CERIAS Tech Report 2009-22

ACHIEVING HIGH SURVIVABILITY IN DISTRIBUTED SYSTEMS THROUGH AUTOMATED RESPONSE

by Yu-Sung Wu
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Yu-Sung Wu

Entitled Achieving High Survivability in Distributed Systems through Automated Response

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

S. Bagchi
Chair

R. Eigenmann

A. Ghafoor

E. H. Spafford

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): S. Bagchi

Approved by: V. Balakrishnan 6/16/2009
Head of the Graduate Program Date

Graduate School Form 20
(Revised 10/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Achieving High Survivability in Distributed Systems through Automated Response

For the degree of Doctor of Philosophy

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Yu-Sung Wu
Signature of Candidate

2009/6/16
Date

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

ACHIEVING HIGH SURVIVABILITY IN DISTRIBUTED SYSTEMS THROUGH AUTOMATED RESPONSE

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yu-Sung Wu

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2009

Purdue University

West Lafayette, Indiana

To my family, for their support, love, and endeavor.

感謝我摯愛的家人

ACKNOWLEDGMENTS

Special thanks to Prof. Saurabh Bagchi and my colleague Bingrui Foo without whose efforts and input this work would not have made its way. Also thanks to Prof. Gene Spafford, who has been providing advice to guide the ADEPTS project. I would also like to thank Prof. Arif Ghafoor and Prof. Rudolf Eigenmann for being on my committee and helping to shape my years long study at Purdue ECE.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

Table Page

Table Page

ABSTRACT

Wu, Yu-Sung Ph.D., Purdue University, August 2009. Achieving High Survivability in Distributed Systems through Automated Response. Major Professor: Saurabh Bagchi.

We propose a new model for automated response in distributed systems. We formalize the process of providing automated responses and the criterion for asserting global optimality of the selection of responses. We show that reaching the globally optimal solution is an NP-hard problem. Therefore we design a genetic algorithm framework for searching for good selections of responses in the runtime. Our system constantly adapts itself to the changing environment based on short-term history and also tracks the patterns of attacks in a long-term history.

Unknown security attacks, or zero-day attacks, exploit unknown or undisclosed vulnerabilities and can cause devastating damage. The escalation pattern, commonly represented as an attack graph, is not known a priori for a zero-day attack. Hence, a typical response system provides ineffective or drastic responses. Our system "conceptualizes" nodes in an attack graph, whereby they are generalized based on the object-oriented hierarchy for components and alerts. This is done based on our insight that high level manifestations of unknown attacks may bear similarity with those of previously seen attacks. This allows the use of history such as effectiveness of each response from past attacks to assist responses to the unknown attack.

This thesis lays down three distinct claims and validates them empirically. The claims are: (i) For automated response, consider a baseline mechanism that has a static mapping from the local detector symptom to a local response. This corresponds to the state-of-the-art in deployed response systems. Now consider our proposed model which takes into account global optimality from choosing a set of responses and also does a dynamic computation of the response combination from the set of detectors and other system parameters (inferences about the accuracy of the attack diagnosis, response

effectiveness, etc.). The survivability of the application system with our proposed model is an upper bound of the survivability achievable through the baseline model. (ii) In some practical situations, the proposed model gives higher survivability than the baseline model. (iii) The survivability with our proposed model is improved when the system takes into account history from prior similar attacks. This kind of history is particularly important when the system deals with zero-day attacks.

1. INTRODUCTION

Distributed systems comprising multiple services interacting among themselves to provide end-user functions are an increasingly important information infrastructure. Examples abound in the commercial domain and in the critical infrastructure domain, such as, e-Commerce systems and SCADA systems, 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. The huge financial stakes involved or the importance to homeland security make them prime candidates for malicious activities, a.k.a. attacks[1]. 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 cyber attacks[1], while realizing that some of the discussion carries over to natural or accidental failures as well[2].

The motivations outlined above have led to substantial interest in securing distributed systems through prevention of failures and successful attacks. Prevention is achieved through careful design and development of the components to eliminate most faults and vulnerabilities. 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 or attacks. This is especially pertinent when the systems have interfaces to external users, as all e-Commerce systems and many SCADA systems do.

---

[1] Here we focus on cyber attacks, while attacks generally include physical attacks as well. We define attacks as malicious attempts to violate the integrity of an information system and perform illegitimate actions. An intrusion is a penetration into the system. Thus, a network based DoS is an attack, but not an intrusion.

[2] A failure implies any violation of the specification of the system, according to the current policy in force, due to natural (or accidental) causes.

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 failures or attacks. This has spurred interest in the defense community as evidenced by DARPA's OASIS program [1] and the follow-on Self Regenerative Systems (SRS) program[2]. Therefore, besides the use of the best possible prevention techniques, a survivable system has to have the ability for incident[3] response when failures or attacks happen. Traditionally, incident response means the intervention of system administrators in which the administrators have to identify the issues, contain the effects from the incident, recover the system, understand the incident, and apply measures which would prevent the reoccurrence of the same incident. However, as distributed systems become ubiquitous and complex and they are often placed in environments difficult to reach for human intervention, automated response tools gain importance. We present the design and instantiation of a series of automated response systems called ADEPTS, which provides a framework for achieving automated response in a distributed system environment.

Providing automated response 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 attacks relevant to the response process and to identify the effectiveness of the deployed response mechanism. The rudimentary response mechanism often bundled with Host-based Intrusion Prevention System (HIPS) [3, 4] or Network-based Intrusion Prevention System (NIPS) [5, 6] 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 HIPS or anti-virus product to quarantine the file and disable all access to the file, or a suspect packet being flagged by a NIPS may cause the specific network connection to be terminated. 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 automated response system for distributed systems, one has to consider the constituent services in the system and the different levels of degradation of

[3] An incident is defined as the event of an attack or a failure.

each individual service due to an incident, 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 use a representation called an Incident Graph (I-GRAPH), where the nodes represent sub-goals for the incident 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 goal corresponding to a node in the I-GRAPH is achieved. The attack model considered in ADEPTS is multi-stage network-based attacks. Some of the stages of the attack must be detectable by detectors embedded in the distributed system. An attack which compromises the entire system in one shot is not considered. Insider attacks may also be considered, though the requirement is that the insider should be not be omnipotent and must not be able to compromise multiple services instantaneously. Brute force DoS attacks which do not penetrate the system are not considered, nor are passive attacks.

Now we outline the key design principles or design requirements that drive our work. The response choice in an automated response system should be dynamically and adaptively determined. The attacks 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 attack; and out-of-band parameters of the response, such as expert system knowledge of an effective response for a specific attack or policy determined response when a specific manifestation occurs. The automated response system 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 or failure sequences is infinite. Translating this to ADEPTS, it 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 incident will ultimately have a manifested goal which is detectable. Any automated response system 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 automated response system 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 automated response system. 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 [7], 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.

An interesting question one might ask is the optimality of the chosen responses by a response system. Specifically, given a set of detector alarms, indicating the current state of an attack on the system, what would be the best response actions a response system should take against the attack? Here one needs to factor in both the cost from deploying a response and also the cost from potential further damage to the system as a result of not deploying a response. We formally define the optimality criterion for responses and show that the problem of optimal response determination (O.R.D.) is an NP-Hard problem. Hence, we develop a genetic algorithm (GA) based framework to approximate the optimal response for a given attack. Since GA has the nice property of passing good chromosomes (solutions) from a parent generation to the child generation, the response solution for an attack can actually be improved across instances of attacks. This means that for an attack, which has been seen before or has been seen somewhere else, very effective response actions can be taken promptly without needing to perform the lengthy GA evolution process much.

Zero-day attacks are attacks which exploit unknown computer vulnerabilities. They pose challenges to the design of a response system due to that the corresponding attack graph for such an attack is not assumed a known priori. On the other hand, exact history information of effective responses against such an attack is not known as well. To address these challenges, we firstly come up with an online process to dynamically populate the attack graph (I-GRAPH) based on detector alarms and knowledge of the configuration of the protected system. Our work is based on existing works on alert correlation [8, 9]. The generated attack graph, however, does not warrant better responses for the zero-day attack due to the lack of information such as response effectiveness

against the attack. We come up with a technique called "conceptualization of attack graph" to enable the use of inexact history information from a closely matched past attack. This is based on our observation that though the detailed mechanism of a zero-day attack is unknown, in most cases the high level manifestation behind such an attack is not necessarily unknown.

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. The metric used to evaluate a survivable 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 incident on the system's functionality. We propose the use of the metric *survivability* [10] to evaluate the effect of responses. 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 security goals (e.g., keep users' private information secure) that are preserved in the event of an incident. A high level transaction depends on certain chains of interactions between multiple functioning services and preserving a high level goal implies thwarting certain goals from being reached.

The design of ADEPTS is realized in an implementation which provides automated 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.

1.1. Thesis Statement

1.1.1 Survivability Metric for Automated Response System

The performance of a response system can be evaluated based on many different metrics such as throughput, reliability, usability, security effectiveness [11], and

survivability [10]. Survivability is qualitatively defined as the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks. The term security effectiveness has sometime been used synonymously with survivability. Our discussion will be focused on survivability.

Survivability can be evaluated based on impact from an attack and from deployed response actions [12] to the protected system. It considers the impact on confidentiality, integrity, and availability of the protected system. Assume for an attack, a response system chooses a set *RC* of response actions. We conceptually defines the expected impact from deploying *RC* to the protected system as :

$$\begin{aligned} |Iv(RC)| = \; & |\,(\text{expected impact from attack with } RC \text{ being present}) + \\ & (\text{impact from response actions in } RC)\,| \end{aligned} \tag{1.1}$$

Assume the system's initial survivability is *C*, then the survivability post the deployment of *RC* should be :

$$\text{survivability}(RC) = C - |Iv(RC)| \tag{1.2}$$

Note that we describe $|Iv(RC)|$ as the "expected" impact from deploying *RC*. This is because the impact from an attack can be non-deterministic because an adversary may or may not take a certain attack step. In addition, a response action may or may not react to the attack successfully. The complete definition of $|Iv(RC)|$ is presented in Sec. 3.6.

1.1.2 BASELINE Model

Table 1.1. BASELINE Model of Automated Response

1. A collection of (detectors, response actions) pairs : $\{(D_1,R_1), (D_2,R_2),\ldots, (D_k,R_k), \ldots, (D_N,R_N)\}$
2. For each pair, a mapping function $f_k : D_k \rightarrow R_k$ is determined prior to the execution of the system
3. f_k is designed based on expert knowledge

Table 1.1 defines a BASELINE Model for Automated Response in Distributed Systems. This corresponds to existing mainstream automated response systems, where stand-alone IDS or IPS boxes are used. An instantiation of this model has each IDS or IPS box trigger some local response(s). In this model, the mapping from detectors to response actions is specific to each pair, and a local detection at one IDS or IPS box cannot flag a response at a different IDS or IPS box. The corresponding mapping function f_k dictates how each IDS or IPS carries out the response actions based on the alerts from the detectors.

1.1.3 Proposed Model

Table 1.2. Proposed Model of Automated Response

1. The set of all the detectors D and the set of all the response actions R.
2. History of past attacks H
3. A mapping function f : (D,H) →R

We propose a new model for automated response in distributed system (Table 1.2). The proposed model considers the whole set of detectors and the whole set of the response actions. The mapping function allows the mapping of any detector into any response action. The model also allows the use of history information from past attacks.

1.1.4 Claims

C1. For a given attack, the proposed model describes a set of response actions, from which the survivability is the upper bound of the survivability from any set of response actions generated from the BASELINE model.

C2. In a practical system, it is possible to identify cases when the proposed model yields a higher survivability than the BASELINE model.

C3. It is possible that the use of history information in the proposed model can further improve the survivability.

Table 1.3. Desirable Properties of Automated Response

1. f is designed to maximize the survivability (Eq. (1.2)) based on the information accumulated in H and detectors D
2. f is designed to tolerate new types of attacks

The proposed model generalizes the BASELINE model. To ensure an instantiation of the proposed model can provide higher survivability beyond the BASELINE model, we introduce two properties in Table 1.3 to guide the design of the mapping function f in the proposed model. The properties are incorporated into the two instantiations we created: the SWIFT system in Sec. 5 and the ORIGIN system in Sec. 6. Essentially, these two properties will help identify cases to support claim C2 and claim C3, which will be presented in Sec. 8.

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 thirty hours, almost all the attacks succeed irrespective of the skill level of the defending system's administrator[13]. 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 [14, 15]. 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 [12], 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 taxonomy of offensive and defensive responses to aid in selection of coherent responses in an automated response system ([16],[17],[18]). Cooperating Security Managers (CSM) [19] 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 [16]. It uses the suspicion level of the user as the only determining factor in the choice of response. A

second system called EMERALD [20] 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 provide a framework for easily incorporating these factors in the automated response determination. Another adaptive IRS is the Adaptive, Agent-based Intrusion Response System (AAIRS) [21]. 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 [22, 23].

A significant volume of work in this domain has focused on using diverse redundant components in building intrusion tolerant systems ([24-27]). 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. 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.

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 [28, 29]. 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 [30, 31]. The notion of privilege graphs introduced in [31] 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. The work lays down a framework to reason about the optimality of the response choices made by these systems, which has not been seen in previous works.

A topic relevant to our work here is multi-stage attack graph generation. Three complementary methods are discernable among the existing work: (i) Using pre-conditions and post-conditions of vulnerabilities and attacker actions to generate the graphs, (ii) using the network topology, connectivity, and/or other physical network attribute, and, (iii) using attack classification/taxonomy and expert knowledge to broadly identify the possible links between attack stages or relations between alerts. Most past research has mainly relied on (i) [32], though they have at times used (ii) as a complement. Any approach that relies on (i) is unable to target zero-day attacks, therefore our approach uses (ii), though additional information can be provided from (iii) to make our approach more refined. A relevant work that also uses (ii) is [12], and the mechanisms described are similar to the initial version of ADEPTS [33]. The significant differences are that their system cannot handle unknown attacks and it is unknown whether their system has been developed further to include more capabilities. The distinguishing feature we call 'conceptualization' is novel and has additional benefits not explicitly desired or considered by other researchers, namely the usefulness in responding to zero-day attacks. For a review of past approaches, one can refer to the survey paper at [34]. The approaches explored so far have not been targeted to unknown attacks, though it is conceivable that one can build on them for this purpose. The use of a Bayesian network as the basic framework for inference is not new [35], and we have applied it to inferencing in the presence of deployed responses.

There have been some efforts at using genetic algorithms for intrusion detection [36-38] and search for vulnerabilities [39]. The results have been promising, but only after careful definition of the syntax of the chromosomes and tuning of the fitness measure. We have not found prior application of GA to intrusion response.

3. MULTI-STAGE SECURITY ATTACKS AND RESPONSES

In a distributed system, an attack at a site (a specific service on a specific host) can obviously cause impacts at that site. However, it can also potentially cause cascading effects on the other services and the other hosts in the system due to the interconnected nature of a distributed system. As a result, many attacks targeted at distributed systems are the multi-stage attacks, in which an attack consists of multiple attack steps that can span across different services and different hosts. In IDS research, there have been systems which utilize the diverse detection alerts from these multiple attack stages for increasing overall detection accuracy and better understanding of the whole behavior from a multi-stage attack [7]. Also there is research on reconstructing the attack stages for an attack from detector alerts [8]. In this section, we lay out the attack model for a multi-stage attack. We then give the model on responses for multi-stage attacks. In the end, we present a metric to evaluate the effect from using responses against an attack.

3.1. Adversary and Attack Model

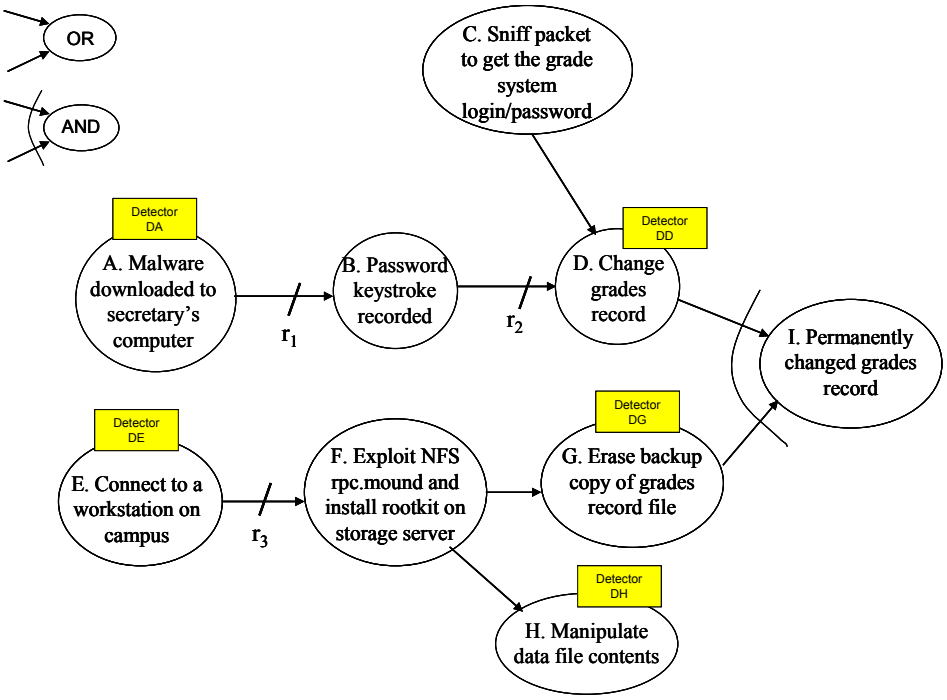


Figure 3.1. An example I-GRAPH

Our model for the target attack is an external multi-stage attack which first compromises the services that have external interfaces and subsequently compromises internal services with the goal of disrupting some transactions supported in the system or violating some of the security goals in the system. This is the model commonly used in the literature for distributed intrusion response systems (IRSs) [12, 33]

A representation called Incident Graph (I-GRAPH) [33] is used for modeling the spread of the attack, which is similar in concept to attack graphs [9, 32, 40]. The final goal of the adversary may be disrupting some high level system functionality, such as "Permanently change grades record" in Figure 3.1. This final goal is achieved through multiple intermediate intrusion goals (attack steps) and each is represented as an I-GRAPH node.

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 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.

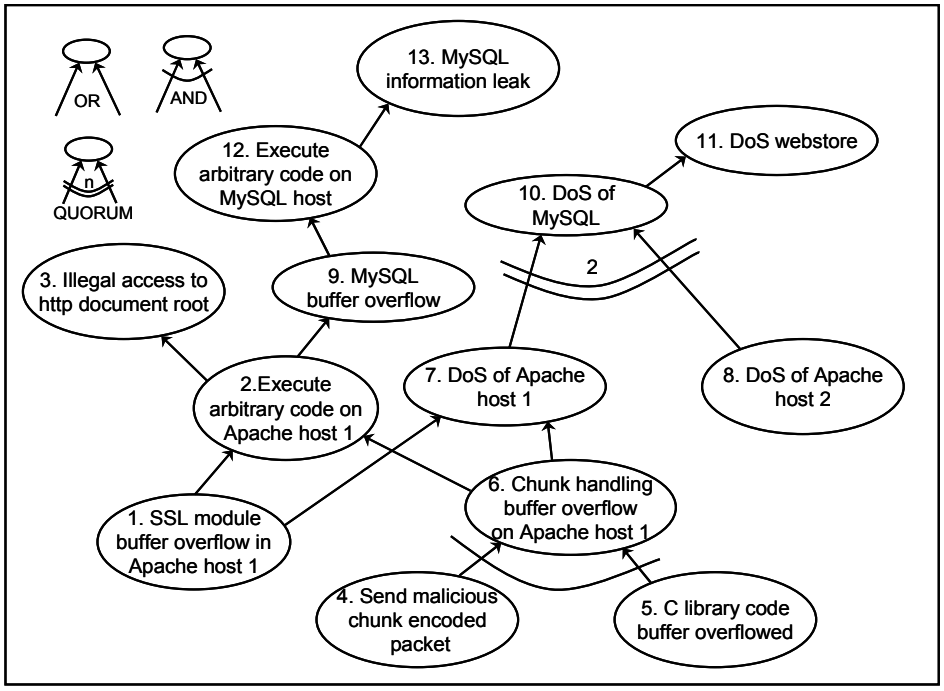


Figure 3.2. A section of the I-GRAPH from our deployed e-Commerce environment

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 Figure 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).

The second input to PIG is the target independent vulnerability descriptions. Information on the vulnerabilities can be created based on installed detectors in the system. For example, a stack overflow detector can correspond to a stack overflow vulnerability. This kind of vulnerability can be generic in nature as the associated

detector can be designed to pick up generic manifestation such as a buffer overflow, regardless of the specific signature used to trigger the manifestation. On the other hand, vulnerabilities can also 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. In the following, we give the formal definition on the structure of our I-GRAPH.

3.2.3 Definition of I-GRAPH G(N,E):

N := {nodes in G} := {{N_A: specific attack manifestation} $\cup$ {N_B: generic attack manifestation} $\cup$ {N_C: high level parameterized manifestation} $\cup$ {N_D: logical inference pseudonodes }}

E := {edges in G} := {(n_1,n_2) | if n_2 is casually dependent on n_1 for $n_1,n_2 \in N$}

N_A : These nodes are constructed directly out of the detector alerts for specific attack manifestations. These manifestations carry specific detector signatures. For example, the Snort rule for detecting Apache chunked encoding memory corruption exploits or some AntiVirus software detecting the binary code of some virus in an infected file.

N_B : These nodes are constructed out of the detector alerts that correspond to attack manifestations which are generic in nature. These manifestations usually span

across multiple different attacks, some of which can be potentially of unknown attack types. For example, stack buffer overflow detectors such as LIBSAFE should generate alerts out of any attack attempts which result in stack buffer overflow, irrespective of the specific attack signature used to achieve the overflow.

N_C : These node corresponds to high level manifestation which usually do not have a corresponding detector alert. However, the manifestation is hypothesized since it directly impacts a system functionality or violates a system goal. For example, "losing customer credit card numbers" could form a node of type N_C.

N_D : These are intermediate nodes used for providing OR/AND/Quorum logics in the I-GRAPH.

In many deployments, the systems may have unknown vulnerabilities and therefore the I-GRAPH is mainly composed of N_B nodes, which can be automatically created based on the available detectors and the knowledge of the interactions among the servers in the target system [32],[40]. It is not necessarily dependent on knowledge about specific attacks or vulnerabilities. For example, if we have an I-GRAPH node corresponding to "Root password on machine M is changed", this does not mean we know *a priori* that the operating system on machine M has the vulnerability which can be used to change the root password. However, it is actually one of ADEPTS' key roles to deal with the uncertainty by constantly adapting to the actual situation, to provide continuous protection to the system.

3.3. Response Model for Multi-Stage Attack

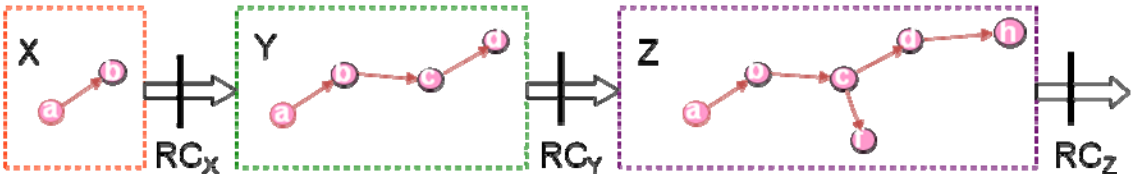


Figure 3.3. Three different snapshots for a given attack. Response combinations RC_X, RC_Y, RC_Z are deployed between snapshots

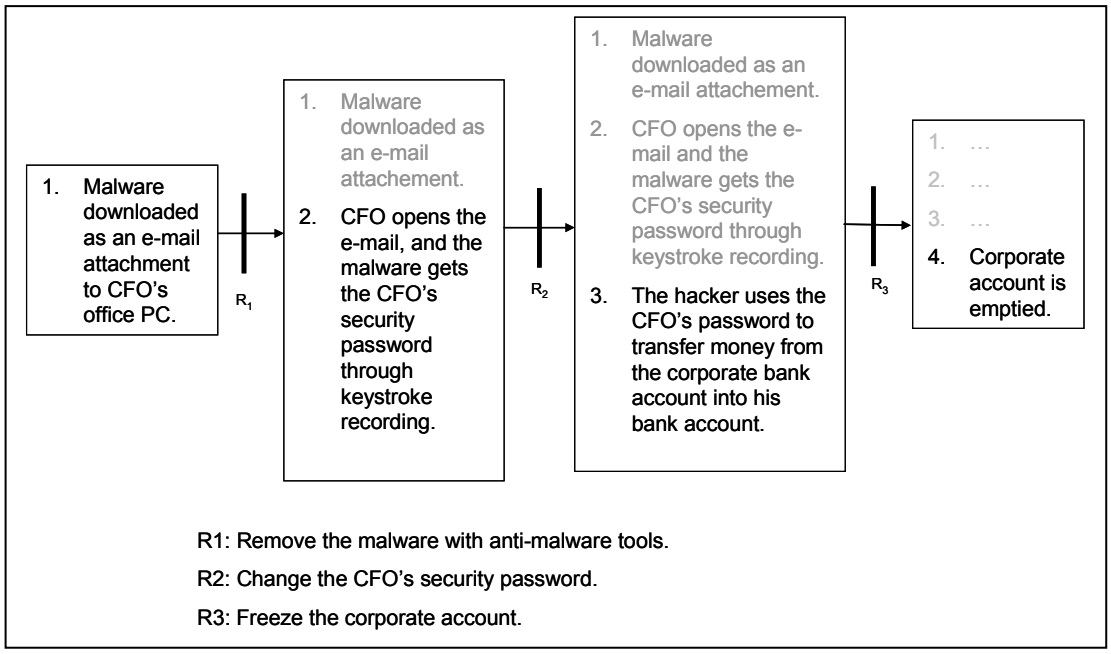


Figure 3.4. Snapshots for an example four-stages attack.

