The Search for Optimality in Online Intrusion Response for a Distributed E-Commerce System

Yu-Sung Wu, Gaspar Howard, Matthew Glause, Bingrui Foo, Saurabh Bagchi, Eugene Spafford+
CERIAS and School of Electrical and Computer Engineering
+CERIAS and School of Computer Science
Purdue University
{yswu, gmodeloh, mglause, foob, sbagchi, spaf}@purdue.edu

Abstract

Providing automated responses to security incidents in a distributed computing environment has been an important area of research. This is due to the inherent complexity of such systems that makes it difficult to eliminate all vulnerabilities before deployment and costly to rely on humans for responding to incidents in real time.

Here we formalize the process of providing automated responses in a distributed system and the criterion for asserting global optimality of the responses. We show that reaching the globally optimal solution is an NP-complete problem. Therefore we design a genetic algorithm framework for searching for good solutions. In the search for optimality, we exploit the similarities among attacks, and use the knowledge learnt from previous attacks to guide future search. The mechanism is demonstrated on a distributed e-commerce system called Pet Store with injection of real attacks and is shown to improve the survivability of the system over the previously reported ADEPTS system.

Keywords: *automated intrusion response, intrusion containment, optimal response, distributed e-commerce system, survivability.*

1 Introduction

Distributed systems comprising multiple services interacting among themselves to provide end-user functions are becoming an increasingly important platform for business-to-business (B2B) and business-to-consumer (B2C) systems. An example is electronic commerce, or e-commerce systems. The huge financial stake involved in e-commerce makes the distributed system infrastructure supporting it a prime candidate for computer security attacks.

This motivation has long led to interest in securing distributed systems through detection of intrusions and of late, through automated responses to intrusions. The rudimentary response mechanisms often bundled with anti-virus or intrusion detection system (IDS) products overwhelmingly consider only immediate local responses that are directly suggested by the detected symptom. For example, a suspect packet being flagged by a network IDS may cause the specific network connection to be terminated. These are applicable in stand-alone systems and do not account for interaction effects among multiple components in a distributed system as a result of which the attack can spread from one service to another.

Our model for the application or payload system being protected is that it comprises multiple services (web service, authentication service, admin service) running on separate hosts and communicating through standardized protocols, such as SOAP. An example is provided by a distributed e-commerce system. Our model for the target attack is that it is an external multi-stage attack which first compromises the services that have external interfaces and using this "foot in the door" compromises internal services with the goal of disrupting some transactions supported in the system or violating some of the goals in the system. This is the model commonly used in the literature for distributed intrusion response systems (IRSs) [4][12].

The few available dedicated IRSs for distributed systems [4]-[9] 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 in a system called ADEPTS, take a greedy approach and do not give a globally optimal solution. 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.

In this paper, we present a framework to reason about the 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 due to the further spread of the attack. Note that this framework has to be 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 service may not be sufficient. The global 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.

We develop three kinds of response actions—recovery-focused, reactive containment-focused, and proactive containment-focused. The first class is used to recover services that have already been impacted, to a more functional state. The second class deploys responses on the services that are currently at the "front" of the attack. Qualitatively, front of an attack is the set of services such that all its predecessor services (the services that are invoked before it in satisfying a user transaction) are estimated to be affected and none of its successor services is. The third class is comprised of responses that are deployed at services that have not yet been affected, but the system estimates will be, based on current knowledge. This is thus a proactive mechanism since alerts have not yet been seen for the services on which the responses are deployed. The techniques are incorporated in our existing IRS called ADEPTS[1]. ADEPTS takes alerts from detectors embedded in the payload system, executes its algorithms, and sends response actions back to the system. The detectors are imperfect and can generate false or missed alarms.

We prove that solving the optimal response determination problem is NP-complete. This is fundamentally because of the dependencies that exist between responses and the universe of possible responses, taking into account the targets of each response, is very large. In our candidate applications subjected to automated multi-stage attacks, it is imperative to deploy prompt responses at runtime. Hence, we come up with an approximate solution to the problem.

The approximate solution relies on history of the attacks seen in the system and the paths they have taken, and estimate of the effectiveness of responses deployed earlier. The former is stored in a structure called the attack phase cache which captures the evolution of previous attack instances using a graph representation. Each graph node represents one snapshot of the attack and the edges represent the evolution from one snapshot to the next. Each node itself is a graph where a node determines the attack goal that is achieved (such as, a root shell is spawned on the Apache web server machine) and the edges represent the causal relationships between the goals (such as, a buffer overflow must be performed before a root shell can be obtained).

To solve the approximate problem, we use ge*netic algorithm* (GA) based search through the universe of possible responses. A critical factor in the performance of a GA based solution is the quality of the initial gene pool used to initiate the search. The initial pool includes responses that are locally optimal for each service. Since GA guarantees that the quality of the final solution is better than any element in the initial pool, we are guaranteed that ADEPTS will never choose a response that is worse (with respect to our optimality criterion) than baseline ADEPTS after incurring any initial loss of performance due to inaccuracy in initial parameter settings. As multiple attack instances of a given type are seen in ADEPTS, the effectiveness of the deployed responses are updated and the quality of the gene pool used to initiate the GA-based search is improved. Thus, ADEPTS adapts to provide better responses as history builds up in the system.

It is widely observed that multi-stage attacks take polymorphic forms and detecting the different forms of a given attack poses a challenge for an IRS. In ADEPTS, we provide an algorithm for approximate matching of the current attack instance with previous instances. The approximate graph matching algorithm enables ADEPTS to "borrow" previously computed effective responses for mitigating the current attack instance.

The ADEPTS system is demonstrated on a distributed three tier e-commerce system called Pet Store that uses the J2EE platform and is developed by Sun Microsystem's Java BluePrints program. The testbed has the classic structure of the web server, the application server (implemented using communicating Enterprise Java Beans (EJBs)), and the database server. We create a library of multi-stage attack scenarios. A set of network-based and host-based detectors generates alarms. The output metric is survivability, a high-level metric that is based on the transactions that are supported and the system goals that are maintained in the application once the attack is injected and the responses determined by ADEPTS are deployed. The relative importance of each is determined by the system owner as weights in the survivability computation. The experiments show the survivability with ADEPTS compared to baseline ADEPTS, the ability of ADEPTS to adapt its responses as increasing numbers of attack instances are seen, its ability to handle polymorphic forms of an attack, and the latency of response determination in the two systems.

The rest of the paper is organized as follows. Section 2 presents the design of the framework for reasoning about optimality. Section 3 describes the algorithms used to search for the optimal responses. Section 4 describes the e-commerce testbed and the attack scenarios. Section 5 presents the experiments and the results. Section 6 discusses some subtle aspects of the presented solution. Section 7 surveys related work and Section 8 concludes the paper.

