The Search for Efficiency in Automated Intrusion Response for Distributed Applications

Authors: Yu-Sung Wu (Purdue University)
Gaspar Modelo-Howard (Purdue University)
Bingrui Foo (Purdue University)
Saurabh Bagchi (Purdue University)
Eugene Spafford (Purdue University)

Contact Author: Yu-Sung Wu (E-mail: yswu@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. Earlier works [9], [10] have shed the light on automated responses. They pick the best local response that stops an attack propagation from its current step to the next step.

Here we propose a new approach where the optimality of responses is considered from a global point of view on "*What's the eventual outcome on the whole system from using a response?*". We formalize the process of providing automated responses and the criterion for asserting global optimality of the set of deployed 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 solutions. In real world, good solutions can change as the problem structure changes. Here the problem structure involves the protected target system and the attacks, both of which can change over time. Our framework constantly adapts itself to the changing environment based on short term history and also tracks the patterns of attacks in a long-term history. We demonstrate the solution on a distributed e-commerce application called Pet Store with injection of real attacks and show that it improves the survivability of the system over previous works [9], [10].

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

Submission Category: Regular Paper

1 Introduction

Distributed systems comprising multiple services interacting among themselves to provide end-user functions are becoming an increasingly important platform. Many of the platforms, such as distributed e-commerce systems, have huge financial stakes involved in them. This 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, anti-virus software can restrict access to virus infected files. Also, most modern web browsers can automatically block connections to potential phishing websites. These are applicable in stand-alone systems due to their simplicity. For distributed systems with nearly exponentially large number of interaction effects among multiple components, pre-configuring these static pairs of detector alarm and response is laborious and can be shown to have inferior runtime performance due to the dynamic workload on the system and the changing nature of attacks.

The few available dedicated IRSs (Intrusion Response Systems) for distributed systems [1]-[6] 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 earlier work in a system called ADEPTS [9][10], 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. We address the problem of optimal response selection in this paper.

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) [1][9].

We present a framework called SWIFT 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 framework 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 [9], [10] 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. 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. This is fundamentally because of the dependencies that exist between responses and between services in a distributed system. 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. To boost the quality of the populated responses, the approximate solution can use history of the attacks seen in the system and the paths they have taken, and estimates of the effectiveness of responses deployed earlier so that the search space can be restricted. To solve the approximate problem, we use genetic algorithm (GA) [23] 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 demonstrated on a distributed three tier e-commerce application called Pet Store that uses the J2EE platform. 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 are deployed. The experiments show the improved survivability with SWIFT compared to baseline ADEPTS, the ability of SWIFT to adapt its responses as increasing numbers of attack instances are seen, and its ability to handle attack variants.

The rest of the paper is organized as follows. Section 2 presents the design of the optimality framework. 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 Global Optimal Response

2.1 Adversary and Attack Model

A representation called Intrusion Graph (I-GRAPH) [9] is used for modeling the spread of the attack, which is similar in concept to attack graphs [25]. The final goal of the adversary may be disrupting some high level system functionality, such as "Permanently change grades record" in Figure 1. This final goal is achieved through multiple intermediate intrusion goals (attack steps) and each is represented as an I-GRAPH node.