In general, a multi-stage attack consists of multiple *attack snapshots*. Each snapshot contains the detector alerts which have been generated thus far, and the fragment of the I-GRAPH with nodes for which alerts have been received. Figure 3.3 shows a general case, where three snapshots X, Y, and Z are created from an attack. In practice, we find that there are groups of alerts that arrive in a batch, corresponding to several closely spaced attack steps of a fast-moving attack and an IRS cannot deploy a response within a batch of alerts. This batch creates a snapshot. Figure 3.4 shows an example of four attack snapshots created from a real-world attack consisting of four attack stages. The attack begins with a company chief financial officer (CFO) downloaded a malware as a e-mail attachment to his office computer followed by the malware sniffing the keystrokes on the CFO's computer and the adversary eventually getting the corporate bank account number and password. The adversary eventually uses the account number and password to deplete the company's bank account. Potentially, responses can be used right after each attack snapshot such as "R1:Remove the malware…". The goal is to deter the attack from inflicting further damages on the system and as a result prevent the next attack snapshot from being created.

Corresponding to the proposed model in Sec. 1.1, for a multi-stage attack consisting of k snapshots $\{s_1,s_2,..s_k\}$, the response mechanism is formally described by $RC_i = Respond(s_i, H_i)$, where s_i is the i^{th} snapshot, H_i is the history information and RC_i is

the response combination decided by an IRS. Therefore, in Figure 3.3, we have $RC_X=Respond(s_X,H_X)$, $RC_Y=Respond(s_Y,H_Y)$, and $RC_Z=Respond(s_Z,H_Z)$.

3.4. Survivability Guarantee by Proposed Model

Table 3.1. Proof for Thesis Claim C1

Proof:

1. Assume an attack indicated as detector alerts $\{\overline{D_1},\overline{D_2},..,\overline{D_N}\}$.
2. In the BASELINE model, assume mapping functions $\{\overline{f_1},\overline{f_2},..,\overline{f_N}\}$ generate sets of response actions $\{\overline{RC_1},\overline{RC_2},..,\overline{RC_N}\}$ with the highest survivability, where $\overline{f_k}:\overline{D_k}\rightarrow\overline{R_k}$
3. In the proposed model, we can have a mapping function $\overline{f}$ constructed as follow

   $\overline{f}$ contains the mapping $(\{\overline{D_1},\overline{D_2},...,\overline{D_N}\},\varnothing)\rightarrow\{\overline{RC_1},\overline{RC_2},...,\overline{RC_N}\}$
4. We now have a mapping function $\overline{f}$ in the proposed model which describes the set of response actions, which yields the same survivability as from $\{\overline{f_1},\overline{f_2},..,\overline{f_N}\}$ in the BASELINE model. This suffices an upper bound on the survivability attainable by any set of response actions from the BASELINE model.

Thesis Claim C1 in Sec. 1.1 states that the proposed response model guarantees system survivability higher than or equal to the system survivability attainable by the BASELINE model. The proof is intuitive and is presented in Table 3.1.

3.5. Impact Vector Metric

Table 3.2. Example E-Commerce Transaction Goals

Name	**Weight**
Browse webstore	10
Add merchandise to shopping cart	10
Place order	10
Charge credit card	5
Admin work	10

Table 3.3. Example E-Commerce Security Goals

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

Conceptually, an attack can impact the normal operation of a system. On the other hand, a response action can also affect the normal operation of system. For example, a firewall rule may accidentally block legitimate traffic as well. To quantify the impact on a system from an attack and from a response action, we define a metric called *Impact Vector*. We assume that the protected target system has a set of transactions (e.g. Table 3.2) that should be supported during system operation and security goals (e.g. Table 3.3) that should be satisfied during its operation. The impact vector *Iv* used in a system of *n*

transactions and m security goals is an $(n+m)$ element vector, with each element representing the impact value on the corresponding transaction or security goal. The higher the value is, the more severe the impact is. The range for the value is $[0, \infty]$.

The dimensions may not all be independent, in which case assigning the *Iv* values has to be done carefully taking the dependence into account. The notion of impact vectors is found in the security domain in several different forms, e.g., as the result of risk analysis[41].

For each response r, there is an associated impact vector $Iv(r)$ which indicates the impact on the system as a result of deploying the response. This may be specified by the system administrator or determined automatically by calculating the services affected by the response and computing which transactions and security goals are violated as a result as in [12]. For each I-GRAPH node n, there is an associated impact vector $Iv(n)$ which gives the impact as a result of this node being achieved by an adversary.

The absolute value of *Iv* is defined as

$$|Iv| = |[a_1\ a_2\ \ldots\ a_n]| = \sum_{i=1,n} a_i,\ a_i \in (0, \infty).$$

The summation of two impact vectors is also an impact vector and is defined as follows:

$$Iv = Iv_1 + Iv_2 = [max(Iv_{1,1}, Iv_{2,1}),\ \ldots,\ max(Iv_{1,n}, Iv_{2,n})]$$

3.6. Impact Vector Metric for Response Combination

Let us assume an attack has resulted in i snapshots $s_1, s_2, .., s_i$. Also assume the I-GRAPH has m nodes $n_1, n_2, .. n_m$. We want to evaluate the cost of the response combination $RC_i = f(s_i, H)$, which consists of n response actions $\{r_1, r_2, .., r_n\}$. Assume the probability of each node being achieved in the attack considering the responses in RC_i is $P(n_1)$, $P(n_2)$,…, $P(n_m)$. Then the cost of RC_i is defined by Eq. (3.1). Under this metric, the optimal response combination to a given attack at a specific snapshot (corresponding to a specific point in time) is the one which yields the minimum value of cost as shown in Eq. (3.2).

$$Cost(RC_i) = | Iv(RC_i) | = \left| \sum_{k=1}^{m} Iv(n_k)\,\mathrm{P}(n_k) + \sum_{k=1}^{n} Iv(r_k) \right| \tag{3.1}$$

$$RC_{i,opt} = \arg\min_{RC_i} Cost(RC_i) \tag{3.2}$$

The summation across the *m* I-GRAPH nodes in Eq. (3.1) corresponds to the expected impact from attack as shown in Eq. (1.1) in Sec. 1.1. The remaining part correspond to the impact from response actions.

3.6.1 Survivability Metric

The survivability metric after deploying response combination RC_i is calculated by subtracting $Cost(RC_i)$ from the initial system survivability and is defined as follow:

$$\begin{aligned} \text{survivability after deploying } RC_i &= \text{initial survivability - } Cost(RC_i) \\ &= \text{initial survivability - } \left| Iv(RC_i) \right| \end{aligned} \tag{3.3}$$

Thus, a lower impact vector value corresponds to a higher survivability. The optimal response combination $RC_{i,opt}$ maximizes the survivability in Eq. (3.3).

3.7. Inference on I-GRAPH : 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 are likely to have been achieved. Each detector provides confidence values for its alerts, termed *alert confidence*. If the detector does not provide an inbuilt 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 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

$$CCI = \begin{cases} alert\ confidence & \text{,no child} \\ f'(CCI_i) & \text{,no detector} \\ f(f'(CCI_i), alert\ conf.) & \text{,otherwise} \end{cases} \quad f' = \begin{cases} \max(CCI_i) & \text{,OR edge} \\ \min(CCI_i) & \text{,AND edge} \\ mean(CCI_i | CCI_i > \tau) & \text{,Quorum edge and quorum met} \\ 0 & \text{, Quorum edge and quorum not met} \end{cases}$$

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

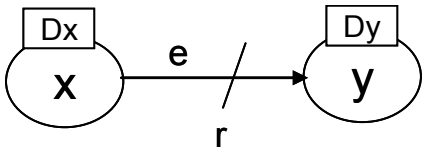
The intuition behind our implementation is that for an OR edge, the node can be achieved if any of its children nodes is achieved and therefore the likelihood (due to its children) is the maximum of all of its children. For an AND edge, all the children nodes have to be achieved and therefore the likelihood 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, the I-GRAPH can be considered to be a Bayesian network, and the conditional probability distributions derived from the edge relations (i.e. OR, AND, Quorum).

In CCI calculation, 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 can't 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.

3.7.1 Effect of Response and Edge Propagation Factor



$x.AV$: detector alert confidence value from Dx

$y.AV$: detector alert confidence value from Dy

$\mathrm{P}(x) \leftarrow x.cci \leftarrow x.AV$

$y.cci \leftarrow f\left(x.cci \times e.EPF \times (1 - r.EI), y.AV\right)$

$\mathrm{P}(y) \leftarrow y.cci$

Figure 3.5. Effect of response and EPF in I-GRAPH inference

In reality, attack may not always propagate across an I-GRAPH edge. This can be due to a less skilled adversary or a more secure system design. We introduce the edge propagation factor (EPF) to model the likelihood of an attack propagating on an edge. EPF is a value between 0 and 1. It is used to attenuate the CCI value from a child node in the CCI calculation. In Figure 3.5, we show an example where the CCI from node x is attenuated by the EPF value *e.EPF* on edge *e*.

The Effectiveness Index (EI) of a response indicates the likelihood of success of the response, For better understanding, Figure 3.5 shows a simple example on how response *r* on an edge *e* affects the CCI values on the parent and the child node on the edge. The CCI values are then used as the estimates of the probabilities of nodes being achieved P(*x*) and P(*y*).

EI and EPF values are estimated by ADEPTS through observation of alerts. Conceptually for a response deployed on an edge, if attack propagation (based on incoming detector alerts) is observed, the EI value of the response will be decreased. Otherwise it will be increased. When the response is not deployed, if attack propagation is observed, then the EPF value will be increased. Otherwise, it will be decreased.

3.7.2 False Alarm Estimation

It is important that the response system not use the alarms from imperfect detectors as the only triggers. The detectors may have both false alarms and missed alarms. 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

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

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

$$false\_alarm\_probability = \alpha \times links\_probability + (1-\alpha) \times history\_probability$$

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.

$$history\_probability = \beta * links\_probability + (1-\beta) * history\_probability$$

3.7.3 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 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,

$$\mathrm{rCCI} = \begin{cases} g(g'(\max(\mathrm{rCCI}_i), ac)) & \text{, node has } d \text{ and } oe \\ g(\max(\mathrm{rCCI}_i)) & \text{, node has no } d \\ g(ac) & \text{, node has no } oe \\ 0 & \text{, otherwise} \end{cases} \qquad g(y) = \begin{cases} y & \text{,AND edges} \\ y \times \dfrac{\mathrm{CCI}_{child}}{\sum \mathrm{CCI}_{children}} & \text{,OR edges} \\ y & \text{, Quorum \& } \mathrm{CCI}_{child} > \tau_N \\ 0 & \text{, Quorum \& } \mathrm{CCI}_{child} \le \tau_N \end{cases}$$

where $rCCI_i$ corresponds to rCCI of the i^{th} 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) $f'(\text{CCI}_i) > k \times \text{alert confidence}$; (ii) $f'(\text{CCI}_i) > \tau_M$; (iii) $\max(\text{rCCI}_i) > k \times \text{alert confidence}$; (iv) $\max(\text{rCCI}_i) > \tau_M$, 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.

3.8. Inference on I-GRAPH: Bayesian Network

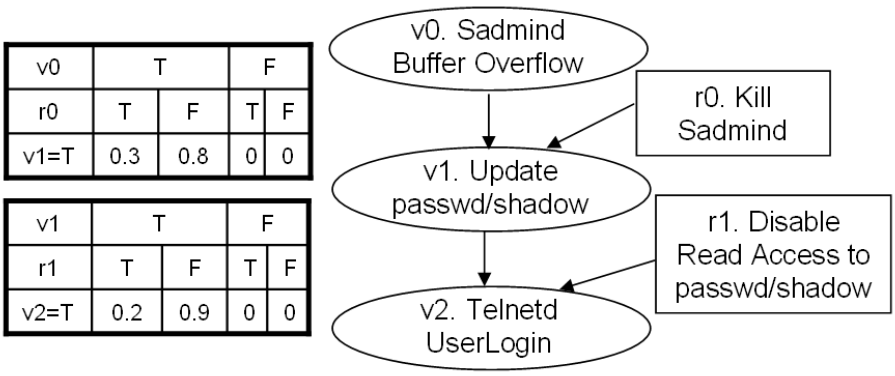


Figure 3.6. Bayesian Network based Attack Graph Model

Bayesian network [42] can also be used to perform inference in the I-GRAPH. A Bayesian Network based I-GRAPH includes nodes that represent attack steps (v0, v1, and v2 in Figure 3.6) and nodes for responses (r0 and r1). For each node, a conditional probability table (CPT) is used to model the probability on the states of a node given the states of its parents. For root nodes, the CPT is used to represent the prior probabilities. For an attack step node, the possible states are true, i.e., the step is achieved or false, i.e., the step is not achieved. For a response node, the states are again true, i.e., the response is deployed and false, i.e., the response is not deployed. Also, for both types of nodes, the

state can be set to NA (not available), which means ADEPTS is not sure about the state of the node.

3.9. Limitations

The concept of attack graph has been widely used in modeling multi-stage attacks. The creation of attack snapshots for an attack from detector alerts is not the focus of this work, and one can rely on techniques from [8, 9, 40] for the construction of attack graph. We thus assume an attack snapshot is provided by the detection framework and is accurate in the sense the detection framework should address excessive false/missed alarms. We present a basic technique to address false/missed alarms in Sec. 3.7. This, however, is not the focus of this work. One should refer to a more comprehensive approach such as [43] on improving detection accuracy of multi-stage attacks.

Many different metrics can be used to evaluate an automated response system [11]. Some examples include throughput of the transactions the system can sustain, reliability of the system, usability, survivability, etc. Our discussion is centered on survivability only. We formally define the impact vector metric to measure the change in survivability. We assume the impact vectors for each response action and each I-GRAPH node are given.

The impact vector metric only looks at the expected value of the resulting costs (impact) from using a response combination for an attack in a system. The individual impact vectors from response actions and achieved I-GRAPH nodes are assumed constants. The proposed response model (Sec. 1.1 and Sec. 3.3) is set to optimize responses based on the expected impact. There can be alternative ways to define the metric for evaluating the survivability. For instance, the impact vector from achieving an I-GRAPH node can be a function of time (e.g. a DDoS attack at midnight is probably less damaging than a DDoS attack during business hours) instead of being a constant. As a result, the corresponding metric requires integration across time of the costs over the I-GRAPH nodes / response actions. This is a more involved calculation requiring many more parameters.

4. AUTOMATED RESPONSES

For distributed systems with nearly exponentially large number of interaction effects among multiple components, pre-configuring static pairs of detector alarm and the corresponding response when the alarm is flagged is laborious. We show (in Sec. 8.3) that this design has inferior runtime performance due to the dynamic workload on the system and due to the changing nature of attacks.

In the following, we present a system called ADEPTS I as a first step in instantiating the proposed model for automated response, which was introduced in Sec. 1.1. In choosing responses, ADEPTS considers both the severity of the current situation (what damage to the system the attack will cause or has already caused) and the effectiveness of the responses. The mapping between responses and detectors is therefore dynamic. In the end, when the attack ceases, ADEPTS will evaluate the actual effectiveness of the deployed responses. Ineffective responses will be ignored by ADEPTS in dealing with future attacks. This corresponds to the use of history information in the proposed model.

It is often a challenge for an automated response system to handle multiple concurrent attacks on a system. The response mechanisms due to the different manifestations of the distinct incidents may interact in arbitrary ways. For example, the response taken due to one incident may make that due to a second incident redundant, or make it easier to proceed. It also becomes difficult to identify the effectiveness of a response when the different incidents are not identified and handled separately. ADEPTS I 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 incident 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 instances whose responses would not

interfere as distinct and separate instances. An attack sub-graph[4] is created from the I-GRAPH for every incident 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*[5] of graphs to determine appropriate *reference response*s.

ADEPTS I is the first step in designing an automated response system beyond the BASELINE model (Sec. 1.1). This will be evidenced by the response location determination in Sec. 4.3. The mapping between response actions and detectors is not restricted within a pair of response actions and detectors. Also the estimation of response effectiveness (Sec. 4.8) involves the use of history and is beyond the BASELINE model. In determining responses, ADEPTS I employs a simple heuristic to pick responses, which are effective and time-efficient (Sec. 4.4). The heuristic does not guarantee the maximal survivability property mentioned in Sec. 1.1. However, empirical results (Sec. 8.4) indicate cases where even such a sub-optimal design is more than enough to out-perform the BASELINE Model.

[4] In ADEPTS, we focus more on dealing with attacks, which is the reason why we use the term "attack sub-graph" rather than "incident sub-graph" here. However, the techniques used in ADEPTS can be extended to handle an incident which could be due to attacks or failures.

[5] Similarly, we use the term "attack template library" rather than the term "incident template library" as in ADEPTS, we focus more on dealing with attacks.

4.1. Design Overview

First, we give an overview figure (Figure 4.1) that shows the different phases in the operation of ADEPTS I. In the following sections, we will explain each of the phases and refer back to this figure.

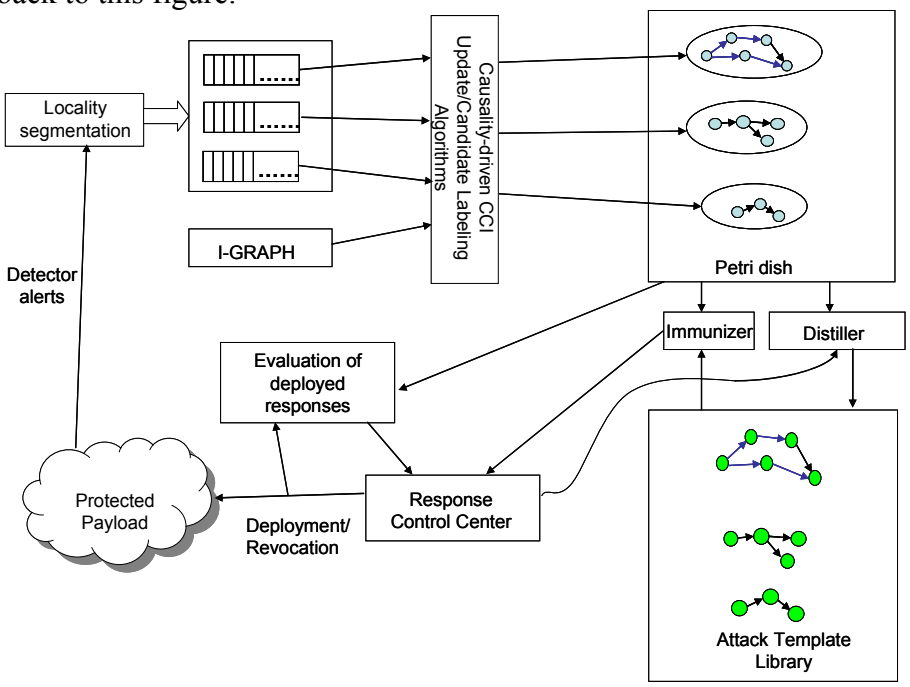


Figure 4.1: Overview of the different phases of ADEPTS

Throughout ADEPTS I, 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.

Table 4.1. Pseudo-code for attack sub-graph creation when new alert event arrives

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) {
1.   bNeedNewSub-graph := true
2.   for each existing sub-graph S do {
         if LocalityTest(S,E) is true then do {
3.           bNeedNewSub-graph := false
4.           Add the nodes and edges induced by E
             into S
5.           CCIUpdate(S,E)
6.       }
7.   }
8.   if bNeedNewSub-graph is true then do {
9.       Create new sub-graph NS
10.      Add node corresponding to E into NS
11.      Add NS into P
12.      CCIUpdate(NS,E)
13.  }
}
```

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

