS is to gracefully degrade the functionality of the system when intrusions are detected using techniques well known in the fault tolerant literature – redundant components built with diversity in mind, and reconfiguration of the failed components. Our work falls in this second class of IRS. In this paper, we lay out the design requirements, present a structured methodology for the design of the IRS to meet the requirements, and show an instantiation of the IRS in a system called ADEPTS that provides resilience to a distributed e-commerce system.

Providing an IRS for a distributed system is very different from providing one for a stand-alone system. One needs to consider the interaction effects among the multiple services both to accurately identify patterns of the intrusions relevant to the response process and to identify the effectiveness of the deployed response mechanism. The rudimentary response mechanism often bundled with anti-virus or intrusion detection system (IDS) products overwhelmingly consider only immediate local responses that are directly suggested by the detected symptom. For example, a file being infected with a virus may cause the anti-virus product to quarantine the file and disable all access to the file (e.g., Norton and McAfee anti-virus products), or a suspect packet being flagged by a network IDS may cause the specific network connection to be terminated (e.g., Snort In-line). This is unsatisfactory since the response may be sub-optimal at best – greater effect may be achieved by deploying it at a site different from the detection, and inaccurate at worst – the response is ineffective since the intrusion goal has already been achieved and the response system plays a fruitless game of catch-up.

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, i.e., an intrusion. 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, we 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. Thus, we focus on the manifestations of the different attacks as they pertain to the services rather than the attack sequence itself. This leads us 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. We will use the term "*a node is achieved*" to mean the attack goal corresponding to a node in the I-Graph is achieved. This representation suggests the attack model that Adepts targets – multi-stage attacks where the different stages may affect different services in the distributed system. An increasingly common subset of this class of attacks is a multi-stage attack comprised of reconnaissance (sending query packets to find out the nature of the services in the payload system), sending malicious network packets (to exploit one or more vulnerabilities in the services), affect services moving from the front-end toward the back-end where critical data may be stored or critical services executed, and finally the clean-up phase to hide the footprints of the attack. Our target attack model *does not* include insider attacks, one-shot attacks that devastate the entire system, brute force denial of service attacks, and covert channels.

Now we outline the key design principles or design requirements that drive our work. The response choice in an IRS should be dynamically determined. The *set of responses* is statically determined while the intrusions may be unanticipated or the dynamic system conditions, such as the frequency of interaction between the services or the load on a service, may vary. 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. The IRS should be capable of providing its service in the face of unanticipated attacks. This is clearly motivated by the observation that the potential universe of attack sequences is infinite. Translating this to Adepts, one should not assume that the I-Graph is complete nor that there is a detector to flag whenever an I-Graph node is achieved. However, we do assume that the intrusion will ultimately have a manifested goal which is detectable. Any IRS needs to consider 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. In Adepts, we take the approach of co-existing with off-the-shelf detectors in the detection system and consider the problem of improving the accuracy of the detectors as orthogonal and outside the scope. Adepts can also make use of correlation based intrusion detection systems, such as Cids previously developed by us [3], that already improves the detection metrics. Adepts provides algorithms to estimate 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.

It is often a challenge for an IRS to handle multiple concurrent attacks incident on a system. The response mechanisms due to the different manifestations of the distinct attacks may interact in arbitrary ways. For example, the response taken due to one attack may make that due to a second attack redundant, or make it easier to proceed. It also becomes difficult to identify the effectiveness of a response when the different attacks are not identified and handled separately. Adepts provides an algorithm to use the factors of locality (spatial in the I-Graph or temporal) and causality (parameters of the packet, such as originating IP) to identify attack instances that need to be handled separately. It is tricky to define what constitutes two separate concurrent attacks, since they may originate from the same source by the same adversary. We bypass this argument by considering attack instances whose responses would not interfere as distinct and separate attack instances. An attack sub-graph is created from the I-Graph for every attack instance. The attack sub-graph is grown in a *Petri dish* as alerts come in and in parallel, it is matched against an *attack template library* of graphs to determine appropriate *reference response*s.

In this paper, our goal is not to present novel response mechanisms for specific classes of attacks. However, there are two characteristics of the responses used in Adepts that are germane to the discussion. In examining different attack scenarios, we uncovered that there are some which can be called *persistent*. This implies that the higher level attack goal, even if achieved, relies on the continual presence of the lower level attack goals. This leads to the insight that a valid response, which may otherwise be rejected due to being too late to prevent the node being achieved, can still be deployed for persistent attacks. Second, it is important to consider a lease for a deployed response, after which the response will expire and can be rolled back. In the absence of such a concept, the effect of the responses will be cumulative over the lifetime of the system, thus progressively degrading the system functionality. In Adepts, the responses have an expiry time, which would in all likelihood be a pre-configured property of the response or specified by the system administrator at a finer granularity. A response is deactivated when its expiry time lapses. Conversely, the expiry time can be extended by Adepts if it is determined that the symptoms of the failure that necessitated the response in the first place still exist. We believe this feature would be important in any IRS that is to operate autonomously over extended periods of time.

Overall, the goal of the paper is to present a structured methodology for automating response actions. Within this methodology, Adepts provides algorithms for determining the appropriate locations and choices for the response, how to deploy the response, and how to evaluate the effectiveness of a response. If we model the response action as a containment action, it has the effect of cutting edges in the I-Graph thereby preventing the higher level nodes from being achieved. This, in turn, is reflected through a containment boundary with the services inside the boundary being considered compromised or suspect and the services outside continuing to provide the critical functionality. The metric used to evaluate an intrusion tolerant system has to be carefully chosen. Low-level metrics, such as the latency of detection or false and missed alarm rates do not fully capture the effect of an intrusion on the system's functionality. We propose the use of the metric called *survivability* [24] for evaluating the effect of Adepts. We define it such that its value depends on the set of high-level system transactions that can be achieved (e.g., allow web store browsing) and the set of high-level system goals (e.g., keep users' private information secure) that are preserved in the event of an intrusion. A high level transaction depends on certain chains of interactions between multiple functioning services and preserving a high level goal implies thwarting certain intrusion goals from being reached.

The design of Adepts is realized in an implementation which provides intrusion response service to a realistic 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 compared with that of a baseline system.

Summarizing, the main contributions in this paper are:

1. We present a methodology for automatic and adaptive intrusion containment for systems of interacting services.
2. We show an instantiation of the methodology in a system called Adepts that is used to protect a complex three-tier distributed e-commerce system.
3. The system has means to learn from past attacks, discard spurious false alerts, and deduce missed alerts from the detection system.
4. The evaluation with real attack scripts shows improvement in the high level survivability metric for the payload system protected by Adepts.

The rest of the paper is organized as follows. Section 2 presents related research. Section 3 presents the design overview of Adeptsand lays out the fundamental data structures. Section 4 presents the algorithms for choice of responses. Section 5 describes the implementation and the e-commerce testbed on which Adepts is deployed. Section 6 presents the experiments and the results. Section 7 concludes the paper with mention of some future work.

2. Related research

In order to guarantee the requirement for continuous availability of the services in a distributed system, it is important to consider how the system reacts once the intrusion is detected. The majority of current IDSs stops with flagging alarms and relies on manual response by the security administrator or system administrator. This results in delays between the detection of the intrusion and the response which may range from minutes to months. Cohen showed using simulated attack scenarios that given a ten hour delay from detection, 80% of the attacks succeed and given 30 h, almost all the attacks succeed irrespective of the skill level of the defending system's administrator [5]. This insight has led to research in survivable systems engineering pioneered by CERT at CMU. Survivability is loosely defined as the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents [4,6]. The researchers identify the four key properties of survivable systems, namely, resistance to attacks, recognition of attacks and damage, recovery of essential and full services after attack, and adaptation and evolution to reduce effectiveness of future attacks. ADEPTS is motivated by the requirement to provide the second, third, and fourth properties.

A majority of the IRSs are *static* in nature in that they provide a set of preprogrammed responses that the administrator can choose from in initiating a response. This may reduce the time gap between detection and response, but still leaves a potentially unbounded window of vulnerability. The holy grail is an IRS that can respond to an attack automatically. A handful of systems provide adaptive responses. In [7], the authors propose a network model that allows an IRS to evaluate the effect of a response on the network services. The system chooses in a *greedy* manner the response that minimizes the penalty. There are some studies which present taxonomies of offensive and defensive responses to aid in selection of coherent responses in an automated response system [8–10]. Cooperating Security Managers (CSM) [11] is a distributed and host-based intrusion detection and response system. CSM proactively detects intrusions and reactively responds to them using the Fisch DC&A taxonomy [8]. It uses the suspicion level of the user as the only determining factor in the choice of response. A second system called EMERALD [12] uses two factors in determining the response – the amount of evidence furnished to support the claim of intrusion and the severity of the response. None of the systems uses record of the past performance of the intrusion detection system as measured by the incidence of false positives and false negatives. None keeps track of the success or failure of the deployed response nor provides a framework for easily incorporating these factors in the automated response determination. Another adaptive IRS is the Adaptive, Agent-based Intrusion Response System (AAIRS) [13]. 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. There is some previous work on protecting distributed systems against flooding based distributed denial of service (DDoS) attacks in an automated manner through rate limiting [14,15].

A significant volume of work in this domain has focused on using diverse redundant components in building intrusion tolerant systems [25–28]. The basic intuition is that the components have design diversity and are unlikely to share vulnerabilities that are exploited by an attack. Using the Sitar system as a representative example, the basic mechanism is reminiscent of active replication techniques. There are proxy servers which interface with the external world mediating access to the actual servers after passing the requests through some checks. The outputs from the servers are passed through validity checks and then voted on. Disagreement during voting acts as a trigger for the reconfiguration module to evaluate the intrusion threat and initiate reconfiguration, such as bringing in a different server. While diverse components were taken for granted in all of this work, more recent work performed under the DARPA SRS program has explored introducing diversity through synthetic means, such as compiler transformations [30–32]. An attractive feature of this approach is it does not concern itself with the way the attack is launched but only on the manifestation, namely divergence in the outputs from the replicas. This philosophy resonates with our philosophy in ADEPTS. An interesting approach of combining tolerance to natural and malicious faults was espoused in the large European Union project called MAFTIA [33].

Fault trees have been used extensively in root cause analysis in fault tolerant systems. They have also been used to a limited extent in secure system design [16,17]. We use an attack graph representation with nodes as intermediate goals since the same intermediate goals show up in several attack paths. Graph theoretic approaches to modeling the temporal nature of security attributes is found in [18,19]. The notion of privilege graphs introduced in [19] has some similarity to our I-Graph. However, they represent only attacks launched by escalating the privilege level of the attacker and the arcs are marked with weights representing the difficulty of the privilege escalation. The weights are dependent on several factors, such as the expertise and resources of the attacker, and therefore difficult to predict.

3. Process flow and fundamental structures in adepts

3.1. Overview

First we give an overview figure (Fig. 1) that shows the different phases in the operation of Adepts. In the course of the paper, we will explain each of the phases and refer back to this figure.

Throughout Adepts, three policy levels are used to control the behavior of the relevant algorithms – aggressive, moderate, and conservative. The three policies can be abstracted to represent a ratio of missed responses to false responses deployed, with the aggressive policy having the lowest ratio and the conservative policy having the highest ratio.