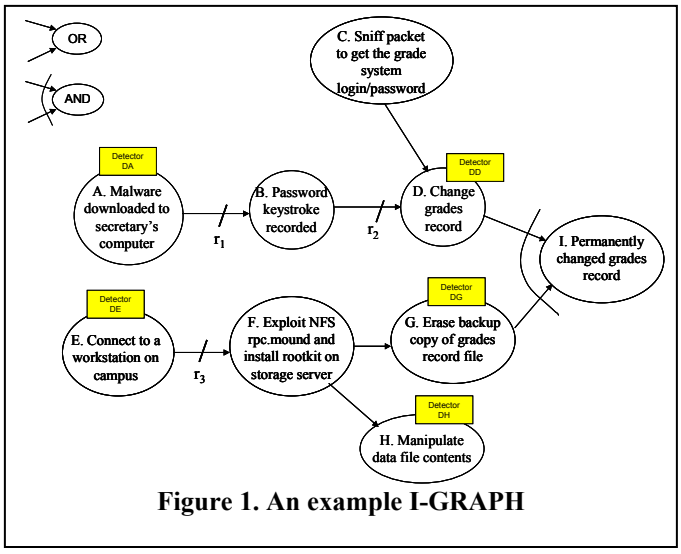


Figure 1. An example I-GRAPH

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

N := {nodes in G} := {{N_A: specific attack manifestation} ∪ {N_B: generic attack manifestation} ∪ {N_C:high level parameterized manifestation} ∪ {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 [26],[28]. It is not 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 SWIFT's key roles to deal with the uncertainty by constantly adapting to the actual situation, to provide continuous protection to the system.

When alerts are received by SWIFT for a node in the I-GRAPH, SWIFT calculates a Compromised Confidence Index (CCI) value for all the nodes. This scheme is identical to that in baseline ADEPTS [9],[10]. Through this paper, we use the CCI as the estimate for the probability P(*n*) that a node *n* has been achieved. In general, P(*n*) is assumed to be provided by the underlying the I-GRAPH model. We can also use alternative choices such as Hidden Markov Model or Bayesian Network based attack graph [27] and the rest of the discussion will remain unchanged.

2.2 Response Model for Multi-Stage Attack

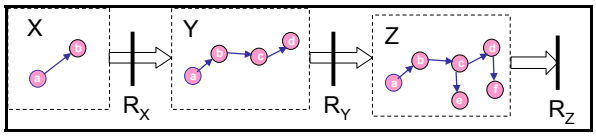


Figure 2. Three different snapshots for a given attack scenario. Responses R_X, R_Y, R_Z are deployed between snapshots

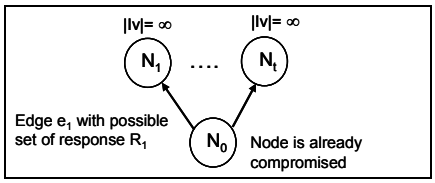
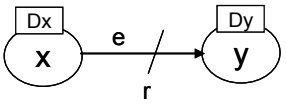


Figure 3. Transformation to map set covering problem to optimal response determination (ORD).

The Effectiveness Index (EI) of a response indicates the likelihood of success of the response, which is determined by SWIFT 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. For better understanding, Figure 4 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. Here detector alert values are assumed to be a real number between 0 and 1, with 1 indicating the full confidence from the detector that the corresponding node is compromised and with 0 indicated the node is not compromised. The CCI values are then used as the estimates of the probabilities of nodes being achieved P(*x*) and P(*y*). This mechanism is the same as in baseline ADEPTS . Further details are thus omitted here.

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 2 shows three snapshots X, Y, and Z of 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 SWIFT cannot deploy a response within a batch of alerts. This batch creates a snapshot.



$x.AV$: detector alert value from Dx

$y.AV$: detector alert value from Dy

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

$y.cci \leftarrow \left[x.cci \times e.EPF \times (1 - r.EI) + y.AV \right] / 2$

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

Figure 4. Effect of response and EPF in I-Graph inference.

Generally, for a multi-stage attack consisting of *k* snapshots $\{s_1, s_2, .. s_k\}$, the response mechanism is formally described by $RC_i = Respond(s_i, H_i)$, where s_i is the i^{th} snapshot, H_i is the history information and RC_i is the response combination decided by SWIFT. Therefore, in Figure 2, we have $R_X = Respond(s_X, H_X)$, $R_Y = Respond(s_Y, H_Y)$, and $R_Z = Respond(s_Z, H_Z)$.

2.3 Impact Vector Metric

We use a metric called *Impact Vector* for evaluating the favorableness of a response combination. Firstly, we assume that the protected target system has a set of transactions and security goals 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 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:

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

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. 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 [1]. 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.

2.4 Intractability of Optimal Response Determination

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 $n_1, n_2, .. n_m$. Now we want to evaluate the cost of the 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 $P(n_1)$, $P(n_2)$,…, $P(n_m)$. Then the cost of RC_i is defined by Eqn. (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.

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

$$RC_{i,opt} = \arg\min_{RC_i} cost(RC_i) \qquad (2)$$

Consider the small I-GRAPH in Figure 3. Let $\boldsymbol{E} = \{e_1, \ldots, e_t\}$. 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 $N_1, \ldots, N_t$ 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 [28]. 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 [29][30]. 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.