```
LocalityTest(S,E) {
1.  D: = the node in the I-GRAPH which
    corresponds to event E.
2.  if D is NULL then return False
3.  for each node X in S {
4.      H: = the node in the I-GRAPH which
        corresponds to X
5.      Dist := shortest distance between D and H
6.      If Dist < T then return (true, S)
7.  }
8.  return false
}
```

4.2. Attack Sub-Graphs

In ADEPTS I, 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 precursor system of ADEPTS [33], where all alerts were made to operate on the I-

GRAPH structure itself. We will refer to the earlier version as version 0. There are several motivations for the current design. In version 0, 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 point 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 Table 4.1. For each alert that is received by ADEPTS, the alert is mapped to a node in the I-GRAPH and then 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 (Sec. 4.3). 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 4.6). Therefore, under this design, all alerts will always be mapped to some node in the I-GRAPH.

In reality, it can be difficult to accurately determine which attack instance an alert belongs to, though some effort has been made in [44] by clustering source IP addresses, destination IP addresses, source ports, destination ports, user accounts and 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.

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

4.3. 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.

4.3.1 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 $> \tau$; (ii) *Weak Candidate* (WC), if CCI $\leq \tau$ but further traversal across only AND edges can reach a SC node; (iii) *Very Weak Candidate* (VWC), if CCI $\leq \tau$ 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 as alerts are available. This is a key differentiator from the BASELINE model.

4.4. 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*.

4.4.1 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. The Response set computation algorithm (Section 4.3) 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.

4.4.2 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 $RI = a.EI - b.DI$, 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 65000 or 16660 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.

4.5. Matching in Attack Template Library

ADEPTS maintains an attack template library of attack patterns (*attack snapshots*), 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.

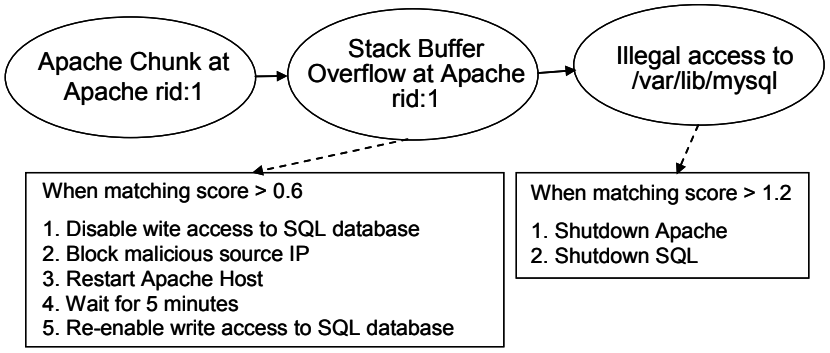


Figure 4.2: Example of a static attack pattern and reference responses

The matching between the static attack pattern and a growing attack sub-graph is handled by the *Immunizer* in Figure 4.1. The algorithm for calculating the matching score is illustrated in Table 4.2. Figure 4.2 shows an example of a static attack pattern.

Raw attack patterns are patterns automatically generated by ADEPTS I from the attack sub-graphs via the *Distiller* in Figure 4.1. 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 4.2, ADEPTS I 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.

4.5.1 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 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 Figure 4.1 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 Table 4.2.

Table 4.2. Algorithm for calculating matching score

// N : a node being added to an existing sub-graph G

// N.S[k] : the per-node matching score for N with respect to pattern I_k

// N.ChildNodeInCCIUpdatePath : the child node of N that contributes to N's CCI. (see $f'(CCI_i)$ in Section 3.7)

// root_nodes_of(G) : nodes in G that do not have out-going edges.

…… After the field N.ChildNodeInCCIUpdatePath is determined……..

1. for k := 1 to M do

   If there's a path P from the corresponding node of N.ChildNodeInCCIUpdatePath to the corresponding node of N in I_k then do

   N.S[k] := N.ChildNodeInCCIUpdatePath.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 $h \in 1..M$
3. Let $p \in 1..M$ such that $\sum_{N \in root\_nodes\_of(G)} N.S[p]$ is maximal.
4. If $\sum_{N \in root\_nodes\_of(G)} N.S[p] >$ preset-threshold , then return responses from I_p.

4.6. 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).

4.7. 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 from the algorithm outlined in Section 4.4. 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.

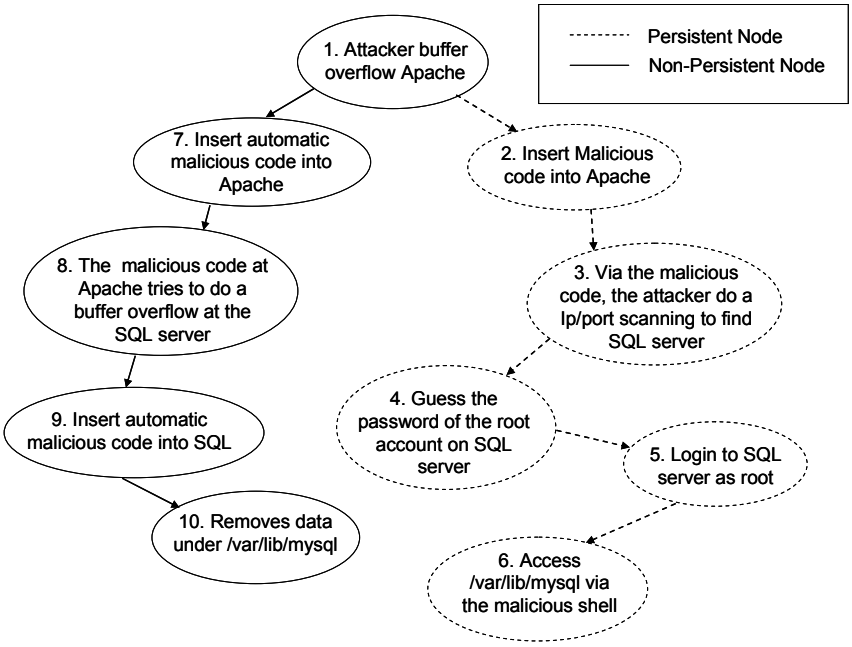


Figure 4.3. Persistent attack example

An example of a persistent attack is shown in Figure 4.3. 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.

4.8. 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.

4.8.1 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 Figure 3.2, 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, $EI_{new} = EI_{old} - \alpha \frac{CCI_7}{CCI_8 + CCI_7} \frac{CCI_1}{CCI_6 + CCI_1}$.

4.8.2 Tuning Response CPT Values in Bayesian Network

If one opts for Bayesian Network for inferencing on I-GRAPH, feedback of deployed responses can be achieved through standard Bayesian Network parameters learning such as EM learning [42]. This results in tuning of values in the conditional probability table entries corresponding to those deployed responses.

4.8.3 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.

4.9. Complexity Analysis

Table 4.3. Notations for complexity analysis

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

The worst case computational complexity follows directly from analysis of the algorithms presented in Sections 3 and 3.1. 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)

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).

4.10. Limitations

We assume reliable and secure communication channels between detectors and ADEPTS. Similarly, reliable and secure communication channels are assumed between response execution agents and ADEPTS. This can be achieved through existing work on secure network service architecture such as [45].

Alerts may arrive at ADEPTS out-of-order, i.e., not following the causality of the attack steps. To handle this, alerts are first put in a reorder buffer and after a time period are processed in ADEPTS. However, if the communication channels cause the alerts to arrive out-of-order beyond the time period for which alerts sit in the reorder buffer, then ADEPTS can mistakenly process an alert before processing its causally preceding alerts.

5. OPTIMAL RESPONSES

The few available dedicated IRSs for distributed systems [12, 19-21, 46, 47] have one or more of the following characteristics—they have a static mapping of symptoms from the detector to the response, do not take feedback into account for determining future responses, assume perfect detectors with no missed and no false alarms, or assume perfect success rate for a deployed response. The complex interactions among the complex software running the distributed applications, the non-determinism in the execution environment, and the reality of new forms of intrusions surfacing would make any one of the above characteristics undesirable. Importantly, the existing work does not present a method for reasoning about or evaluating the optimality of a chosen set of responses. The presented protocols, including our earlier work ADEPTS I, take a heuristic approach and do not give a globally optimal response solution prescribed in our proposed response model in Sec. 1.1. By globally optimal we mean the set of responses that maximizes the system survivability. How far each solution is from the optimal is also not clear. Optimality is an important metric because it allows a system designer to reason about how well a given set of responses with which the IRS is populated can work for the target attack scenarios. This may point to modification of the response repository in the IRS. We address the problem of optimal response selection in this chapter.

We present a system called SWIFT (ADEPTS II) to reason about the global optimality of a chosen set of responses in a distributed system of interacting services. The optimality criterion takes into account the impact of a deployed response to the services in the system and the impact of not deploying a response to the services which could result in further spread of the attack. This system is probabilistic since the future spread of the attack and the effectiveness of a response are unknowns and can only be estimated. The optimality of a response set is a global or system-wide property and thus optimizing the response choice on each compromised service individually as seen in [33],[48] may not be sufficient. The globally optimal solution must account for the fact that there exist dependencies between responses available at the different services. For example, blocking all traffic from a specific subnet at the ingress point will make it redundant to

impose restrictions at an internal service on traffic from a host within the subnet. Also the effectiveness of a response depends on the time to deploy the response.

We prove that solving the optimal response determination problem is NP-hard. Both the number of responses and the number of services (including replicas) can grow large with increasing system size and complexity. Since it is imperative to deploy prompt responses at runtime to counteract automated attacks, we design an approximate solution. Our solution employs genetic algorithm (GA) [49] based search through the universe of possible responses. As multiple attack instances of an attack type or its variants are seen, SWIFT updates the effectiveness of the deployed responses and the quality of the chromosome pool used to initiate the GA-based search. Thus, SWIFT adapts to provide better responses as history builds up in the system. SWIFT can respond to attack variants through an approximate graph matching algorithm and population of chromosomes from the approximate match. Attack variants are particularly relevant for distributed applications where different order of observing alerts from different machines may give the impression of an attack variant.

The SWIFT system is an instantiation of the proposed model of automated response incorporating the two properties of maximizing survivability and tolerating new types of attacks (Sec. 1.1). The experiments (Sec. 8.6 and Sec. 8.8) indicate improved survivability from SWIFT compared to ADEPTS I, the ability of SWIFT to adapt its responses as increasing numbers of attack instances are seen, and its ability to handle attack variants (Sec. 8.9).

5.1. Deficiency and Sub-Optimality in ADEPTS I Response

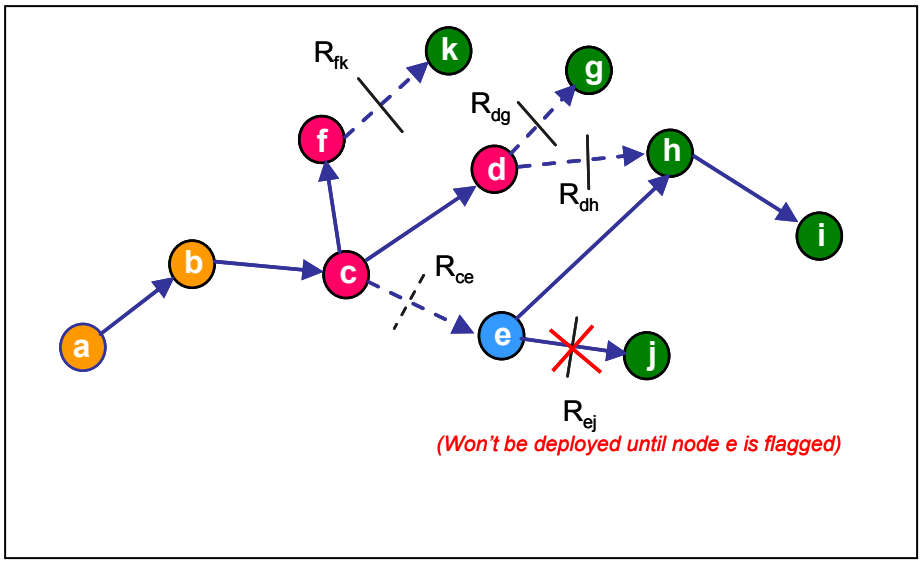


Figure 5.1. Deficiency in ADEPTS I response location

A deficiency in ADEPTS I response mechanism is that the set of the potential responses on an edge e are restricted to the responses whose opcodes match with the channel type of edge e and whose operands are compatible with the alerts on the source node of edge e, and this constraints ADEPTS I from applying precautionary responses. For example, in Figure 5.1, let's assume that node a, b, c, d, and f are achieved by the attacker. Also, assume that nodes c, d, and f are flagged by ADEPTS for deploying responses. Consequently, only responses R_{fk}, R_{dg}, R_{dh}, and R_{ce} will be consider by ADEPTS I at this point in time.

Let's define Iv_x to be the overall impact from the compromised node a, b, c, d, and f and the deployed responses R_{fk}, R_{dg}, R_{dh}, and R_{ce}. Also, let's define the impact from losing node e is Iv_e, and the impact from deploying response R_{ej} to be Iv_r. Finally, let's define the impact from losing node j to be Iv_j.

Now, let's assume that responses R_{fk}, R_{dg}, and R_{dh} are 100% effective, which means they are always effective in blocking the propagation of the intrusion on the corresponding edges. And let's assume that response R_{ce} is a partially effective response, which has a probability of p in successfully blocking the attack propagation. Let's assume that R_{ej} takes the same time as the attack propagating from node e to j to deployment, which means that R_{ej} will be effective when it is deployed before node e is flagged and

ineffective otherwise. So depending on whether R_{ej} will be pre-deployed, the expected impact on the end-scene from this response mechanism with respect to this intrusion will be:

If R_{ej} is pre-deployed,

$$\begin{aligned}\text{Expected impact} &= Iv_1 \\ &= Iv_x + Iv_r + p*Iv_e \\ &= (Iv_x + p*Iv_e) + Iv_r\end{aligned}$$

If Rej is not pre-deployed (current ADEPTS),

$$\begin{aligned}\text{Expected impact} &= Iv_2 \\ &= Iv_x + p*(Iv_e + Iv_j + Iv_r) \\ &= (Iv_x + p*Iv_e) + [p*Iv_r + p*Iv_j]\end{aligned}$$

Therefore, in the case when $|Iv_r|$ is small, $|Iv_j|$ is high and p is high, one would expect $|Iv_2| > |Iv_1|$. And in this case, the ADEPTS I response would be sub-optimal. The adaptation process in ADEPTS I won't be able to fix this deficiency as the current adaptation mechanism doesn't deal with this kind of precautionary responses (responses acting on nodes which haven't been reached by the attacker).

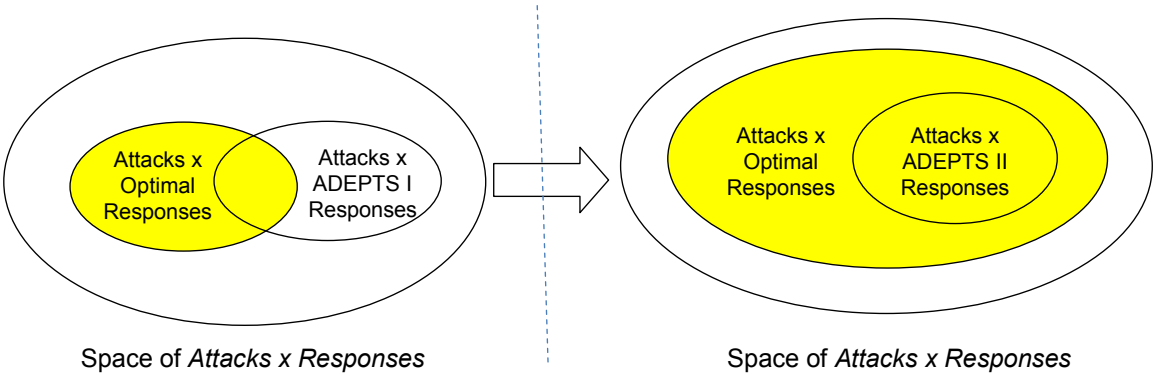


Figure 5.2. ADEPTS I/II with respect to optimal response

The relation of ADEPTS I with respect to the optimal response in the space of attacks x responses can be seen as Figure 5.2. Here we are going to present a improved version of ADEPTS named SWIFT (ADEPTS II) to pursue the optimal response set.

5.2. Framework for Global Optimal Response

SWIFT is set to pursue the global optimal response combination $RC_{i,opt}$ mentioned in Sec. 3.6. This involves selecting response actions that maximize the survivability (Eq. (3.3) in Sec. 3.6). We prove this optimal response determination to be an NP-Hard problem as shown in the following. SWIFT therefore uses genetic algorithm to approximate the optimal response combination. The details are presented in Sec. 5.7.

5.2.1 Intractability of Optimal Response Determination

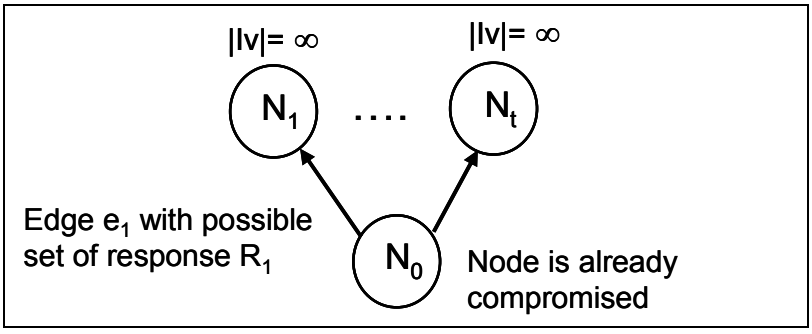


Figure 5.3. Transformation to map set covering problem to ORD

Consider the small I-GRAPH in Figure 5.3. Let $\boldsymbol{E} = \{e_1, ..., e_t\}$. Each edge in $\boldsymbol{E}$ has a set of possibly overlapping responses. Each response has the same probability of success and identical *Iv*'s. The *Iv* of each node $N_1, ..., N_t$ is ∞. Thus ORD will deploy a response on each edge in $\boldsymbol{E}$. By definition of ORD, it will generate a response combination $\boldsymbol{R}$ such that the cost is minimized, which for the special settings implies that the number of responses is minimized. Thus the responses in $\boldsymbol{R}$ cover the set $\boldsymbol{E}$. This is the solution to the set covering problem. The reduction is obviously polynomial. Hence, ORD is an NP-hard problem in terms of the input size of number of responses and number of nodes.

In practice, for a reasonable-sized distributed system, there are many possible attack steps and therefore many possible response steps. For example, there are several research efforts aimed at scalable generation of attack graphs with tens of thousands of nodes [40]. Also, there are many possible services and therefore attack graph nodes. Again, notice the significant research efforts aimed at diagnosing root cause problem in services which aim at scalability to a large number of services [50, 51]. The intractability is observed in practice not just for a few corner cases, but in the average cases as well. This is due to the dependences between responses and attack steps.

5.3. Design Overview

The overall execution flow in SWIFT's search for optimal response combination is shown in Figure 5.4.

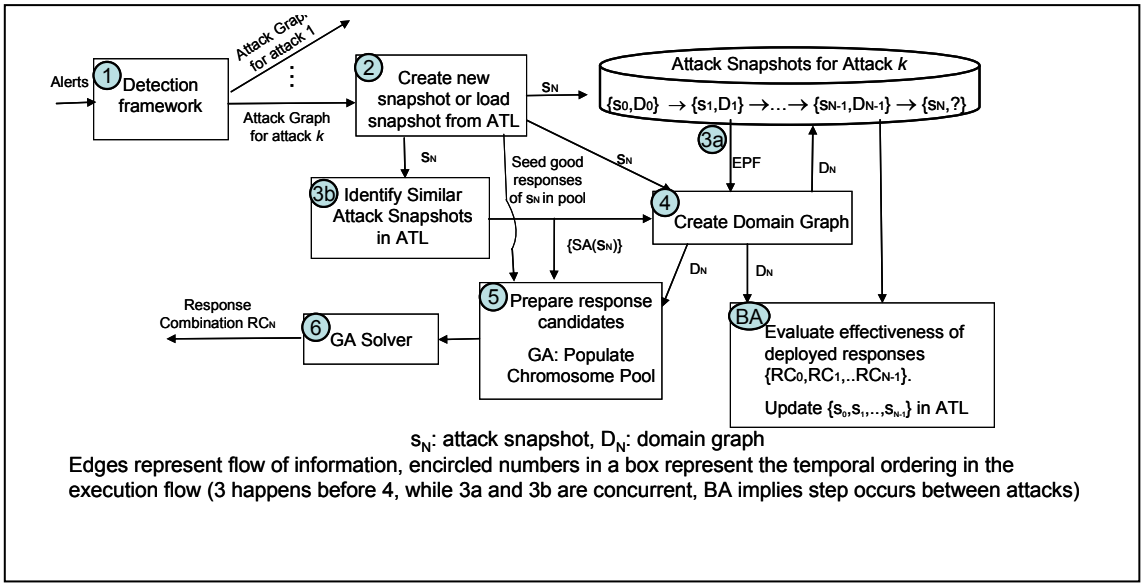


Figure 5.4. Overall flow for the steps in SWIFT to respond to an attack

5.4. Attack Template Library (ATL) and Attack Snapshots

SWIFT seeks to adapt its responses based on previous attack snapshots. Thus it is important to store the history of attack snapshots and prior responses. This is maintained in the *Attack Template Library* (ATL).

The ATL houses snapshots of attacks seen so far. Each snapshot entry *s* in the template library contains the following information: *s.g*: the sub-graph of the I-GRAPH

with nodes that have been achieved at snapshot *s* and the corresponding edges; *s.predict*: the path prediction table used to predict the propagation trend in the I-GRAPH from the snapshot *s* (Section 5.5); *s.rc*: the most effective response combinations previously found by SWIFT for snapshot *s*, *s.r*: the responses used previously for this attack snapshot and their EI values. Thus, the EI value of a response is maintained per snapshot, rather than globally for the response. This acknowledges that a response's effectiveness also depends on how far ahead of the attack front reaching the response node, i.e, on the time to successfully deploy a response. Also when the EI value is used by SWIFT, it picks it up from a Normal distribution with the mean and the variance of the EI observed so far. This design, called **fuzzy EI**, ensures that a response that falsely has a low EI value will eventually be redeemed, deployed in a response combination, and its EI reevaluated.

When the detection framework sends attack graph g_N to SWIFT, SWIFT will check in the ATL if there is an existing attack snapshot s_e with $s_e.g = g_N$. If it does, s_e is loaded from the ATL as s_N (step 2, Figure 5.4) for subsequent SWIFT operations. Otherwise a new snapshot is created. If space is a constraint, SWIFT deletes snapshots from the ATL by various criteria—by time of creation or time of last access, frequency of access, or the snapshot with the lowest cumulative $|Iv|$ of its nodes.

5.5. Predicting the Escalation of Attack

Given an attack snapshot *s*, while there are many possible follow-on attack steps, in practice, some are much more likely. SWIFT estimates the likely follow-on steps so that the search space is restricted and unnecessary responses are not deployed. The attack snapshot prediction table and the edge propagation factor tuning algorithms are used for this purpose.

To track the likelihood of follow-on steps, SWIFT maintains a prediction table *s.predict* for each snapshot *s*. The table entry *s.predict[e]*, which is called edge propagation factor for edge *e* (*e*.EPF), tracks the likelihood of an attack propagating on the edge *e*.EPF is a real number in the range [0, 1] and is used in the creation of the so called *Domain Graph* in next paragraph, which defines the search space explored by SWIFT in making the response decision. SWIFT increases EPF on an edge if attack propagation is perceived on the edge and decreases EPF otherwise. For example, in Figure 3.5, assuming response *r* is not deployed and detector *Dx* fires, *e.EPF* will be increased if detector *Dy* fires subsequently. Otherwise, it will be decreased. EPF on edge *e* is used to tone down the contribution to the probability P(*y*) from node *x*. Therefore, if

the EPF value *e.EPF* is low, this would decrease the likelihood of SWIFT deploying responses around *y*.

If Bayesian Network is used to provide inference on I-GRAPH, *s.predict[e]* is modeled by the corresponding entry in the conditional probability table (CPT) of the destination node of edge *e* as shown in Figure 3.6. There's no need to use additional structures to hold the values. The equivalence to EPF tuning is the Bayesian Network parameter learning for the CPT entries[42].

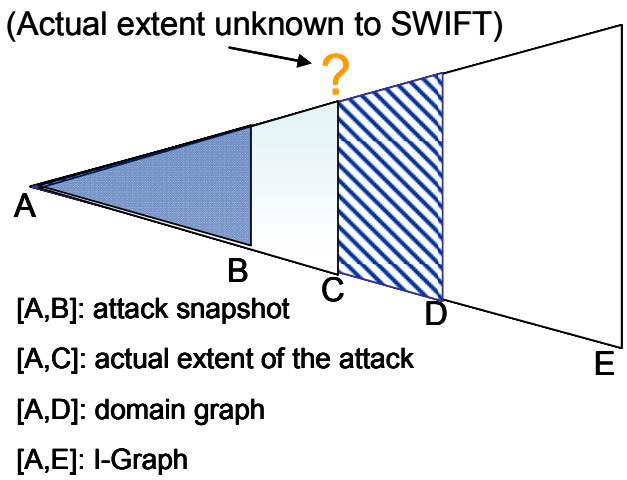


Figure 5.5. Relations between attack snapshot/domain graph/I-Graph

5.5.1 Domain Graph

The Domain Graph $D(s) \supseteq$ s.g and is a subgraph of I-GRAPH, which provides an approximate bound on the nodes that may be reached by an adversary from a snapshot *s* (Figure 5.5). In Eq.(3.1), when we calculate the expected impact vectors due to the nodes in the I-GRAPH, we consider all the nodes in the I-GRAPH. This can adversely affect the performance since the I-GRAPH is likely a large structure for any large real-world distributed systems and many nodes in it will have vanishingly low probability of being achieved based on the current snapshot. The Domain Graph subsets the nodes to be considered so that a more timely and more accurate reaction to the attack can be taken.

Given the I-GRAPH *I* and a snapshot *s*, the Domain Graph $D(s) = (V, E)$ where $V = \{\{\text{node } n \in I \text{ such that } P(n) \times |Iv(n)| \text{ is greater than a given threshold } T\} \cup \{\text{node } n \in I \text{ such that } n \text{ is on the path from } n_x \text{ to } n_y \text{ in } I \text{ where } n_x, n_y \in V\}\}$ and $E=\{e|e \in edges(I) \text{ and } e: (u, v)\}$, where $u, v \in V\}$. This is computed in step 4 of Figure 5.4.

Essentially, domain graph gives the worst case estimate, assuming no responses are going to be deployed, on the extent of an attack and bounds the search space of the Genetic Algorithm that we discuss next. The estimation of domain graph is refined through the tuning of the EPF values (Section 5.5) and the EI values of the responses already deployed. In the ideal case, the estimated domain graph should coincide with the actual extent of an attack. (e.g. C=D in Figure 5.5)

5.6. Similarity of Attack Snapshots SA(S_N)

Similar attack snapshots $SA(s_N)=\{s_X: s_X \in ATL \text{ and } |s_X.g\text{-}s_N.g| < threshold\}$ are used in the creation of the domain graph and the preparation of response candidates (Step 3b in Figure 5.4). The objective is to rely on knowledge learnt previously from similar attack snapshots for helping provide responses to the current attack snapshot s_N. This is useful when s_N by itself does not contain enough historical information for deriving low-cost responses.

The difference $|g_x\text{-}g_y|$ between two I-GRAPH fragments g_x and g_y is defined as

$$|g_x\text{-}g_y| = 1 - \frac{(\#\text{ nodes in } g_x \cap g_y) + (\#\text{ edges in } g_x \cap g_y)}{\#\text{ nodes and edges in } g_x \cup g_y}$$

5.7. Genetic Algorithm (GA)-based Response Mechanism

As the problem of deciding the optimal response combination (ORD) for an attack snapshot has been proved to be NP-hard, we focus on an approximate solution using a GA framework [49]. Following Figure 5.4 step 6, this corresponds to designing a response mechanism `Respond(.)` (algorithm shown in Table 5.1), which takes the snapshot s_N from Step 2 and generates the approximate optimal response combination RC_N. The history information used here is embedded in s_N and $R_{deployed}$, the responses deployed thus far.

Within this framework, we map each response combination onto a chromosome, and the problem of searching for the best response for an attack snapshot is then translated into looking for the best chromosome from the chromosome pool over multiple evolutions. Often using genetic algorithm to perform optimization is an expensive process [38] due to the requirement of search through a huge chromosome pool over many evolution cycles to get a good solution. We reduce the execution time by selectively initializing the chromosome pool.

SWIFT only considers the responses within the Domain Graph that have not been deployed yet. This set of applicable responses is given by R_A. The encoding scheme is that each chromosome c is an $|R_A|$-sized bit vector, with each bit uniquely mapped to a response $r \in R_A$.

To populate the chromosome pool (Step 5 in Figure 5.4), first, SWIFT relies on the history information from the snapshot, namely $s_N.rc$ and $s_N.r$ (i.e., the best response combination found so far for this snapshot and responses deployed and their EIs). Second, SWIFT relies on this same information from past similar attacks. Third, SWIFT populates the chromosome pool with heuristic-based responses from ADEPTS I and fourth, with a set of randomly filled chromosomes.

The *fitness* of a chromosome c, is determined by the response combination RC for c. The fitness of chromosome c is defined as $fitness(c) = 10^{-|Iv(RC)|/\text{dimension}(Iv)}$. This fitness function satisfies some desirable properties – high $|Iv|$ translates to low fitness and $|Iv|$ of zero or infinity are handled. A Genetic Algorithm Solver (Step 6 in Figure 5.4) is then invoked to systematically probe through the space of response combination RC through the typical GA evolution process [49]. The high-level concept here is those response combinations that yield low cost values will be returned by the GA Solver in the end.

The ORD problem is a NP-Hard combinatorial global optimization problem. Techniques such as Simulated Annealing [52] or Monte Carlo Method can also be used. We chose GA because it has been widely used and is easy to implement. The framework of automated response and the ORD problem are generic with respect to the optimization techniques used to solve them.

Table 5.1. GA based response mechanism

Algorithm: Respond
Input: latest attack snapshot s_N
Output: approximated optimal response combination RC_N
Pre-defined Constants:
chromosome_pool_size: a constant on the chromosome pool size.
v% : the percentage of top chromosomes to be kept in the history.
max_evolutions: maximum number of evolutions per iteration for the GA.
rc_size: the maximum size of the set $s_N.rc$ of best response combinations previously found.
$R_{deployed}$: responses deployed thus far
R_A : The set of applicable responses
Method:
1. Create Domain Graph $D_N=D(s_N)$.
2. Derive R_A from $R_{deployed}$ and D_N.
3. Initialize GA chromosome pool through four sources defined in Section 5.7.
   pool = GA_PopulateChromosomePool (ATL, s_N, D_N, *chromosome_pool_size*).
4. Perform GA evolution cycles
   for *i*=1 to *max_evolutions* {
      pool = GA_NextChromosomeGeneration(*pool*).
   }
5. Update the best response combinations
   best_chromosomes = {the top *v%* of chromosomes in *pool* (wrt fitness)}.
   $s_N.rc$ = the top *rc_size* chromosomes from ($s_N.rc \cup$ *best_chromosomes*).
6. Find chromosome $RC_N \in s_N.rc$ with highest fitness.
7. Return RC_N.

5.8. Limitations

Both CCI Inference (Sec. 3.7) and Bayesian Network Inference (Sec. 3.8) use the statistical means of past observations to estimate the attack escalation and response effectiveness. This assumes a probability distribution with small variance. In our experiments (e.g. Sec. 8.8), we notice non-smooth patterns in the |Iv| plot due to non-deterministic attack escalation and response effectiveness. If the variances of the corresponding distributions are too large, further unstable performance from the system can be expected. For future work, it may be useful to estimate the variance and account for it as well.

The current design on dealing with attack variants in SWIFT assumes that similar attacks share similar EPF values (attack escalation parameters) and EI values (response effectiveness). This is by and large an open research question, which requires a thorough

study of existing multi-stage attacks to prove or disprove it. Such a dataset is unfortunately not available in the public domain at present.

6. RESPONDING TO ZERO-DAY ATTACKS

Zero-day attack [53] is the kind of attack which exploits undisclosed vulnerabilities or vulnerabilities for which patches are not yet known. The typical way to deal with it is to limit the possibility of zero-day attack. One approach is to strip unnecessary functionalities thus eliminating potential vulnerabilities [45]. Another approach is to harden the existing system through tighter security policies such as SELiunx. However, both approaches can significantly hamper the flexibility and the diverse functionality of a system. Another approach is to use detection technologies that rely on detecting the "manifestations" of each attack stage rather than the mechanism of launching the attack, which is unknown for zero-day attacks. One example is code emulation based malware detection [54], where they determine if a malware is malicious or not based on its observed behaviors when running on an emulator. They usually have high false positive rates and therefore are typically used on an advisory basis mostly.

A zero-day multi-stage attack has two characteristics: (1) some or all the stages are based on unknown exploits; (2) some or all the interconnections among the stages are unknown. In short, the complete mechanism behind a multi-stage zero-day attack is not known *a priori*. Current work on dealing with multi-stage attacks fall short of handling zero-day multi-stage attacks. Existing work on alert correlation and attack graph generation [9, 55] and prevention based on attack graph analysis [32, 40] are explicitly predicated on known vulnerabilities. Automated intrusion response systems such as SWIFT or ADEPTS I are designed to work on a prebuilt attack graph that captures all escalation paths for attacks. Host-based solution such as pH [56] provides only local responses to intrusions detected on the corresponding host and is not designed to deal with multi-stage attacks. Most existing IRS [19, 46, 56-58] are static in that the response mechanism is tied to the specific alerts by a rule and cannot adapt to new kinds of attacks.

Here, we come up with a solution that enables automated response to zero-day multi-stage attacks. *There are two major contributions from this work.*

The first contribution is a modeling technique for zero-day attacks. This comprises an object-oriented hierarchical model for the component, the detector alerts in

the system and the interconnections between the components that allow attack escalation. The model is used to represent a system configuration specification for the protected system, which is currently a manual input. This is then used for online attack graph generation. Distinct from the extensive body of work on attack graph generation, we generate the graph at runtime based on received alerts, and the graph only captures the local part of the system for which the alerts have been observed. Also, we map the alerts to manifestations of each attack stage, rather than the mechanism of achieving the stage. For example, an alert may flag that the access control list has changed (maliciously in this case), but not the steps used by the adversary to achieve this.

The second contribution is a technique called *conceptualization* of an attack graph. We observe that even for zero-day attacks, the concepts behind them are not always new. For example, a conceptual description of many distinct attacks is memory overflow followed by data execution. This concept is so commonly seen that to deal with it processor manufacturers have introduced the NX-bit, and OS manufacturers have introduced the idea of data execution prevention. We conceptualize the component and the detector associated with an attack graph node by moving each up to a super-class in the object-oriented hierarchy. For example, an Apache Web Server and a Microsoft IIS Web Server can both be conceptualized into a "Web Server" or simply a "Program". Similarly, an alert corresponding to "Java array out of bound exception" can be conceptualized into a "Buffer Overflow", a "Memory", or a "Got Effect" (something is wrong) alert. Essentially, we use conceptualization to map two distinct attacks into the same (or closely similar) attack graph at a high-enough level of conceptualization. Then we use the ability of our IRS, ADEPTS to leverage information from the previously-seen attack to deploy effective responses to the zero-day attack. The proposed system is named ADEPTS III / ORIGIN.

6.1. Design Overview

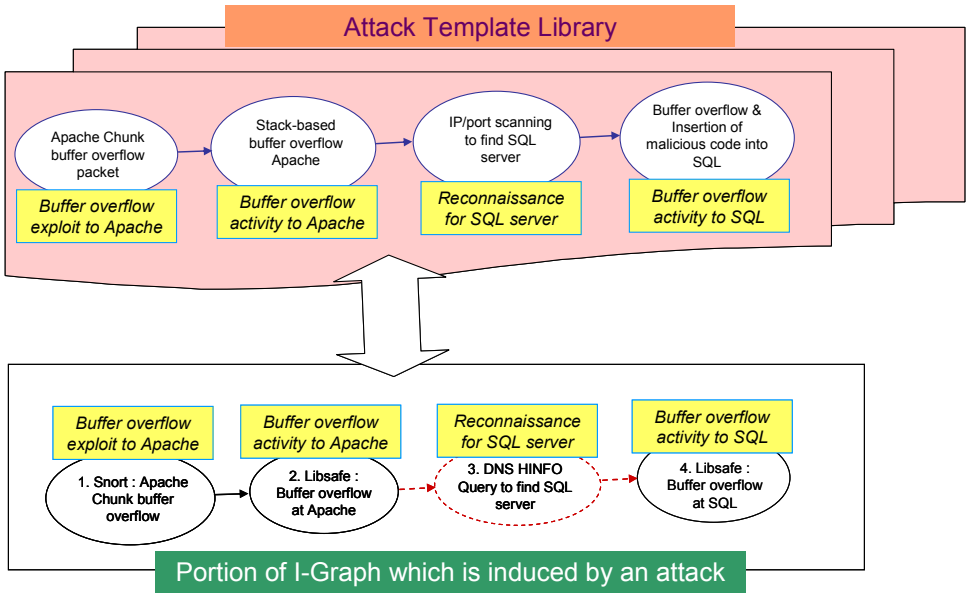


Figure 6.1. Abstraction on attack steps

An issue concerning IDS and IRS is their effectiveness on handling zero-day attacks. A classical approach in IDS application is the use of looser detection rules. By that it means the use of a detection rule which can potentially detect a whole class of attacks, and not just a single attack. For example, using a Snort rule signature pattern '/bin/sh' can potentially match against a couple of different variants of shell creation exploits, some of which can be owing to unknown zero-day attacks.

Typically, the nodes in I-GRAPH are specific to a detector alert, so if the detector rule for this alert is too strict, a slight variation in an attack could cause a missed mapping between an I-GRAPH node and an attack step, which in turn could result in a less effective response action.

To address this issue, our approach relies on the abstraction of I-GRAPH nodes. As shown in Figure 6.1, the rectangular boxes contain the abstractions on the corresponding oval-shaped I-GRAPH nodes. Therefore, what happen here is that when an exact mapping between an alert and an I-GRAPH node is impossible, ORIGIN will fall back to the abstractions and try to perform the mapping on the abstraction layers. By doing so, we hope ORIGIN can better tolerate variations in attacks and provide equally well responses to the unknown zero-day variants of an attack. For instance, in the example in Figure 6.1, there's no '*DNS-HINFO Query to find SQL server*' node, which could result in a

mismatching with the attack pattern in the template library. That attack pattern, however, resembles the attack instance almost perfectly. By using the abstractions, this mismatching can then be resolved, and the optimized responses from that attack pattern can be applied to this attack instance as well.

Table 6.1 gives a high-level summary of how different types of IRS technologies perform against different types of attacks through the life-time of attacks. Existing local response technologies take responses based on an alert flagged by a detector on the local machine and the response is hard-coded statically for each kind of alert. This is the most prevalent form of IRS in use today, such as, anti-virus software quarantining a file when the detector matches a virus signature in the file. Existing multi-stage capable IRSs such as SWIFT perform significantly better than the local response technologies for known attacks. This is due to the fact that they consider the global effect from a multi-stage attack and from the use of responses. However, for zero-day attacks, they are no better than the local response technologies, since they do not find the attack in the pre-built knowledge structure, such as the I-GRAPH used in SWIFT.

Table 6.1. Capability of ORIGIN/ADEPTS III for different kinds of attacks

	First Attack Instance	Repeated Attack Instances	
		IRS Learning Phase	IRS Fully Adapted
Known Attack	Good / Excellent	Excellent	Excellent
Zero-day Attack with **known concept**	Good / Excellent	Good / Excellent	Excellent
Zero-day attack with **new concept**	Poor / Moderate	Moderate / Good	Excellent
Local responses : "Poor" for all 9 cases **ADEPTS I/II** : "Poor" for the 6 Zero-day attack cases.			

(A range of performance is shown for some cells since the exact performance depends on the parameter setting of the IRS.)

The contribution of our work is highlighted in the bold box in Table 6.1, which corresponds to the cases of zero-day attacks. We categorize zero-day attacks into two types: one with known concepts, and another with new concepts. The first kind corresponds to the case, where after conceptualization, the zero-day attack will become identical (or closely similar) to a previously seen attack. The second kind corresponds to the case, where not even a conceptually similar attack has been seen before. ORIGIN greatly improves the state-of-the-art for the first kind of zero-day attacks, while it still improves the state-of-the-art, but less significantly, for the second kind of zero-day attacks. Note also that as in previous ADEPTS systems, ORIGIN shows the capability of learning – the effectiveness of deployed responses, the escalation paths of attacks – through observing multiple instances of an attack type and therefore the performance improves with repeated instances, till it is fully adapted to the attack type.

Figure 6.2 shows the overall system design of ORIGIN, which we will describe in the next few sections.

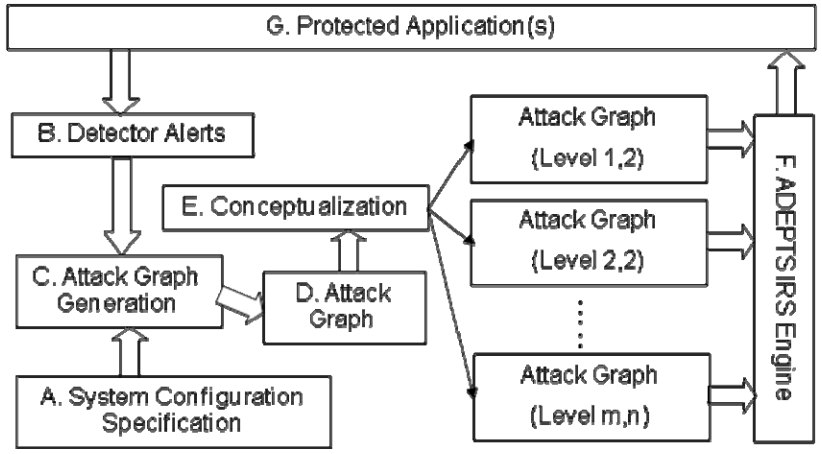


Figure 6.2. Building Blocks and Operational Flow in ADEPTS III / ORIGIN

6.2. System Configuration Specification

Rather than assuming prior knowledge of attacks by utilizing a pre-built attack graph as done in existing work [32, 40, 59], ORIGIN is designed to rely on the knowledge about the configuration of a system. Our hypothesis is that for a system owner it is easier to provide the system configuration, than to enumerate all possible attacks against her system. It is still a challenge to represent the configuration of a real world system which

can consist of multiple heterogeneous components with complex dependency relationships among them. While prior works have introduced the idea of using system configuration information for attack graph generation, they use a simple and coarse system model. For example [8, 40, 60] use each host in the system as the unit in the configuration.

In our specification, an application system is viewed as a collection of components. A component is a generic term for any resource in the system and can be an OS, a program, a file, etc. To model the interconnections between two components, we define a connection between them, which can allow the spread of attack.

We use C++ [61] as the specification language, which allows us the use of an inheritance hierarchy for components. Then we can move common specifications to a base component (base class in C++ terminology). One example is two machines running the same operating system but having different customizations (e.g. different programs installed and different settings). In this case, there can be a base component representing the original OS and two derived components representing the customized OSs (machines). The specification is compiled into a dynamic linked library (DLL), which can be loaded/unloaded from ORIGIN at runtime.

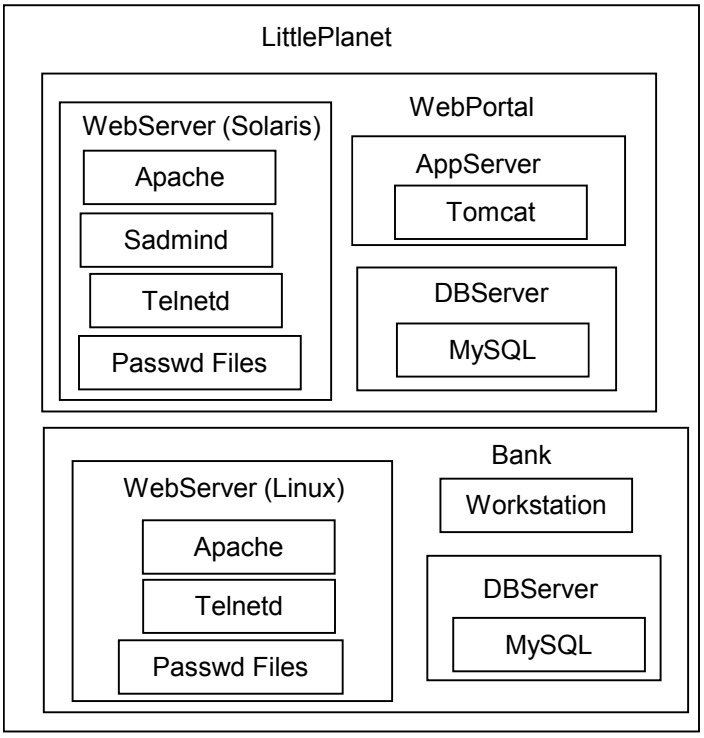


Figure 6.3. Example of Component Memberships

6.2.1 Component Definition

Table 6.2. Component Definition