2 Framework for Optimality

2.1 Modeling Spread of an Attack: Background

A representation called an Intrusion Graph (I-GRAPH) is used for modeling the spread of the attack. 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 goal is achieved through multiple intermediate intrusion goals and each is represented as an I-GRAPH node. The node in the I-GRAPH may be a predicate corresponding to a low-level attack manifestation, or

[1] We use the name ADEPTS to denote the augmented system with the features that we are presenting here. The earlier version of the system is referred to as baseline ADEPTS.

propositional with parameterized variables, corresponding to a higher level manifestation. The intrusion goals have mutual dependency relationships which are modeled using the I-GRAPH edges (such as, a buffer overflow must be performed before a root shell can be obtained). 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 (parent nodes) need to be achieved before the goal at the tail (child node) can be achieved. An I-GRAPH node may have zero, one, or more detectors whose alerts map to it. The I-GRAPH representation is derived from the idea of attack graphs and fault trees. Our work can leverage efforts at building complete and succinct attack graphs, e.g., using model checking approaches. Automatic ways to build a graph based on specifications are shown in [8]. We do not focus on generation of the graph, rather on using the representation for automated response.

When alerts are received at ADEPTS for a node in the I-GRAPH, ADEPTS calculates a Compromised Confidence Index (CCI) value for all the nodes. CCI is a measure of the likelihood that the node has been achieved ("a node is achieved" implies the goal represented by the node has been achieved by the attack). For a leaf node, the CCI value comes from the alert confidence corresponding to the alert that is mapped to the node. Each detector has a confidence value for its alerts, termed *alert confidence*. This can be provided by the meta-detector which correlates alarms from multiple simple detectors [1] or can be a value calculated by ADEPTS through its missed and false alarm detection algorithms ([13] Sec. 4.1.1, 4.1.2). Through this paper, we use the term CCI of a node and the probability that a node has been achieved synonymously.

2.2 Response Model

The response mechanism incorporates three kinds of responses. The first two kinds are both containment-oriented responses which block the attack propagation from one node to the other node in I-GRAPH. In our response model, we use the failure probability of these responses to attenuate the CCI values on the child nodes from the parent nodes. Specifically, $CCI_{child} = CCI_{parent}$ * P(the response X on the edge from the parent node to the child node fails). The probability measure of the response is the complement of the Effectiveness Index of a response which is determined by ADEPTS through observation of alerts. This mechanism is the same as in baseline ADEPTS [12] and is thus not described here. In Figure 1, responses R_P and R_Q fall in this category, and $CCI_B = CCI_A * P(R_P \text{ fails})$.

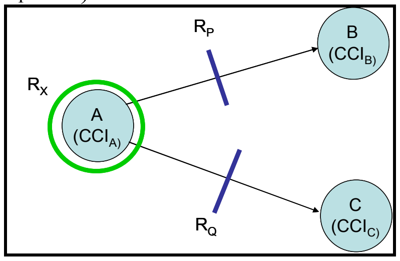


Figure 1. Response model for fragment of I-GRAPH. Node A has been achieved, and containment responses are being evaluated to block nodes B and C from being reached.

The second type of response is the recovery response. This kind of response is used to recover a compromised node back to a more functional state. In our model, these responses have the effect of resetting the CCI value on a node to a default value. We assume the effect from the recovery responses is not persistent. Thus if the node is compromised again right after some recovery response was deployed, the CCI value of that node will be increase to reflect the new attack effect. This is in contrast to the blocking response, which has a prolonged effect on deterring the attack propagation. Response R_X in Figure 1 falls into the recovery response category. The effect from response R_X would be $CCI_A \leftarrow CCI_{A\_default}$. The default value of CCI is dependent on the response that is deployed.

2.3 Response Mechanism for a Multi-Stage Attack

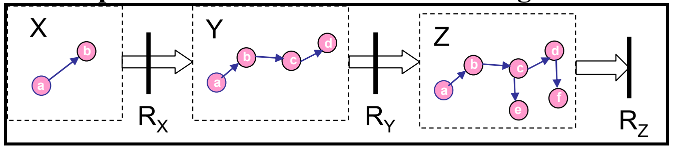


Figure 2. Three different snapshots for a given attack scenario. Three responses R_X, R_Y, R_Z are deployed between the snapshots.

In general, a multi-stage attack consists of multiple 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. This fragment of the I-GRAPH is called the *Attack Sub-Graph* (ASG). Figure 2 shows three snapshots X, Y, and Z of an attack scenario. Generally, for a multi-stage attack consisting of k snapshots $\{s_1,s_2,..s_k\}$, the response mechanism is formally described by

$$RC_i = f(s_i, H)$$

s_i is the i^{th} snapshot, H generally speaking is the history information. With respect to our Genetic Algorithm framework, it corresponds to the EI of the responses and the snapshots in the ATL.

RC_i is the response combination decided by ADEPTS.

Therefore, in Figure 2, we have R_X=f(s_X,H), R_Y=f(s_Y,H), and R_Z=f(s_Z,H).

In general, there is more than one way to partition a multi-stage attack into its snapshots. One extreme is to treat each incoming detector alert as creating a new snapshot, while the other extreme is to consider the whole attack as a single snapshot. Practically speaking, we can assume there are groups of alerts that arrive in a batch and ADEPTS cannot deploy a response within a batch of alerts. This batch creates a snapshot.

2.4 Impact Vector Metric

We come up with a metric called *Impact Vector* for evaluating the favorableness of a response set. Firstly, we assume that the protected target system has a set of transactions and security goals that would be desirable to meet 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 dimensions may not be independent, in which case assigning the Iv values has to be done carefully taking the dependence into account. The higher the value is, the more severe the impact is. The range of Iv values is arbitrary with 0 being the lower bound. The summation of two impact vectors is also an impact vector and is defined as follows:

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

For each response r, there is an associated impact vector $Iv(r)$ which indicates the impact value 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. For each I-GRAPH node n, there is an associated impact vector Iv(n) which gives the impact value as a result of this node being achieved by an adversary.

2.5 Optimality of a Response Combination

Let us assume an attack has resulted in i snapshots $s_1,s_2,..,s_i$. Let us assume the I-GRAPH has m nodes $n_1,n_2,..n_m$. Then we evaluate the cost of a response combination $RC_i = f(s_i,H)$, which consists of n responses $\{r_1,r_2,..,r_n\}$. Assume the probability of each node being achieved in the attack considering the responses in RC_i is Prob(n_1), Prob(n_2),…, Prob(n_m). Then the cost of RC_i is defined by Eqn. (1).

$$Cost(RC_i) = |Iv(RC_i)| = \left|\sum_{k=1}^{m} Iv(n_k)\Pr ob(n_k) + \sum_{k=1}^{n} Iv(r_k)\right| \quad (1)$$

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 |Iv|.

$$RC_{i,opt} = \underset{RC_i}{\arg\min} \, |Iv(RC_i)| \quad (2)$$

This optimization is under the constraint that $RC_p \cap RC_q$, $p \neq q$, only contains recovery responses. This is from the assumption that containment responses are permanently deployed. We have traded off the additional power of considering responses with fixed lifetimes for simplicity.