3 Design of Search Algorithms

The overall execution flow in SWIFT's search for optimal response combination is shown in Figure 5.

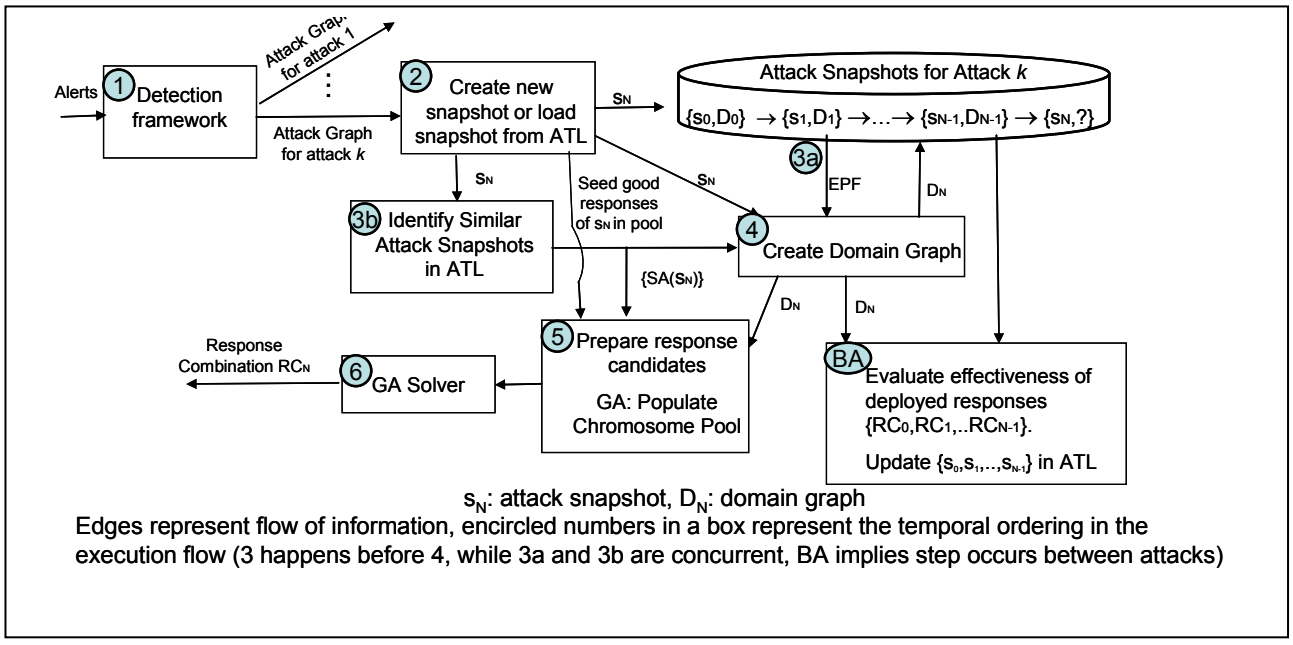


Figure 5. Overall flow for the steps in SWIFT to respond to an attack

3.1 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* (Sec. 3.2); *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) 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.

3.2 Attack Snapshot Prediction Table and Edge Propagation Factor (EPF) Tuning

Given an attack snapshot *s*, while there are many possible follow-on attack steps, in practice, some are much more likely. SWIFT attempts to estimate 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.

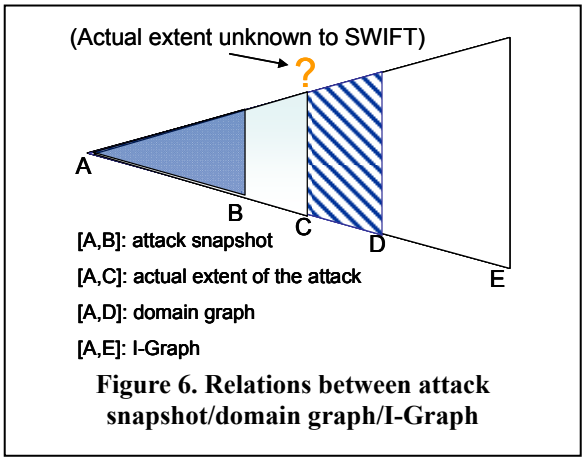


Figure 6. Relations between attack snapshot/domain graph/I-Graph

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* (Sec. 3.3), 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 4, 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*.