```
class ComponentName : baseComponentName
/// baseComponentName is optional
{
      GetID()
      GetHost()
      EnumerateDetectors()
      EnumerateMembers()
      EnumerateOutConnections(d)
      GetIV(d)
}
```

Table 6.2 shows a component definition. We omit the detailed C++ syntax keywords/types/qualifiers for presentation simplicity. GetID returns the identifier for this component. GetHost returns the hosting component of this component. For example, a Linux component can be the hosting component of a sendmail component installed on that Linux box. EnumerateDetectors returns the detectors associated with this component. EnumerateMembers returns the member components. In ORIGIN, the membership relation between components x and y means x is hosted within y, either physically (e.g., in a machine) or logically (e.g., the sendmail component as a member of the Linux component.) Figure 6.3 shows an example of the member associations of some of the components in our testbed. In Figure 6.3, a component x within the rectangle of component y means x is a member of y. EnumerateOutConnections(d) returns the applicable outgoing connections where attack effect can propagate outward from this component given the detector alert from detector d. For example, Apache.EnumerateOutConnections(StackOverflow) may have a connection leading to a code execution on that machine while it is unlikely for Apache.EnumerateOutConnections(NetworkDoS) to have a connection leading to code execution. GetIV(d) returns the impact vector Iv for this component given an alert from detector d. This represents the damage cost to the component when the alert is triggered. We have defined Iv in Section 3.5 to be a vector with each element value $\in[0, 1]$ representing the damage to each transaction and each security goal in the system. The object-oriented

design allows ORIGIN to concisely represent components using inheritance of shared properties.

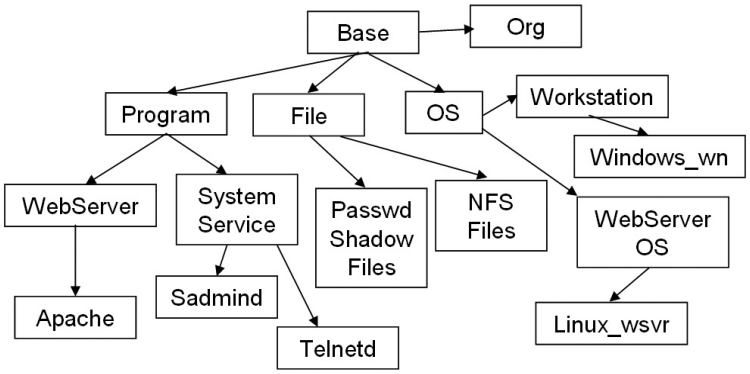


Figure 6.4. Component Inheritance Chart

Figure 6.4 shows an excerpt of the inheritance hierarchy of the components in our prototype implementation.

6.2.2 Connection Definition

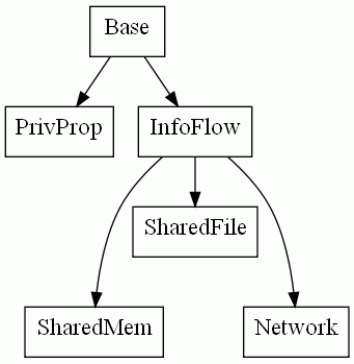


Figure 6.5. Connection Inheritance Chart

In a multi-stage attack, attack effect can propagate from one component to another following certain conditions / rules [8, 32]. To incorporate this concept into our design, we introduce the *information flow connection*. We use an *information flow connection* from component X to component Y if data can be transmitted from X to Y. Typically, this assumes a network connection from X to Y [40, 60, 62] though this can also be a shared memory, a shared file, or a procedure call between X and Y.

Less obviously, and distinct from prior work, we use another kind of connection between components – a *privilege propagation connection*. This captures spread of attacks without any explicit computing connection, but say through social channels, or by breaking into an email account and accessing privileged information to say access an online bank account. Formally, if compromising X can give an attacker the ability to carry out actions on Y, which are otherwise not possible, a privilege propagation connection is created from X to Y.

A connection also has an inheritance hierarchy (Figure 6.5) starting from a base class baseConnectionName. Its members are: source and destination components, *PF* – the propagation factor ∈ [0, 1] which gives the prior likelihood of attack propagation on this connection (refer to EPF in Section 3.7); *Responses* for the responses that can halt the attack propagation on this connection. Each response has an associated effectiveness index (EI) value (Section 3.7) indicating how effective it is believed to be when used on this connection.

6.2.3 Detector Definition

A component can have associated detectors, which are returned by the EnumerateDetectors function of a component. Each associated detector generates alerts that pertain to certain attack manifestations observed on the component.

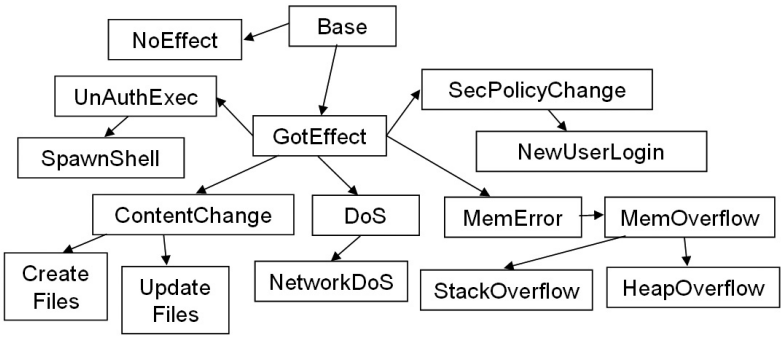


Figure 6.6. Detector Inheritance Chart

Figure 6.6 shows an excerpt of the inheritance relationship among detectors in our prototype implementation. Right below the base, there are two derived detectors : *NoEffect* and *GotEffect*. The *NoEffect* detector is used in the attack graph generation process (Section 6.3) to model the "silent" nodes on an attack path that do not generate

actual detector alerts, either because detectors are not present or the installed detectors failed to generate alerts, but can logically fit into an attack path. In the complete chart, the level beneath GotEffect has eight high level manifestations {VirusPattern, UnAuthExec, SecPolicyChange, OtherRuntimeError, MemError, DoS, ContentChange, and ConfidentialityLoss} that we find in practice have high coverage for all different attack stages, even for zero-day attacks. We show only the part of the inheritance hierarchy which applies to the attack scenarios we use later in the evaluation section.

6.3. Online Attack Graph Generation Process

In the attack graph, each node v has the following fields: the detector alert (v.detector), the component where the detector is installed (v.component), the identifier (v.id), the conceptualized detector alert after the node has gone through the conceptualization process (v.cdetector), the conceptualized component (v.ccomponent), the CPT table (v.cpt), and state of the node (v.state). A response node r has the following fields: the response command (r.cmd), the impact vector for this response (r.iv), and state of the node (r.state).

6.3.1 Attack Graph Generation

Table 6.3. Updating Attack Graph

UpdateGraph(G,d_k,NodeState)

// G: an existing attack graph which was created based on prior alerts $d_1,d_2,\ldots,d_{k-1}$.

// d_k: a detector alert

{

1. Create node v_k
2. v_k.detector := d_k
3. v_k.component := Component where d_k is generated from.
4. v_k.id = string_concate(v_k.detector, v_k.component)
5. if v_k already exists in G (as identified by v_k.id) discard v_k and return.
6. Connect node $v_s \in G$ to v_k through edge e_{Pm} : $v_s \rightarrow v_k$ for each path P: $v_s \rightarrow \{v_{sk1} \rightarrow v_{sk2} \rightarrow \ldots \rightarrow v_{skq}\}_{Pm} \rightarrow v_k$ such that the following conditions hold:

   6.1. $v_{ski} \notin G$ for i = 1..q
   6.2. q is minimized
   6.3. For each consecutive nodes pair $v_a \rightarrow v_b$ on P, there exists a connection $c \in v_a$.component.EnumerateOutConnections(v_a.detector) such that c.GetDstComponent() = v_b.component.
   6.4. v_{ski}.detector = NoEffect for i = 1..q
   6.5. All the nodes on P are distinct with respect to the nodes' identifiers.
7. For each consecutive nodes pair $v_a \rightarrow v_b$ on P
   7.1. c := the connection from v_a to v_b
   7.2. For each response R in c.EnumerateResponses()
        If !∃ response node r such that r.cmd == R {
            Create node r in G
            Set r.cmd:=R and r.iv := impact vector of R.
            r.state = NA
        }
        Connect r to v_k
8. Initialize the v_k.cpt with default values or load it from attack template library if a past attack exists.
9. v_k.state := NodeState

}

Table 6.3 shows the algorithm for updating an attack graph when receiving a new detector alert. At the beginning, G is set to be empty, so calling UpdateGraph(…) for each incoming detector alert essentially generates the resulting attack graph.

The idea behind this algorithm is to chain up detector alerts that are considered to have causal relationships. Our assumption on the casual relations between two alerts is based on connections between the underlying components where the alerts originate from. This assumption is widely used in prior work [34, 40, 62] though prior work has dealt with offline analysis of known vulnerabilities.

Table 6.4. Generating Attack Graph

```
GenAttackGraph({d1,d2,..,dk},{sd1,sd2,…,sdm})
// {d1,d2,…,dk}:detector alerts received in order
// {sd1,sd2,…,sdk}:speculative detector alerts
{
1.  G := NULL
2.  For i = 1 to k
        UpdateGraph(G,di,true)
3.  For i=1 to m
        UpdateGraph(G,sdi,NA)
4.  return G
}
```

Table 6.4 shows the algorithm which generates the attack graph where $\{d_1,d_2,\ldots,d_k\}$ are detector alerts received and $\{sd_1,sd_2,\ldots,sd_m\}$ are "speculative" detector alerts. Speculative alerts are not actually triggered yet, but are provided here so that the attack graph can grow to include them. Speculative alerts are used to populate those attack graph nodes which are likely to be but haven't been achieved by the adversary in an ongoing attack (Region [B,D] in Figure 5.5). This is required so that the IRS engine can consider a pro-active response, i.e., a response on a component that has not been affected yet, but due to the spread of the current attack. They are also useful to evaluate how effective deployed responses have been in preventing those nodes from being reached.

Similar to ADEPTS I / II, ORIGIN maintains an attack template library (ATL) of attack graphs of past attacks. Assuming an ongoing attack has triggered detector alerts $\{d_1,d_2,\ldots,d_k\}$ and there's a past attack with "triggered" detector alerts $\{d_1,d_2,\ldots,d_k,d_{k+1},\ldots,d_{k+j}\}$. ORIGIN will then use $\{d_{k+1},d_{k+2},\ldots,d_{k+j}\}$ as part of the speculative alerts for the generation of attack graph for this attack.

At Step 6.2 of the algorithm in Table 6.3 we assume attack escalates through the shortest path (minimum number of connections). Technically, one needs to consider all possible escalation paths. In practice, we find that many of these paths share some of the connections used in the shortest path, which means the responses from the connections on the shortest path suffice to break the corresponding edge in the attack graph.

6.4. Conceptualization of Attack Graph

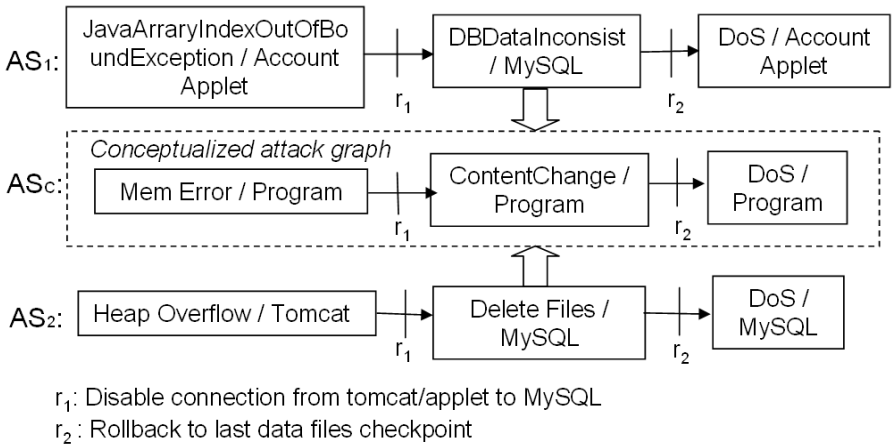


Figure 6.7. Example of conceptualization

Existing attack graph models are typically bound to specific vulnerabilities or specific detector alerts. This becomes an issue when dealing with zero-day attacks where the information about the underlying vulnerabilities is not known. Even if there are detectors that flag the manifestations from the attack, these do not map to the pre-built attack graphs available to the IRS results in either ineffective or disruptive response choices due to the lack of prior knowledge about the attack and consequently, no opportunity for the IRS to learn based on past responses. Typical disruptive responses are of the form of killing a process or shutting down a machine, responses that can be deployed with little knowledge of the specifics of an attack.

We therefore come up with a technique that allows ORIGIN to link a zero-day attack with past knowledge. The idea is to conceptualize an attack graph by generalizing a node's component ID (n.component) and detector ID (n.detector). After conceptualization, ORIGIN can potentially find from its database a past attack which, after a similar process, looks similar to the new attack graph. ORIGIN can then leverage the CPTs learned from

the past attack, to more accurately estimate the attack escalation and to provide better responses. We first provide an example of the conceptualization process before giving the algorithm.

In Figure 6.7, there are two distinct attacks to the WebPortal site. The attack in the top part of the figure (AS_1) has the attacker exploit a bug in the user account management application, which triggers a JavaArrayIndexOutOfBound exception. This bug causes a user data corruption in the back-end database, which triggers a DBDataInconsist alert from MySQL. As the user data is corrupted, a new instance of the Java based user account management application refuses to start, thereby causing a DoS to the WebPortal site. The attack in the bottom part of Figure 6.7 (AS_2) has the attacker exploit a bug in the Tomcat Java application server, which leads to a privileged access to the machine running MySQL with the side-effect of a heap buffer overflow alert. The attacker accesses the MySQL machine and deletes the data files used by the MySQL server. This causes the server to stop functioning correctly due to the missing files, again resulting in a DoS to the WebPortal site.

The two attacks AS_1 and AS_2 are radically different from the point of view of the alerts generated and the vulnerabilities exploited. When a traditional IRS such as SWIFT faces the two attacks in sequence, it will consider both of them to be two distinct fresh zero-day attacks and deploy ineffective or disruptive responses. However, looking at the center part of Figure 6.7 (AS_C), we see that AS_1 and AS_2, after being conceptualized with component level = 2 and detector level = 3, look identical. (Details on the conceptualization process is mentioned in next paragraph.) At this level of generalization, AS_1 and AS_2 share the concept which has a sequence of three steps – causing a memory error at some program, followed by changing some contents at some program, and eventually a DoS of some program. From ORIGIN's point of view, the two attacks can share the same effective response r_1 and r_2. Since it takes time for an IRS such as ADEPTS to adapt its responses, the mapping of a zero-day attack to a known concept allows it to deploy an effective response on the first instance of the attack. Thus, if AS_1 has been seen in ORIGIN, then a subsequent occurrence of AS_2 can be handled better than in existing IRS such ADEPTS I or SWIFT.

Table 6.5. Update Node IDs

```
UpdateNodeIDs(G)
////   G(N,E): An existing attack graph
{
1.   Sort nodes in G into topological order {n1,n2,…,nN}

2.   for i = 1..N   {
     2.1.   m := number of parent nodes of ni
     2.2.   Sort parent nodes of ni into {p1,p2,..,pm} by the parent nodes IDs
     2.3.   count := 0
     2.4.   do {
                 ni.id := Hash( {p1.id, p2.id,..,pm.id} , {ni.ccomponent, ni.cdetector}, count )
                 count := count + 1
            }while (ni.id is already used by a node in G)
     }
}
```

Table 6.6. Conceptualization of Attack Graph

```
Conceptualize(G, ComponentConceptLevel, DetectorConceptLevel)
// G: an attack graph to be conceptualized
// ComponentConceptLevel: concept level for component
// DetectorConceptLevel: concept level for detector
{
1.   For each attack step node v in G do
     1.1.   ConceptualizeNode(v, ComponentConceptLevel, DetectorConceptLevel)

2.   UpdateNodeIDs(G)
}
```

Table 6.7. Conceptualization of Attack Graph Node

```
ConceptualizeNode(v, ComponentConceptLevel, DetectorConceptLevel)
// v: attack graph node to be conceptualized
// ComponentConceptLevel: concept level for component
// DetectorConceptLevel: concept level for detector
{
1.  v.ccomponent = v.component
2.  v.cdetector := v.detector

3.  depth_c := depth of v.compoent in the component inheritance chart (Figure 6.4)
    from the root "Base".
4.  depth_d := depth of v.detector in the detector inheritance chart (Figure 6.6) from the
    root "Base".