2.6 Optimal Response Determination is NP-Complete

We prove that the problem of optimal response determination (ORD) for a given system is NP-hard while the decision version of the problem is NP-complete (NPC). The knowledge that can be assumed is the I-GRAPH and the probability of success of each response. We prove that ORD is NPC by showing first that it is NP and then reducing the set covering problem which is known to be NPC using a polynomial time transformation to ORD.

Given a response set $\boldsymbol{R}$ and a cost number k, it is possible in polynomial time to determine if the cost of $\boldsymbol{R}$ is less than k (decision version of problem). This is essentially the calculation in Eqn. (1) which is linear in complexity.

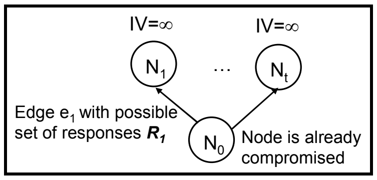

IV=∞
IV=∞
N1
...
Nt
Edge e1 with possible set of responses R1
N0
Node is already compromised

Figure 3. Transformation to map set covering problem to optimal response determination (ORD). In the simple I-GRAPH here, solving ORD solves set covering.

For the polynomial time reduction of set covering, consider the small I-GRAPH in Figure 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 IVs. 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 set $\boldsymbol{R}$ 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.

2.7 Domain Graph

The domain graph $D(s) \supseteq$ ASG and is a subgraph of I-GRAPH, which provides an approximate and a conservative bound on the nodes that may be reached by an adversary from the snapshot s. In Eqn. (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. Practically, this will adversely impact 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 observed alerts. The domain graph subsets the nodes to be considered so that a more timely reaction to the attack can be deployed.

DEFINITION: DOMAIN GRAPH

Given the I-GRAPH I and an snapshot s, the domain graph $D(s) = (V, E)$ where $V = \{$node $n \in I$ such that $Prob(n) \times |n.Iv|$ is greater than a given threshold $T\}$ and $E=\{e | e \in I.E$ and (u, v) where $u, v \in V\}$.

Prob(n) is the probability of node n being achieved by the adversary based on snapshot s and is estimated by a call to GEN_PROB(n,s,I, r_{NULL}).

ALGORITHM: GEN_PROB(n,s,I,rc)

```
/* Notation: n: node, s: the snapshot, I: I-GRAPH, rc: response
combination deployed
Resp(x,n)∈rc: the response on the edge from node x to node
n, epp(x,n): the probability of a propagation on the edge from
x to n.
*/
{
    for node n ∈ ASG
        Prob(n) = Alert confidence for alert for node n;
    for node n ∉ ASG
        if n has no parent nodes,
            Prob(n) = 0
        if n has OR-type parent nodes,
            Prob(n) = max { Prob(x) * (1- Resp(x,n).EI)*epp(x,n)
                        | x ∈ parent(n) }
        if n has AND-type parent nodes,
        Prob(n) = min { Prob(x) * (1-
                    Best_Resp(x,n).EI)*epp(x,n) |
                    x ∈ parent(n)
    }
```

This algorithm is based on the observation that for an AND node all parent nodes will have to be achieved and therefore a min operator is taken for probability of achieving the child node from all its parent nodes. For an OR node, since any path works, a max operator is used. The edge propagation probability calculation is discussed in Section 3.2. For generation of the Domain Graph, no response is taken into account so that the largest possible sub-graph of the I-GRAPH is used during the response determination process. This algorithm is approximate in that it does not take into account the possibility that two OR/AND edges leading to a child node may be dependent. Such information is not available in the I-GRAPH and therefore is not available to the GEN_PROB algorithm either.

In view of the restriction on the search space to the Domain Graph, we restate the optimality criterion. Assuming an attack snapshot P, an attack domain D, the response repository R, and the set of deployed response $RC_{deployed}$, an optimal assignment of responses is a selection $RC_{opt} \in (R-RC_{deployed})$ =

$$\arg\min_{RC_i} |Iv(RC_i \cup RC_{deployed})| + E[|Iv(\text{nodes in } (D-P) \text{ that will be achieved by the adversary with } RC_i \cup RC_{deployed} \text{ in place})|] + E[|Iv(\text{nodes in P that will be recovered by the responses } RC_i\ |] \qquad (3)$$

2.8 Attack Template Library (ATL)

ADEPTS seeks to adapt its responses based on previous attack scenarios. 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 is a directed graph where each node corresponds to an attack snapshot of an attack scenario. An edge, say from X to Y, represents the evolution of the attack scenario from snapshot X to snapshot Y. Responses, if any, that were deployed between the two snapshots are associated with the ATL edge. Figure 4 shows an example ATL.

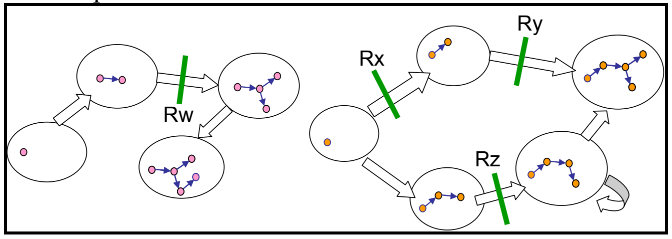


Figure 4. Example of Attack Template Library (ATL). Here two different attack scenarios are shown with 5 and 4 attack snapshots. 4 responses have been deployed.

3 Design of Search Algorithms

3.1 Maintenance of Attack Template Library

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* (more detailed information in Section 3.2); *s.best_genes*: the previous responses used by ADEPTS for snapshot s.

<u>Matching and Creation of a New Snapshot</u>

Without loss of generality, let us assume that the current snapshot in the attack is s_0. The detection framework observes activity in the ongoing attack, which translates into a subgraph g_{new} in the I-GRAPH being flagged with respect to the new changes. A new snapshot s_1 is created with $s_1.g = s_0.g \cup g_{new}$.

Now, ADEPTS checks in the ATL if there is an existing s_x such that $s_x.g = s_1.g$. If there is, s_1 is discarded and s_x is made the current snapshot for the attack. A directed edge from s_0 to s_x is added if it did not exist in the ATL. Otherwise, if such a s_x does not exist, the new node s_1 is added to the ATL with $s_1.best\_genes$ = NULL and $s_1.predict$ = 0. A edge from s_0 to s_1 is added.

<u>Deletion of Attack Snapshot</u>

If space is a constraint, ADEPTS deletes snapshots from the ATL by various criteria-by time of creation or time of last access (the oldest is deleted), frequency of access, or the snapshot with the lowest cumulative Iv of its nodes.