3.2. I-Graph

3.2.1. I-Graph structure

The I-Graph is used as the underlying representation for knowledge about intrusions, as they spread achieving progressively wider set of goals. In the I-Graph representation, each intrusion goal is represented by one node in the graph. The final goal of the intrusion may be disrupting some high level system functionality, such as "Denial of service achieved against the online store". This final step will be achieved through multiple small to moderate sized steps. A successful execution of a step is looked upon as achieving an intermediate intrusion goal and captured as an I-Graph node. The intrusion goals have dependency relationships between one another. For example, in order to corrupt the

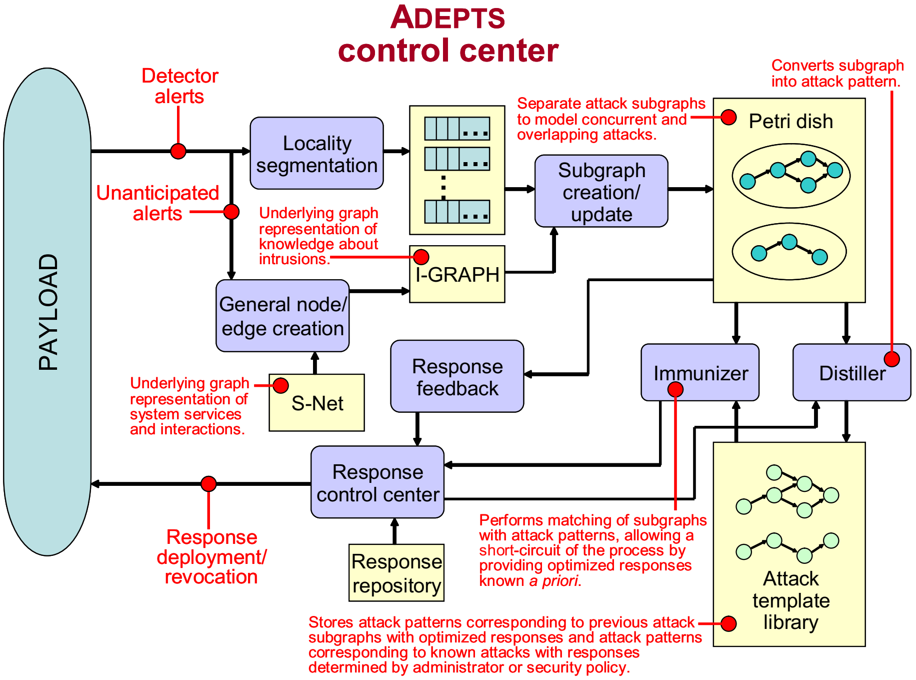

ADEPTS
control center
PAYLOAD
Detector alerts
Unanticipated alerts
Locality segmentation
Underlying graph representation of knowledge about intrusions.
I-GRAPH
General node/ edge creation
Underlying graph representation of system services and interactions.
S-Net
Separate attack subgraphs to model concurrent and overlapping attacks.
Subgraph creation/ update
Converts subgraph into attack pattern.
Petri dish
Response feedback
Immunizer
Distiller
Response control center
Response deployment/ revocation
Response repository
Performs matching of subgraphs with attack patterns, allowing a short-circuit of the process by providing optimized responses known a priori.
Attack template library
Stores attack patterns corresponding to previous attack subgraphs with optimized responses and attack patterns corresponding to known attacks with responses determined by administrator or security policy.

Fig. 1. Overview of the different phases of Adepts.

data in the backend database server, one may need to exploit a vulnerability in the front-end web server. The edges are used to model this kind of dependency. The *parents* of a node are the nodes reached by the outgoing edges of the node. They correspond to higher goals relative to the goal of the node. The *children* of a node are the nodes with outgoing edges to the node. They correspond to lower goals relative to the goal of the node.

In the I-Graph, edges are categorized into three types – OR, AND, and Quorum edges. For a node with incoming OR edges to be achieved, at least one of its child nodes needs to be achieved, while for AND edges, all the child nodes have to be achieved. For Quorum edges, one can assign a Minimum Required Quorum (MRQ) on it, which represents the minimum number of child nodes whose goals need to be achieved in order for the node with incoming Quorum edges to be achieved. Conforming to the traditional definition of quorums in fault tolerant systems, one may think MRQ as the minimum number of service replicas whose loss will affect the functionality of the service. An example fragment of the I-Graph used in our payload system, a distributed e-commerce system, is shown in Fig. 2.

3.2.2. I-Graph generation

A key issue in the usability of Adepts is the ease with which the I-Graph can be generated and updated as system configuration changes or new vulnerabilities are brought to light. We employ a *semi-automated* method called Portable I-Graph Generation (PIG) for this. PIG requires two inputs – vulnerability descriptions and system services description (SNet). Of the two inputs, the SNet is target system dependent. This is a directed graph, in which each node represents an individual service in the target system and an edge from node A to node B represents an *intrusion-centric channel*. An intrusion-centric channel means if A is compromised, then the intrusion can spread to B through the channel. An intrusion-centric channel may be of five kinds – (i) DOS channel: if the source service is subjected to a successful DoS attack, then the destination service can also be subjected to DoS; (ii) Network channel: there is a network data connection between A and B; (iii) Shared file channel; (iv) Shared memory channel; (v) Super channel: which combines the functionality of all of the above. The SNet is currently manually created for the target system, though in the future, some tool which can perform service discovery and interaction discovery can perform this task automatically. This is an active area of research especially in the industry, such as for IrDA and Bluetooth Service Discovery Protocol (SDP), Sun's Jini, and Microsoft's Universal Plug and Play (UPnP).

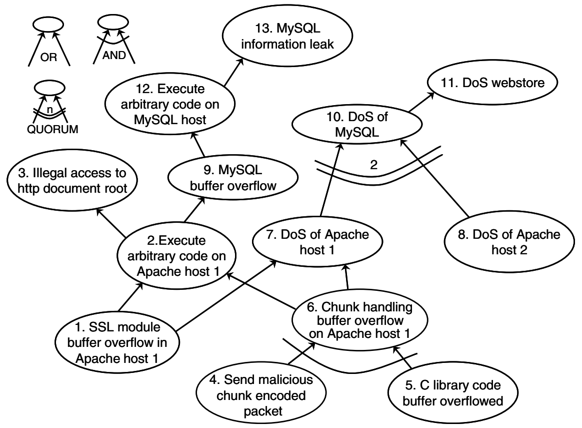

OR
AND
n
QUORUM
13. MySQL information leak
12. Execute arbitrary code on MySQL host
11. DoS webstore
10. DoS of MySQL
2
9. MySQL buffer overflow
3. Illegal access to http document root
2.Execute arbitrary code on Apache host 1
7. DoS of Apache host 1
8. DoS of Apache host 2
6. Chunk handling buffer overflow on Apache host 1
1. SSL module buffer overflow in Apache host 1
4. Send malicious chunk encoded packet
5. C library code buffer overflowed

Fig. 2. A section of the I-Graph from our deployed e-commerce environment.

The second input to PIG is the target independent vulnerability descriptions. Information on the vulnerabilities can be obtained by querying the common vulnerability databases, such as CERT, Bugtraq, and CERIAS-VDB. For use in PIG, the vulnerability is specified through four fields – (i) *Name*: which is primarily useful for human reference. (ii) *Affected service*: which gives the service(s) in the SNet affected by the vulnerability; (iii) *Manifestation*: this is a Boolean expression in disjunctive normal form composed of five elementary manifestations, namely, leaking of information, execution of arbitrary code, incorrect behavior of service, DoS, and service termination. (iv) *Dependent vulnerability and services*: which denotes the dependence on other vulnerabilities and services that have to be compromised to exploit this vulnerability. The vulnerability definitions are analogous to the virus definitions used in anti-virus products. They can be developed either by the Adepts developer or by a third party. The basic idea behind the I-Graph generation algorithm is that when a vulnerability description is read in, a corresponding node in the I-Graph is created, thus creating a one-to-one map. In the next step, the algorithm checks for nodes in the I-Graph that this newly created node can get connected to. For this step, it relies on information from both the SNet and the vulnerability descriptions to decide whether spread of the intrusion is possible from the newly created node to the other nodes and vice-versa.

It is noted that though temporary countermeasures are usually provided (before patches appear) in some vulnerability databases (e.g., CERT), they typically require disabling affected services. Without ADEPTS, disabling affected services when a vulnerability is discovered can be quite disruptive since a large number of services may have already been affected.

3.3. Attack sub-graphs

In ADEPTS, the I-GRAPH is made a read only data structure and is used to create attack sub-graphs corresponding to each attack instance. *Attack sub-graph* instances are created and modified during runtime to separately model concurrent and overlapping attacks. Concurrent attacks are defined to be attacks that occur around the same time and overlapping attacks are those for which the intersection of their affected I-GRAPH nodes is not null. By using attack sub-graphs, ADEPTS is able to handle such attacks in parallel and optimize its response to each attack. Each attack sub-graph is grown in a Petri dish as alert events are received. The use of attack sub-graphs is a departure from the design in the earlier version of ADEPTS [29] where all alerts were made to operate on the I-GRAPH structure itself. We will refer to the earlier version as version 1. There are several motivations for the current design. In version 1, multiple attack instances will incorrectly affect each other's response determination though the response at a given node may have different effects depending on which attack it is targeted at. For example, a response at a "Libsafe buffer overflow on Apache" node may be to kill the Apache process. This may be useful for an attack which tries to inject malicious code through the buffer overflow but not useful for a denial of service attack against Apache. Similarly, attack instances at different points in time will interfere with one another to different degrees as well. The different attack sub-graphs can also be processed in parallel by multiple threads on the same processor or different processors and the read-only nature of the I-GRAPH eliminates a synchronization bottleneck.

The pseudo-code for handling a new alert event is given in Fig. 3. For each alert that is received by ADEPTS, the alert is mapped to a node in the I-GRAPH and then the sub-graph creation algorithm determines whether the alert belongs to an existing attack (and therefore existing sub-graph) or whether it is from a new attack. If it is the former, the alert will be placed into the corresponding sub-graph and the sub-graph will be evaluated by the system for response determination as described in Section 4 (the call to CCIUpdate). If it is the latter, the algorithm will create a new sub-graph beginning from the node mapped to by the alert. Note that in the I-GRAPH, we have general nodes for mapping generic alerts for a service or generic alerts from a host machine (see Section 4.4). Therefore, under this design, all alerts will always be mapped to some node in the I-GRAPH.

In reality, it is difficult to accurately determine which attack instance an alert belongs to, though some effort has been made in [20] by clustering source IP addresses, destination IP addresses, source ports, destination ports, user accounts and

A Petri-dish P is used to house sub-graphs. When a new alert event E comes in, the following algorithm will be initiated to either push E into some existing sub-graph(s) or create a new sub-graph NS with E in it.

```
EnterDEvent(P, E) {
    bNeedNewSub-graph := true
    for each existing sub-graph S do
        if LocalityTest(S,E) is true then do
            bNeedNewSub-graph := false
            Add the nodes and edges induced by E into S
            CCIUpdate(S,E)

    if bNeedNewSub-graph is true then do
        Create new sub-graph NS
        Add node corresponding to E into NS
        Add NS into P
        CCIUpdate(NS,E)
}
```

Check whether alert event E is local to sub-graph S. Assuming I-GRAPH I and a user-defined threshold T.

```
LocalityTest(S,E) {
    D: = the node in the I-GRAPH which corresponds
         to event E.
    if D is NULL then return False
    for each node X in S
        H: = the node in the I-GRAPH which
             corresponds to X
        Dist := shortest distance between D and H
        If Dist < T then return (true, S)
    return false
}
```

Fig. 3. Pseudo-code for attack sub-graph creation when new alert event arrives.

initiated processes. To avoid this problem our algorithm uses locality of the alerts to reduce the uncertainty, and is designed such that inaccuracies are tolerated. All existing active sub-graph instances are considered to be possible choices for an alert. This is because when an attack is determined to have concluded or contained, the attack instance is removed. The *necessary* condition for an alert to belong to a sub-graph (which corresponds to a specific attack instance) is that the shortest spatial distance with respect to the nodes in the I-Graph (i.e., spatial locality) be within a user-defined threshold. The minimum spatial distance is the minimum pair-wise distance in the I-Graph between the node mapped to by the alert and the nodes in the sub-graph. In the case when none of the existing sub-graphs passes the LocalityTest, a new sub-graph is created to house the alert event.

4. Responses in Adepts

In this section, we present the different mechanisms in Adepts to choose the appropriate responses and the locations to deploy them after attack sub-graphs are created to handle unanticipated attacks, and to provide feedback to the responses.

4.1. Determining response locations

For any given node in the attack sub-graph we can consider there are two kinds of responses associated with it, one set associated with the outgoing edges which have the role of preventing higher level goals from being achieved, and the second set with incoming edges which have the role of preventing continued achievement of the node goal.

4.1.1. CCI computation algorithm

The goal of the algorithm is to determine, based on the received alerts from the detectors, which of the I-Graph goal nodes (and corresponding sub-graph nodes) are likely to have been achieved. Each detector provides confidence values for its alerts, termed *alert confidence*. If the detector does not provide an built in confidence value with the alert, then the alert confidence value is set to one. The alert confidence provided by a detector is then moderated based on the likelihood it is a false alarm. Adepts has a mechanism to determine this on a per-alert basis and will adjust the alert confidence as described in detail in the next section.

The *Compromised Confidence Index* (CCI) of a node in a sub-graph is a measure of the likelihood that the node has been achieved. It is computed based on the alert confidence corresponding to the alert that is mapped to the node and the CCI of its immediate children nodes. Mathematically, the CCI of a node is given by

$$\mathrm{CCI} = \begin{cases} \mathit{alert\ confidence}, & \text{no child,} \\ f'(\mathrm{CCI}_i), & \text{no detector,} \\ f(f'(\mathrm{CCI}_i, \mathit{alert\ conf.}), & \text{otherwise,} \end{cases}$$

$$f' = \begin{cases} \max(\mathrm{CCI}_i), & \text{OR edge} \\ \min(\mathrm{CCI}_i) & \text{AND edge} \\ \mathrm{mean}(\mathrm{CCI}_i | \mathrm{CCI}_i > \tau), & \text{Quorum edge and quorum met} \\ 0, & \text{Quorum edge and quorum not met} \end{cases}$$

where CCI_i corresponds to the CCI of the *i*th child and τ is a per node threshold.