5.  while(depth_c > ComponentConceptLevel) {
      5.1.  v.ccomponent := base_component(v.ccomponent)
      5.2.  depth_c := depth_c – 1
    }

6.  while(depth_d > DetectorConceptLevel) {
      6.1.  v.cdetector := base_detector(v.cdetector)
      6.2.  depth_d := depth_d - 1
    }
}
```

Table 6.6 shows the algorithm for conceptualizing an attack graph. The parameters ComponentConceptLevel and DetectorConceptLevel determines the degree of conceptualization. The word "conceptualization" means moving the component ID or the detector ID of an attack graph node upward toward the respective base of the inheritance hierarchy (Figure 6.4 and Figure 6.6). ComponentConceptLevel specifies the target level a component should be moved to (base is at level 1). Similarly, DetectorConceptLevel specifies the target level a detector should be moved to. The algorithm in Table 6.7 is invoked by the algorithm in Table 6.6 to carry out the actual conceptualization process for each node. The UpdateNodeIDs(G) (Table 6.5) used in Table 6.6 generates the identifier for each attack step node in the attack graph. A node's ID is the hash value of its parent nodes' IDs and its conceptualized component and detector fields. This design ensures a node's ID to be representative of the graph topology

leading to it. Conceptually, two nodes from two separate attacks bear the same IDs, then ORIGIN knows they have the same preceding attack steps. This makes identifying similarities between attacks easier.

6.5. Limitations

The conceptualization level (DetectorConceptLevel and ComponentConceptLevel in Sec. 6.4) is considered an input value to the system. The system administrator needs to set the value at the start of the system. Future research should look at how to determine the appropriate conceptualization level automatically.

The proposed approach to deal with zero-day attack is built on the assumption that two similar attacks will share similar attack escalation and response effectiveness. This requires an exhaustive study of a database of multi-stage attacks. This kind of database is not available in the public domain, and the assumption is still an open problem.

7. IMPLEMENTATION OF ADEPTS AND TESTBED

7.1. Description of e-Commerce Application

Figure 7.1 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 7.1.

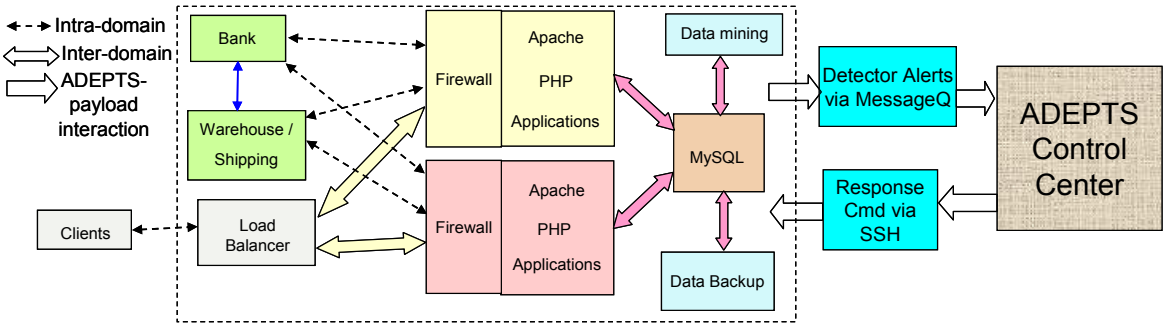


Figure 7.1. Layout of e-Commerce testbed

We set certain security goals for the system, the complement of which are specified in Table 7.2, along with the weights. Thus adding the word "prevent" before each gives the goal. The attached weights to the transactions and security goals are used for survivability computation in Section 8.3. 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).

Table 7.1. 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
Admin work	Variable	10

Table 7.2. List of e-Commerce security goals

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

7.2. 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 Section 8.1, 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.

7.3. 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. Each step of the attack scenario may be detected by none, one, or more of the detectors. We show in Table 7.3 ~ Table 7.6 and Figure 7.2 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 attack scenarios shown in Table 7.3 ~ Table 7.6. 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.

Table 7.3. Attack Scenario 0

Scenario 0
1. Exploit Apache mod buffer overflow.
2. Insert malicious code.
3. IP/port scanning to find vulnerable SQL server.
4. Buffer overflow MYSQL to create a shell (/bin/sh).
5. Use malicious shell to steal information stored in MySQL.

Table 7.4. Attack Scenario 1

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

Table 7.5. Attack Scenario 4

Scenario 4
1. DDoS attack via issuing huge amount of legal transactions (i.e. product search)

Table 7.6. Attack Scenario 8

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

We also test ADEPTS with other attack scenarios (e.g. Figure 8.13, Figure 8.18, Figure 8.19, Figure 8.20, and etc.) 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 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 Figure 2.

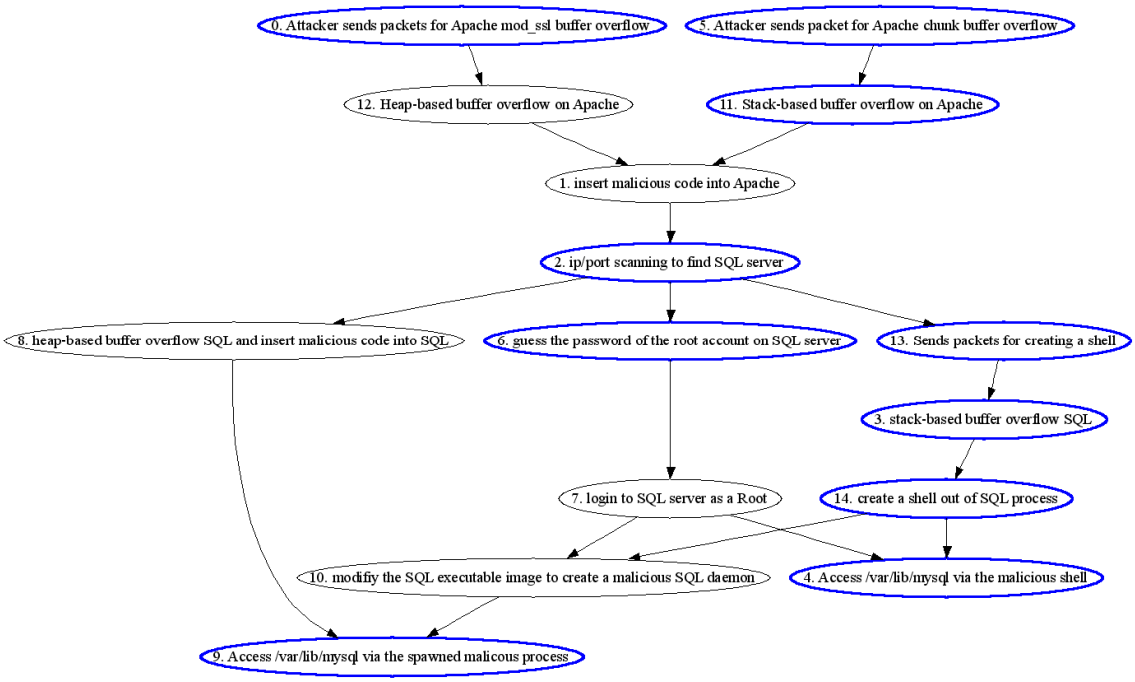


Figure 7.2. Example of a dynamic attack scenario (Attack Scenario 9)

[Thicker blue circles denote an associated detector]

7.4. 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. 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.

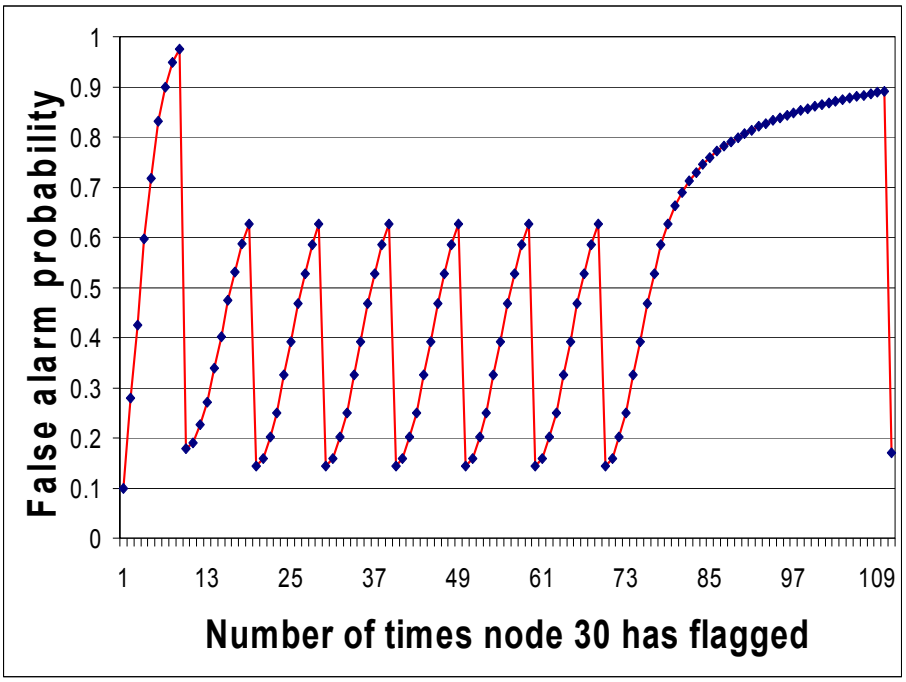
8. EXPERIMENTS AND RESULTS

We perform experiments on the e-Commerce testbed using both synthetic and real-world attack scenarios. 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, (v) Comparing global optimal response determination (ADEPTS II / SWIFT) with heuristic-based response determination (ADEPTS I), (vi) Effectiveness of (ADEPTS III / ORIGIN) to deal with zero-day attacks.

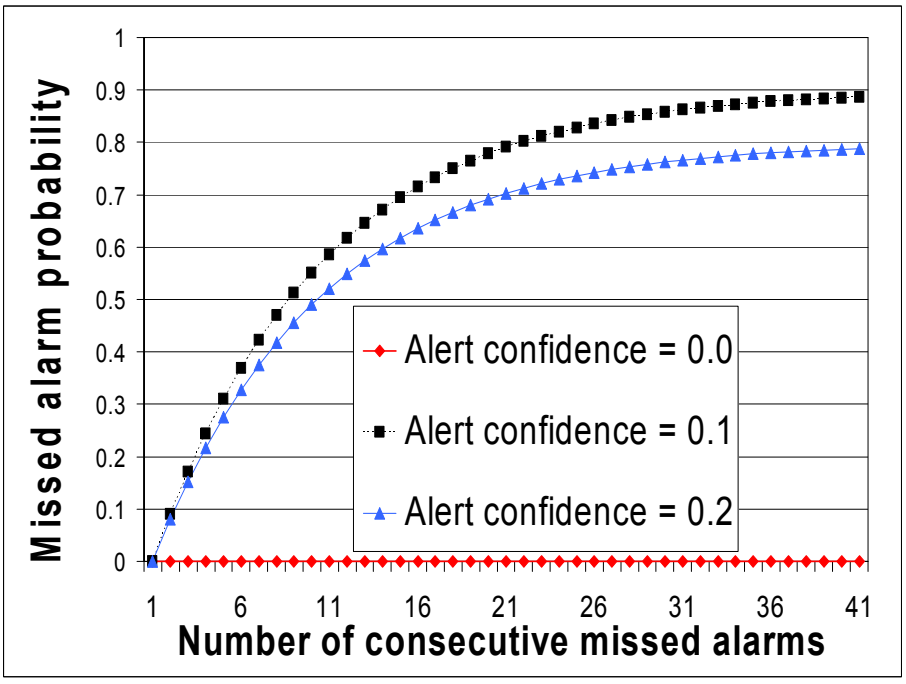
Due to the constraints of space, the results for a sample number of attack scenarios are shown. Comparing ADEPTS to other dedicated IRSs is difficult since they are not publicly available. For the experiments, survivability is defined as 1000 - |Iv| or more specifically 1000 – Σ unavailable transactions – 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 (e.g. Table 7.1 and Table 7.2). Transactions become unavailable due to responses, such as rebooting a host, or attacks. Security goals may be violated due to the successful execution of an attack step. After a run of an attack scenario is completed, any drop in survivability (or increase in |Iv|) due to non-permanent security goal violations (e.g. running a malicious process only to reach another goal) is reversed.

8.1. Missed Alarm and False Alarm Estimation

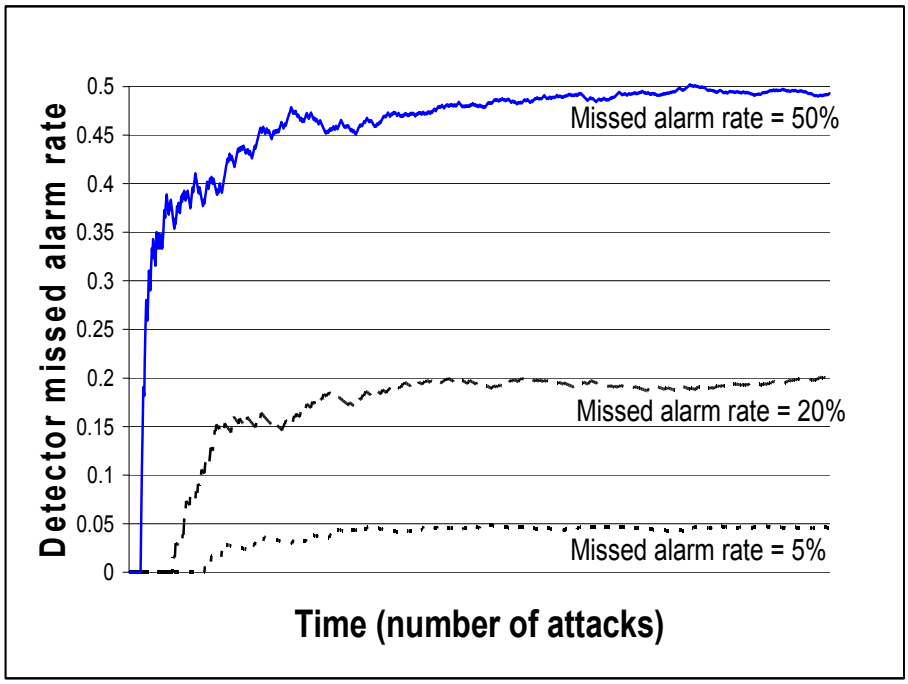
The objective of this experiment is to demonstrate the behavior of the false alarm and missed alarm algorithms given in Section 3.7. The scaling parameter γ_q and γ_p are set at 0.2.


False alarm probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
13
25
37
49
61
73
85
97
109
Number of times node 30 has flagged

(a)


Missed alarm probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
6
11
16
21
26
31
36
41
Number of consecutive missed alarms
Alert confidence = 0.0
Alert confidence = 0.1
Alert confidence = 0.2

(b)



(c)

Figure 8.1. Behavior of false and missed alarm computation algorithms

Figure 8.1 (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.

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.

Figure 8.1 (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.

Figure 8.1 (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.

8.2. 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 (attack snapshot) and the state regarding the effectiveness of the responses is maintained in this pattern.

Four instances of attack scenario 9 shown in Figure 7.2 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.

Table 8.1. 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 8.2. 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[6]	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

In Table 8.3, we show the different responses taken after each instance of the attack. The second column gives 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 Figure 8.2. 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.

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

Table 8.3. Response adaptation for attack scenario 9

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

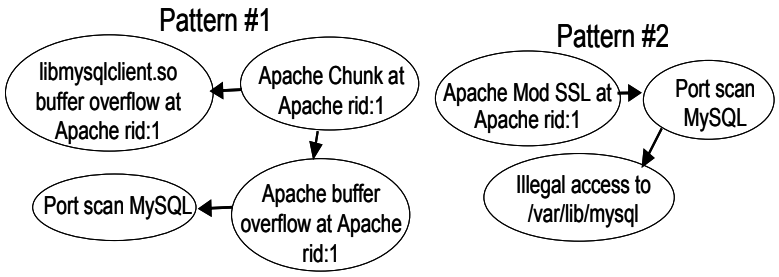


Figure 8.2. Raw Patterns #1 and #2 after instance 1 and 2 of attack scenario 9

Table 8.4. Responses associated with the static attack pattern in Figure 8.3.

[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 minutes before deploying next response for this node. – Re-enable access to "/var/lib/mysql" on IP_M	– Shutdown IP_A – Shutdown IP_M

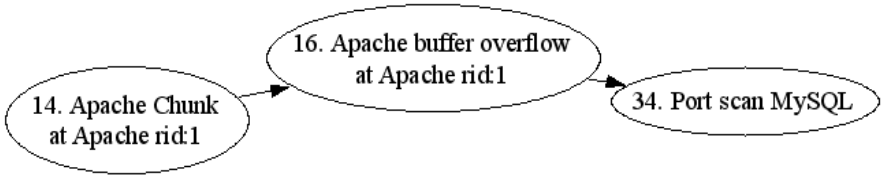


Figure 8.3. Static attack pattern with optimized responses for experiment 2

As mentioned earlier in Section 4.5, 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 Figure 8.3. The associated responses are shown in Table 8.4. We use a preset threshold 0.5 for the threshold parameter in the algorithm outlined in Table 4.2. 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 8.3. 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

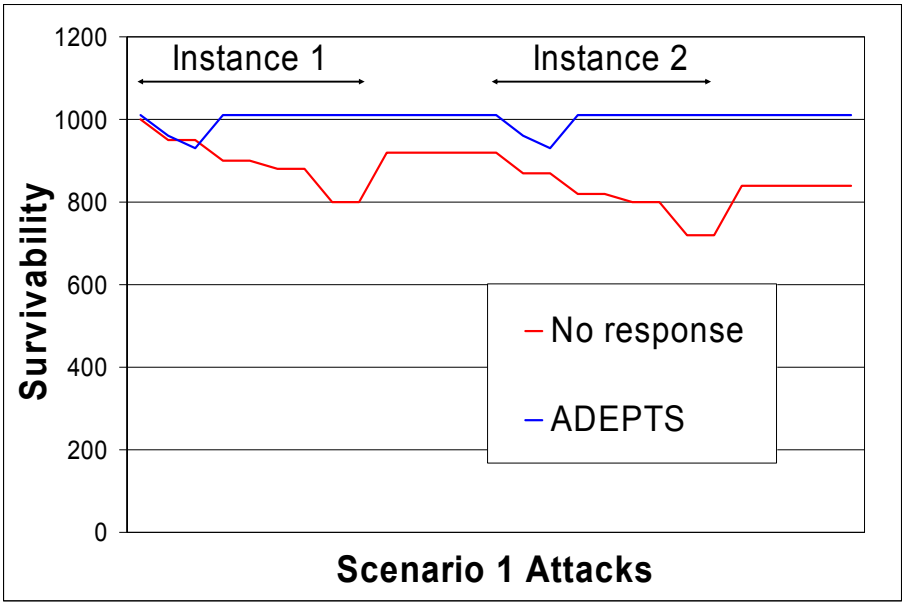
successful response “Reboot Apache’s host machine” is suppressed for demonstration. The result in Table 8.5 (column 2) shows that the attack moves one step further and the responses stored on the node [34. Port scan MySQL] are deployed in which both the Apache server and MySQL server are both shut down.

Table 8.5. Response selection with matching against static attack pattern

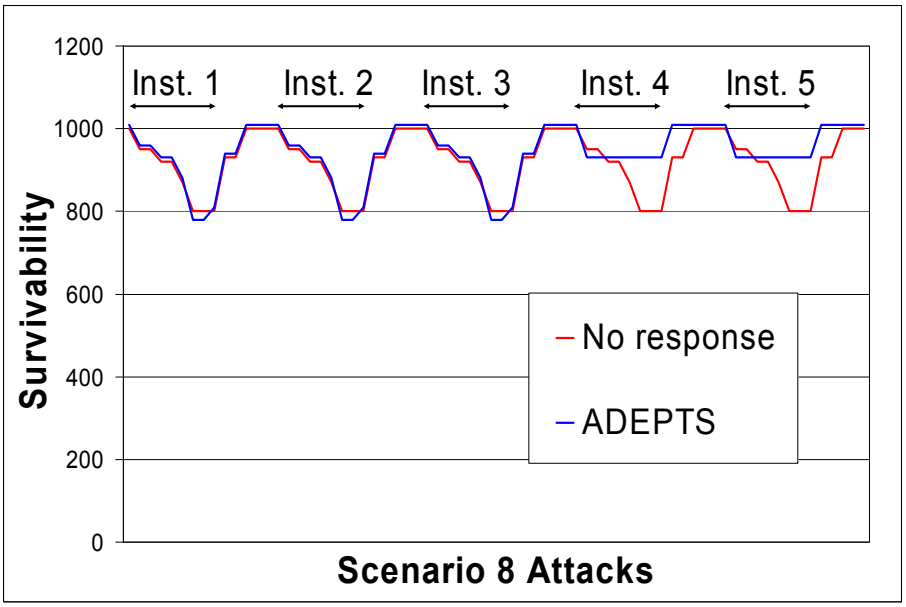
Instance of Attack Scenario	**Matching score, Step at which match is successful**	**Step : Responses taken (S|F)**	**Steps executed before AS is stopped**
I1	0.64, S11	S11 : R1(S), R2(S), R3(S), R4(S) … after 15 minutes R5(S)	S5 => S11 =>S1
I2 (Response “Reboot Apache machine” is suppressed)	0.64, S11 0.72, S2	S11 : R1(S), R2(S), R3(F) S2 : R6(S), R7(S)	S5 => S11 => S1 => S2

8.3. Survivability Improvement from Automated Response (ADEPTS)

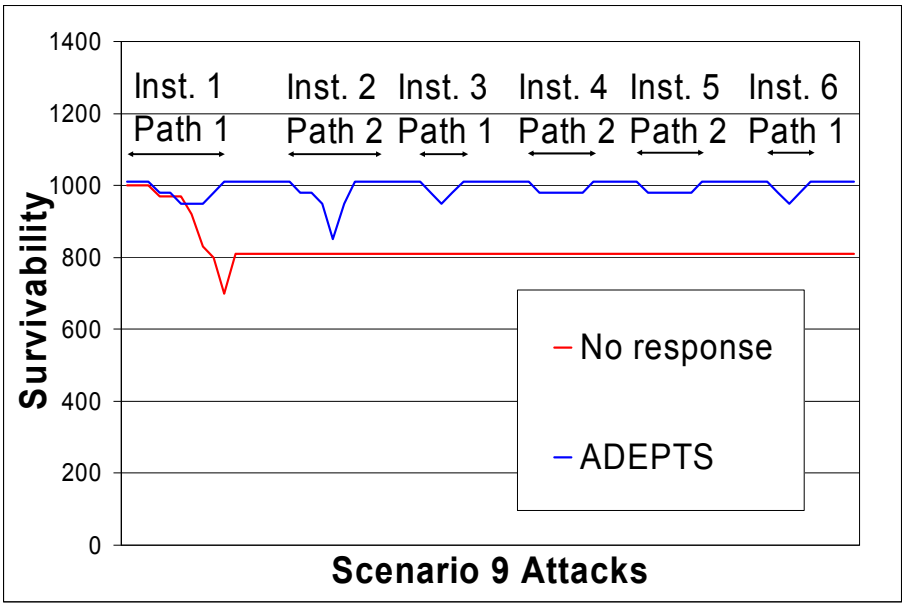
In this experiment, the objective is to show how the survivability of the e-Commerce system is affected by repeatedly stressing the system through the injection of successive instances of a given attack scenario. The results of running three scenarios are shown, where static scenarios 1 and 8 cover two 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 fixed as 1000 while with ADEPTS it is 1010, so as to provide clarity in the graphs through non overlapping plot lines. 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.


1200
1000
800
600
400
200
0
Survivability
Instance 1
Instance 2
No response
ADEPTS
Scenario 1 Attacks

(a)


1200
1000
800
600
400
200
0
Survivability
Inst. 1
Inst. 2
Inst. 3
Inst. 4
Inst. 5
No response
ADEPTS
Scenario 8 Attacks

(b)



(c)

Figure 8.4. Effect of attack scenarios on survivability

(x-axis corresponds to logical time)

In Figure 8.4(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 8.6 shows the cause of the survivability drop during an instance of scenario 1. In Figure 8.4(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 rebooting 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 Figure 8.4(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.

Table 8.6. Cause of survivability drop with and without ADEPTS 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]	

8.4. Survivability Improvement : ADEPTS v.s BASELINE Local Response

This experiment shows the comparative performance of ADEPTS in maintaining the survivability of the e-commerce system with respect to having no responses and only BASELINE local responses. Two different attack scenarios are executed and the survivability calculated at each step of the attack scenario. For the local response case, the responses that came with the deployed detectors are used – Snort (IP blocking) and bank monitor (freeze credit card).

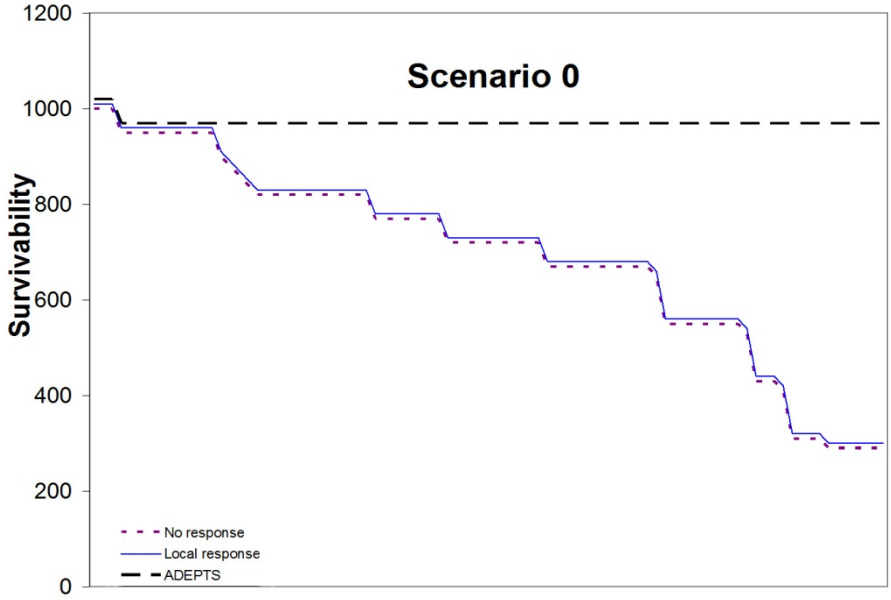


Figure 8.5. Survivability vs. Attack Steps from Attack Scenario 0

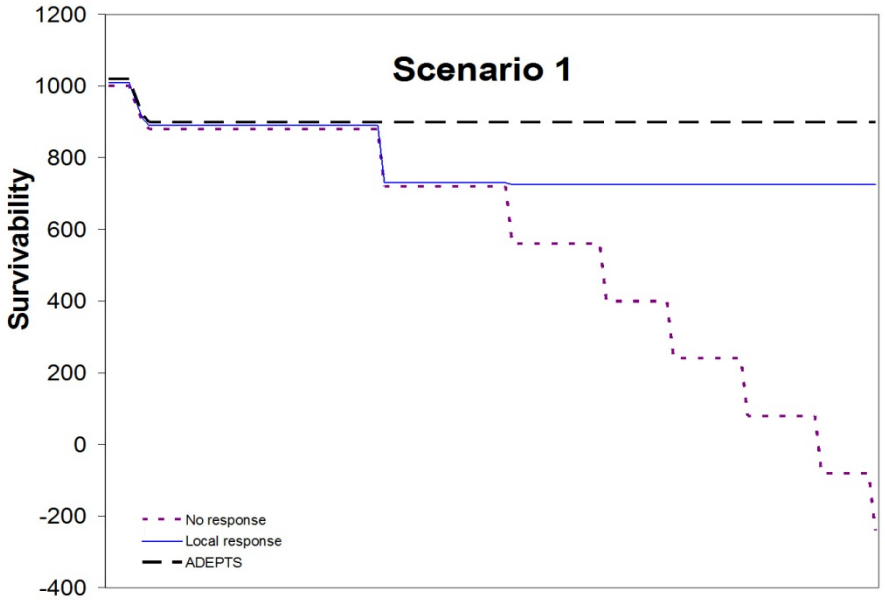


Figure 8.6. Survivability vs. Attack Steps from Attack Scenario 1

For Attack Scenario 0 (Figure 8.5), ADEPTS far outperforms the other two. The File Access Monitor detects a malicious shell being created with Apache privileges while Snort detects an Apache SSL module buffer overflow packet. Consequently, ADEPTS deploys aggressive responses to kill the process and block all following incoming packets from the attacker. The inability of the BASELINE local response implemented by Snort to

drop the IP packets in time causes the attack to continue to spread. For Attack Scenario 1 (Figure 8.6), the performance of the local response is noticeably worse than ADEPTS. ADEPTS deploys a successful response of disallowing shell commands with Apache privileges, earlier than the local response at the bank monitor.

8.5. Scalability of ADEPTS

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 an early prototype of ADEPTS (referred to as version 0 in Section 4.2) that lacks this ability and therefore operates on the entire I-GRAPH.