3.2 Attack Snapshot Prediction Table

Given an attack snapshot *s*, while there are an exponentially many possible next snapshot for the attack, in practice, some are much more likely. It would be useful to estimate the possible next snapshots for deploying the proactive containment responses. For tracking this likelihood, ADEPTS maintains a prediction table *s.predict* for each snapshot. The table entry *s.predict[e]* tracks the number of traversals of the edge *e* in the I-GRAPH following the snapshot *s*.

Now we wish to calculate the edge propagation probability for two edges in the I-GRAPH.

For an edge *e* which connects node *a* to node *b* in the I-GRAPH, the edge propagation probability *epp(a,b)* (as used in GEN_PROB(n,s,I,r)), can be calculated from *s.predict* as follows:

$$epp(a,b) = epp(e) = \frac{\tan^{-1}\left(\frac{\text{s.predict[e]} + d_{Bias}}{d_{Max} + d_{Bias}} \times d_{Scale}\right)}{\pi/2}$$

$$d_{Max} = \max_{e \in edges[s.g]} s.predict[e]$$

d_{Scale} and d_{Bias} are tunable values. Till d_{Max} becomes comparable to d_{Bias}, the epp value is relatively insensitive in discriminating between the edges. If the predict values are small, then d_{Scale} is used to scale the *epp* value to close to 1 so that the range of the function becomes (0, 1). ADEPTS uses values 9 and 10 respectively for d_{Scale} and d_{Bias}.

The reason for the above function for *epp* is that it converts large values (*s.predict[e]*) to a small range and using the factor in the denominator epp is constrained to be ∈ (0, 1).

3.3 Genetic Algorithm Framework

As the problem of deciding the optimal responses for an attack snapshot *s* has been proved to be a NP-Complete problem, we focus on an approximation solution using a GA framework. In this framework, we have an instantiation of the general function $f(s_k,H)$ as Respond(s_{k-1}, ASG). Assume the attack has undergone k snapshots together with k responses from the system as $\{s_1 \rightarrow f(s_1,H_1) \rightarrow s_2 \rightarrow f(s_2,H_2) \ldots s_k \rightarrow f(s_k,H_k)\}$.

Within this framework, we map each response combination onto a gene, and the problem of searching for the best response for an attack snapshot is then translated into looking for the best gene from the gene pool over multiple evolutions. Often using genetic algorithm to perform optimization is an expensive process [23] due to search through a huge gene pool over many evolution cycles to get a good solution. Our framework reduces the execution time in two ways. First, ADEPTS relies on the history information from the snapshot, namely s_k.best_genes. Second, ADEPTS relies on the information from similar attacks, namely s_x.best_genes for $s_x.g \approx s_k.g$. These are added to the initial gene pool to speed the convergence of the GA. The basic execution flow of the GA framework is shown below. Here ADEPTS is trying to determine the optimal response combination at snapshot s_k.

```
/*
R: response repository; R_deployed: deployed responses; ATL:
attack template library; gene_pool_size: a constant on the
gene pool size; v% : the percentage of top genes to be kept in
the history; max_evolutions: maximum number of evolutions
per iteration for the GA
*/
Respond($s_k$, $R_{deployed}$)
{
  Create domain graph D=D($s_k$);
  pool = GA_PopulateGenePool (ATL, $s_k$, D, gene_pool_size);
        /* defined in 3.3.3 */

  for i=1 to max_evolutions
      pool = GA_NextGeneration(pool); /* through operations
            defined in 3.3.4 */
      best_genes $\in$ {the top v% of genes in pool (with respect
                  to the fitness metric on gene)};
      $s_k$.best_genes = the top x% of genes from
                  ($s_k$.best_genes $\cup$ best_genes); //
                  updating history and x% keeps size of
                  $s_k$.best_genes constant

  Find gene c $\in$ best_genes with the highest fitness;
  Return(response combination RC corresponding to gene c);
}
```

3.3.1 Relation of gene to response combination

Here we describe how to create the genes prior to calling Respond(s_k, $R_{deployed}$). We only consider responses within the domain graph which are not deployed yet. This set of applicable responses is given by

$$R_A = \left\{ r \middle| r \in \left(R - R_{deployed}\right) \cap \left(D(s_k).resp\right) \right\} \qquad (4)$$

Here $D(s_k).resp$ is the set of responses on the nodes or edges of domain graph $D(s_k)$. The gene with respect to the this run of Respond(.) uses the encoding scheme such that each gene *c* is an R_A-bit vector, with each bit uniquely mapped to a response $r \in R_A$.

3.3.2 Definitions: Fitness and Similarity

The fitness of a gene *c*, is determined by the response combination *RC* for *c*. The fitness of gene *c* is defined as

$$fitness(c) = exp(exp(1/exp(Cost(RC)/S))) \qquad (5)$$

where *S* is a scaling factor. The curious function satisfies some desirable properties – high cost translates to low fitness, cost of zero or infinity are handled, and the outermost exponentiation has the desired effect of spreading out the range of fitness which is needed for the GA to discriminate between good and bad responses.

Given two attack snapshot s_a and s_b, the similarity metric to compare the two is defined as in

$$\text{Similarity}(s_a,s_b) = \frac{(\#\text{common nodes in } s_a.g \text{ and } s_b.g) + (\#\text{common edges in } s_a.g \text{ and } s_b.g)}{\#\text{ nodes and edges in } s_a.g \cup s_b.g} \qquad (6)$$

The range of Similarity(s_a,s_b) is between 0 and 1.

3.3.3 Populating the Gene Pool

The gene pool is populated through the following algorithm. As mentioned earlier, we use both the history information corresponding to the attack snapshot s_k and the history information from similar snapshots to increase the convergence speed of the GA algorithm. Note that the initial pool includes the greedy responses that would have been chosen by baseline ADEPTS. Choosing genes from a similar snapshot enables ADEPTS to respond to attack variants.

```
GA_PopulateGenePool (ATL, $s_k$, D, gene_pool_size) {
  pool_sec1 = {genes with the corresponding binary
              encodings filled with random bit stream};
  pool_sec2 = $s_k$.best_genes;
  pool_sec3 = $\bigcup$ $s_x$.best_genes for $s_x \in$ ATL with
              Similarity($s_x$,$s_k$) $\geq 0.7$;
  pool_sec4 = Genes corresponding to selection of
              responses by baseline ADEPTS;
  return the top gene_pool_size genes from
  {pool_sec1 $\cup$ pool_sec2 $\cup$ pool_sec3 $\cup$ pool_sec4};
}
```

3.3.4 Evolution from One Generation to the Next

We employ the standard GA algorithm here for generating the next generation gene pool from the current one. The standard procedures include three steps:

(i) *c = crossover(pool)*, which is to generate a new child from two parents randomly picked from current gene pool with probability of each parent being chosen proportional to the fitness values.
(ii) Mutate(*c*), which is to incur mutation onto the child gene *c* by flipping bits in the bit vector with a given probability which is typically kept very low.
(iii) Elitism, which is to keep the top *x*% of genes from the previous generation in the pool for the new generation.