The intuition behind our implementation is that for an OR edge, the node can be achieved if any of its children nodes is achieved and therefore the likelihood (due to its children) is the maximum of all of its children. For an AND edge, all the children nodes have to be achieved and therefore the likelihood is as much as the least likely child node. For Quorum edges, if the quorum is not met, then the higher goal is *not* achieved, but if met, the likelihood of it being achieved only depends on the children nodes that have achieved the quorum. The function *f* allows various weights to be assigned to determine the relative effect of the alert confidence and the children's CCI. The function for the current design is the statistical mean. Though our specific implementation is defined above, the framework can accommodate formal models as well. For example, a sub-graph can be considered to be a Bayesian network, and the conditional probability distributions derived from the edge relations (i.e., OR, AND, Quorum).

When new alerts arrive, the nodes corresponding to these alerts are put in a reorder buffer. Adepts enforces the condition that the difference between the time when Adepts is invoked with an alert to the timestamp on the alert put in at the host where the alert was generated has to be at least a given time bound. This implies alerts can be reordered when they sit in the reorder buffer before being passed to Adepts. The I-Graph is traversed in breadth-first-search (BFS) order starting from the nodes with the earliest alerts, and the CCIs of the nodes are computed until each reachable node has been traversed *at most once*. This prevents infinite cycling to occur even though there may be cycles in the I-Graph. The disadvantage of such a traversal (or even a DFS traversal) is that the traversal may lead to a node being processed before all its predecessor nodes are processed and therefore the CCI computed will be a lower bound. Thus, some causal relations between nodes may be lost. However, the alerts are usually temporally ordered according to the order in which the events occurred, thus the causal order is more likely to be obeyed in the CCI computation. Since the CCI of a parent node is dependent on that of its child nodes, a BFS traversal starting from the earliest node with an alert, rather than DFS, is more justified. A performance optimization is terminating the CCI computation when the CCI value goes below a threshold since this gives confidence that a response at nodes beyond this point is unnecessary.

As each node can potentially contain multiple alert events, which has an individual alert confidence, the alert confidence used to update the CCI is chosen based on policy. For an aggressive policy, the maximum alert confidence in the alert queue is used; for a moderate policy, the maximum of a subset of alert confidences based on the most recent alerts is chosen; for a conservative policy, the alert confidence corresponding to the most recent alert is chosen. No matter which case, we call the chosen alert event as the *active alert event*. Before a child node is used for CCI computation, Adepts will check whether the *active alert events* on the child and the parent obey a causal relation. Causal relation is defined as causality in the information such as packet source IP, destination IP, and process ID that is included in the alerts. By comparing the information, the causal relation between two alerts can be validated. For example, knowing that the SSH server listens on port 22, the CCI calculation for the event 'buffer overflow at SSH server' can depend on the CCI value from the child node 'detecting malicious packet bound for port 22'. On the other hand, the buffer overflow event cannot depend on the CCI value from the child node 'detecting malicious packet bound for port 80', since there's no causal relation between port 80 and the SSH server. Since a parent node can potentially have more than one child node, alternative child nodes will be used in case the causal relation is found not to match. If no child node can pass the causality validation, the parent node will be regarded as a leaf node in the CCI update path, and its CCI value will only depend on its own alert confidence. Impreciseness in the causality validation can be treated as missed/false alarms and can be inherently tolerated by the CCI update algorithm.

4.1.2. False alarm estimation

It is important that the response system not use the alarms from imperfect detectors as the only triggers. It is well known that intrusion detection has a tricky balancing act between the occurrence of false alarms and missed alarms. While anomaly based detectors typically have a high rate of false alarms, misuse based detectors have a high rate of missed alarms. The effectiveness of Adepts is therefore dependent on being able to work with the imperfect detectors. Adepts attempts to estimate when either of the two events has happened and either suppress its operation (false alarm) or trigger its operation (missed alarm). The false alarm detection algorithm attempts to detect false alarms for a given detector and a given node in the attack sub-graph by considering both present and past evidence. Our objective for detecting false alarms is to prevent needless invocation of Adepts and prevent useless responses from being deployed. As a result, it will mitigate DoS attacks targeting Adepts by injecting spurious alarms. To achieve this goal, based on the probability of an alert being a false alarm, the confidence of the alert is modified. Alerts with extremely low confidence are discarded, which allows obvious false alarms to be conveniently ignored. The algorithm is designed to be conservative in nature, that is, a lot of evidence is required to conclude that an alert is false, but not as much evidence is required to conclude that a false alert is actually true. This bias is easily controlled using two parameters (α, β). Also, the rate of increase (decrease) of the false alarm probability increases with successive false (true) alarms giving a convex (concave) curve. The shapes of the curves are also controlled by the parameters (α, β).

When alerts are passed to Adepts from the detectors, a false alarm probability (calculated a priori and initially set to 0) is recalculated for each alert. Based on this probability, the alert confidence is modified using the following equation

$$alert\_confidence = alert\_confidence * (1 - false\_alarm\_probability).$$

The recalculation of the false alarm probabilities is as follows. For each alert, the false alarm probability is

$$\mathit{false\_alarm\_probability} = \alpha \times \mathit{links\_probability} + (1-\alpha) \times \mathit{history\_probability}.$$

The *links probability* represents the lack of evidence linking the alert to other alerts. It is defined as

> 1 − max(probability that a link exists) = 1 − max(probability of temporal linkage, probability of spatial linkage).

The probability of spatial linkage is $1/(1+\gamma_q \times q)$, where γ_q is a scaling parameter and q is the minimum spatial distance between alerts in the same attack sub-graph. The probability of temporal linkage is similarly given by $1/(1+\gamma_p \times p)$, where γ_p is a scaling parameter and p is the minimum temporal distance between the alert and other alerts in previous iterations that occurred spatially close to the alert. The temporal distance is in terms of the number of invocations of ADEPTS that separates the two alerts. The *history probability* is a combination of past link probabilities, and it is recalculated given the present links probability using the following equation:

$$\mathit{history\_probability} = \beta * \mathit{links\_probability} + (1-\beta) * \mathit{history\_probability}.$$

4.1.3. Missed alarm estimation

The missed alarm detection algorithm first attempts to determine the possible locations of missed alarms. Then it uses the methods described in the false alarm detection algorithm to recalculate the missed alarm probability using other link evidence. This means that all the formulas used are the same as described in the previous section, except that the links probability is defined to be 1 − max(ratio of alert confidence to combined confidence of successors, ratio of alert confidence to combined confidence of predecessors). ADEPTS introduces alerts corresponding to the missed alarms into the system in the next iteration. This is more efficient than re-computing the CCI in the present iteration, as this may lead to multiple re-computations and can be exploited by an attacker. The alerts introduced will have their alert confidences as inversely proportional to their missed alarm probability.

The algorithm is run asynchronously with respect to the other algorithms, with the exception that it must run after the CCI computation algorithm because it uses the updated CCI values to determine the possible locations of missed alarms. In a nutshell, the algorithm determines the locations by doing a reversed CCI computation by traversing a sub-graph in reverse order, where for each incoming edge of a node

$$\mathrm{rCCI} = \left\{\begin{array}{ll} g(g'(\max(\mathrm{rCCI}_i), ac)), & \text{node has } d \text{ and } oe, \\ g(\max(\mathrm{rCCI}_i)), & \text{node has no } d, \\ g(ac), & \text{node has no } oe, \\ 0, & \text{otherwise}, \end{array}\right\},$$

$$g(y) = \begin{cases} y, & \text{AND edges}, \\ y \times \frac{\mathrm{CCI}_{\mathrm{child}}}{\sum \mathrm{CCI}_{\mathrm{children}}}, & \text{OR edges}, \\ y, & \text{Quorum } \mathrm{CCI}_{\mathrm{child}} > \tau_N, \\ 0, & \text{Quorum } \mathrm{CCI}_{\mathrm{child}} \leqslant \tau_N, \end{cases}$$

where rCCI_i corresponds to rCCI of the ith outgoing edge, "*ac*" is alert confidence, "*d*" is detector, "*oe*" is outgoing edge.

The function g' is the statistical mean of the two inputs, where the return value of the function g' represents the likelihood a node has been achieved based on evidence from its detectors and its parents. After the computations are completed, the locations of the missing alarms are those nodes for which all the following conditions are satisfied. (i) $f'(\mathrm{CCI}_i) > k \times$ alert confidence; (ii) $f'(\mathrm{CCI}_i) > \tau_{\mathrm{M}}$; (iii) $\max(\mathrm{rCCI}_i) > k \times$ alert confidence; (iv) $\max(\mathrm{rCCI}_i) > \tau_{\mathrm{M}}$, where k and τ_{M} are constants. A possible missed alarm location is determined based on whether the ratio of evidence of children being achieved to the direct evidence that the node has been achieved and the ratio of evidence of parents being achieved to the direct evidence that the node has been achieved is high (conditions (i) and (iii)). Conditions (ii) and (iv) are required to ensure that there is enough evidence to suggest a missed alarm occurred there.

4.1.4. Response set computation algorithm

The purpose of this algorithm is to determine the nodes where the current attack is and will most likely spread to. This will allow the response algorithm to deploy appropriate responses at those locations. Each sub-graph is traversed in reverse order of the CCI computation algorithm, continuing until all reachable nodes are traversed at most once. During the traversal, each node is labeled as one of: (i) *Strong Candidate* (SC), if CCI > τ; (ii) *Weak Candidate* (WC), if CCI $\leqslant \tau$ but further traversal across only AND edges can reach a SC node; (iii) *Very Weak Candidate* (VWC), if CCI $\leqslant \tau$ but further traversal across any type of edge can reach a SC node; (iv) *Non-Candidate* (NC), otherwise. If the CCI of a node is computed to be greater than τ, the system concludes the node has been achieved, where τ is a deployment parameter. Therefore the SC label on a node is a strong indicator that the node has been achieved, while the WC or VWC label indicates smaller likelihoods due to evidence from their parents.

Next, some nodes are placed in a *response set*, indicating to the response system where responses should be deployed. For an aggressive policy, all SC nodes, and WC and VWC nodes which have at least one immediate NC parent node are placed in the response set. For a moderate policy, all SC and WC nodes that have at least one immediate NC parent node are chosen. For a conservative policy, all SC nodes that have at least one immediate NC parent node are chosen. The aggressive, moderate, and conservative policies provide increasingly less disruption as well as less protection. It is important to note that in Adepts, responses may be deployed even in nodes for which no direct evidence of alerts are available. This is a key differentiator from local responses.

4.2. Response deployment

In this section we focus on the mechanisms needed to deploy a response. The deployment of the response is achieved by a *Response Repository*, a *Response Control Center*, and distributed *Response Execution Agents*.

4.2.1. Response infrastructure

The Response Repository stores the responses available for deployment in a payload system. Each response in the repository consists of an opcode and one or more operands, with wildcards allowed for each. The opcode is the response command, and the operands are the different parameters that need to be specified in order to execute the response. For example, the opcode for the response command of dropping incoming packets from a remote IP to a local port is DROP_INPUT, and the corresponding operands are REMOTE_IP and LOCAL_PORT. The opcode and the operands together make up a complete response command. The response structure allows Adepts fine-grained customization of the available responses.

The opcode is selected based on the ability of the opcode to cut off the *intrusion-centric channels* as defined in Section 3.2.2. The Response set computation algorithm (Section 4.1.4) sends to the Response Control Center the list of I-Graph nodes which are candidates for the deployment of responses. For each node, the Response Control Center selects a set of candidate response opcodes that can be used to prevent attacks from spreading via the node's outgoing intrusion-centric channels. The choice is determined by the type of the channel. For example, the file access based opcodes, such as DENY_FILE_ACCESS or DISABLE_WRITE, are selected as candidate response opcodes if an outgoing shared file channel is present.

After the opcodes have been chosen, the Response Control Center generates a list of complete response commands by collecting suitable operands. For this, it examines the alert events stored in the *alert queue* of the node and uses them to fill in the operands that are required by the selected opcodes. An opcode can be combined with multiple operands during this phase. For example, for an opcode KILL_PROCESS, the control center may extract PID#1 from alert event#1 and PID#2 from alert event#2, both in the alert queue. Then, the response command KILL_PROCESS PID#1, PID#2 is generated for subsequent evaluation.

4.2.2. Choosing responses

For each selected response command, the Response Control Center computes the *Response Index* (*RI*). The RI takes into the account the estimated effectiveness of the response to the particular attack, measured by the *Effectiveness Index* (*EI*), and the perceived disruptiveness of the response to legitimate users of the system, measured by the *Disruptiveness Index* (*DI*). The EI and the DI are both specific to the response command (opcode–operand combination) and the node in the I-Graph to which the response is mapped. The RI is given by $RI = a \cdot EI - b \cdot DI$, where a and b are deployment parameters.