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.

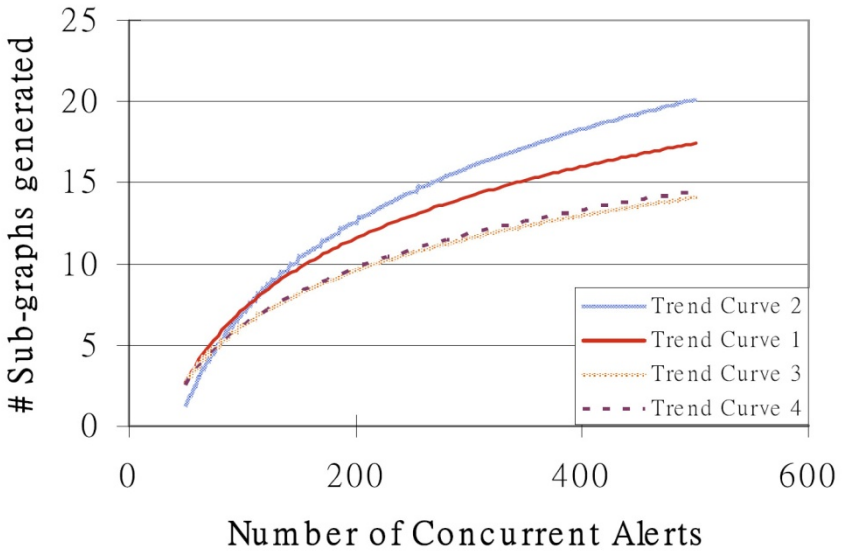


Figure 8.7. Degree of parallelization in ADEPTS I

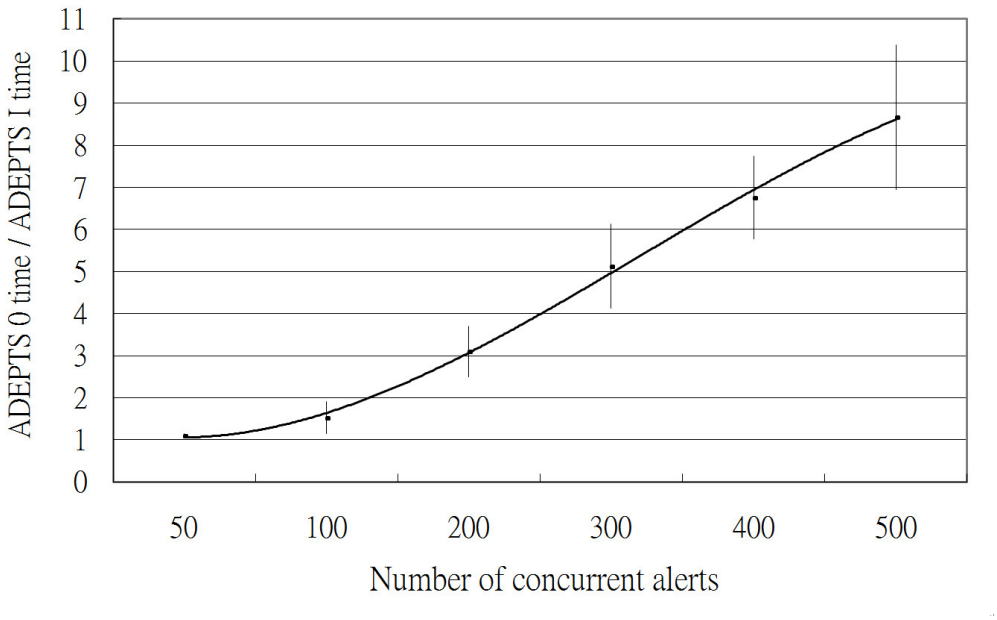


Figure 8.8. Speed up in ADEPTS I with increasing number of concurrent alerts

From Figure 8.8, we see that ADEPTS I gives considerable speedup over ADEPTS 0, with an increasing trend as the number of concurrent alerts increases. Looking at the absolute values of times in ADEPTS I, 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 Figure 8.7 (only results from 4 out of the total 8 I-GRAPHS are shown for presentation clarity). However, the growth is sub-linear and therefore the relative speedup between ADEPTS I and ADEPTS 0 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. Figure 8.8 gives the average speed up of all the 8 cases between ADEPTS I and ADEPTS 0. The vertical bar shows 2 standard deviations. With a small number of alerts (say, 50), ADEPTS I performs only slightly better than ADEPTS 0. This is due to the inherently higher constant overhead of ADEPTS I nullifies the performance gain from limited parallelization of 3-6 generated sub-graphs. However, from around 100 alerts, ADEPTS I starts to significantly out-perform ADEPTS 0. 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.

8.6. Survivability for Micro-Benchmark (SWIFT v.s ADEPTS I)

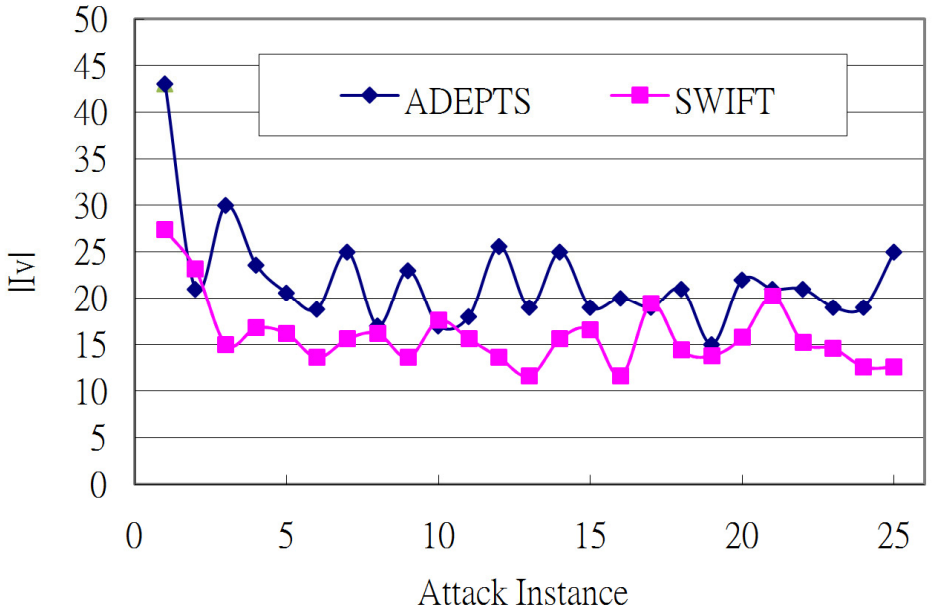


Figure 8.9. Improvement in lowering |Iv| with SWIFT for Micro-benchmark

We consider as a micro-benchmark an attack scenario that has the form shown in Table 8.7. This is a regular structure with each node representing a unique service being affected. The multi-stage attack starts at svc0 and proceeds through all the four possible paths with the goal of achieving svc21. There are 'single-node' responses on each node which if successful has the effect of preventing the node and its children nodes from being achieved. The other responses are 'dual-node' responses, which can contain the attack on two nodes at a time. In general, a dual-node response has lower cost than the total cost from two counterpart single-node responses but has higher cost than an individual single-node response. Still, one has to consider the overall effectiveness and the overlapping cost from other responses. This is one of the key strength of SWIFT in judging the whole situation and seeking for the global optimal response combination. The attack scenario is injected individually into SWIFT and ADEPTS I at the root node and is executed multiple times. The initial EI values for all responses are taken to be 1, a consciously chosen overly optimistic decision to investigate how the system *unlearns* it.

The survivability result from the experiment is shown in Figure 8.9. Overall, SWIFT chooses responses which yield lower |Iv| than those from ADEPTS. This clearly shows the advantage from considering responses in a system-wide global manner in SWIFT (Eq. (3.2)). This is true even for the first attack instance where no history information is available as shown in Figure 8.9. With the history built up over each attack instance, we can see the decreasing of |Iv| from both cases due to the adaption processes employed. Over the 25 attack instances, SWIFT yields an averaged |Iv| of 15.9 while ADEPTS yields an averaged 21.9, a 27% improvement.

Table 8.7. Detailed attack snapshots from attack instance 24

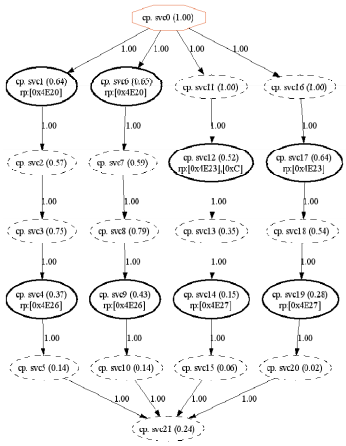
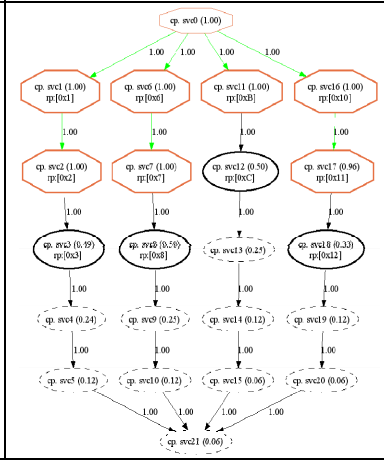
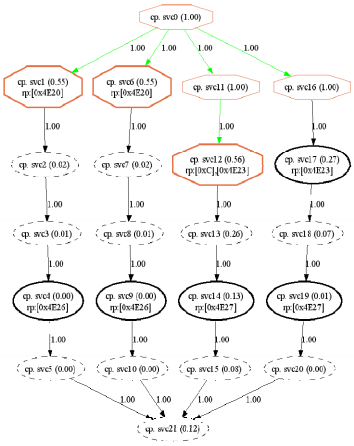
Attack Instance 24		
(a) SWIFT **After attack snapshot 1**	**(b) ADEPTS I (|Iv|=19)** **At the end of the attack**	**(C) SWIFT (|Iv|=12.6)** **At the end of the attack**
		

Table 8.7 shows the selected attack snapshots at different time points for SWIFT and ADEPTS for attack instance 24. Octagonal node means adversary has achieved the node, elliptical means it has not; solid node means response has been deployed. In (a), we see the response of SWIFT after only the first attack snapshot has been observed. SWIFT has already deployed proactive responses, as far ahead as the fourth stage of the attack. Having seen 23 previous attack instances for this specific attack, SWIFT has deduced that responses in the fourth stage (at nodes svc4, svc9, svc14, svc19) have to be deployed early enough to be successful. (b) and (c) show the cases at the end of the attack for ADEPTS I and SWIFT respectively. ADEPTS I selects locally optimal responses and therefore prefers the single-node responses, deploying a total of 11 responses and effectively preventing the end goal of the adversary from being achieved. However, SWIFT due to the property of searching for globally optimal responses, selects 4 dual-

node responses (ID: 0x4E20, 0x4E23, 0x4E26, 0x4E27) and 1 single-node response (ID: 0xC), again preventing the end goal from being achieved, but at a lower cost.

8.7. SWIFT : Learning from History to Reduce Search Space Size

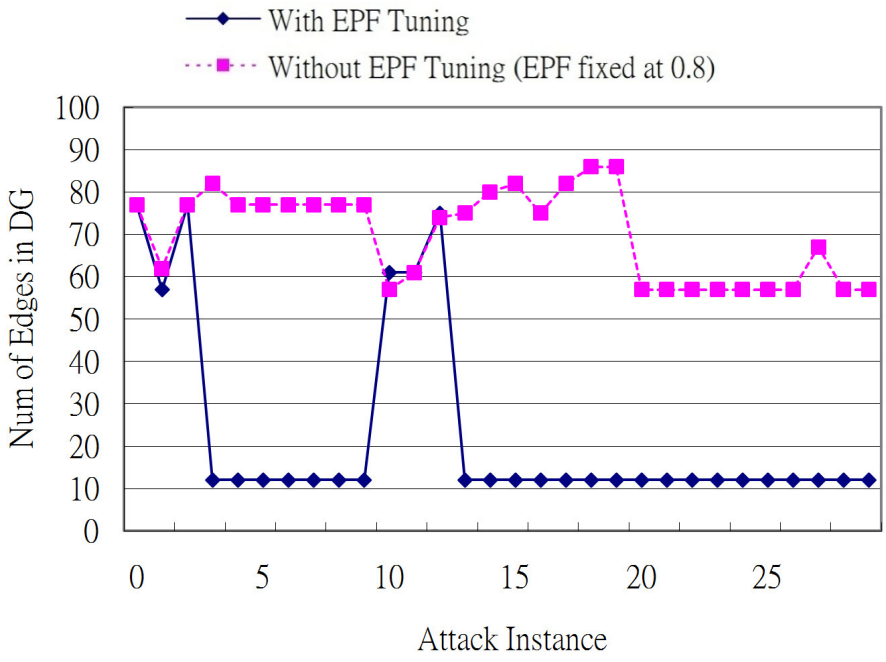


Figure 8.10. # of edges in the domain graph generated out of the 3rd snapshot

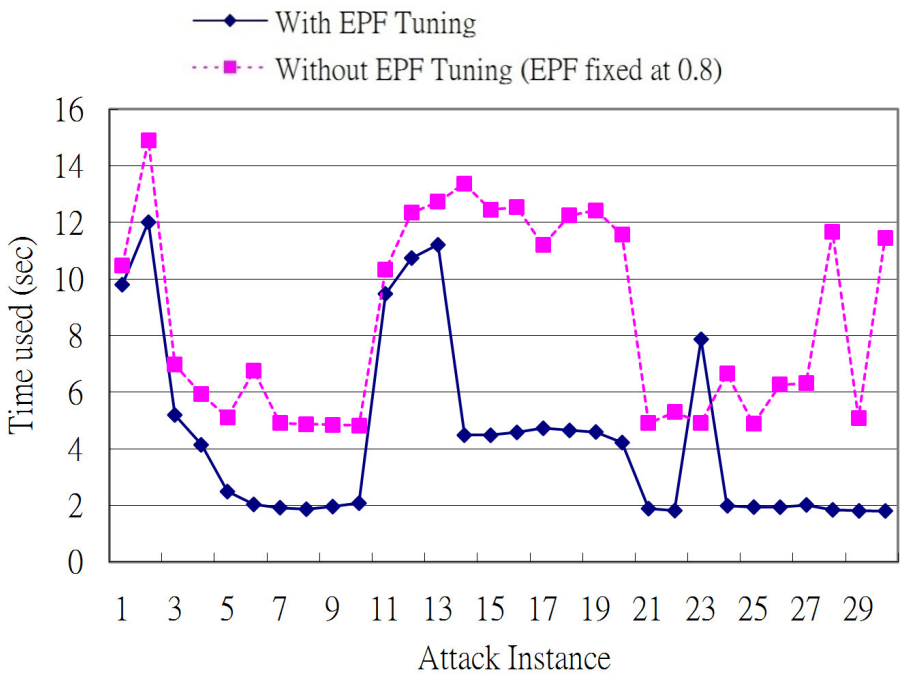


Figure 8.11. Time used by SWIFT in response decision

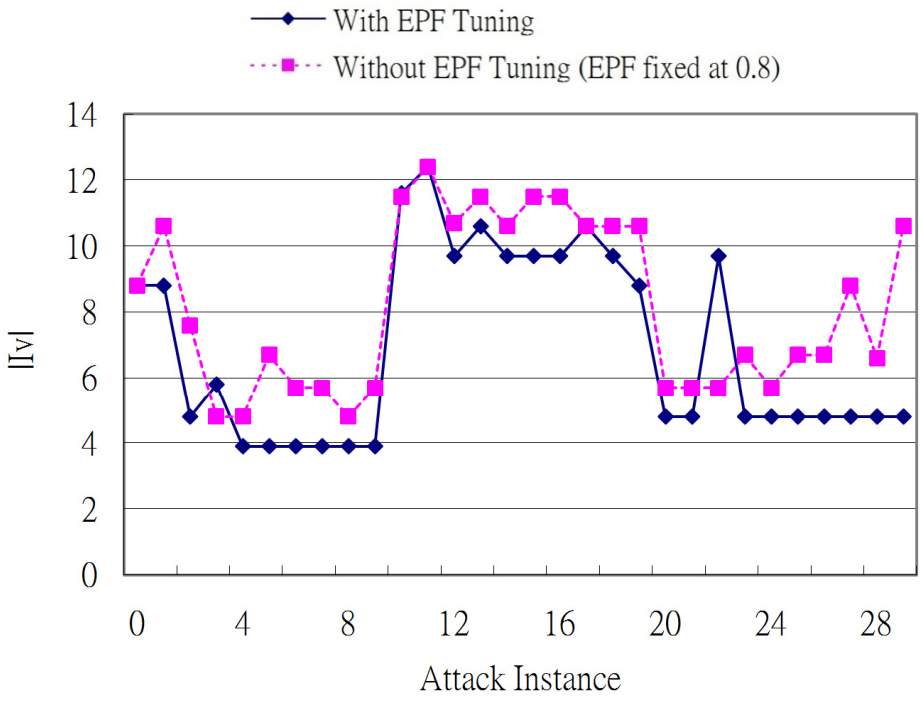


Figure 8.12. |Iv| v.s Attack Instance

This experiment shows the effect of EPF tuning on reducing the size of the domain graph for an attack scenario as SWIFT gets adapted to the attack steps (Section 5.5.). Here we assume a system with an I-GRAPH containing 42 nodes and 103 edges. We use two attack scenarios EPFAS.1, which can be potentially deterred with deployed responses, and EPSAS.2, which doesn't have any applicable responses available on its attack paths and can't be deterred. 30 attack instances are injected into the system. Attack instances 0-9 follow attack scenario EPFAS.1, 10-19 follow EPFAS.2, and 20-29 revert to EPFAS.1. Here we discuss the results on the 3rd attack snapshot from a few representative attack instances. (In the last few attack instances, when SWIFT fully adapts itself to the attack, the attack is only able to populate three attack snapshots before being effectively stopped by SWIFT. Therefore, for presentation consistency, we use the 3rd attack snapshot even though in the first few attack instances, there do have more than three attack snapshots available.)

As we can see EPF Tuning not only reduces the size of the domain graph, which speeds up the execution time of SWIFT, but also improves the quality of the generated response solutions i.e., reduces the overall system $|Iv|$. This happens since SWIFT searches through follow-on attack steps which are more likely and avoids deploying responses on nodes that are unlikely. From Figure 8.10, we can see a clear decreasing trend in the size of the domain graph from 77 edges to 12 edges for the first 10 attack instances with EPF tuning. On the other hand, the number of edges without EPF tuning is significantly higher. The fluctuation of the number of edges without EPF tuning is due to the different responses deployed prior to the 3rd attack snapshot for each different attack instance.

From Figure 8.12 we can see that for attack instances 10-19, all the responses are totally ineffective, which translates into the higher $|Iv|$ values. From Figure 8.10, we see the sudden increase in the size of the domain graph at instance 10 as the unseen attack scenario EPFAS.2 emerges. With EPF tuning, SWIFT adapts itself quickly and the size drops to 12 edges per domain graph starting from attack instance 13 again. When the system is injected with EPFAS.1 again (instances 20-29), we observe that SWIFT is able to use its memory of EPFAS.1—the domain graph is small and the $|Iv|$ does not shoot up. The spike in $|Iv|$ at attack instance 22 is due to the probabilistic nature of the occasional failure of the response on [svcs3].

Overall, we conclude that reducing the size of a domain graph through EPF tuning not only improves the efficiency in response searching but also improves the quality of the resulting responses.

8.8. SWIFT : Survivability for Real Attack Scenarios

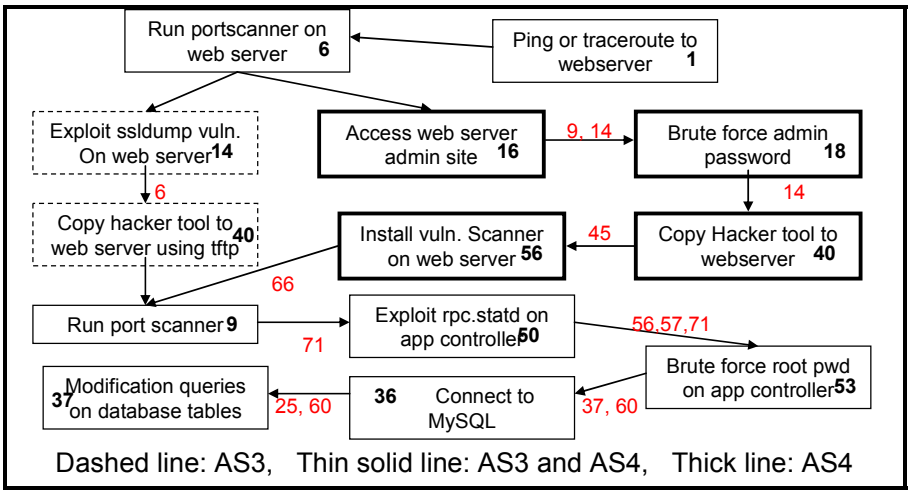


Figure 8.13. Attack scenarios 3 and 4 (AS3, AS4)

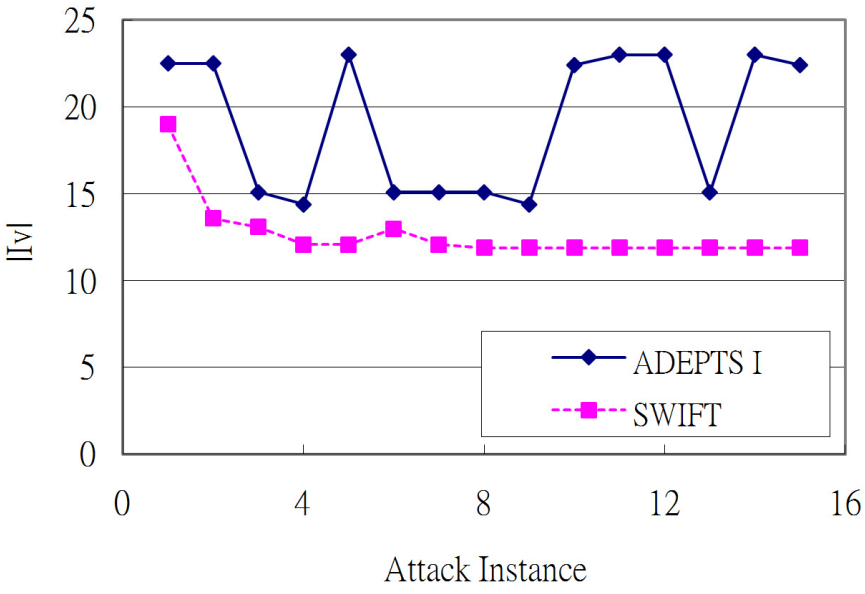


Figure 8.14. |Iv| for AS3

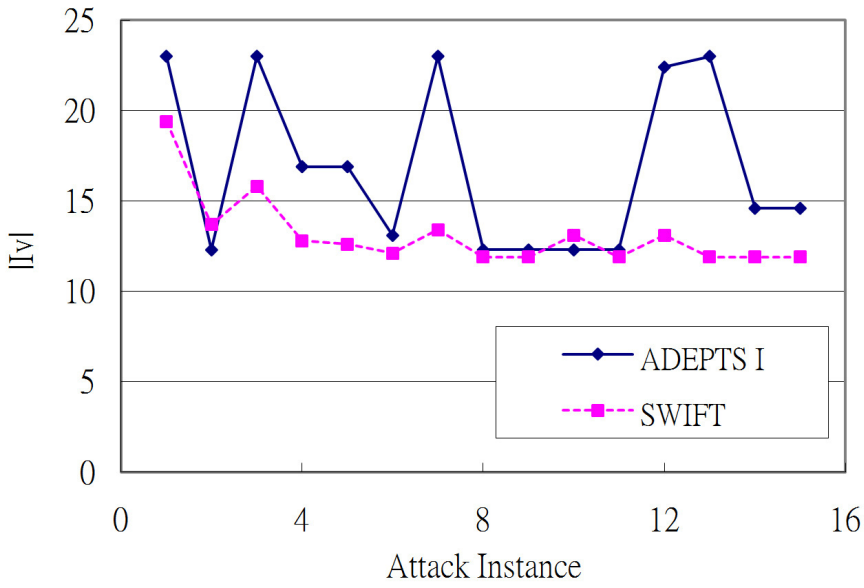


Figure 8.15. | Iv | for AS4

Figure 8.13 shows the two attack scenarios AS3 and AS4 used in this experiment. These are real in so far as they are created from the publicly available vulnerability and attack databases by chaining individual attack steps. The numbers on the edges correspond to the response IDs which can prevent propagation of the attack. Some responses (R_9, R_{25}, R_{56}, R_{57}, and R_{66}) require longer lag time for effective deployment. They are useful for SWIFT due to its ability to deploy them proactively, but useless for ADEPTS I, which considers only local optimal responses. Besides, we have initial EI value for R_{60} set erroneously low and those of the other responses set overly high. The goal is to see if SWIFT can recover from this situation. The end node N_{37} is a critical node with a high $|Iv|$. We inject 15 instances each of AS3 and AS4 and compare the achieved survivability at the end of each attack instance for ADEPTS I and SWIFT. Figure 8.14 shows that ADEPTS I's performance is widely fluctuating for AS3. This is primarily due to the fact that ADEPTS I considers responses close to the nodes that have been achieved. For example, R_{71} has about 50% probability of success in deterring the propagation from node N_{50} to N_{53} when it is deployed by ADEPTS I at the time when N_9 is flagged. SWIFT consistently has lower $|Iv|$ than ADEPTS I. This is due primarily to SWIFT's ability to redeem R_{60} through the fuzzy EI mechanism (Section 5.4) even though it had a low initial value. In ADEPTS I R_{60} is not considered till the EIs of the other responses also diminish to this low value. For AS4 (Figure 8.15), while the general pattern is similar to that of AS3, the difference in the $|Iv|$ is negligible for some instances. This is due to the fact that

there are more available responses in AS4, and therefore ADEPTS I does not suffer as much from underestimated response R_{60}.

8.9. SWIFT : Responding to Attack Variants

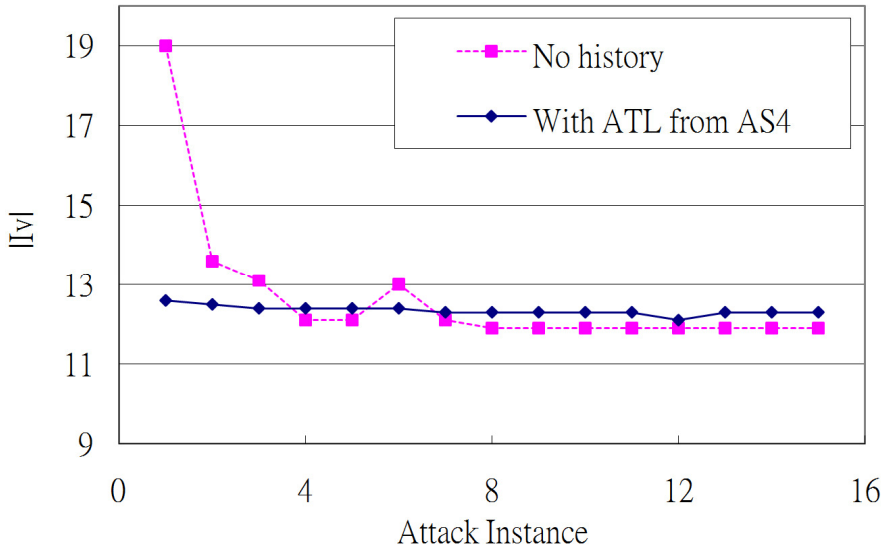


Figure 8.16. |Iv| with SWIFT leveraging history from an attack variant (AS3)

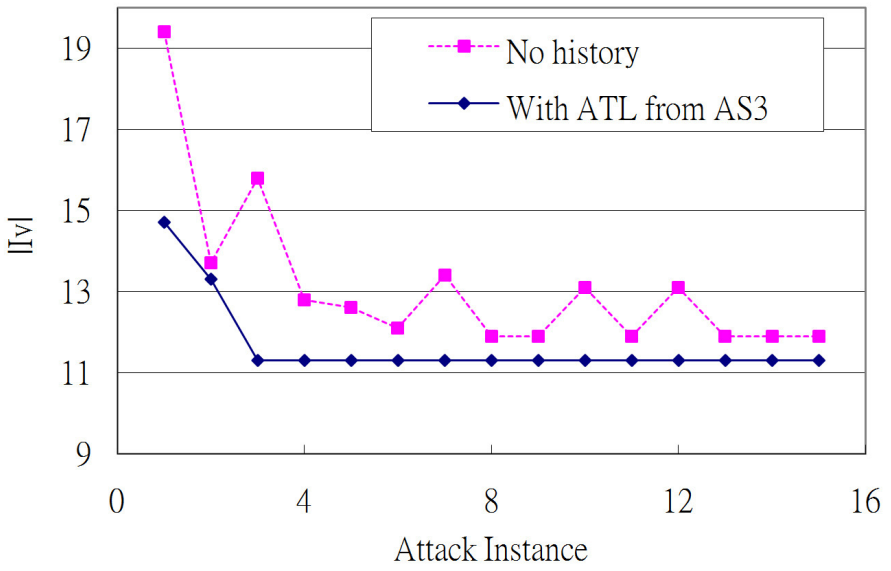


Figure 8.17. |Iv| with SWIFT leveraging history from an attack variant (AS4)

In this experiment we consider AS3 and AS4 to be variants of each other (due to their shared nodes as shown in Figure 8.13). The results are shown in Figure 8.16 and Figure 8.17. In the first sub-experiment, we execute AS4 15 times and use its snapshot from the ATL (which includes the optimized responses that SWIFT had determined) in responding to AS3. In the second sub-experiment, we reverse the roles of AS3 and AS4. The key difference between using history and not using it expectedly lies in the first attack instance. In both AS3 and AS4, SWIFT is able to use the historical information from the variant and limit the damage to the system from the first attack instance compared to the case without history. This would be valuable in dealing with very destructive attacks when they are observed for the first time.

8.10. ORIGIN : Responding to Zero-Day Attacks

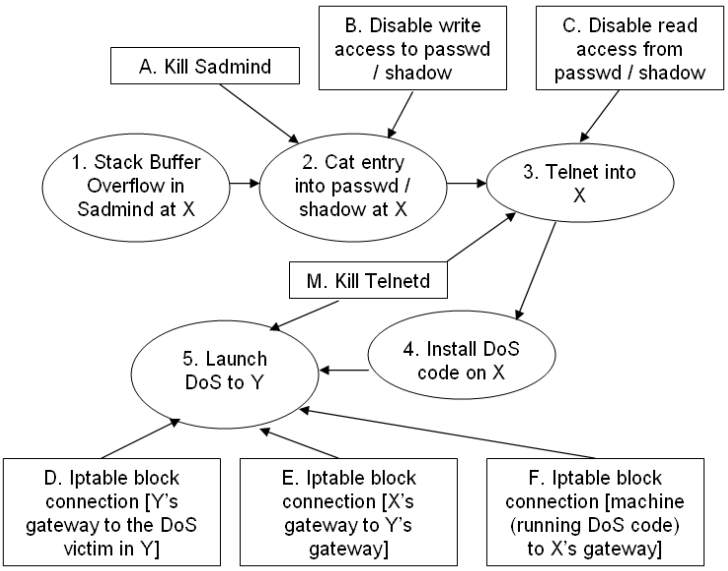


Figure 8.18. AS: MIT LLDoS

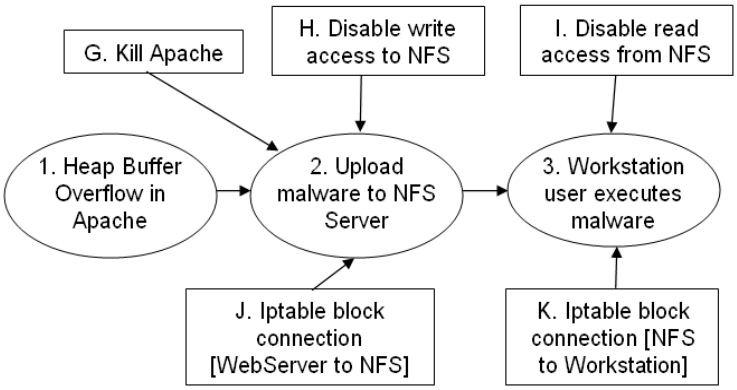


Figure 8.19. AS: MalExec

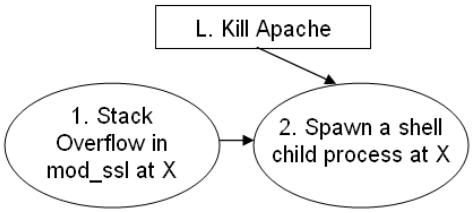


Figure 8.20. AS: ModSSL

To evaluate the design, we inject attacks from different attack scenarios into our testbed (the application system) which is protected by ORIGIN. We use two attack scenarios from previous representative works in this field. One is the MIT LLDoS attack scenario (Figure 8.18) as seen in [8, 62]. The other (Figure 8.19) is from the work by Ou in a system called MulVAL [40]. We create a third attack scenario called "ModSSL" (Figure 8.20). In the attack scenario, the oval nodes correspond to attack steps. The rectangular nodes correspond to responses. In each attack scenario, the adversary follows pre-specified probabilities which determine how likely he is to proceed from one step to the next step in the absence of any response. For both LLDoS and MalExec, from step 1 to step 2 the value is 0.8, and for the rest, the probabilities are 0.9. For ModSSL, from step 1 to step 2, the value is 0.1. For responses, the probability of success is a function of the timing when they are deployed (response can take time to be fully deployed.). None of these values are known to ORIGIN and it is left to the learning mechanism of ORIGIN.