4 Experimental Testbed

The experimental testbed deployed for evaluating ADEPTS is an e-commerce system, where users interact through a web browser with a three-tier server structure. First, a load balancer distributes the incoming traffic to a pair of Apache Tomcat web servers. Two JBoss application servers hold the J2EE application, one running as the application controller and the other as the component repository. The application is Sun Microsystem's Java Pet Store (version 1.4). In the backend, a MySQL database server runs as a repository of information, including customer accounts, product catalog and inventory, and order history. Figure 5 shows the testbed used for the experiments.

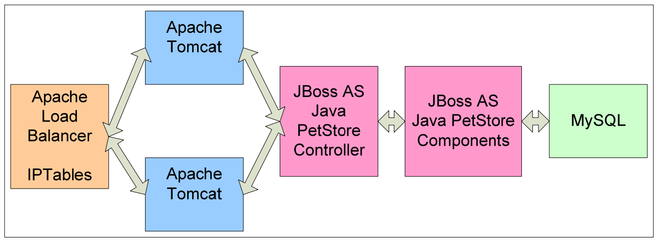

Apache Load Balancer
IPTables
Apache Tomcat
Apache Tomcat
JBoss AS Java PetStore Controller
JBoss AS Java PetStore Components
MySQL

Figure 5. Layout of three-tier e-commerce testbed for ADEPTS. Each box runs on a separate host. (AS: Application Server, Tomcat: Web Server)

The testbed emulates the common features of many service-oriented e-commerce systems. The Pet Store application provides a separate web interface for administrators, regular users and suppliers. We deployed many common TCP/IP services, such as FTP, SNMP, SSH and VNC, on these boxes so that they contribute vulnerabilities and therefore enable attack scenarios. Also, common configuration errors were induced, such as sharing of user accounts and passwords among many hosts or using weaker security policies on the internal network, as compared to access from external network. The objective was to replicate the complexity and lack of strong security policies often found in real e-commerce systems.

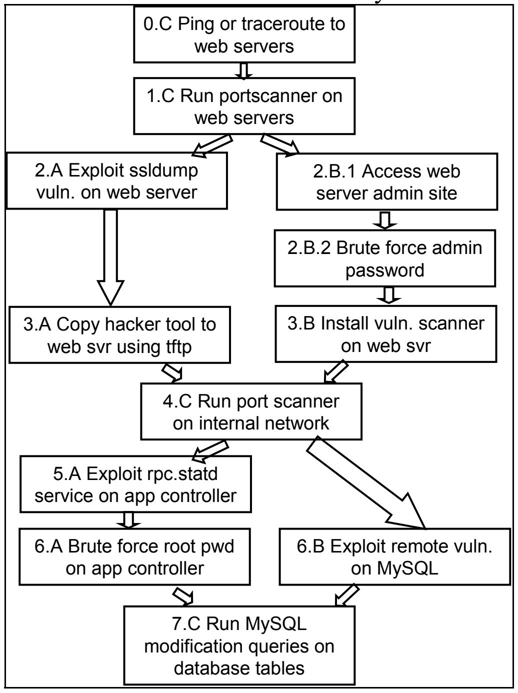

0.C Ping or traceroute to web servers
1.C Run portscanner on web servers
2.A Exploit ssldump vuln. on web server
2.B.1 Access web server admin site
2.B.2 Brute force admin password
3.A Copy hacker tool to web svr using tftp
3.B Install vuln. scanner on web svr
4.C Run port scanner on internal network
5.A Exploit rpc.statd service on app controller
6.A Brute force root pwd on app controller
6.B Exploit remote vuln. on MySQL
7.C Run MySQL modification queries on database tables

Figure 6. Attack scenarios 3 and 4, used for experimental evaluation. Boxes with A and B denote the stages for scenario 3 and 4 respectively, while C denotes stages common to both.

The ADEPTS implementation is tested against a set of attack scenarios based on popular vulnerabilities published by the electronic payment industry [24], the web security community [25], and in the CVE dictionary [26]. Six attack scenarios were developed to extensively test ADEPTS. Each attack scenario is made up a set of individual stages that could be run independently or mixed to form dynamic attack scenarios. Dynamic attack scenarios mimic the real-world attacks where an attacker might have several options to continue the intrusion after reaching a certain intermediate goal, or has to choose alternate stages since a previous one failed. For each service there are multiple detectors, but these are imperfect. This is simulated by artificially generating false alarms or suppressing alerts according to a given rate.

Figure 6 shows attack scenarios 3 and 4 both of which have the end goal of corrupting or leaking information from the database. These scenarios share several stages in common and can thus be merged to form a dynamic attack scenario.

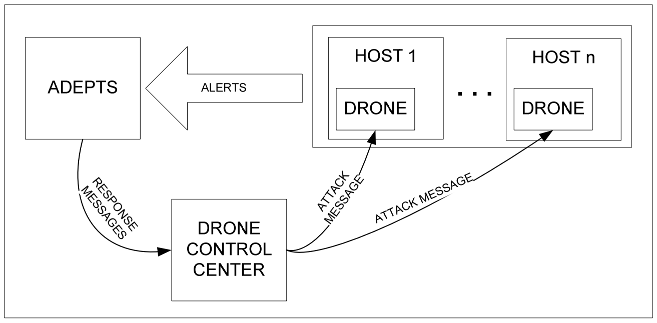


Figure 7. A drone-based implementation for attack scenarios in the e-commerce testbed. A drone allows for the manifestation of an attack stage without needing to exploit the vulnerability.

To implement the attack scenarios, a java-based drone was implemented on each host (Figure 7). A drone shows the manifestation of a successful attack stage without having to go through the exploit to achieve this. It thus provides an easy tool for creating different attack scenarios. A drone control center was also developed to centrally control each drone and launch an attack scenario by sending commands to drones on individual hosts. It can vary parameters like time between attack stages and probability of success between stages. Each attack stage could trigger an alarm on the corresponding drone, depending on the false alarm/missed alarm rates defined for the detector. On receiving an alert from a drone, ADEPTS calculates a set of responses according to the algorithms described in Section 3. The resulting set of responses is sent to the control center.