Note that EI of an identical response command may differ for different attacks that map to different I-Graph nodes. For example, blocking port 65,000 or 16,660 may be useful against the *stacheldraht* DDoS attack but is unlikely to be effective against the TFN DDoS attack. The two attacks can be differentiated by their packet signatures. The control center chooses the response with the highest RI among the candidate responses, with a threshold being used to suppress a response that falls below it. The Response Execution Agents, one on each managed node, are used to deploy the responses.

If no response is chosen for a particular node, then the next higher level node is searched for possible responses. When Response Execution Agents on a particular compromised host have been disabled, responses will be taken at other hosts, as determined by the spread of the attack through the I-Graph. If an ineffective response is deployed, then by the response feedback process outlined in Section 4.6.1, the choice for future attacks will be biased against taking this response. Also the ineffective response will be deactivated according to the mechanisms outlined in Section 4.6.2.

4.3. Matching in attack template library

Adepts maintains an attack template library of attack patterns, similar in structure to the attack sub-graphs which are created at runtime. The attack patterns can be categorized into two types: *static attack pattern* and *raw attack pattern*.

The *static attack pattern* is created from previously seen attack patterns for which the "best" responses for each node in the pattern have been determined a priori by an expert system or by a security administrator. These responses would therefore be chosen over an automatically determined response if the matching score between the static pattern and the growing attack sub-graph exceeds a user-defined threshold. Additionally, the reference response may be determined by policy decisions made by a corporation or by a public body, e.g., sample data as a result of an incident may be automatically mailed to a central monitoring site for use in corporate-wide profiling and monitoring. Alternatively, if certain types of classified data are exposed, the system may notify appropriate investigators so as to begin an official investigation. This mechanism is powerful in letting Adepts learn its responses from domain specific knowledge, acquired knowledge over previous attack instances, or regulatory policy.

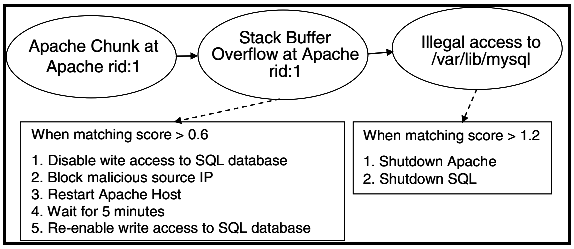

Apache Chunk at Apache rid:1
Stack Buffer Overflow at Apache rid:1
Illegal access to /var/lib/mysql
When matching score > 0.6
1. Disable wite access to SQL database
2. Block malicious source IP
3. Restart Apache Host
4. Wait for 5 minutes
5. Re-enable write access to SQL database
When matching score > 1.2
1. Shutdown Apache
2. Shutdown SQL

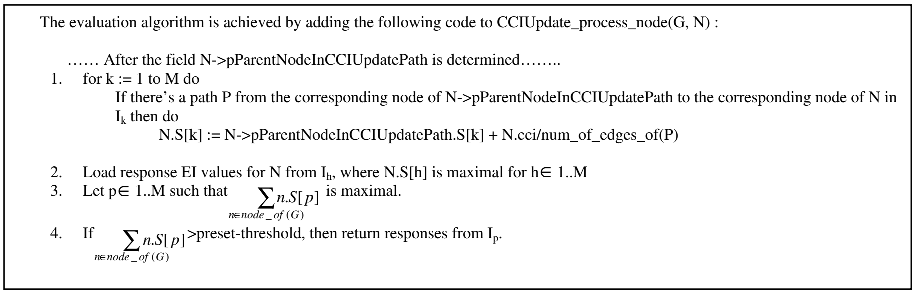
Fig. 4. Example of a static attack pattern and reference responses.

The matching between the static attack pattern and a growing attack sub-graph is handled by the *Immunizer* in Fig. 1. The algorithm for calculating the matching score is illustrated in Fig. 5. Fig. 4 shows an example of a static attack pattern.

Raw attack patterns are patterns automatically generated by adepts from the attack sub-graphs via the *Distiller* in Fig. 1. The raw attack patterns are used to store the pattern from an attack sub-graph and the responses which have been used in that attack sub-graph. Most importantly, the EI values for those responses are stored in the raw attack pattern as well. As mentioned in Section 3.3, the current version of Adepts features the distinction of EI values from different attack instances. Since the EI value is used to quantify the effectiveness of a response against a certain type of attack, it is necessary to make sure the attack sub-graphs corresponding to the same type of attack will be using the same copy of EI values. At the time when the first instance of a type of attack comes into the system, there's no corresponding raw attack pattern in the attack template library, and the default EI values will be used for the responses in the attack sub-graph which is being grown for that instance of attack. After the attack ceases (the attack sub-graph stops growing for a pre-defined expiration time), the attack sub-graph will be distilled into a raw attack pattern in the attack template library by the Distiller. What will happen when a later instance of the same type of attack comes into the system is that the Immunizer will match the growing attack sub-graph against the patterns in the attack template library. As it identifies the best-matched raw attack pattern, the EI values for the corresponding responses will then be loaded from that pattern into the growing attack sub-graph. At the time when the second instance of the attack stops, the Distiller will then merge the attack sub-graph back into the best-matched raw attack pattern. During the merge process, the Distiller writes back the new EI values into the attack pattern and optionally adds new nodes and new edges to the attack pattern, as some degree of non-determinism can be expected in a different run of the same type of attack.

4.3.1. Immunizer

The Immunizer matches a growing attack sub-graph against the patterns in the template library. As a suitable raw attack pattern is matched, the Immunizer uploads the EI values for the corresponding nodes from the attack pattern to the still



```
The evaluation algorithm is achieved by adding the following code to CCIUpdate_process_node(G, N) :

  …… After the field N->pParentNodeInCCIUpdatePath is determined……..
1. for k := 1 to M do
       If there's a path P from the corresponding node of N->pParentNodeInCCIUpdatePath to the corresponding node of N in $I_k$ then do
           N.S[k] := N->pParentNodeInCCIUpdatePath.S[k] + N.cci/num_of_edges_of(P)

2. Load response EI values for N from $I_h$, where N.S[h] is maximal for h∈ 1..M
3. Let p∈ 1..M such that $\sum_{n \in node\_of(G)} n.S[p]$ is maximal.
4. If $\sum_{n \in node\_of(G)} n.S[p]$ >preset-threshold, then return responses from $I_p$.
```

Fig. 5. Algorithm for calculating matching score.

growing attack sub-graph. On the other hand, when a suitable static attack pattern is matched, the Immunizer passes the pre-stored responses to the Response Control Center in Fig. 1 for guiding further choice. Now, assuming there are M patterns $I_1, I_2, \ldots, I_M$ in the attack template library. For each node N from a sub-graph G, we keep a vector S[1..M] which records the matching score for G with respect to each of the M patterns till the addition of node N. A match is concluded when the matching score exceeds a threshold. The algorithm for calculating the matching score is given in Fig. 5.

4.4. Handling unknown alerts

In a real-world deployment, it is quite probable that all possible attack paths have not been anticipated and therefore the I-Graph for the payload system is incomplete. Thus, Adepts would be unable to map an incoming alert from a detector to a node. To handle this situation, Adepts has the provision of a general node per host. The alert would be mapped to the general node for the host that is the destination of the attack. It is assumed, with reason we believe, that the host is easily deducible from the alert. Since the general node represents unknown vulnerabilities, it is connected to all others nodes related to the services running on the particular host. Thus the effect of a general node flagging will be felt through increased CCI for other nodes related to the same host. The responses attached to the general node form a pre-specified fixed set called the *general responses*. The general responses are the commands that would be possible to deploy with very little knowledge of the operands, such as killing a process (need process ID), shutting down a service (need service ID), or restarting a host (need host ID).

General responses are only applied to nodes which are associated with the important services in the system, for example, the database which stores customers' account information. This is to minimize the negative impact from applying general response with respect to the potential false alarms from the detectors. We are currently working on the construction of new nodes and new edges in the I-Graph for unknown attacks, which models the attack paths of the unknown attacks [34]. Knowing the attack path can help determine the potential impact from an alert. (e.g., seeing a buffer overflow at the web server does lead to a disruption in the backend database server). This would give Adepts the ability to deploy general responses on nodes which by themselves are not associated with important services but are being used as stepping stones for attacking some other essential services in the system. By understanding how the unknown attacks relate to other known attacks, we are also able to expand on the set of possible responses by including responses that cut links between the general nodes and extant nodes in the I-Graph.

4.5. Response chains and persistent attacks

In real-world attack scenarios, there exist attack actions whose success depends on the continued presence of a previous attack action. In I-Graph terminology, a higher level attack goal can only continue to be achieved, if the lower level goal also continues to be achieved. This means a persistent connection has to be maintained between the attack agents that achieve the higher and lower level goals for the attack action to persist successfully. An administrator can flag each I-Graph node that requires the lower level goals to continue to be met, as a *persistent node*. The connected persistent nodes form a persistent attack path.

When the Response Control Center sees that the I-Graph node on which a response is to be deployed is a persistent node, it performs an action different

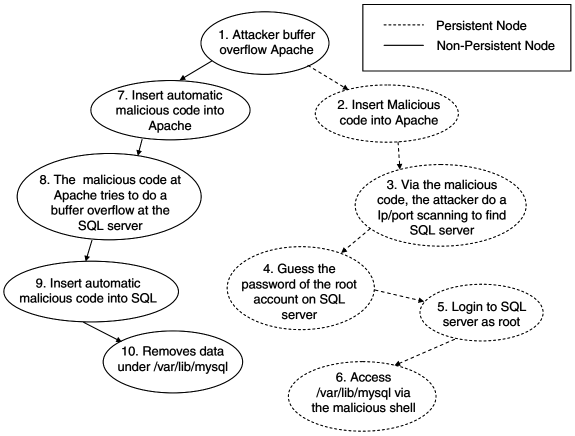

Persistent Node
Non-Persistent Node
1. Attacker buffer overflow Apache
7. Insert automatic malicious code into Apache
2. Insert Malicious code into Apache
8. The malicious code at Apache tries to do a buffer overflow at the SQL server
3. Via the malicious code, the attacker do a Ip/port scanning to find SQL server
9. Insert automatic malicious code into SQL
4. Guess the password of the root account on SQL server
5. Login to SQL server as root
10. Removes data under /var/lib/mysql
6. Access /var/lib/mysql via the malicious shell

Fig. 6. Persistent attack example.

from the algorithm outlined in Section 4.2.2. Instead of taking response on the node, it searches downward along the persistent attack path and identifies the non-persistent nodes that terminate the path. Then, the response against these non-persistent nodes are deployed. For an AND node, one path is searched, and for an OR node, all paths are searched. In practice, it is possible that the response taken at the first encountered non-persistent node does not succeed, and it may be desirable to deploy responses on the other nodes on the attack path. In Adepts, for the aggressive policy, responses will be deployed both at the top-level node with which the response algorithm is invoked and the lowest level non-persistent nodes.

An example of a persistent attack is shown in Fig. 6. Here, node 2, 3, 4, 5, and 6 are persistent nodes. On this attack path, it requires the attacker progressively embed malicious codes onto Apache and MySQL. These embedded codes act like relay stations such that the attacker is able to remotely control a root privileged shell on the SQL server.

4.6. Providing feedback to responses

Feedback to the response system is crucial for Adepts, providing the runtime mechanism to bias response choices in favor of those that have been effective in the past.

4.6.1. Varying EI

The feedback is provided by dynamically varying the EI of the response. After a response has been deployed, the feedback system checks to see if any active response action is deployed on an edge that can be used to reach a node in the currently computed response set. If such a response action exists, it is indication that the response action possibly failed and its EI is decreased.

The amount by which the EI of the response is decreased depends on whether the response is on an AND edge, OR edge, or Quorum edge to the node in the response set. If it is on an AND edge, then it is certain that the response failed and thus the node was achieved. Therefore, the EI is decreased by a fixed fraction for responses on all the edges. If the response is on an OR or Quorum edge, then the EI is decreased in the proportion of the CCI values of the nodes, the maximum decrease being the same as in the AND case. When a response expires or when an administrator manually deactivates a response, the EI of the response action is increased by a fixed percentage under the intuition that the response was successful since further alerts were not observed.

Referencing Fig. 2, suppose an active response is present on the edge between node 1 and 7, and node 10 is in the response set. Suppose the fixed fraction to decrease is α. Then for the active response, $EI_{\text{new}} = EI_{\text{old}} - \alpha \frac{\text{CCI}_7}{\text{CCI}_8+\text{CCI}_7} \frac{\text{CCI}_1}{\text{CCI}_6+\text{CCI}_1}$.

4.6.2. Deactivating responses

To reduce disruptiveness of an ineffective response, each response gets a pre-assigned time-to-live (TTL). After the response has been deployed, the Response Control Center (RCC) periodically checks the activated responses and deactivates it when they are expired. On the other hand, the RCC will extend the TTL of an expired response temporarily if it finds the response is successful and the payload system is still under attack. This is determined by investigating that the alerts mapped to the current node in I-Graph disappears, the immediate upper level node has not been compromised, but the immediate lower level node still gets alerts.