Table 8.8. Overall |Iv| for each attack scenario injected to the testbed (IRS in absence)

LLDoS	MalExec	ModSSL
30	21	10

After we inject an attack scenario into the system, we sum the impact vectors from the achieved attack steps and the impact vectors from the deployed responses. Table 8.8 shows the |Iv| for each attack scenario when ORIGIN is not present and all the attack steps are achieved. Note that these values result from a subjective decision by the sysadmin about the importance of the different transactions and the security goals in the testbed system. The absolute values do not matter but the reduction by IRS is important.

We run ORIGIN with the conceptualization ON/OFF and study the results from both cases. Since we are assuming all three attack scenarios are unknown to ORIGIN at the beginning, there is no pre-built attack graph for any of them. Thus for previous IRS systems including our previous ADEPTS I and ADEPTS II, no response will be considered, and the attack will always go through. This means they can only achieve the same |IV|s as for an unprotected system as given in Table 8.8.

Table 8.9. Nodes (in component ID / detector ID pair) generated by ORIGIN

LLDoS: 1.Sadmind (StackOverflow) => 2.PasswdShadowFiles (UpdateFiles) => 3.Telnetd (NewUserLogin) => 5.Apache (NetworkDoS)
MalExec: 1.Apache (HeapOverflow) => 2.NFSFile (CreateFiles) => 3.Windows_wn (UnAuthExec)
ModSSL: 1.Apache (StackOverflow) => 2.Linux_wsvr (SpawnShell)

Table 8.10. Conceptualized nodes (Conceptualize(G,2,3))

LLDoS: 1.Program (MemError) => 2.File (ContentChange) => 3.Program (SecPolicyChange) => 5.Program (DoS)
MalExec: 1.Program (MemError) => 2.File (ContentChange) => 3.OS (UnAuthExec)
ModSSL: 1.Program (MemError) => 2.OS (UnAuthExec)

Table 8.11. Conceptualized nodes (Conceptualize(G,1,2))

LLDoS: 1.Base (GotEffect) => 2.Base (GotEffect) => 3.Base (GotEffect) => 5.Base (GotEffect)
MalExec: 1.Base (GotEffect) => 2.Base (GotEffect) => 3.Base (GotEffect)
ModSSL: 1.Base (GotEffect) => 2.Base (GotEffect)

Table 8.9 shows the nodes (in component ID / detector ID pair) and the edges created by the attack graph generation process for each of the three attack scenarios. For LLDoS, there is no corresponding node for step 4. “Install DoS code on X” because in our system configuration specification, there’s no detector to detect that step. Table 8.10 shows the nodes and edges after conceptualization with Conceptualize(G,2,3). Each node shows the conceptualized component and the conceptualized detector alert. As shown in Table 8.11, all three attack scenarios become identical in terms of their prefixes with Conceptualize(G,1,2).

Next we present the results showing situations when conceptualization gives benefits and also situations when conceptualization has drawbacks. For each experiment, we conduct three batches of executions and take the average. In each batch of experiment, we inject 100 instances of attacks (from the same scenario) into the testbed. We then plot the averaged |Iv| readings for each corresponding attack instance.

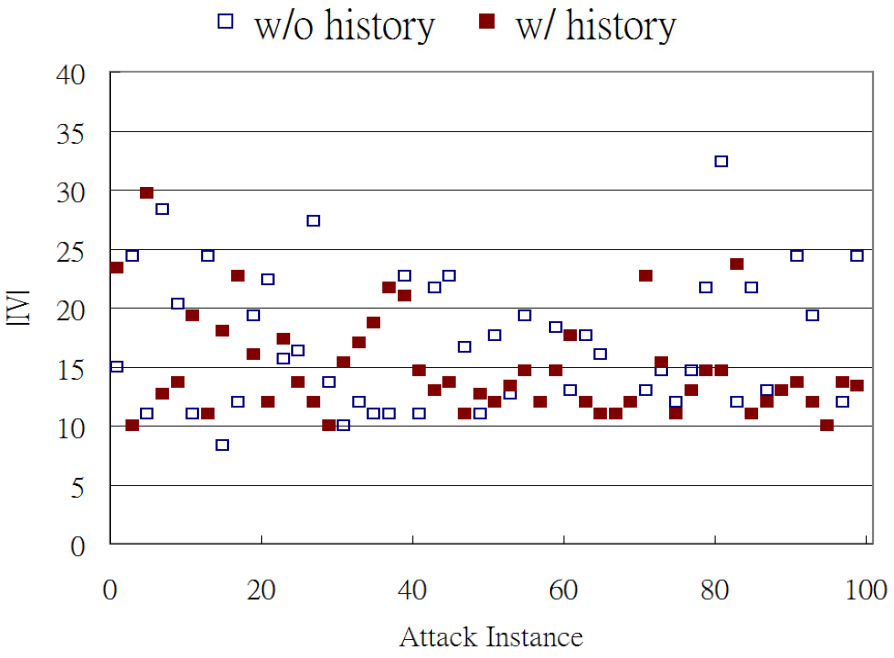


Figure 8.21. w/o conceptualization. LLDoS w/ and w/o history from MalExec

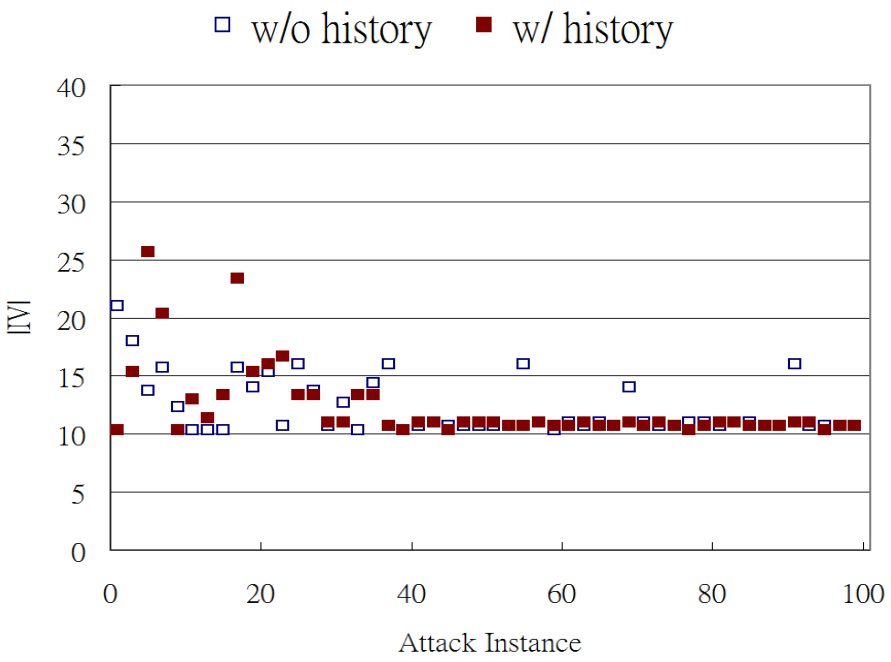


Figure 8.22. w/o conceptualization. MalExec w/ and w/o history from LLDoS

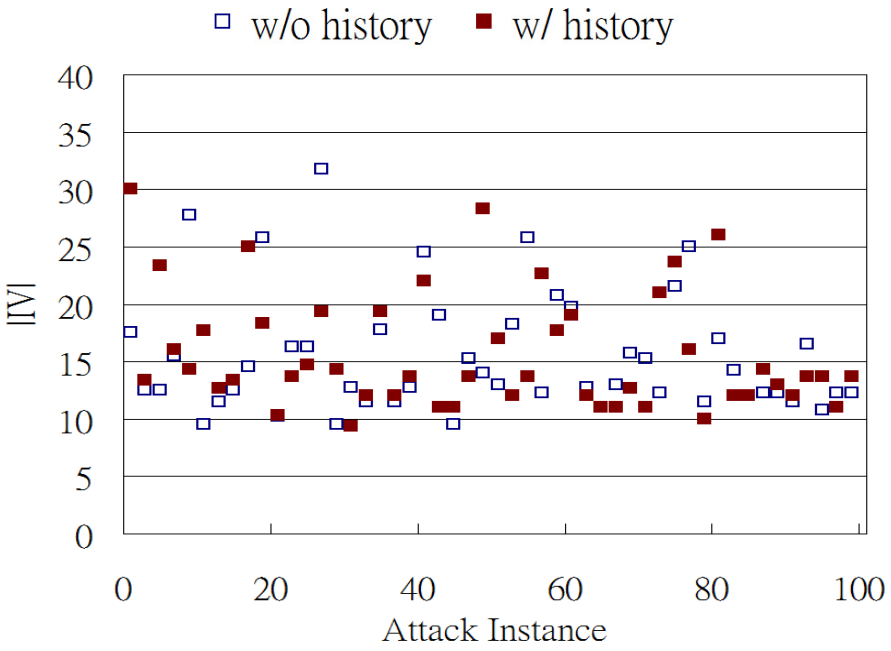


Figure 8.23. Conceptualize(G,2,3). LLDoS w/ and w/o history from MalExec

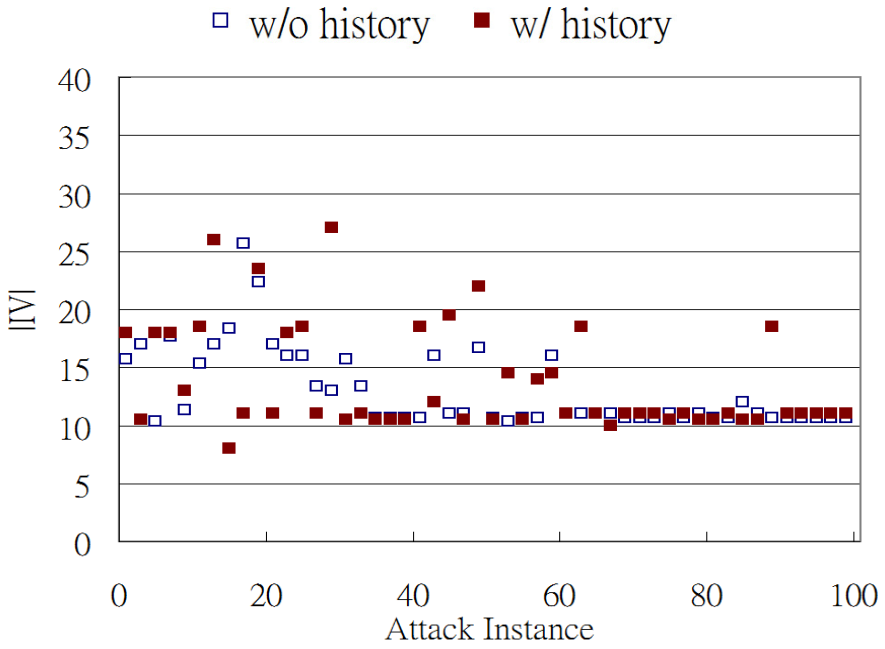


Figure 8.24. Conceptualize(G,2,3). MalExec w/ and w/o history from LLDoS

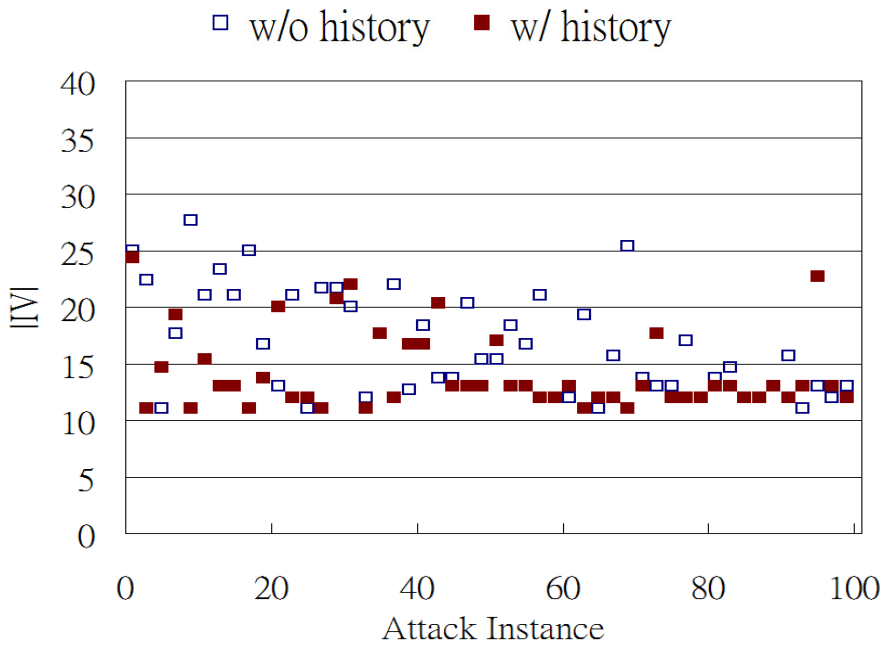


Figure 8.25. Conceptualize(G,1,2). LLDoS w/ and w/o history from MalExec

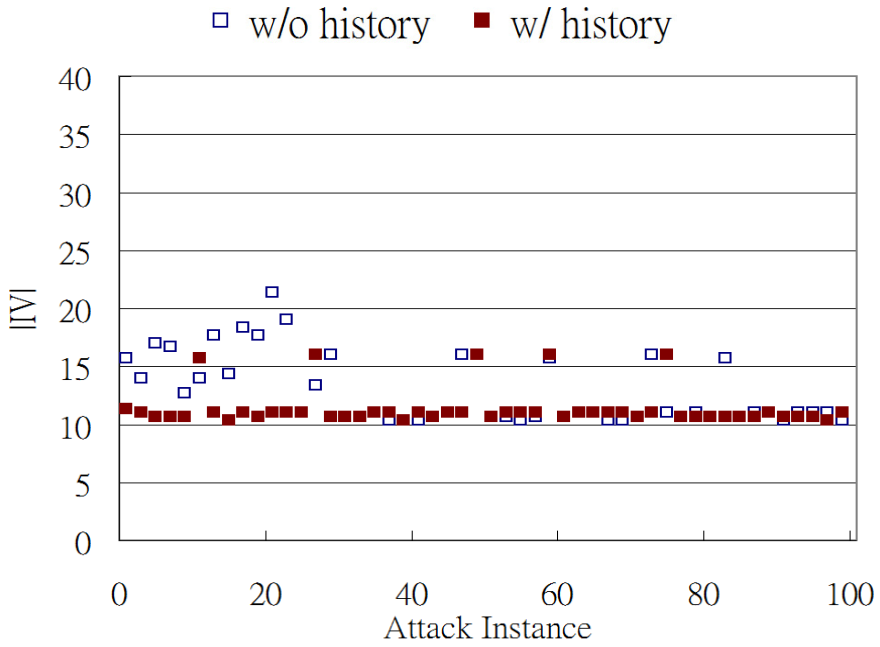


Figure 8.26. Conceptualize(G,1,2). MalExec w/ and w/o history from LLDoS

8.10.1 Benefits from Conceptualization

For this experiment, we use LLDoS and MalExec as the attack scenarios. This is because we found they share similarities after being conceptualized. Specifically, they share the first two steps in Table 8.10 and the first three steps in Table 8.11. To start with, we first experiment on LLDoS and MalExec without any conceptualization. The results are shown in Figure 8.21 and Figure 8.22. Figure 8.21 shows the result on injecting LLDoS into the testbed without any history in ORIGIN's ATL and also pre-injecting MalExec followed by injecting LLDoS (referred to as "with history"). The significant discontinuities in the data points are due to the non-determinism in both the attack escalations (the attacker can fail at any step) and the deployed responses (a response can also fail). Like previous adaptation-capable IRS such as [48], we see the |Iv| value decreases as more attack instances are seen. This is due to ORIGIN having a more accurate estimate on both the attack escalations and effectiveness of responses for the specific attack.

From both Figure 8.21 and Figure 8.22, we see no significant differences between with and without history. This is understandable as the vulnerabilities exploited by

LLDoS and MalExec are completely different. Therefore, even with history from one attack, the other attack is still regarded as a zero-day attack.

For the next run, we proceed with Conceptualize(G,2,3) in ORIGIN. The conceptualized attack graph nodes are shown in Table 8.10. The results are shown in Figure 3.5 and Figure 5.3. Still not much difference is observed between when history is available and when history is not available. This is because the Conceptualize(G,2,3) conceptualized attack graphs for LLDoS and MalExec only shares the first two nodes. And after checking with Figure 8.18 and Figure 8.19, we know that they share only two responses: 'Kill XXX process' and 'Disable Write access to YYY'. A further investigation turns out that these two responses are not very effective. So knowing one attack does not really facilitate the response to the other attack.

We then proceed further to Conceptualize(G,1,2). This results in attack graph nodes as shown in Table 8.11. The results are shown in Figure 8.25 and Figure 8.26. Eventually, we observe distinctive improvement in the case where history is available. This means that the originally zero-day attack is no-longer totally unknown to the IRS. The improvement is more significant for the case of injecting MalExec with history from LLDoS (Figure 8.26) and less significant in the other case (Figure 8.25). This is due to the fact that even with the history from MalExec, ORIGIN still needs to figure out the likelihood of escalation to Step 5 and the effectiveness of response M,D,E, and F in LLDoS (Figure 8.18). While with the history from LLDoS, ORIGIN only needs to figure out if responses J and K (Figure 8.19) are effective. Actually, we found that in both cases, the most effective response is the "Disable Read Access", which corresponds to responses C and I. These responses are in the shared part of the conceptualized attack graphs for the two attacks and hence history benefits from them.

In summary, ORIGIN is able to respond to a zero-day attack most notably MalExec, having seen a distinct attack LLDoS before.

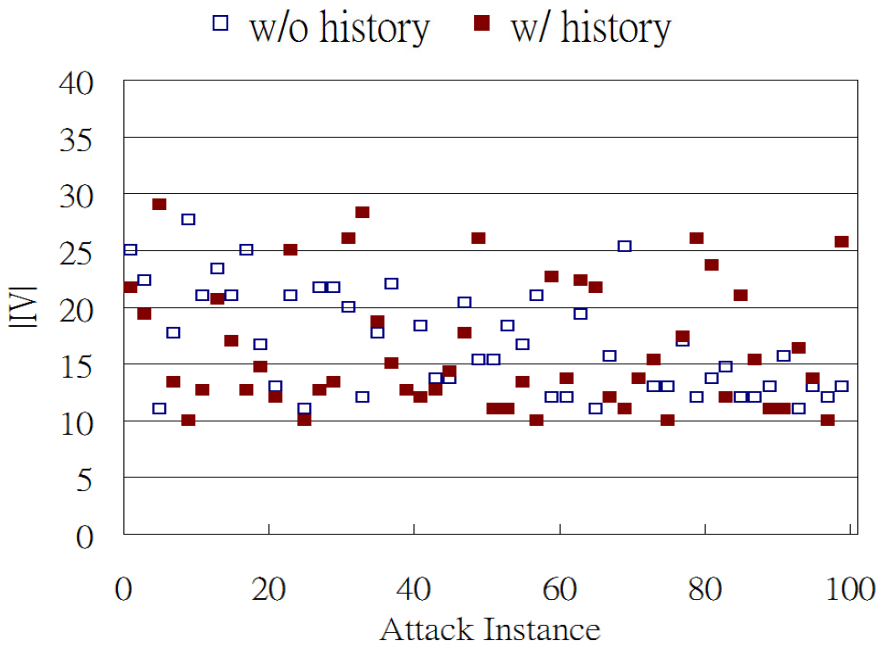


Figure 8.27. Conceptualize(G,1,2). LLDoS w/ and w/o history from ModSSL

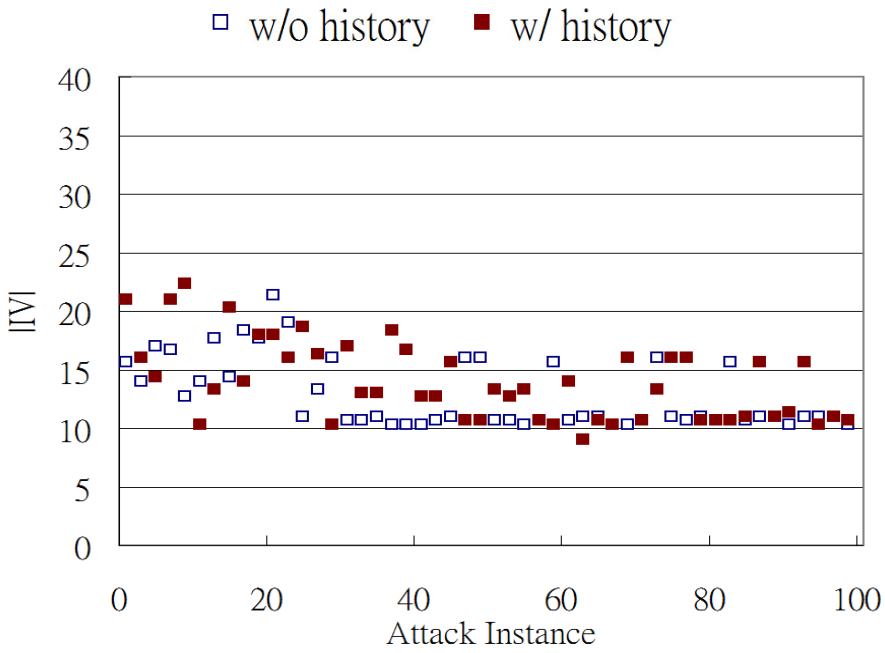


Figure 8.28. Conceptualize(G,1,2). MalExec w/ and w/o history from ModSSL

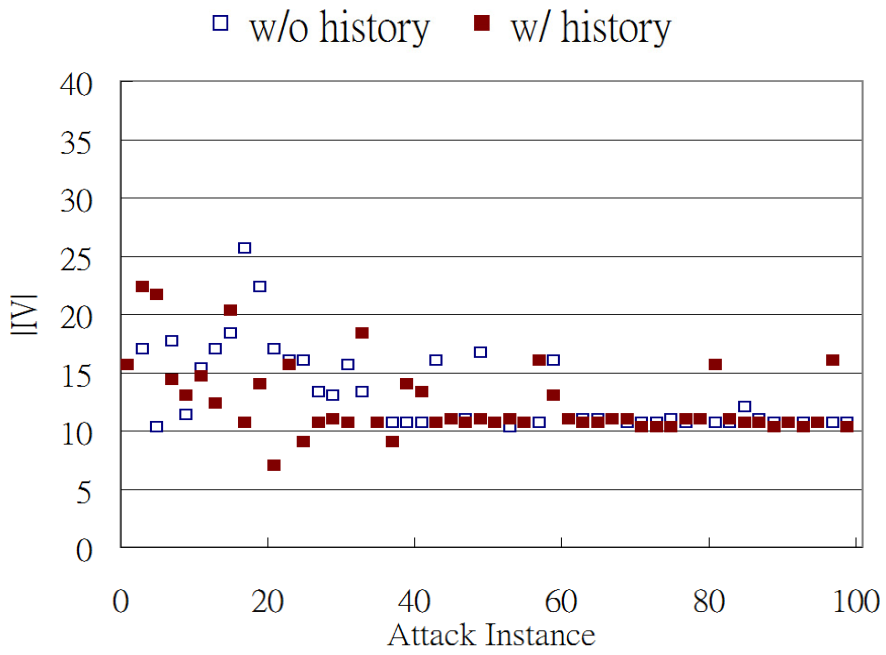


Figure 8.29. Conceptualize(G,2,3). MalExec w/ and w/o history form ModSSL

8.10.2 Drawbacks from Conceptualization

In our experiment, we found that for both LLDoS and MalExec, the likelihood of attack propagation from the first step to the second step is roughly 80%. We create the attack ModSSL for further investigation into the capabilities of conceptualization (Figure 8.20). When injecting the ModSSL attack, the attacker is intentionally restrained from proceeding to step 2 from step 1 with a low propagation probability of 10%. Figure 8.27 and Figure 8.28 present the result from first injecting ModSSL to the system and then followed by LLDoS and MalExec respectively. The conceptualization level is set at Conceptualize(G,1,2) as this is the level when the system will consider all three attack scenarios as identical concept.

From both Figure 8.27 and Figure 8.28, we see worse performance with history from ModSSL. This is because the pre-injected ModSSL has made ORIGIN believe that propagation from step 1 and step 2 for attacks of that concept is very unlikely (around 10%). This decreases the likelihood of early proactive responses such as C and I when the first attack step is achieved. These responses are useful for both LLDoS and MalExec for they have a much higher chance of escalating from step 1 to step 2.

In Figure 8.29, we increase the concept level to Conceptualize(G,2,3). At this level, ModSSL is no longer following the same concept as either LLDoS or MalExec (Figure 8.18, Figure 8.19, and Figure 8.20). Therefore, the performance against MalExec with ModSSL pre-injected is back to normal as if ModSSL has not been seen.

One thing to note is that even in the cases of Figure 8.27 and Figure 8.28, ORIGIN is still able to self-correct itself and approach the better response choice as more attack instances are being seen.

Overall, there exists an optimal conceptualization level for different cases the experiments indicate. The fact that conceptualization is causing a degradation in response quality can be automatically deduced by ORIGIN since the values in CPTs will change sharply. Therefore, ORIGIN can increase the conceptualization levels (deconceptualization) or turn it off completely for specific multi-stage attacks. This is left as a topic for future work.

9. CONCLUSIONS

We propose a new model for automated response in distributed systems. The proposed model is first instantiated by the design and implementation of an automated response systems called ADEPTS I. ADEPTS I uses a directed graph representation called I-GRAPH to model the spread of the failure and attack through the system. It provides algorithms to determine the possible path of spread and appropriately choose the response. The mapping between detectors and response actions are dynamic and not restricted to a pre-assigned set pairs. ADEPTS I creates attack sub-graphs from the I-GRAPH for each incident instance and processes each sub-graph independently, thus making it scalable with the number of alerts. ADEPTS I 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. Empirical results comparing ADEPTS I and BASELINE (Sec. 8.4) constitutes the evidence to thesis claim C2.

We develop a system named SWIFT (ADEPTS II) as an improved instantiation of the proposed model over ADEPTS I to pursue globally optimal responses. The optimality criterion takes into account the impact on the whole system from a deployed response in reducing functionality and from the spread of the attack. We proved that the optimal response determination problem for multi-stage attacks is NP-hard, fundamentally because responses at different services are inter-dependent. Hence, we proposed using a Genetic Algorithm (GA) based framework. The proposed GA framework enables the use of history information from past attacks that are similar to the current one through seeding the initial chromosome pool with the learnt effective response combinations from those similar attacks. The evaluation brings out the fact that survivability improves with the global response determination process of SWIFT over a heuristic based response determination (e.g. 27% improvement based on experiment in Section 8.6) used in ADEPTS I. History information from past similar attacks is used to deal with attack variants in SWIFT with corresponding empirical results (Sec. 8.9) supporting thesis claim C3.