The I-GRAPH is generated manually based on the attack scenarios covering all the attack goals. A subset of the nodes have associated detectors. The detectors used are (i) Snort: detects attack patterns in network traffic; (ii) Libsafe: detects buffer overflow attacks of protected C library functions; (iii) Process monitor: monitors unauthorized process invocation and execution based on a specified white-list; (iv) File access monitor: monitors and compares file access attempts of selected processes against preset rules; (v) Password brute-force detector: detects failed authentication attempts; (vi) EJB monitor: separate EJB that monitors other running EJBs in the same process space and flags alerts when unexpected behavior is observed (e.g. termination of an essential EJB). The responses specified for each node or edge in the I-GRAPH are manually selected from the response repository in [13]. The impact vectors for these responses are created based on the effect the responses have on the e-commerce testbed. The I-GRAPH has 55 nodes, 96 edges, 5 nodes with no detectors, and 72 responses. The max, min, and average in-degree and out-degree are (7, 0, 1.7) and (5, 0, 1.7).

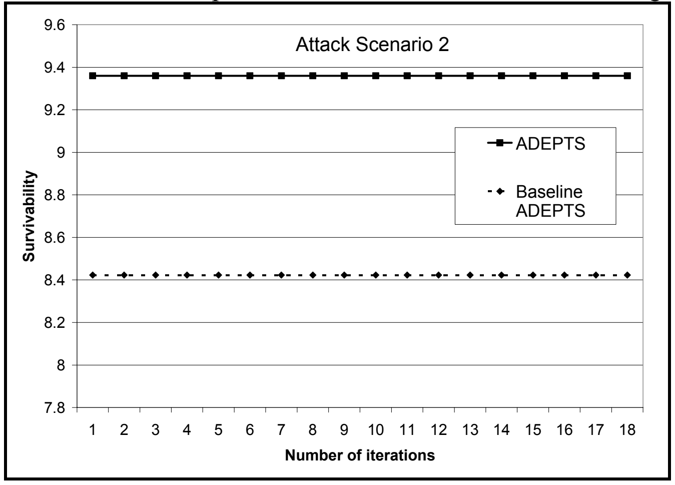
It may be argued that a more realistic experiment would be to actually exploit the vulnerabilities, and deploy the detectors and the responses on the hosts. However, this would restrict the universe of attack scenarios we can try since vulnerabilities are often quite quickly patched in production systems such as ours. Also it would make it exceedingly difficult to vary experimental parameters such as missed and false alarm rates from the detectors and success rate of the responses. Ultimately this would not bring out the strengths and weaknesses of the proposed algorithms in an experimentally rigorous manner. The presentation of overall ADEPTS capabilities with real vulnerabilities being exploited and actual detectors put in place is left for a follow-on paper.

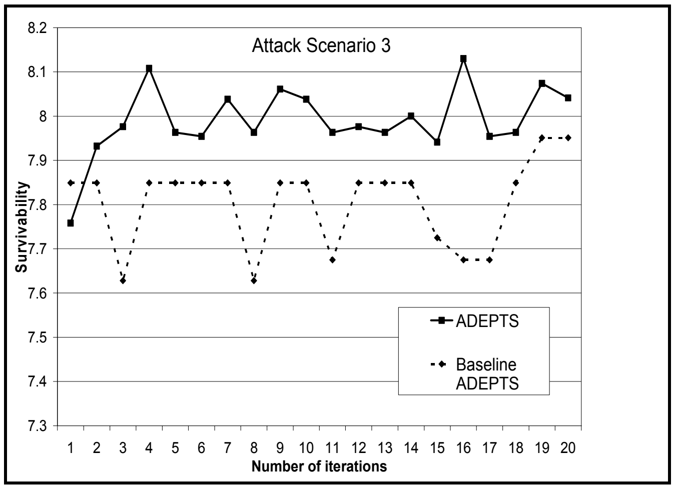
5 Results

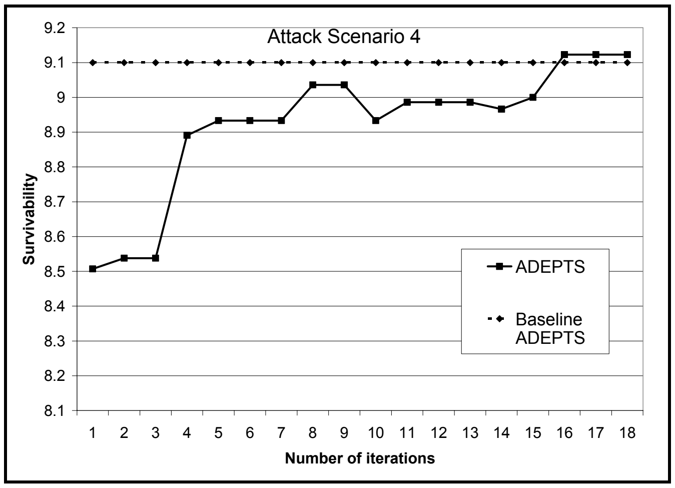
The output metric used in the experiments is survivability. It is qualitatively meant to capture the value of the system to the owner in terms of the transactions that can be supported and the system goals that are met when the attack and the responses have occurred. Quantitatively, it is given by:

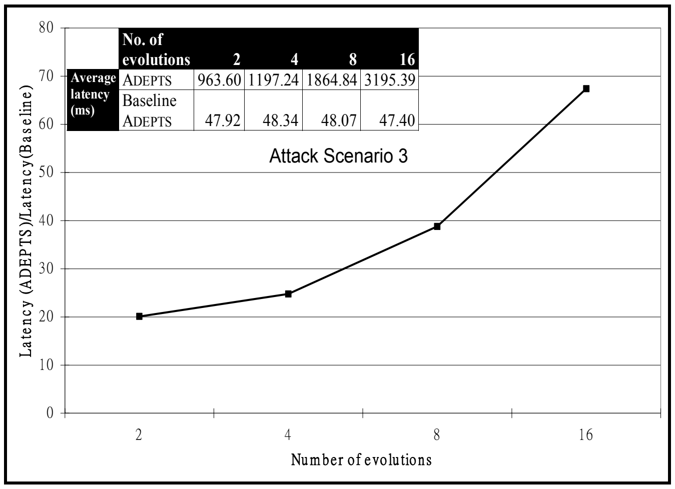
$$C - \sum_{i=1}^{m+n} Iv[i] \qquad (7)$$

Where C is a scaling constant representing the perfect survivability and $Iv = \sum Iv$(deployed responses)$+\sum Iv$(achieved I-GRAPH nodes). The dimension of Iv for PetStore is 24, divided equally between transactions and system goals. $Iv[i]$ denotes the i^{th} dimension. The other relevant metric is latency, which is measured from the time an alert arrives at ADEPTS to when the response is communicated to the drone control center. Thus, it does not include the time to actually deploy the response, which is justified since that is a characteristic of the response and not ADEPTS' algorithm.