4.7. Complexity analysis

The worst case computational complexity follows directly from analysis of the algorithms presented in Sections 3 and 4.

We use the notations from Table 1. Sub-graph creation: $O(tsv)$; CCI update: $O(v^2ep)$; False alarm determination: $O(ta)$; Missed alarm determination: $O(ve)$; Response set computation: $O(ve)$; Optimal response selection: $O(roq)$; Varying EI algorithm: $O(cve)$.

Table 1
Notations for complexity analysis

Max number of existing alerts in a sub-graph	a
Number of existing sub-graphs	s
Max number of nodes in a sub-graph	v
Max number of edges in a sub-graph	e
Max number of existing active responses in a sub-graph	c
Number of new alerts	t
Max number of outgoing response from a node	o
Max number of alerts in an alert queue	q
Number of nodes in response set	r
Number of patterns in attack template library	p

Thus, the worst case time complexity of an execution of ADEPTS is: $O(tsv + v^2ep + ta + roq + p + cve)$.

The process of updating a raw attack pattern from a sub-graph can be done offline and is $O(v^3 + vp)$.

The low computational complexities of the algorithms allow the system to perform reasonable real time response decision and deployment. By allowing local responses from individual detectors, fast attacks can be stopped. Then ADEPTS has more time to reason about the attacks and consider whether the responses are reasonable. It can also override these local responses. A hierarchal arrangement containing multiple ADEPTS systems allows for scalability of the payload system. A performance evaluation of ADEPTS was not shown in the Experiments section due to space constraints, but the reader is referred to [29].

5. Implementation of ADEPTS and testbed

5.1. Description of e-commerce application

Fig. 7 depicts the testbed that we use for experiments. The payload system mimics an e-Commerce webstore, which has two Apache web servers running webstore applications, which are based on Cubecart (http://www.cubecart.com) 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 2.

Table 2
List of e-commerce transactions

Name	Services involved	Weight
Browse webstore	Apache, MySQL	10
Add to shopping cart	Apache, MySQL	10
Place order	Apache, MySQL	10
Charge credit card	Warehouse, Bank	5
Remote admin work	Variable	10

Table 3
List of e-commerce security goals

Illegal read of file (20)	Illegal process being run (50)
Illegal write to file (30)	Corruption of Apache docs/ MySQL db (70)
Unauthorized credit card charges (80)	Confidentiality leak of customer info (100)
Cracked administrator password (90)	Unauthorized orders created or shipped (80)

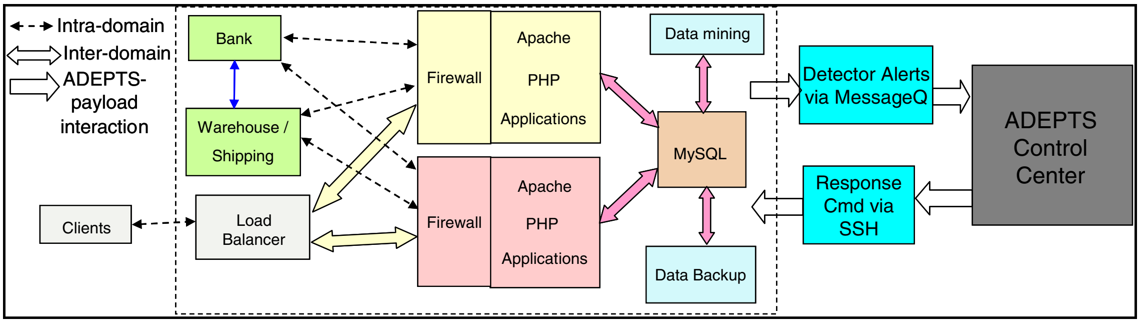


Fig. 7. Layout of e-commerce testbed.

We set certain security goals for the system, the complement of which are specified in Table 3, along with the weights. Thus adding the word "prevent" before each gives the goal. The attached weights for the transactions and security goals are used for survivability computation in Section 6.4. If the initial survivability value of the fully functional system is normalized to 1000 (an arbitrary choice) and due to an attack, it is no longer possible to perform administrative update to the system remotely, then the survivability will become $1000 - 10 = 990$. The weights are hypothetical but the magnitudes represent the relative importance to the overall system. In an actual deployment, these weights would be set by the system owner using methods such as analysis of the Total Cost of Ownership (TCO).

5.2. Detectors

For our testbed, multiple detectors which communicate with ADEPTS through secure channels are used. We use two off-the-shelf detectors – *Snort* and *Libsafe*, and create three home-grown detectors. Snort is used for detecting intrusion patterns in network traffic while Libsafe is used to detect buffer overflows in protected C-library calls. We create a kernel-based *File Access Monitor*, which can detect file access attempts of monitored processes and compare these access attempts against preset rules to detect illegitimate activity. Also, we create a *Transaction Response Monitor*, which monitors the transaction response time of the webstore using requests from the Apache Benchmark (http://httpd.apache.org/docs-2.0/programs/ab.html). Finally, there is an *Abnormal Account Activity Detector* at the Bank, which detects abnormal account activities such as excessive number of credit card transactions on one account. The detectors used are all imperfect ones, with the possibility of missed alarms and false alarms. In experiment 1, false alarms and missed alarms are artificially generated to test the detection algorithms. The frequency of false alarms is controlled manually and missed alarms are generated by non-deterministically discarding real alerts (or artificially setting a low alert confidence) based on a user-defined missed alarm probability for an alert. The detectors are not optimized for each attack scenario that the system is tested with. This is because the process is clearly labor-intensive and relies heavily on administrator expertise. For the off-the-shelf detectors, the rules are taken from the public distribution, else the rules are created by a researcher separate from the group that generates the attack scenarios.

5.3. Attack scenarios

The ADEPTS implementation is tested with different attack scenarios classified into three categories – illegal transaction, DoS, and leaking/corrupting information. Each attack scenario consists of a set of attack steps, with an ultimate high-level goal. The steps of the attack scenario are based on reported attacks and vulnerabilities from the common databases, such as, CVE and CERT advisories. For example, CERT advisory 2002–23 results in buffer overflow of the OpenSSL module in Apache leading to step 1 of scenario 8. Individual steps are chained together in a logical manner (post-condition of stage i matches the pre-condition of stage $i+1$) to come up with the multi-stage attack

Table 4
Attack steps for three static attack scenarios

Scenario 8
1. Buffer overflow Apache
2. Create a shell with Apache Privilege
3. Attacker issues crontab command to exploit a vulnerability which can create a root privilege shell
4. Root privilege shell created out of the vulnerable cron deamon
5. Attacker corrupts the data stored in web server document root
Scenario 4
1. DDoS attack via issuing huge amount of legal transactions (i.e., product search)
Scenario 1
1. Apache php_mime_split buffer overflow
2. 'ls' to list webstore document root and identify code regarding warehouse shipments
3. Send shipping request to warehouse, crafting request form to cause buffer overrun to fill form with victim's credit card number
4. Make unauthorized orders

scenarios. Each step of the attack scenario may be detected by none, one, or more of the detectors. We show in Table 4 and Fig. 8 one sample scenario from each category – Scenario 1 is placing unauthorized orders (illegal transaction), Scenario 4 is a DoS attack on the webstore, and Scenario 8 is vandalizing webstore (leaking/corrupting information).

Scenario 9, which is stealing/corrupting the SQL database (leaking/corrupting information) is different from the other three attack scenarios shown in Table 4. The difference is that Scenario 9 is a *dynamic attack scenario* while the other three are the *static* ones. In a dynamic attack scenario, an adversary proceeds through the scenario graph in a depth first manner and if any step is unsuccessful, possibly due to a successful deployed response, the adversary attempts an alternate path. Thus, a branch out point indicates multiple alternate strategies available to the adversary. Dynamic scenarios are used to better reflect the actions from a real-world adversary.

We also test ADEPTS with other attack scenarios involving buffer overflow attacks to steal client info, and other DoS attack scenarios entailing memory exhaustion in the Apache mime handling components. The entire I-GRAPH automatically generated by the PIG algorithm consists of 57 nodes and 1148 edges and is too large to be shown. A fragment of the I-GRAPH was shown in Fig. 2.

5.4. Response repository

Four types of response commands are included in the Response Repository – *general*, *file*, *network*, and *denial-of-service types*. The *general-type* commands can be deployed to block any types of *intrusion-centric* channels in the I-GRAPH, corresponding to the super channel. The other types of commands have a one-to-one map to the kinds of intrusion channels introduced in Section 3.2.2. The implementation of the file-type commands is achieved by using the *Linux Intrusion Detection System* (LIDS) version 2.2.0. The implementation of the network-type commands is performed by using *iptables*. The general type commands are killing a process and restarting or shutting down a service or a host. The file-type commands are to deny any access to a file, or selectively disable read, write, or execute access. The network-type commands are to block incoming or outgoing network connections, parameterized by source or destination port, IP, or protocol. The DOS-type commands are to limit the rates of various types of packets, such as SYN, ICMP echo, ICMP host not reachable, and SYN-ACK.

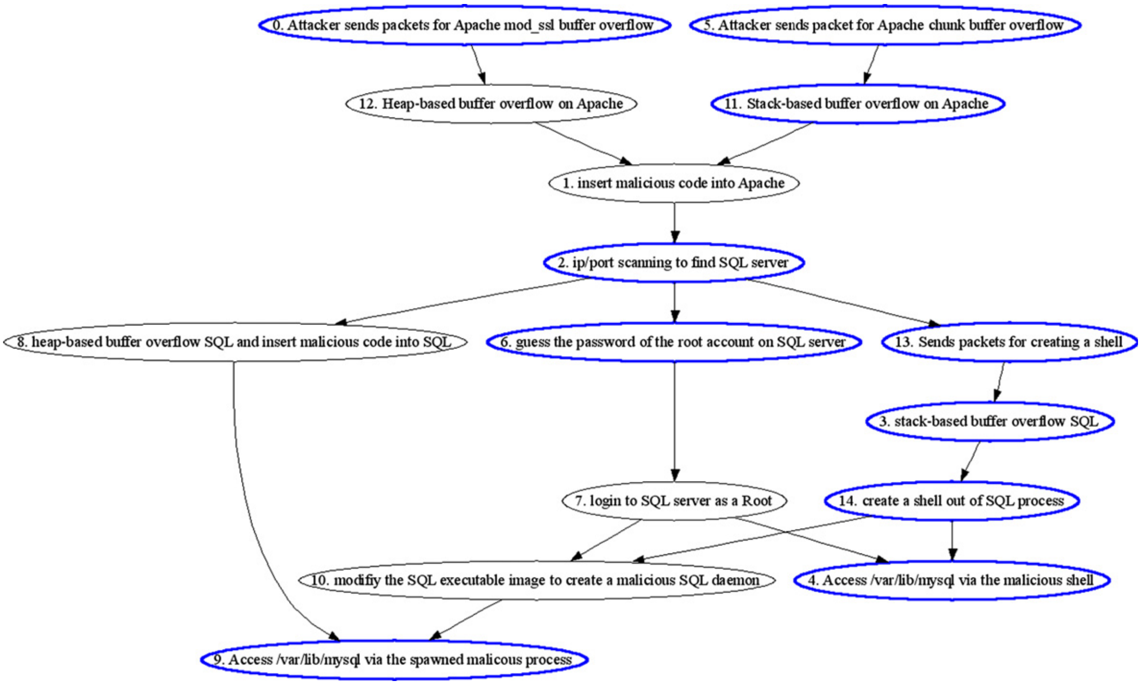

0. Attacker sends packets for Apache mod_ssl buffer overflow
5. Attacker sends packet for Apache chunk buffer overflow
12. Heap-based buffer overflow on Apache
11. Stack-based buffer overflow on Apache
1. insert malicious code into Apache
2. ip/port scanning to find SQL server
8. heap-based buffer overflow SQL and insert malicious code into SQL
6. guess the password of the root account on SQL server
13. Sends packets for creating a shell
3. stack-based buffer overflow SQL
7. login to SQL server as a Root
14. create a shell out of SQL process
10. modifiy the SQL executable image to create a malicious SQL daemon
4. Access /var/lib/mysql via the malicious shell
9. Access /var/lib/mysql via the spawned malicous process

Fig. 8. Example of a dynamic attack scenario (attack Scenario 9).

6. Experiments and results

We perform four sets of experiments on the e-commerce testbed using 9 synthetic attack scenarios, a mix of static and dynamic. The experiments have the goals of demonstrating the following – (i) Ability of the missed alarm and false alarm detection algorithms to identify inaccuracies in the detectors, (ii) Ability of Adepts to adapt the responses without and with reference attack patterns, (iii) Scalability of Adepts, (iv) Effect of survivability with time as multiple instances of an attack scenario impact the payload.