3.3 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 6). In Eq. (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 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 deployed.

Given the I-GRAPH *I* and a snapshot *s*, the Domain Graph $D(s) = (V, E)$ where $V = \{\{$node $n \in I$ such that $P(n) \times |n.Iv|$ is greater than a given threshold $T\} \cup \{$node $n \in I$ such that *n* is on the path from n_x to n_y in *I* where $n_x, n_y \in V\}\}$ and $E=\{e|e \in edges(I)$ *and e: (u, v)* , where $u, v \in V\}$. This is computed in step 4 of Figure 5.

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 (Sec. 3.2) 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 6)

3.4 Genetic Algorithm (GA)-based Response Mechanism

As the problem of deciding the optimal response combination for an attack snapshot has been proved to be NP-hard, we focus on an approximate solution using a GA framework [23]. Following Figure 5 step 6, this corresponds to designing a response mechanism `Respond(.)` (algorithm shown in Figure 7), 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 [20] 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), 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 greedy locally optimal responses from baseline ADEPTS 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) is then invoked to systematically probe through the space of response combination RC through the typical GA evolution process [23]. The high-level concept here is those response combinations that yield low cost values (Eq. (1)) will be returned by the GA Solver in the end.

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.
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 Sec. 3.4.
   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.

Figure 7. GA based response mechanism

4 Experimental Testbed

The experimental testbed deployed for evaluating SWIFT is an e-commerce system (Figure 8), where users interact through a web browser with a three-tier server structure. 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. The testbed emulates the common features of many service-oriented e-commerce systems, including mis-configurations and weak security policies on the networks. The SWIFT implementation is tested against a set of attack scenarios based on vulnerabilities published by the electronic payment industry [21], the web security community [22], and in the CVE dictionary [24]. The I-GRAPH has 55 nodes, 96 edges, 5 nodes with no detectors, and 72 responses (all containment focused responses). The max, min, and average in-degree and out-degree are (7, 0, 1.7) and (5, 0, 1.7). A subset of the nodes have associated detectors, such as Snort, Libsafe, Process and File access monitor, Brute-force password cracking detector, and EJB monitor.

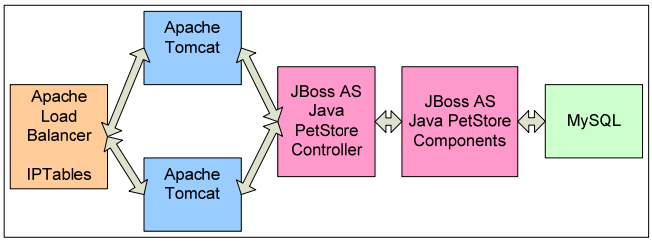


Figure 8. Layout of three-tier e-commerce testbed for SWIFT. Each box runs on a separate host. (AS: Application Server, Tomcat: Web Server)

5 Results

The output metric we choose for evaluation is survivability. Qualitatively it captures 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 are deployed. Quantitatively, it can be computed as :

$$\text{Survivability } s \;:=\; C - |\,Iv(RC_i)\,| \qquad (3)$$

C is a scaling constant representing the perfect survivability. Since *C* is application specific and in the absence of such an application-specified value it will represent an arbitrary scaling. Therefore, we use |*Iv*| instead as the indicator of performance. The reader should keep in mind that |*Iv*| and survivability have an inverse relation. Unless otherwise specified, SWIFT is executed with a chromosome pool of size 10 and 4 evolutions per snapshot.

5.1 Survivability for Micro-Benchmark

We consider as a micro-benchmark an attack scenario that has the form shown in Table 1. 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 baseline ADEPTS 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.