	No. of evolutions	2	4	8	16
Average latency (ms)	ADEPTS	963.60	1197.24	1864.84	3195.39
	Baseline ADEPTS	47.92	48.34	48.07	47.40

Figure 8. Results from attack injection (a)-(c) show the performance of ADEPTS relative to baseline ADEPTS for three different attack scenarios (Experiment 1). (a) has perfect choice of initial responses, (c) mimics inaccurate initial settings by

the sysadmin. In all the survivability of ADEPTS outperforms that of baseline ADEPTS. (d) shows the increased latency in ADEPTS due mainly to the multiple generations of the genetic algorithm (Experiment 2)

In the interest of space, here we provide the results of injecting two attack scenarios—scenarios 2, 3, and 4 (Figure 6). These scenarios are complex, multi-stage, and touch all three tiers of the e-commerce system. Additionally they share some stages and therefore can be used to test the ability of ADEPTS to learn from attack variants. The perfect survivability value (constant *C* in Eqn. (7)) is 10.

For experiment 1 (Figure 8(a)-(c)) we compared survivability between ADEPTS and baseline ADEPTS, by running them against attack scenarios 2, 3, and 4. Each attack scenario was executed multiple times (# iterations on the plots) and history was cleared for each attack scenario at the start of the experiment. Thus the performance of ADEPTS at the beginning is dependent on default EI values for the responses. The GA runs two evolutions per iteration. For attack scenario 3 (AS3), ADEPTS consistently performed better than baseline ADEPTS, except for the very first iteration. ADEPTS is more pessimistic at the beginning, by considering the possibility of responses failing and therefore deploying proactive responses. However, later ADEPTS updates the EI values based on observed performance of the responses and therefore outperforms baseline ADEPTS. For AS2, both ADEPTS and baseline ADEPTS provided a set of responses that do not change over the iterations. This is due to the fact that the responses chosen at the outset were close to perfect. Still ADEPTS came up with a better set of responses due to its ability to consider the implication of the response over the entire domain graph rather than the greedy approach of baseline ADEPTS. This allows further propagation of the attack since it is determined that the cost of preventing is higher than the cost incurred if the node is achieved. As opposed to this, the limited responses provided by baseline ADEPTS cannot make this decision.

For AS4, ADEPTS evolves over time from a bad starting point. Baseline ADEPTS performs better over several iterations until ADEPTS gains enough history (iteration 16). The case here is of poor assignment of initial EI values, say by an inexperienced sysadmin. The experiment demonstrates that ADEPTS is robust to such errors since with growing history it relies less and less on the default initial assignments.

Experiment 2 (Figure 8(d)) is a comparison of the latency of response determination between ADEPTS and baseline ADEPTS. For this experiment, AS3 is run for different number of evolutions per iteration and for each run the history is cleared. The primary contributor to the latency is the processing of the GA as it processes through multiple generations. Therefore, the number of generations is kept as the control parameter. Results shows a higher latency for ADEPTS, as compared to that from baseline ADEPTS by as much as a factor of 67 at 16 evolutions per iteration. However, the absolute value for ADEPTS is less than 3.2 seconds for the highest value of 16 evolutions per iteration. The latency can be partially hidden by performing the calculation of gene pool generations offline, when no attack is occurring. There is a tradeoff between the number of iterations and the number of evolutions per iteration that the GA may run. With more iterations, the EI values will be updated more accurately, while more evolutions per iteration will make the GA perform better but take more time per iteration. The optimal value depends on the particular attack scenario and further exploration of this tradeoff is needed.

Experiment 3 (Figure 9) explores the ability of ADEPTS to learn from attack scenario variants. This experiment is run with AS4. For the first case, AS4 is run without having run its variant AS3 before. For the second case, AS3 is run for 20 iterations. Then, the algorithms can use the previous ATL containing the snapshots of the attack stages and the best genes for AS3, and the EI values for responses. We observe that AS4 with history outperforms AS4 without history until about the 8th iteration where their performances converge. This implies that the 8th iteration of AS4 with history corresponds to the 20th iteration of its variant, namely AS3. If the variants are closer (in terms of the similarity metric, say), then the point at which the two cases will converge will be higher.

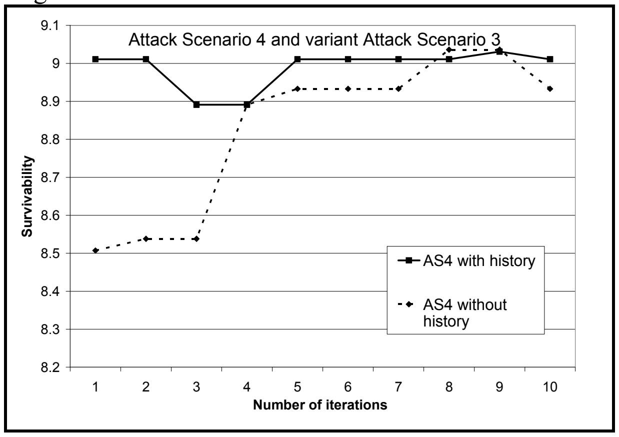


Figure 9. Experiment for evaluating adaptation capability of ADEPTS to learn from attack variant. For one case, AS4 is run after running its variant AS3 and generating history. For the other, AS4 is run without such history.

6 Discussions

This paper has presented the algorithm in ADEPTS to decide on optimal responses. Several other aspects of an IRS are needed to support the presented algorithm, but they cannot all be described in the confines of this paper. Some of these aspects have been described in other publications, such as, diagnosis of the node(s) likely to have been achieved, creation of I-GRAPH, populating the response repository, and updating the effectiveness of responses in [12], and tolerating imperfect detectors and handling unknown alerts in [13]. Some other aspects are under investigation, such as handling unanticipated attacks and concurrent attacks.

Several design decisions described here lend themselves to further experimentation and refinement. For the recovery oriented responses, ADEPTS resets the CCI of the node at which the response is deployed. However, some attacks rely on a sequence of nodes being achieved, e.g., a Trojan which relies on a continuous network connection to leak information, and thus a response on one node may reset the CCI of a chain of nodes. The I-GRAPH structure used here is static and is implicitly assumed to be complete. However, in the face of unanticipated attacks it would be imperative to grow the I-GRAPH. A candidate approach would be using machine learning mechanisms that create nodes for alerts and edges between correlated alerts. A challenging issue with observing events in any distributed system is that the order in which they are observed may not be the order in which they have occurred. This is due to the asynchronous nature of the communication medium and uncertain delays at the computational nodes. Thus, corresponding to a given attack scenario, alerts may be observed in different orders. The ability of ADEPTS to respond to attack variants can handle the reordering to some extent depending on the similarity value of the match. However, it may be required to reorder alerts based on alert-specific attributes and domain knowledge of causality to effectively respond to the attack. Here we have seen a level of dependency between responses. Yet another level of non-determinism is introduced by concurrent attacks since the response to one attack may suffice to contain both attacks. The presented framework can be extended to discriminate between distinct attacks as in [11] and handle them at the expense of expanding the GA search space. The expansion factor will be the number of concurrent attacks. A common drawback for a solution that relies on history of attacks is that it is unable to handle a hitherto unseen attack of devastating impact. For ADEPTS, history helps the GA to converge faster but is not strictly necessary. The EI values will be less calibrated and the GA has to run longer to arrive at an acceptable solution. By setting the Iv of a node to a suitably high value, ADEPTS will deploy a response, even if drastic, to prevent the node from being achieved.