Due to the constraints of space, the results for a sample number of attack scenarios are shown. The results show the behavior of Adepts under a limited number of parameter configurations. They are not meant to bring out trends in the performance of Adepts or provide predictability under new attack scenarios or different parameter configurations. Further experimental work needs to be done for this. Comparing Adepts to other dedicated IRSs is difficult since they are not publicly available. For the experiments, survivability is defined as 1000 – Σ unavailable transactions – Σ failed security goals. Each response is pre-configured with an expiry time. When a transaction becomes unavailable or the security goal is violated, the survivability drops by its corresponding weight, which was given in Tables 2 and 3. Transactions become unavailable due to responses, such as rebooting a host, or attacks and they become available again when the response expires. Security goals may be violated due to the successful execution of an attack step. If a security goal is violated multiple times during an attack, then each violation causes a decrease in the survivability. After a run of an attack scenario is completed, any drop in survivability due to non-permanent security goal violations (e.g., running a malicious process only to reach another goal) is reversed. For all the experiments, the moderate policy is used.

6.1. Experiment 1: missed alarm and false alarm estimation

The objective of this experiment is to demonstrate the behavior of the false alarm and missed alarm algorithms given in Sections 4.1.2 and 4.1.3. The scaling parameter γ_q and γ_p are set at 0.2.

Fig. 9(a) shows how the false alarm probability fluctuates as real attacks are interspersed with false alarms. Repeatedly, sets of nine false alarms are generated successively at node 30 with attack Scenario 14 run after each set of false alarms. Scenario 14 is a static scenario where the attacker attempts to buffer overflow Apache and MySQL through their process stacks, to try and illegally access the MySQL database. Scenario 14 generates a real alert at node 30, along alerts at other nodes. The last set of false alarms has forty false alarms instead of nine. As can be seen, false alarms will cause the false alarm probability for alerts at node 30 to increase. The increase with the first set is the highest since there is no prior evidence of real attacks.

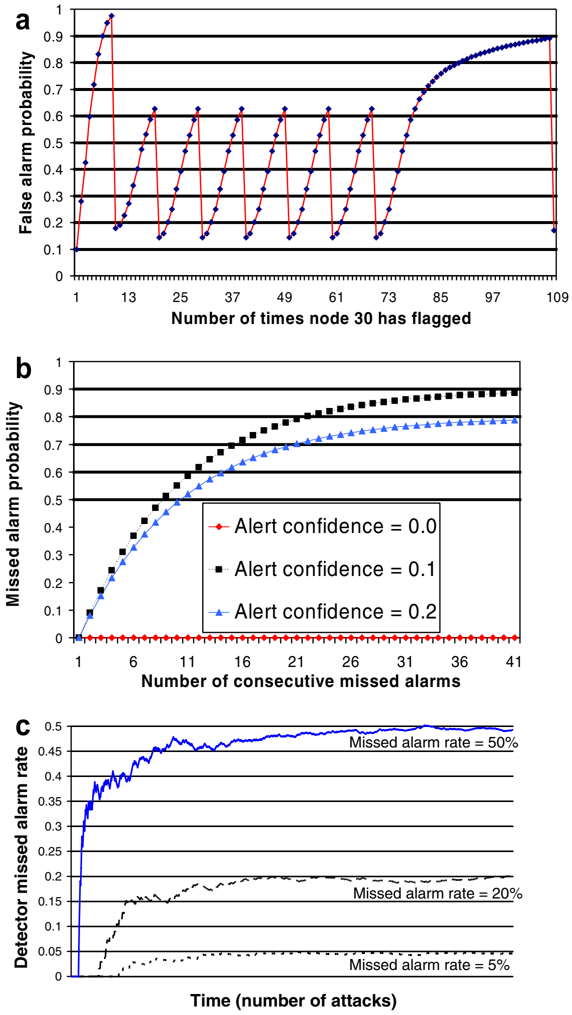


Fig. 9. Behavior of false and missed alarm computation algorithms.

Due to the variation of the (α, β) bias parameters (they are dependent on the present and past links probabilities), the rate of increase increases as more false alarms occur but decreases as it converges to one. The large drop when a real alarm occurs is due to the conservative nature of the algorithm which tries not to miss alarms for which response action is to be taken.

Fig. 9(b) shows how the missed alarm probability of a node in the present I-Graph varies when consecutive missed alarms are continuously generated. This is achieved by repeatedly running attack Scenario 14 with node 30 having hardwired alert confidences of 0.0, 0.1, and 0.2. Due to the highly connected nature of the I-Graph, the compromised child node of node 30 is also connected to the compromised parent node of node 30. This results in a failure to detect a missed alarm when the alert confidence is 0.0 (i.e., no alert occurs at node 30). This is explained by the fact that a compromised child node can lead to a parent of node 30 being compromised bypassing node 30 because of the existence of an edge between them. Therefore it is not possible for the algorithm to determine that there was a missed alarm at node 30 without any other evidence. If the I-Graph was not that connected, or probabilities were assigned to each I-Graph edge based on the likelihood of traversal, then it is very likely that the completely missed alert (alert confidence 0.0) would be detected. Only when the alert confidences are small non-zero values were missed alarms detected. The growth rate is inversely proportional to the alert confidence. The missed alarm probability can grow to a maximum 1-alert confidence.

Fig. 9(c) shows the Libsafe detector's calculated missed alarm rate as attack Scenario 14 is repeatedly executed with varying missed alarm rates. This rate is calculated by taking the number of times Adepts concludes there is a missed alarm with probability greater than 0.5 and dividing it by the number of times the attack scenario has been run. This mimics the situation where a detector is unpredictable and misses a fraction of alerts corresponding to different variants of an attack. Every time Scenario 14 is run, the alert confidence of node 30 is set to 0.1 with a probability equal to the missed alarm rate (5%, 20%, 50% in the experiments) and to 1.0 otherwise. As we can see, the missed alarms are detected with regularity, resulting in a calculated missed alarm rate that asymptotically tends to the actual missed alarm rates. For the experimental setting of Adepts, only when the missed alarm probability is greater than 50% will a possible missed alert be considered an actual missed alert. As a result, the initial missed alarm rate is at 0% and gradually grows later.

6.2. Experiment 2: adaptation of response action

In this experiment we demonstrate the adaptation in Adepts in taking more appropriate responses as multiple instances of an attack are observed in the system. Recollect that an attack sub-graph is induced from the I-Graph for each attack instance. After the attack instance ceases, the sub-graph is distilled into a raw reference attack pattern and the state regarding the effectiveness of the responses is maintained in this pattern.

In the experiment, four instances of attack Scenario 9 shown in Fig. 8 are executed. The instances are not identical since the dynamic attack Scenario 9 allows for diversity of paths. Instance 1 and 3 follow the same attack steps while instance 2 and 4 follow the same attack steps. Raw attack pattern 1 is created after instance 1 and reused in instance 3, while raw attack pattern 2 has the same role for instances 2 and 4.

The steps for the attack instances are as follows. Attack instances 1 and 3 have the steps: S5 – Attacker sends packet for Apache chunk buffer overflow; S11 – Stack-based buffer overflow on Apache; S1 – Insert malicious code into Apache; S2 – IP/Port scanning to find SQL server; S13 – Send packets to SQL server for creating a shell; S3 – Stack-based buffer overflow SQL; S14 – Create a shell out of SQL process; S4 – Access /var/lib/mysql via the malicious shell. For attack instances 2 and 4, the steps are: S0 – Attacker sends packets for Apache mod_ssl buffer overflow; S12 – Heap-based buffer overflow at Apache; S1, S2 – Same as above; S6 – Guess the root password on SQL server; S7 – Login to SQL server as a root. S10 – Modify SQL executable image to create a malicious SQL daemon; S9 – Access /var/lib/mysql via the spawned malicous process.

In Table 5, we give the mapping between the numeric IP address to the symbolic address used subsequently in Table 6. Table 6 gives the notation for responses used in the testbed. In Table 7, we show the different responses taken after each instance of the attack. The second column gives

Table 5
Placement of testbed services(symbolic addresses are used subsequently)

Client	IP_C:128.10.247.110
Apache Replica 1	IP_A:128.10.247.105
MySQL Server	IP_M:128.10.247.106

Table 6
Explanation of the responses for attack Scenario 9

R0	iptables -A INPUT -p tcp -j DROP -s IP_{C} – dport 80	Block attacker's IP from accessing port 80 on Apache Server Replica 1
R1	./lids-file.sh IP_{A} READ /usr/local/apache2/bin/httpd	Make Apache Replica 1′s image read-only
R2, R5, R8, R9[a]	iptables -A INPUT -j DROP -s IP_{C}	Block attacker's IP from accessing Apache Server Replica 1
R3, R7	restart.sh IP_{A} /usr/local/apache2/bin/httpd	Restarting Apache Server Replicate 1
R4	iptables -A INPUT -p tcp -j DROP -s IP_{C} – dport 443	Block attacker's IP from accessing port 443 on Apache Server Replica 1
R6	restart.sh IP_{M} /usr/sbin/mysqld	Restart MySQL Server

[a] Although R2, R5, R8, R9 are all the same, they are actually four independent responses deployed in the four instances of Scenario 9.

Table 7
Response adaptation for attack Scenario 9 without static attack pattern matching (Experiment 2)

Instance of attack	(Response, EI)	(Response taken, after which step) ($S\|F$)	Steps executed before AS stopped
I1	Before: (R0, 1.1); (R1, 1.1); (R2,1.1); R3(,1.1) After: (R0, 0.935); (R1, 0.935)	R0,11 (F) R1,11 (F) R2,2(S) R3,2 (S)	S5 ⇒ S11 ⇒ S1 ⇒ S2
I2	Before: (R4, 1.1); (R5, 1.1); (R6, 1.1) After: (R4, 0.935)	R4,2 (F) R5,9 (S) R6,9 (S)	S0 ⇒ S12 ⇒ S1 ⇒ S2 ⇒ S6 ⇒ S7 ⇒ S10 ⇒ S9
I3	Load EI values from Raw Attack pattern #1 EI(R1): 0.935 ⇒ Another response as R1 is less favorable in this attack instance. EI(R3). 1.1 ⇒ Set R7′s EI to 1.1 because R7 is the same as R3 Before: (R7, 1.1); (R8, 1.1) After: (All responses are successful. No decreasing of EI values occurs.)	R7,11 (S) R8,11 (S)	S5 ⇒ S11
I4	Load EI values from Raw Attack pattern #2 EI(R4): 0.935 ⇒ Another response as R4 is less favorable in this attack instance. EI(R5): 1.1 Before: (R9, 1.1) After: (All responses are successful. No decreasing of EI values occurs.)	R9,2 (S)	S0 ⇒ S12 ⇒ S1 ⇒ S2

the tuples with the responses and the EIs, both before and after the attack. The third column gives the response that was taken and if it was a success or a failure. Only a response that is deployed has its EI changed. The fourth column gives the steps in the attack instance that were executed before it was contained and therefore the attack sub-graph (AS) creation was stopped. After attack instances 1 and 2, the two raw attack patterns are created in the template library, which are shown in Fig. 10. In instance 2, responses R5 and R6 are noted as successful because they prevent data on the SQL Server from being accessed henceforth, though the attacker's goal of accessing some data has already been achieved. After instance 3, a more precise and effective set of responses is chosen using the raw pattern and the attack is stopped two steps ahead compared to instance 1. Similarly for instance 4, a more effective response R9 is chosen and the attack is stopped four steps ahead compared to instance 2.

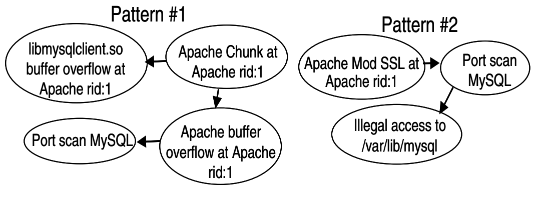

Pattern #1
libmysqlclient.so buffer overflow at Apache rid:1
Apache Chunk at Apache rid:1
Port scan MySQL
Apache buffer overflow at Apache rid:1
Pattern #2
Apache Mod SSL at Apache rid:1
Port scan MySQL
Illegal access to /var/lib/mysql

Fig. 10. Raw patterns #1 and #2 after instance 1 and 2 of attack Scenario 9.

As mentioned earlier in Section 4.3, one can populate static attack patterns with optimal responses in the attack template library. For this part of the experiment, we build a static attack pattern as shown in Fig. 11. The associated responses are shown in Table 8. We use a preset threshold 0.5 for the threshold parameter in the algorithm outlined in Fig. 5.