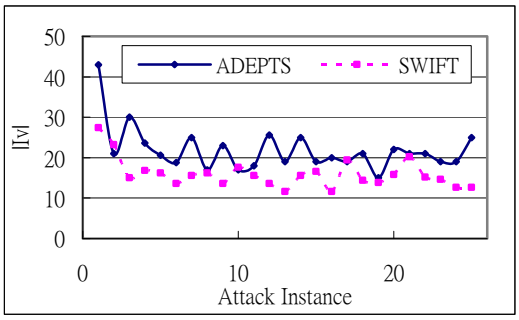


Figure 9. Improvement in Survivability with SWIFT for Micro-benchmark.

The survivability result from the experiment is shown in Figure 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. (2)). This is true even for the first attack instance where no history information is available as shown in Figure 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.

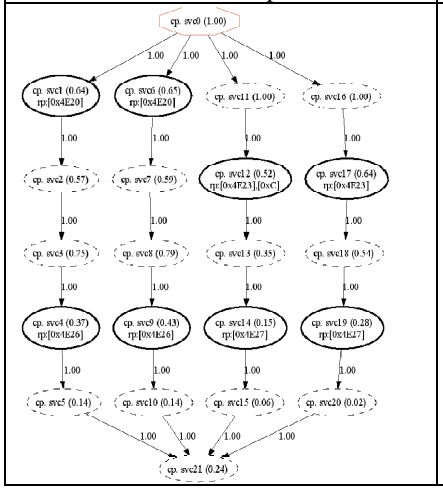
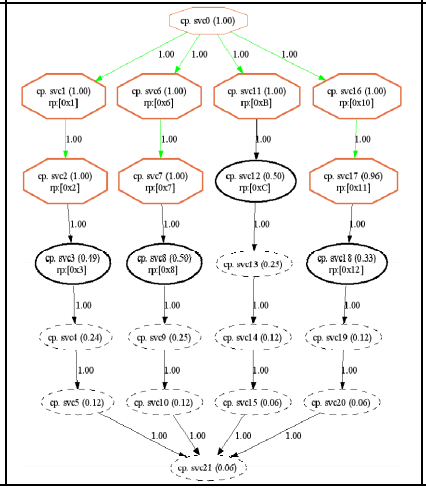
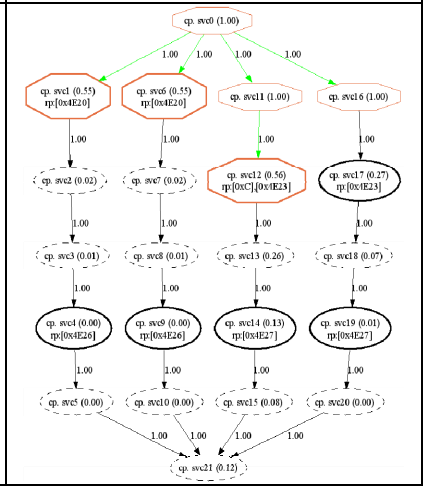
Attack Instance 24		
(a) SWIFT After attack snapshot 1	(b) Baseline ADEPTS (|Iv|=19) At the end of the attack	(C) SWIFT (|Iv|=12.6) At the end of the attack
		

Table 1. Detailed attack snapshots from attack instance 24

Table 1 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 and SWIFT respectively. The baseline 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.

5.2 Learning from History to Reduce Search Space Size

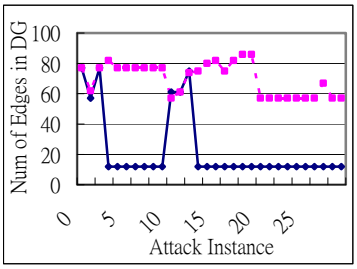


Figure 10. # of edges in the domain graph generated out of the 3rd snapshot.

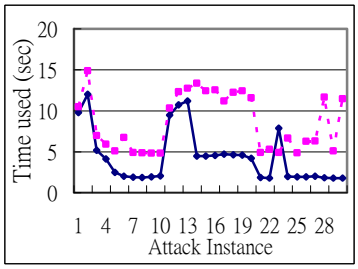


Figure 11. Time used by SWIFT in response decision

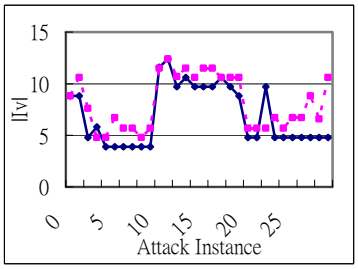
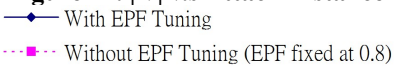


Figure 12. |*Iv*| v.s Attack Instance



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 (Sec. 3.2.). 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 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 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 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.