7 Related Research

With increasing complexity and ubiquity of distributed systems, IRSs for such systems have been gaining interest. The general principles followed in the development of the IRS naturally classify them into four categories.

1. *Static decision making*. This class of IRS provides a static mapping of the alert from the detector to the response that is to be deployed. The IRS includes basically a look-up table where the administrator has anticipated all alerts possible in the system and an expert indicated responses to take for each. The systems in [14]-[16] fall in this category.
2. *Dynamic decision making*. This class of IRS reasons about an ongoing attack based on the observed alerts and determines an appropriate response to take. The first step in the reasoning process is to determine which services in the system are likely affected, taking into account the characteristics of the detector, the network topology, etc. The actual choice of the response is then taken dependent on a host of factors, such as, the amount of evidence about the attack, the severity of the response, etc. The third step is to determine the effectiveness of the deployed response to decide if further responses are required for the current attack or to modify the measure of effectiveness of the deployed response to guide future choices. A wide variety is discernible in this class based on the sophistication of the algorithms. The systems in [4]-[9], including ADEPTS, fall in this category.
3. *Intrusion tolerance through diverse replicas*. This class of IRS implicitly provides the response to an attack by masking the effect of the response. The basic approach is to employ a diverse set of replicas to implement any given service. The fault model is that the replicas are unlikely to share the same vulnerabilities and therefore not all will be compromised by any given attack. An advantage of this approach is the system can continue operation without a disruption as in the active replication technique. The systems in [17]-[19] fall in this category.
4. *Responses to specific kinds of attacks*. This class of IRS is customized to respond to specific kinds of attacks, most commonly, distributed denial of service (DDoS) attacks. The approach is to trace back as close to the source of the attack as possible and then limit the amount of resources available to the potentially adversarial network flows. The system reported in [10] fall in this category.

The concept of survivability was pioneered by SEI at CMU. It 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 ([2],[3]). 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. The part of the ADEPTS system presented in this paper provides the second and the fourth properties.

The work presented here differs from previous IRS work in that it lays down a framework to reason about the optimality of the response choices made by these systems. The approach here can be applied to evaluate any available IRS. Our previous work with the ADEPTS system also did not have any design to choose globally optimal responses.

There have been some efforts at using genetic algorithms for intrusion detection [21]-[23] and search for vulnerabilities [20]. The results have been promising, but only after careful definition of the syntax of the chromosomes and tuning of the fitness measure of the chromosomes. We have not found any prior application of GA to intrusion response systems.

8 Conclusion

In this paper, we introduced the notion of optimality of responses deployed by an intrusion response system. We developed a framework for reasoning about optimality of responses deployed on a growing set of attack snapshots for a multi-stage attack. A genetic algorithm based search was proposed to search for the optimal response set. The chromosomes for the initial gene pool and the carry over from one generation to the next are designed to guarantee the solution is better than the locally optimal response selection done by the baseline ADEPTS IRS. The claims were experimentally validated on a three tier e-commerce system through injection of multi-stage attacks.

9 References

[1] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi, "Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS", *ACSAC* 2003.

[2] R. Ellison, R. Linger, T. Longstaff, and N. Mead, "Case Study in Survivable Network System Analysis", *Technical Report CMU/SEI-98-TR-014*, SEI, CMU, 1998.

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

[4] T. Toth and C. Kruegel, "Evaluating the Impact of Automated Intrusion Response Mechanisms", *ACSAC* 2002.

[5] G. White, E. Fisch, and U. Pooch, "Cooperating Security Managers: A Peer-based Intrusion Detection System", *IEEE Network*, vol 10, no. 1, 1996, pp. 20-23.

[6] P. Porras and P. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *NISSC*, pp. 353-365, 1997.

[7] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch, "Adaptation Techniques for Intrusion Detection and Intrusion Response Systems", *Int. Conf. on Systems, Man, and Cybernetics*, pp. 2344-2349, 2000.

[8] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using specification-based intrusion detection for automated response," *RAID*, pp. 136–154, 2003.

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

[10] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, and T. Reid, "Autonomic Response to Distributed Denial of Service Attacks", *RAID* 2001.

[11] C. Carver, J. Hill, and U. Pooch, "Limiting Uncertainty in Intrusion Response", *IEEE Workshop on Info. Assurance and Security*, 2001.

[12] B. Foo, Y-S. Wu, Y-C. Mao, S. Bagchi, and E. H. Spafford, "ADEPTS: Adaptive Intrusion Response using Attack Graphs in an E-Commerce Environment," DSN, pp. 508-517, 2005.

[13] Y-S. Wu, B. Foo, Y-C. Mao, S. Bagchi, and E. H. Spafford, "Automated Adaptive Intrusion Containment in Systems of Interacting Services," In Elsevier Journal on Computer Networks (in press), Spring 2007.

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

[15] Symantec Corp., "Norton Antivirus."

[16] T. Ryutov, C. Neuman, K. Dongho, and Z. Li, "Integrated access control and intrusion detection for Web Servers," ICDCS, pp. 394-401, 2003.

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

[18] C. Cachin, "Distributing trust on the Internet," DSN, pp. 183-192, 2001.

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

[20] Jacobs, S., D. Dumas, W. Booth, M. Little, "Security Architecture for Intelligent Agent Based Vulnerability Analysis," 3rd Annual Fedlab Symposium on Advanced Telecommunications/Information Distribution Research Program, pp. 447-451, February 1999.

[21] W. A. Jansen, "Intrusion detection with mobile agents," Computer Communications, Volume 25, Issue 15, pp. 1392-1401, 2002.

[22] G. Helmer, J. Wong, V. Honavar and L. Miller, "Automated discovery of concise predictive rules for intrusion detection," Journal of Systems and Software, Volume 60, Issue 3, pp. 165-175, 2002.

[23] Ludovic Me, "GASSATA: A genetic algorithm as an alternative tool for security audit trails analysis," RAID '98.

[24] PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard. Version 1.1. http://pcisecuritystandards.org.

[25] The Open Web Application Security Project. The Ten Most Critical Web Application Security Vulnerabilities. 2004, www.owasp.org.

[26] The MITRE Corporation. Common Vulnerabilities and Exposures. http://cve.mitre.org.