For this part of experiment, we still use Scenario 9 and we take the same attack path as the one used in Instance 1. It is observed that with the pre-configured static attack pattern, the attack is stopped one step earlier when compared to the result of instance 1 in Table 7. The recovery step of re-enabling access to "/var/lib/mysql" is also advantageous since it improves the system survivability. For demonstrating the matching with static attack pattern even better, another instance of Scenario 9 is executed and the successful response "Reboot Apache's host machine" is suppressed for demonstration. The result in Table 9(column 2) shows that the attack moves one step further and the responses stored on the node [34. Port scan MySQL] are deployed in which both the Apache server and MySQL server are both shut down.

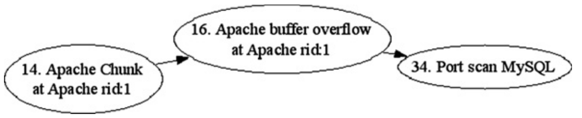

14. Apache Chunk at Apache rid:1
16. Apache buffer overflow at Apache rid:1
34. Port scan MySQL

Fig. 11. Static attack pattern with optimized responses for experiment 2.

Table 8
Responses associated with each node of the static attack pattern in Fig. 11

[16. Apache buffer overflow at Apache rid:1]	[34. Port scan MySQL]
– iptables -A INPUT -j DROP -s IP_C –./lids-file.sh IP_M DENY "/var/lib/mysql" – Reboot Apache's host machine – Wait for 15 min before deploying next response for this node. – Re-enable access to "/var/lib/mysql" on IP_M	– Shutdown IP_A – Shutdown IP_M

6.3. Experiment 3: scalability of Adepts

The design principle of creating multiple attack sub-graphs gives the advantage of handling incoming alerts and performing responses in parallel at the level of sub-graphs. We performed scalability experiments to bring out the benefit as the number of concurrent alerts increases. A concrete way of showing the benefit is to show comparative performance with the earlier version of Adepts (referred to as Adepts version 1 or Adepts 1 to distinguish it from the current Adepts version 2). Adepts 1 worked with the entire I-Graph and therefore could not make use of the parallelism. For the experiment, we synthesized 8 random I-Graphs, each with 700 nodes and 1050 edges differing in topology. For each run of this experiment, we insert a given number of concurrent alerts into Adepts. We then measure the time for processing them measured as the time between receiving the alerts and determining the nodes for responses. It is assumed that there exist enough computational resources to work on the parallelized parts of the computation in parallel. The result plots and detailed discussions are in Appendix. The important trend is that the growth in the number of parallelizable sub-graphs with the number of concurrent alerts is sub-linear since the spatial locality algorithm tends to cluster multiple alerts in the same attack sub-graph. With a small number of alerts (say, 50), Adepts 2 performs only slightly better than Adepts 1. This is due to the inherently higher constant overhead of Adepts 2 that nullifies the performance gain from limited parallelization of 3-6 generated sub-graphs. However, from around 100 alerts, Adepts 2 starts to significantly out-perform Adepts 1.

6.4. Experiment 4: survivability of E-commerce testbed

In this experiment, the objective is to show how the survivability of the e-commerce system is affected by repeatedly stressing the system through

Table 9
Response selection with matching against static attack pattern

Instance of attack scenario	Matching score, step at which match is successful	Step: responses taken ($S\|F$)	Steps executed before AS is stopped
I1	0.64, S11	S11 : R1(S), R2(S), R3(S), R4(S) … after 15 min R5(S)	S5 ⇒ S11 ⇒ S1
I2 (Response "Reboot Apache machine" is suppressed)	0.64, S11 0.72, S2	S11 : R1(S), R2(S), R3(F) S2 : R6(S), R7(S)	S5 ⇒ S11 ⇒ S1 ⇒ S2

the injection of successive instances of a given attack scenario. The results of running three scenarios are shown, where static Scenarios 1 and 8 cover two of the three general attack categories (DoS attacks are not shown here) and Scenario 9 is a dynamic scenario. The survivability is measured with and without ADEPTS. The static attack template library was kept empty. The initial survivability value without ADEPTS is arbitrarily fixed as 1000 while with ADEPTS it is 1010, so as to provide clarity in the graphs through non-overlapping plot lines. The exact survivability value does not carry any inherent meaning. Instead, the values are supposed to be compared against each other to denote the magnitude of the effect on survivability for different instances. In the graphs displayed, when the survivability returns to its initial value, it means that a single instance of an attack scenario has ended and responses that were deployed have expired. Permanent violations of security goals (e.g., illegal transaction, corruption of a database) will not result in the survivability returning to its initial value unless the administrator resets/restores/repairs the system. Multiple violations of a security goal are assumed to be from different attackers and therefore cause a decrease in survivability multiple times.

In Fig. 12(a), attack Scenario 1 is executed twice by different attackers. With ADEPTS, the response (restart httpd) deployed during step 2 of the attack scenario prevents the leakage of system information regarding warehouse shipments. This results in the termination of the attack scenario and a return to the initial survivability after the response has expired. This contrasts with the survivability without ADEPTS, which further degrades due to the continuing attack that finally results in unauthorized orders being made. The survivability degrades further because a separate illegal transaction has occurred in the second instance. Since the response was effective in the first instance, it is deployed again in the second instance. Table 10 shows the cause of the survivability drop during an instance of Scenario 1. In Fig. 12(b), attack Scenario 8 is executed five times. Due to the ineffective nature of the responses deployed by ADEPTS initially, the survivability degrades similarly (a little worse due to deployment of ineffective responses) to the system without ADEPTS. Survivability returns to the initial value during the first three instances because manual intervention occurs, that is, an administrator repairs the system. During the fourth and the fifth instances, due to the feedback from earlier instances, a relatively disruptive response of rebo-

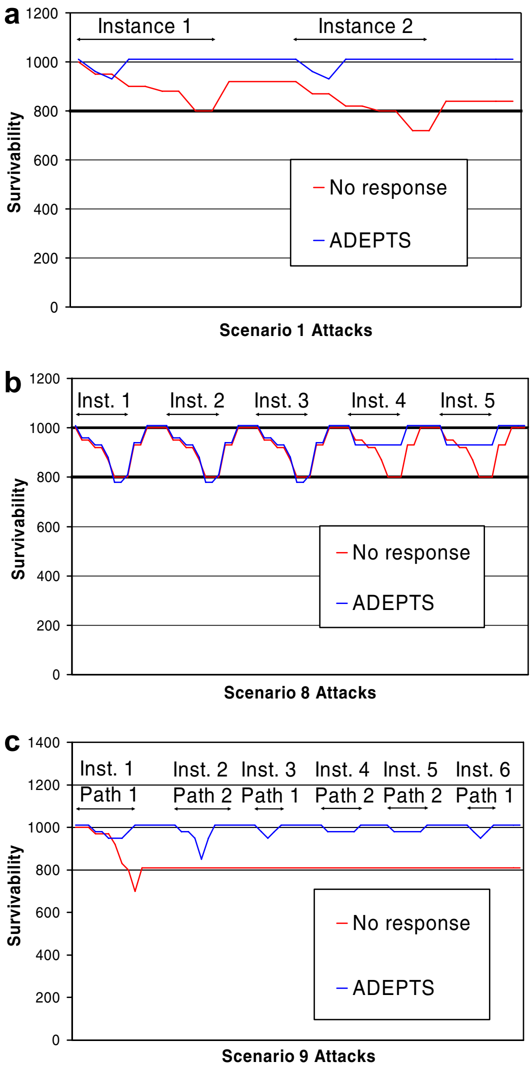


Fig. 12. Effect of attack scenarios on survivability.

Table 10
Cause of survivability drop with and withoutADEPTS in Scenario 1

Cause of survivability drop in Scenario 1 [Penalty] Without ADEPTS	Cause of survivability drop in Scenario 1 [Penalty] With ADEPTS
Compromised Apache invoking unauthorized program (bin/bash) [−50]	Compromised Apache invoking unauthorized program (bin/bash) [−50]
Compromised Apache invoking unauthorized program (bin/ls) [−50]	Restart/usr/local/apache2/bin/httpd [−30]
Illegal read in /usr/local/apache2/htdocs [20]	
Illegal order created [−80]	

oting the Apache host machine is deployed much earlier, resulting in an effective termination of the attack scenario. The webstore transactions are unavailable for the period when the response is active, resulting in a lower survivability.

In Fig. 12(c), dynamic attack Scenario 9 is executed six times. Due to the dynamic nature of the attack, different optimal responses are learned for different attack paths taken by the attacker. Two different attack paths are tried in this experiment. That is why an effective response in the first instance does not apply to the second instance. As more instances occur, the optimal responses are determined based on feedback. For path 1, because the attack does not require a persistent connection from outside the network, the initial responses that block incoming packets fail. Due to the earlier failures, another response that restarts the http daemon is deployed. The response is effective in stopping the attack because a clean copy of the daemon will be running after the restart. Through feedback,ADEPTS deploys this response earlier in instance 3 and instance 6 when the attacker uses path 1 again. The survivability degradation is still the same, but by deploying the response earlier, the likelihood that the malicious code is successfully injected is minimized. The initial steps through path 2 consist of causing a buffer overflow using the heap, which is undetectable by the available detectors in the system. This allows the attacker to compromise Apache silently. Then the attacker determines the IP address of the MySQL server and the port it is listening to. This is detected, but the response of blocking incoming TCP packets from the attacker's IP to port 443 fails because the attacker is using another port to communicate with the malicious Apache process. The attacker then buffer overflows the heap of the MySQL daemon using another vulnerability, and this is undetected. Then the attacker illegally accesses /var/lib/mysql and is detected. The effective response deployed is to restart the MySQL daemon. This relatively late response results in a significant drop in survivability. In instance 4 and instance 5 when path 2 is repeated, instead of blocking packets specific to port 443, the effective response of blocking all incoming packets from the attacker is deployed. The result is a smaller drop in survivability. Without ADEPTS, the survivability only degrades once because the attack is successful and the database is corrupted and not repaired.

7. Conclusions

In the paper we have presented the design and implementation of an automated intrusion containment system called ADEPTS. ADEPTS uses a directed graph representation called I-GRAPH to model the spread of the failure through the system. It provides algorithms to determine the possible path of spread of the intrusion, appropriate services where to deploy the response, and appropriately choose the response. ADEPTS creates attack sub-graphs from the I-GRAPH for each attack instance and processes each sub-graph independently, thus making it scalable with the number of alerts. Historical information learned through previous attack instances as well as domain specific knowledge of optimal responses to well-known attacks and regulatory policies can be used to guide the response choice. ADEPTS is demonstrated on an e-commerce system with real attack scenarios. The effect on the system is measured through a high level survivability metric which captures the effect of transactions that can be supported as well as system goals that are preserved under the attack. The effect of the attacks on survivability is shown in ADEPTS and in a baseline system with no intrusion response.

A key question which still remains to be answered is how efficient ADEPTS is. Considering there can be other automated response systems for distributed systems in the future to be compared with ADEPTS, we are now looking at the problem of defining analytically the idealized form of an automated response system and try to compare ADEPTS against it. Specifically, this will lead to a better understanding of the limitations of ADEPTS and also a more clear understanding of the advantage of ADEPTS with respect to other forms of automated response (e.g., local responses which function independently).

Seeing that unanticipated attacks are inevitable for real-world distributed systems, it is also interesting to see how well ADEPTS reacts to them. After all, one of the selling points of using automatic response is the advantage of being able to respond to zero-day attacks automatically and efficiently. The approach outlined here can handle such attacks, but in a very sub-optimal manner. We are now looking at the abstraction of attack goals such that unknown attacks which are similar to known attacks can still be effectively handled by ADEPTS [34].

Ideally, we would like to see ADEPTS adapt itself to an attack such that the eventual response actions with respect to that type of attack would be comparable to the best possible response actions that one can take based on the universe of response opcode–operand combinations in ADEPTS' database. So far we have shown the positive result from ADEPTS adaptation on some example attack scenarios. Empirically, we will need to extend the range and class of the attack scenarios. We need to see whether the existing or the future improved adaptation mechanisms are effective in yielding better responses over time with respect to attack scenarios from a richer sample set. Potentially, we may be able to provide a theoretical analysis of the adaptation mechanism as this would provide a better guarantee with respect to the unbounded space of all possible attacks. The detailed plan and some of the preliminary result on these pursuits is available in [35].

Appendix. Scalability experiment

In this experiment, we examine the performance benefits that accrue from this capacity for parallelism. The benefit is brought out by comparing the performance of ADEPTS with the ability to handle multiple independent attack instances with multiple attack sub-graphs, one for each attack instance, against a version of ADEPTS (referred to as version 1 in Section 3.3) that lacks this ability and therefore operates on the entire I-GRAPH. For the exposition in this section, we will refer to the current version of ADEPTS as ADEPTS 2 and the earlier version as ADEPTS 1.

In this experiment, we synthesized 8 random I-GRAPHS, each with 700 nodes and 1050 edges differing in topology. For each run of this experiment, we insert a given number of concurrent alerts into ADEPTS. We then measure the time for processing them measured as the time between receiving the alerts and determining the nodes for responses. It is assumed that there exist enough computational resources to work on the parallelized parts of the computation in parallel.