5.3 Survivability for Real Attack Scenarios

Figure 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 kind of useless for the baseline, 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 baseline and SWIFT. Figure 14(a) shows that the baseline's performance is widely fluctuating for AS3. This is primarily due to the fact that the baseline 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 the baseline at the time when N_9 is flagged. SWIFT consistently has lower $|Iv|$ than the baseline. This is due primarily to SWIFT's ability to redeem R_{60} through the fuzzy EI mechanism (Section 3.1) even though it had a

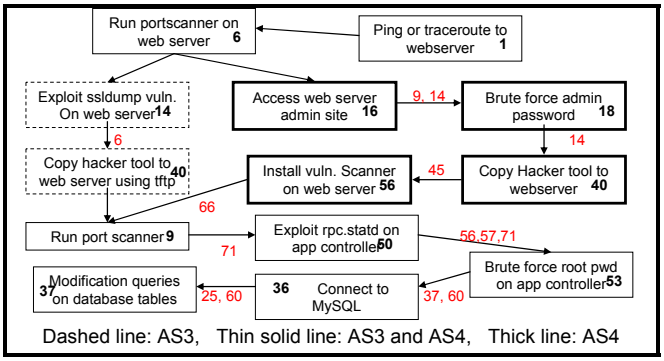


Figure 13 Attack scenarios 3 and 4 (AS3, AS4)

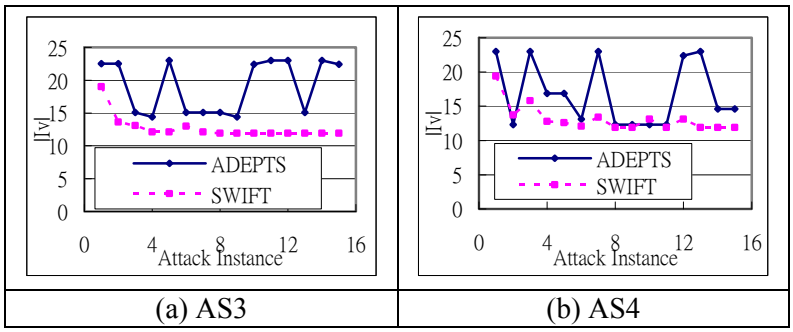


Figure 14. Experiment 3: Survivability for AS3 and AS4

low initial value. In the baseline system R_{60} is not considered till the EIs of the other responses also diminish to this low value. For AS4 (Figure 14 (b)), 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 the baseline does not suffer as much from underestimated response R_{60}.

5.4 Responding to Attack Variants

In this experiment we consider AS3 and AS4 to be variants of each other (due to their shared nodes as shown in Figure 13). The results are shown in Figure 15. 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 baseline system. This would be valuable in dealing with very destructive attacks when they are observed for the first time.

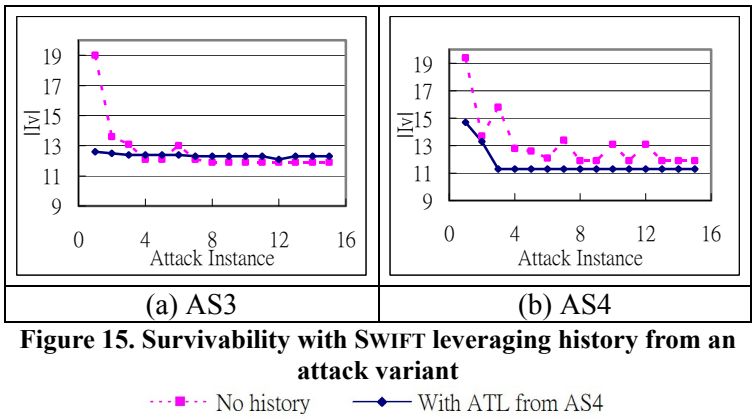


Figure 15. Survivability with SWIFT leveraging history from an attack variant

6 Discussions

This paper has presented the algorithms in SWIFT to decide on optimal responses. Several other aspects of an IRS are needed to support the presented algorithms, but they cannot all be described in the confines of this paper. Some aspects are under current investigation, and some have been presented in earlier work [9][10].