We develop a third system named ORIGIN (ADEPTS III) as a further improved instantiation of the proposed model. The focus of ORIGIN is to provide automated response for zero-day multi-stage attacks. For a zero-day attack, the corresponding attack graph is not known *a priori*. Therefore, existing IRSs, which use pre-built attack graphs, are ineffective for such kind of attacks. A multi-stage attack can have non-deterministic escalations from one stage to another and many response choices may be available to deter the escalation. Therefore knowledge from prior attacks is useful. However, for zero-day attack such prior knowledge is not available to the IRS. We firstly propose an object-oriented system configuration specification methodology. We also present an online attack graph generation process to generate a Bayesian Network based attack graph based on detector alerts and the system configuration specification. We propose a technique called "conceptualization of attack graph" which manages to establish linkage between a zero-day attack and past attacks to improve the performance of intrusion response for the zero-day attacks. We validate the ORIGIN system by two representative attack scenarios used in two independent previous papers [40, 62] and one custom-built attack scenario. We show that conceptualization to an appropriate level enables ORIGIN to respond effectively to LLDoS and MalExec when they are zero-day attacks. This is another evidence when history may help reduce the impact from an attack (thesis claim C3). However, we find that the performance is sensitive to the conceptualization level chosen and how to determine the appropriate conceptualization level remains an open problem.

There are many issues yet to be addressed following this work. For example, the current design does not consider the use of recovery responses. Typically, recovery of compromised parts of a system can be achieved through system checkpoints and rollback. Existing work such as virtual machine snapshot and volume shadow copy service (on MS Windows) can serve as the building blocks for this task. However, a response action, whether containment or recovery, may have unintended consequences, such as disruption of normal functionality and violation of system integrity, and these will have to be factored in carrying out a recovery response. In addition, another important work is reconfiguration of a system after an attack, which is also not addressed in this work. The goal of reconfiguration is to fix the vulnerabilities exploited by the attack and prevent the same attack from compromising the system again. Reconfiguration can be simply changing the settings in system configuration files or applying software patches. A recent work [63] attempts to generate patches for software bugs automatically through genetic programming, which can potentially be extended to generate patches for security vulnerabilities and serve as a means to reconfigurate the system.

LIST OF REFERENCES

LIST OF REFERENCES

[1] DARPA-IPTO, "Organically Assured and Survivable Information Systems (OASIS)," (http://www.darpa.mil/ipto/programs/oasis)

[2] DARPA-IPTO, "Self-Regenerative Systems (SRS)," (http://www.darpa.mil/ipto/programs/srs)

[3] N. Provos, "Improving host security with system call policies," in *USENIX Security Symposium*, 2003.

[4] N. MacDonald, "Host-Based Intrusion Prevention Systems (HIPS) Update: Why Antivirus and Personal Firewall Technologies Aren't Enough," Gartner.com, 2007.

[5] N. Desai, "Intrusion prevention systems: The next step in the evolution of IDS," 2003.

[6] G. Young and J. Pescatore, "Magic Quadrant for Network Intrusion Prevention System Appliances, 1H08," Gartner.com, 2008.

[7] Y. S. Wu, B. Foo, Y. Mei, and S. Bagchi, "Collaborative intrusion detection system (CIDS): a framework for accurate and efficient IDS," in *IEEE Annual Computer Security Applications Conference*, 2003, pp. 234-244.

[8] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *ACM conference on Computer and Communications Security*, 2002, pp. 245-254.

[9] S. Noel, E. Robertson, and S. Jajodia, "Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances," in *IEEE Annual Computer Security Applications Conference*, 2004, pp. 350-359.

[10] CMU, "Survivable Network Technology," (http://www.sei.cmu.edu/organization/programs/nss/surv-net-tech.html)

[11] NSSLabs, "McAfee M-8000 Network IPS Product Certification," 2009.

[12] T. Toth and C. Kruegel, "Evaluating the Impact of Automated Intrusion Response Mechanisms," in *IEEE Annual Computer Security Applications Conference*, 2002, pp. 9-13.

[13] F. Cohen, "Simulating Cyber Attacks, Defenses, and Consequences," (http://all.net/journal/ntb/simulate/simulate.html)

[14] R. Ellison, R. Linger, T. Longstaff, and N. Mead, "Case Study in Survivable Network System Analysis," SEI, CMU, Technical Report1998.

[15] A. H. R. Anderson, and R. Hundley, "Studies of Cyberspace Security Issues and the Concept of a U.S. Minimum Essential Information Infrastructure," in *Information Survivability Workshop*, 1997.

[16] E. Fisch, "Intrusion Damage Control and Assessment: A Taxonomy and Implementation of Automated Responses to Intrusive Behavior," College Station, TX: Texas A&M University, 1996.

[17] C. A. Carver and U. W. Pooch, "An Intrusion Response Taxonomy and its Role in Automatic Intrusion Response," in *Proceedings of IEEE Workshop on Information Assurance and Security*, 2000, pp. 129-135.

[18] U. Lindqvist and E. Jonsson, "How to systematically classify computer security intrusions," in *IEEE Symposium on Security and Privacy*, Oakland, CA, 1997, pp. 154-163.

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

[20] P. A. Porras and P. G. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997, pp. 353–365.

[21] D. J. Ragsdale, C. A. Carver Jr, J. W. Humphries, U. W. Pooch, I. Technol, O. Center, and U. S. M. Acad, "Adaptation techniques for intrusion detection and intrusionresponse systems," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

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

[23] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling High Bandwidth Aggregates in the Network," AT&T Center for Internet Research at ICSI (ACIRI)2001.

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

[25] C. Cachin, "Distributing trust on the Internet," in *Proceedings of International Conference on Dependable Systems and Networks*, 2001, pp. 183–192.

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

[27] V. Stavridou, B. Dutertre, R. A. Riemenschneider, and H. Saidi, "Intrusion tolerant software architectures," in *DARPA Information Survivability Conference & Exposition II*, 2001.

[28] P. Brooke and R. Paige, "Fault Trees for Security System Analysis and Design," *Elsevier Journal of Computers and Security,* vol. 22, pp. 256-264, May 2003 2003.

[29] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and 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.

[30] M. Dacier, Y. Deswarte, and M. Kaaniche, "Quantitative Assessment of Operational Security: Models and Tools," LAAS Research Report 96493May 1996 1996.

[31] R. Ortalo, Y. Deswarte, and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *IEEE Tranctions on Software Engineering,* pp. 633-650, 1999.

[32] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *IEEE Symposium on Security and Privacy*, 2002, pp. 273-284.

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

[34] R. P. Lippmann and K. W. Ingols, "An Annotated Review of Past Papers on Attack Graphs," 2005.

[35] Y. Zhai, P. Ning, P. Iyer, and D. S. Reeves, "Reasoning about complementary intrusion evidence," in *IEEE Annual Computer Security Applications Conference*, 2004, pp. 39-48.

[36] W. A. Jansen, "Intrusion detection with mobile agents," *Computer Communications,* vol. 25, pp. 1392-1401, 2002.

[37] G. Helmer, J. S. K. Wong, V. Honavar, and L. Miller, "Automated discovery of concise predictive rules for intrusion detection," *The Journal of Systems & Software,* vol. 60, pp. 165-175, 2002.

[38] M. E. Ludovic, "GasSAtA, a Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis," in *Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium*, 1998.

[39] S. Jacobs, D. Dumas, W. Booth, and M. Little, "Security Architecture for Intelligent Agent Based Vulnerability Analysis," in *Third Annual Fedlab Symposium on Advanced Telecommunications/Information Distribution Research Program*, 1999, pp. 447-451.

[40] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *ACM conference on Computer and Communications Security*, 2006, pp. 336-345.

[41] T. R. Peltier, *Information security risk analysis*, 2 ed.: Auerbach Publications, 2005.

[42] R. E. Neapolitan, *Learning Bayesian Networks*: Prentice Hall, 2004.

[43] G. Modelo-Howard, S. Bagchi, and G. Lebanon, "Determining Placement of Intrusion Detectors for a Distributed Application through Bayesian Network Modeling," in *International Symposium on Recent Advances in Intrusion Detection*, 2008, pp. 271-290.

[44] C. A. Carver, J. M. D. Hill, and U. W. Pooch, "Limiting Uncertainty in Intrusion Response," in *Proceedings of the IEEE Workshop on Information Assurance and Security*, 2001, pp. 5-6.

[45] E. Bryant, J. Early, R. Gopalakrishna, G. Roth, E. H. Spafford, K. Watson, P. William, and S. Yost, "Poly^2 paradigm: a secure network service architecture," in *IEEE Annual Computer Security Applications Conference*, 2003, pp. 342-351.

[46] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using Specification-Based Intrusion Detection for Automated Response," in *International Symposium on Recent Advances in Intrusion Detection*, 2003.

[47] W. Lee, "Toward cost-sensitive modeling for intrusion detection and response," *Journal of Computer Security,* vol. 10, pp. 5-22, 2002.

[48] Y. S. Wu, B. Foo, Y. C. Mao, S. Bagchi, and E. H. Spafford, "Automated adaptive intrusion containment in systems of interacting services," *Computer Networks,* vol. 51, pp. 1334-1360, 2007.

[49] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley Professional, 1989.

[50] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, 2002, pp. 595-604.

[51] G. Khanna, I. Laguna, F. A. Arshad, and S. Bagchi, "Distributed Diagnosis of Failures in a Three Tier E-Commerce System," in *IEEE Symposium on Reliable Distributed Systems*, 2007.

[52] P. J. v. Laarhoven and E. H. Aarts, *Simulated Annealing: Theory and Applications*, 1 ed.: Springer.

[53] Wikipedia, "Zero day attack," (http://en.wikipedia.org/wiki/Zero-Day_Attack)

[54] P. Szor and P. Ferrie, "Hunting for Metemorphic," *Virus,* vol. 123, 2001.

[55] P. Ning, Y. Cui, and D. S. Reeves, "Analyzing Intensive Intrusion Alerts via Correlation," *Lecture Notes in Computer Science,* pp. 74-94, 2002.

[56] A. Somayaji and S. Forrest, "Automated Response Using System-Call Delays," in *USENIX Security Symposium*, 2000.

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

[58] P. Uppuluri and R. Sekar, "Experiences with Specification-Based Intrusion Detection," *Lecture Notes in Computer Science,* pp. 172-189, 2001.

[59] Y.-S. Wu, G. Modelo-Howard, B. Foo, S. Bagchi, and E. Spafford, "The Search for Efficiency in Automated Intrusion Response for Distributed Applications," in *IEEE International Symposium on Reliable Distributed Systems (SRDS)* Naples, Italy, 2008.

[60] O. Sheyner and J. Wing, "Tools for Generating and Analyzing Attack Graphs," *Lecture Notes in Computer Science,* pp. 344-371, 2004.

[61] B. Stroustrup, *The C++ Programming Language*, 3rd ed.: Addison-Wesley, 2000.

[62] P. Ning, D. Xu, C. Healey, and R. S. Amant, "Building attack scenarios through integration of complementary alert correlation methods," in *Annual Network and Distributed System Security Symposium*, 2004, pp. 97-111.

[63] S. Forrest, W. Weimer, T. Nguyen, and C. L. Goues, "A Genetic Programming Approach to Automated Software Repair," in *Genetic and Evolutionary Computing Conference*, 2009.

VITA

VITA

Yu-Sung Wu was born in Hsin Chu, Taiwan. He got his Bachelor degree in Electrical Engineering from National Tsing Hua University, Taiwan in 2002. He soon came to Purdue and got his Master degree in Electrical and Computer Engineering in 2004. He continued his study at Purdue and got his Ph.D. degree in Electrical and Computer Engineering in 2009.

Yu-Sung Wu's research has been focused on information security and system dependability. His work includes the design and implementation of various intrusion detection systems and intrusion response systems for distributed applications. At Purdue, he has collaborated with faculty and students from CERIAS and Computer Science department on many security-related projects. During summers, Yu-Sung has worked at Avaya Labs in New Jersey, where he participated in projects related to Voice-over-IP system security.