From Fig. 14, we see that ADEPTS 2 gives considerable speedup over ADEPTS 1, with an increasing trend as the number of concurrent alerts increases. Looking at the absolute values of times in ADEPTS 2, we find that the time increases as the number of alerts increases even though unlimited computation resources are assumed to be available. This is because the number of parallelizable sub-graphs grows with the number of alerts as shown in Fig. 13. However, the growth is sub-linear and therefore the relative speedup between ADEPTS 2 and ADEPTS 1 increases only sub-linearly with increasing number of alerts. The sub-linear growth is explained by the fact that the spatial locality algorithm will tend to cluster alerts close by in the I-GRAPH into the same attack sub-graph. The second comparatively less significant contributor to the increasing time with increasing number of alerts is that the non-parallelizable part of the computation – determining which sub-graph an incoming alert belongs to – becomes more resource intensive. Fig. 14 gives the average speed up of all the 8 cases between ADEPTS 2 and ADEPTS 1. The vertical bar shows 2 standard deviations. With a small number of alerts (say 50), ADEPTS 2 performs only slightly better than ADEPTS 1. This is due to the inherently higher constant overhead of ADEPTS 2 nullifies the performance gain from limited parallelization of 3-6 generated sub-graphs. However, from around 100 alerts, ADEPTS 2 starts to significantly out-perform ADEPTS 1. Of course, it is not reasonable to expect such a huge number of concurrent alerts for a relatively small scale testbed like ours, but could be close to reality were ADEPTS to be deployed on a huge corporate system.

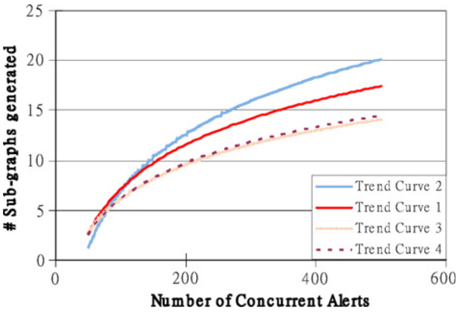


Fig. 13. Degree of parallelization in ADEPTS.

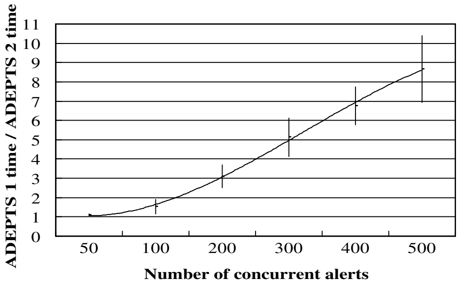


Fig. 14. Speed up in ADEPTS 2 with increasing number of concurrent alerts.

References

[1] Wikipedia, SCADA http://en.wikipedia.org/wiki/scada.

[2] David Bailey, Edwin WrightPractical SCADA for Industry, Elsevier, 2003.

[3] Y.-S. Wu, B. Foo, Y. Mei, S. Bagchi, Collaborative intrusion detection system (CIDS): a framework for accurate and efficient IDS, in: Proceedings of ACSAC, 2003.

[4] R. Ellison, R. Linger, T. Longstaff, N. Mead, Case study in survivable network system analysis, Technical Report CMU/SEI-98-TR-014, SEI, CMU, 1998.

[5] F. Cohen, Simulating cyber attacks, defenses, and consequences. Available from: <http://all.net/journal/ntb/simulate/simulate.html>, 1999.

[6] R. Anderson, A. Hearn, R. Hundley, Studies of cyberspace security issues and the concept of a US minimum essential information infrastructure, in: Proceedings of the 1997 Information Survivability Workshop, CERT, 1997.

[7] T. Toth, C. Kruegel, Evaluating the impact of automated intrusion response mechanisms, in: Proceedings of ACSAC, Dec 2002.

[8] E. Fisch, Intrusion damage control and assessment: a taxonomy and implementation of automated responses to intrusive behavior, Ph.D. dissertation, Texas A&M University, College Station, TX, 1996.

[9] C. Carver, U. Pooch, An intrusion response taxonomy and its role in automatic intrusion response, in: Proceedings of IEEE Workshop on Information Assurance and Security, USMA, West Point, NY, 2000.

[10] U. Lindqvist, E. Jonsson, How to systematically classify computer security intrusions, in: Proceedings of IEEE Security & Privacy, Oakland, CA, pp. 154 – 163, 1997.

[11] G. White, E. Fisch, U. Pooch, Cooperating security managers: a peer-based intrusion detection system, IEEE Network 10 (1) (1996) 20–23.

[12] P. Porras, P. Neumann, EMERALD: event monitoring enabling responses to anomalous live disturbances, in: Proceedings of the 20th National Information Systems Security Conference, 1997, pp. 353–365.

[13] D. Ragsdale, C. Carver, J. Humphries, U. Pooch, Adaptation techniques for intrusion detection and intrusion response systems, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2000, pp. 2344–2349.

[14] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, T. Reid, Autonomic response to distributed denial of service attacks, in: Proceedings of RAID, 2001.

[15] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, AT&T Center for Internet Research at ICSI (ACIRI), February 2001.

[16] P. Brooke, R. Paige, Fault trees for security system analysis and design, Journal of Computers and Security 22 (3) (2003) 256–264.

[17] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, R. Lutz, A software fault tree approach to requirements analysis of an intrusion detection system, in: Proceedings of the 1st Symposium on Requirements Engineering for Information Security, 2001.

[18] M. Dacier, Y. Deswarte, M. Kaaniche, Quantitative assessment of operational security: models and tools, LAAS Research Report 96493, May 1996.

[19] R. Ortalo, Y. Deswarte, M. Kaaniche, Experimenting with quantitative evaluation tools for monitoring operational security, IEEE Transactions on Software Engineering 25 (5) (1999) 633–650.

[20] C. Carver, J. Hill, U. Pooch, Limiting uncertainty in intrusion response, in: Proceedings of IEEE Information Assurance and Security, 2001.

[21] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, M. Zissman, Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation, DISCEX 02 (2) (2000) 1012–2000.

[22] DARPA IPTO, Organically assured and survivable information systems (OASIS). Available from: <http://www.darpa.mil/ipto/programs/oasis>.

[23] DARPA Information Processing Technology Office, Self-regenerative systems (SRS). Available from: <http://www.darpa.mil/ipto/programs/srs>.

[24] CMU Software Eng. Institute, Survivable network technology. Available from: <www.sei.cmu.edu/organization/programs/nss/surv-net-tech.html>.

[25] D. Wang, B.B. Madan, K. Trivedi, Security analysis of SITAR intrusion tolerance system, in: SSRS, 2003, pp. 23–32.

[26] C. Cachin, Distributing trust on the Internet, in: Proceedings of DSN, 2001, pp.183–192.

[27] P. Pal, F. Webber, R. Schantz, J. Loyall, Intrusion tolerant systems, in: IEEE Information Survivability Workshop (ISW), 2000.

[28] V. Stavridou, B. Dutertre, R.A. Riemenschneider, H. Saidi, Intrusion tolerant software architectures, DISCEX01, in: DARPA Information Survivalability Conference and Exposition II, May 2001.

[29] B. Foo, Y-S. Wu, Y-C. Mao, S. Bagchi, E.H. Spafford, ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment, in: Proceedings of DSN, 2005, pp. 508–517.

[30] A.N. Sovarel, D. Evans, N. Paul, Where's the FEEB? The effectiveness of instruction set randomization, in: 14th USENIX Security Symposium, 2005.

[31] W. Hu, J. Hiser, D. Williams, A. Filipi, J.W. Davidson, D. Evans, J.C. Knight, A. Nguyen-Tuong, J. Rowanhill, Secure and practical defense against code-injection attacks using software dynamic translation, in: Virtual Execution Environments Conference, June 2006.

[32] B. Demsky, M. Rinard, Automatic data structure repair for self-healing systems, in: First Workshop on Algorithms and Architectures for Self-Managed Systems, San Diego, CA, June 2003.

[33] R. Stroud, I. Welch, J. Warne, P. Ryan, A qualitative analysis of the intrusion-tolerance capabilities of the MAFTIA architecture, in: International Conference on Dependable Systems and Networks (DSN), July 2004, pp. 453 – 461.

[34] B. Foo, Y-S. Wu, S. Bagchi, E.H. Spafford, Automated intrusion response with unanticipated attacks, Submitted to ACM Conference on Computer and Communications Security (CCS), 2006.

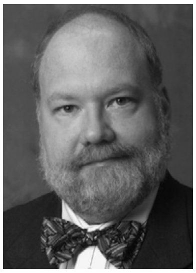
[35] Y-S. Wu, Achieving high survivability in distributed systems through automated intrusion response, Ph.D. thesis, Preliminary report, School of Electrical and Computer Engineering, Purdue University.

Yu-Sung Wu is currently a Ph.D student in the School of Electrical and Computer Engineering at Purdue University in West Lafayette, Indiana. He works in the Dependable Computing Systems Lab, headed by Prof. Saurabh Bagchi. His research interests fall in the area of Intrusion Detection/Tolerance, Fault Detection/Tolerance, VoIP Security, Distributed System, and Information Security. Yu-Sung received his Master degree in Electrical and Computer Engineering from Purdue University and his Bachelor degree in Electrical Engineering from National Tsing Hua University, Hsin Chu, Taiwan.

Bingrui Foo is currently a PhD student in the School of Electrical and Computer Engineering at Purdue University in West Lafayette, Indiana. Presently, he is involved in two research projects. The first is in the field of network security, specifically the design of intrusion tolerant systems and automated response mechanisms. The second is in the field of statistical modeling, consisting of extending mixture models by adding hierarchal structure to images and videos. His papers have appeared in DSN and ACSAC.

Yu-Chun Mao was born in Taipei, Taiwan. She obtained her bachelor degree in the department of Engineering Science and Ocean Engineering at the National Taiwan University in Taipei, Taiwan, and her masters degree in the school of Electrical and Computer Engineering at Purdue University in West Lafayette, Indiana. While she studied at Purdue, she worked on the Cyber Defense Technology Experimental Research Network Project, and she was a member of Professor Saurabh Bagchi's Dependable Computing Systems Research Group. She is currently working as a software engineer at Microsoft.

Saurabh Bagchi is currently an Assistant Professor in the School of Electrical and Computer Engineering at Purdue University in West Lafayette, Indiana with a courtesy appointment in the Department of Computer Science. He leads the Dependable Computing Systems Lab where he carries out research in design and implementation of dependable distributed systems and sensor networks. His work is supported by NSF, Indiana 21st Century Research and Technology Fund, and Purdue Research Foundation, with equipment grants from Intel and Avaya. His papers have been runner up for the best paper in HPDC (2006), DSN (2005), and MTTS (2005). He has been an Organizing Committee member and Program Committee member for the Performance and Dependability Symposium (PDS) track of the Dependable Systems and Networks Conference (DSN) and the Symposium on Reliable Distributed Systems (SRDS). He has organized a workshop called DIWANS on dependability of ad-hoc networks at DSN '04 and Mobicom '06 and a panel on dependability of software at the International Symposium on Software Reliability Engineering (ISSRE '04). Before joining Purdue in 2002, he did his Ph.D. from the Computer Science department of the University of Illinois at Urbana-Champaign with Prof. Ravishankar Iyer at the Coordinated Science Laboratory. His Ph.D. dissertation was on error detection protocols in distributed systems and was implemented in a fault-tolerant middleware system called Chameleon. The ideas were demonstrated and extensively evaluated in an implementation, which was used by Motorola in their wireless telephone network controller and by NASA in their Remote Exploration and Experimentation (REE) testbed.



Eugene H. Spafford is on the faculty at Purdue University, where he has been since 1987. He has a joint appointment as a professor of Computer Sciences and of Electrical and Computer Engineering. He is also the founder and Executive Director of the Center for Education and Research in Information Assurance and Security (CERIAS). CERIAS is a campus-wide multi-disciplinary Center, with a broadly-focused mission to explore issues related to protecting information and information resources. Spaf has written extensively about information security, cybercrime, software engineering, and professional ethics.

His current research is directed towards issues of public policy and information security, architecture and construction of highly-secure systems, and cyberforensic technologies. Dr. Spafford is a Fellow of the ACM, Fellow of the AAAS, and Fellow of the IEEE. In 2000, he was named as a CISSP, honoris causa. He was the year 2000 recipient of the NIST/NCSC National Computer Systems Security Award, generally regarded as the field's most significant honor in information security research. Among his many activities, Spaf is chair of the ACM's U.S. Public Policy Committee, is a member of the Board of Directors of the Computing Research Association, and was a member of the most recent President's Information Technology Advisory Committee (PITAC). More information is available on his website.