A question arises as to whether it is reasonable to assume an attack can occur multiple times such that the adaption process can work. First, we found that the adaption process in our experiments takes only 4-5 attack instances to get adequately adapted. We can consider the possibility of sharing attack history information across systems which share similar configurations (e.g. companies using the same three tier configuration), as this is commonly done with anti-virus signatures. Also, with automated multi-stage attacks, multiple instances of an attack are also seen for a single system. An administrator may not have the ability to patch the system right after

the first instance of an attack. This is particularly true for third-party software where availability of patches is outside the control of the system administrator. 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 SWIFT, history helps the GA to converge faster but is not strictly necessary. By setting the *Iv* of a node to a high value, SWIFT can deploy a response, even if drastic, to prevent the node from being achieved. This shows up in our experiments in the better performance of SWIFT compared to the baseline.

The I-GRAPH structure used here is implicitly assumed to be complete with respect to the N_B-type nodes. One can leverage existing work [26],[28] to automatically generate the I-GRAPH from a given system configuration and set of available detectors. A 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 [8] and handle them at the expense of expanding the GA search space.

7 Related Research

With increasing complexity and ubiquity of distributed systems, IRSs for such systems are gaining interest. They can broadly be classified into four categories. *Static decision making*. This class of IRS [11]-[13] provides a static mapping of the alert from the detector to the response to be deployed. The IRS includes basically a look-up table where the admin has anticipated all alerts possible in the system and an expert indicated responses to take for each. *Dynamic decision making*. This class of IRS reasons about an ongoing attack based on the observed alerts and determines an appropriate response to take. A wide variety is discernible in this class based on the sophistication of the algorithms. The systems in [1]-[6] fall in this category. *Intrusion tolerance through diverse replicas*. This class of IRS implicitly provides the response to an attack by masking the effect of the attack. The approach is to employ a diverse set of replicas to implement any service. The fault model is that the replicas are unlikely to share the same vulnerabilities and therefore not all will be compromised by a single attack. An advantage is that the system can continue operation without a disruption as in the active replication technique. The systems in [14]-[16] fall in this category. However, in practice, it's challenging to assume diverse implementations of all the critical services are available to be set up as replicas. *Responses to specific kinds of attacks*. This class of IRS is customized to respond to specific kinds of attacks, most commonly, DDoS attacks. The approach is to trace back as close to the source of

the attack as possible and then limit the resources available to the potentially adversarial network flows. The system reported in [7] fall in this category.

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. The baseline ADEPTS system does not have the design to choose globally optimal responses.

There have been some efforts at using genetic algorithms for intrusion detection [18]-[20] and search for vulnerabilities [17]. 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.

8 Conclusion

In this paper, we introduced the notion of global optimality of responses deployed by an intrusion response system. 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 initial chromosome pool and the design of carry over of chromosomes from one generation to the next are designed to improve the solution quality over the locally optimal responses of the state-of-the-art ADEPTS system. The proposed GA framework also 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 IRS performance was evaluated on a three tier e-Commerce system through injection of multi-stage attacks. The evaluation brings out the fact that survivability improves with the global response determination process of SWIFT over a greedy local response determination (e.g. 27% improvement based on experiment in Sec. 5.1). In our experiments, on average it takes only about 4~5 attack instances for SWIFT to adequately adapt to an attack. This number can be further decreased if the history information of attacks can be shared across systems which have similar configurations.

9 References

[1]. T. Toth and C. Kruegel, “Evaluating the Impact of Automated Intrusion Response Mechanisms”, ACSAC 2002.
[2]. G. White, E. Fisch, U. Pooch, “Cooperating Security Managers: A Peer-based Intrusion Detection System”, IEEE Network, vol 10, no. 1, 1996, pp. 20-23.

[3]. P. Porras and P. Neumann, “Emerald: Event Monitoring Enabling Responses to Anomalous Live Disturbances,” NISSC, pp. 353-365, 1997.
[4]. D. Ragsdale, C. Carver, J. Humphries, U. Pooch, “Adaptation Techniques for Intrusion Detection and Intrusion Response Systems”, Int. Conf. on Systems, Man, Cybernetics, pp. 2344-2349, 2000.
[5]. I. Balepin, S. Maltsev, J. Rowe, K. Levitt, "Using specification-based intrusion detection for automated response," RAID, pp. 136–154, 2003.
[6]. W. Lee, W. Fan, M. Miller, S. J. Stolfo, E. Zadok, "Toward cost-sensitive modeling for intrusion detection and response," Journal of Computer Security, vol. 10, pp. 5-22, 2002.
[7]. D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, T. Reid, “Autonomic Response to Distributed Denial of Service Attacks”, RAID 2001.
[8]. C. Carver, J. Hill, U. Pooch, “Limiting Uncertainty in Intrusion Response”, IEEE Workshop on Info. Assurance and Security, 2001.
[9]. B. Foo, Y-S. Wu, Y-C. Mao, S. Bagchi, E. H. Spafford, “ADEPTS: Adaptive Intrusion Response using Attack Graphs in an E-Commerce Environment,” DSN, pp. 508-517, 2005.
[10]. Y-S. Wu, B. Foo, Y-C. Mao, S. Bagchi, E. H. Spafford, “Automated Adaptive Intrusion Containment in Systems of Interacting Services,” In Elsevier Journal on Computer Networks, vol. 51 #5, pp. 1334-1360, April 2007.
[11]. W. Metcalf et al., "Snort-inline."
[12]. Symantec Corp., "Norton Antivirus."
[13]. T. Ryutov, C. Neuman, K. Dongho, Z. Li, "Integrated access control and intrusion detection for Web Servers," ICDCS, pp. 394-401, 2003.
[14]. D. Wang, B. B. Madan, K. S. Trivedi, "Security analysis of SITAR intrusion tolerance system," in ACM workshop on Survivable and self-regenerative systems, pp. 23-32, 2003.
[15]. C. Cachin, "Distributing trust on the Internet," DSN, pp. 183-192, 2001.
[16]. F. B. Schneider and L. Zhou, "Implementing trustworthy services using replicated state machines," IEEE Security & Privacy Magazine, vol. 3, pp. 34-43, 2005.
[17]. 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.
[18]. W. A. Jansen, “Intrusion detection with mobile agents,” Computer Communications, Volume 25, Issue 15, pp. 1392-1401, 2002.
[19]. 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.
[20]. Ludovic Me, “GASSATA: A genetic algorithm as an alternative tool for security audit trails analysis," RAID '98.
[21]. PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard. Version 1.1. http://pcisecuritystandards.org.
[22]. The Open Web Application Security Project. The Ten Most Critical Web Application Security Vulnerabilities. 2004, www.owasp.org.
[23]. Genetic Algorithm : (http://en.wikipedia.org/wiki/Genetic_algorithm)
[24]. The MITRE Corporation. Common Vulnerabilities and Exposures. http://cve.mitre.org
[25]. S. Noel, E. Robertson, S. Jajodia, “Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances,” IEEE Annual Computer Security Applications Conference, 2004
[26]. O. Sheyner, J. Haines, S. Jha, R. Lippmann, JM Wing, “Automated generation and analysis of attack graphs, “ IEEE Symposium on Security and Privacy, 2002
[27]. Yu Liu, Hong Man, “Network vulnerability assessment using Bayesian Networks,” Proceedings of SPIE, Volume 5812, 2005
[28]. Xinming Ou, Wayne F. Boyer, Miles A. McQueen, “A scalable approach to attack graph generation,” ACM conference on Computer and communications security, 2006
[29]. M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, E. Brewer, "Pinpoint: problem determination in large, dynamic Internet services," Dependable Systems and Networks (DSN), pp. 595-604, 2002.
[30]. G. Khanna, I. Laguna, F. A. Arshad, S. Bagchi, "Distributed Diagnosis of Failures in a Three Tier E-Commerce System," Symposium on Reliable Distributed Systems (SRDS), pp.185-198, 2007.