Secure Configuration of Intrusion Detection Sensors for Changing Enterprise Systems

Gaspar Modelo-Howard, Jevin Sweval, and Saurabh Bagchi

Dependable Computing Systems Laboratory, Purdue University
465 Northwestern Avenue, West Lafayette, IN 47907, USA
{gmodeloh,jsweval,sbagchi}@purdue.edu

Abstract. Current attacks to distributed systems involve multiple steps, due to attackers usually taking multiple actions to achieve their goals. Such attacks are called multi-stage attacks and have the ultimate goal to compromise a critical asset for the victim. An example would be compromising a web server, then achieve a series of intermediary steps (such as compromising a developer's box thanks to a vulnerable PHP module and connecting to a FTP server with gained credentials) to ultimately connect to a database where user credentials are stored. Current detection systems are not capable of analyzing the multi-step attack scenario. In this document we present a distributed detection framework based on a probabilistic reasoning engine that communicates to detection sensors and can achieve two goals: (1) protect the critical asset by detecting multi-stage attacks and (2) tune sensors according to the changing environment of the distributed system monitored by the distributed framework. As shown in the experiments, the framework reduces the number of false positives that it would otherwise report if it were only considering alerts from a single detector and the reconfiguration of sensors allows the framework to detect attacks that take advantage of the changing system environment.

Key words: Distributed intrusion detection, multi-stage attacks, Bayesian reasoning, sensor reconfiguration.

1 Introduction

Current computer attacks against distributed systems involve multiple steps, thanks to attackers usually taking multiple actions to achieve their ultimate goal to compromise a critical asset. Such attacks are called multi-stage attacks (MSA). As today's enterprise systems are structured to protect their critical assets, such as, a mission-critical service or private databases, by placing them inside the periphery, MSAs have gained prominence. Examples include the breach of a large payment processing firm [1] and the breaches published by the U.S. Department of Health & Human Services [24]. MSAs are characterized by progressively achieving intermediate attack steps and progressing using these as stepping stones to achieve the ultimate goal(s). Thus, prior to the critical asset being compromised, multiple components are compromised. Logically, therefore,

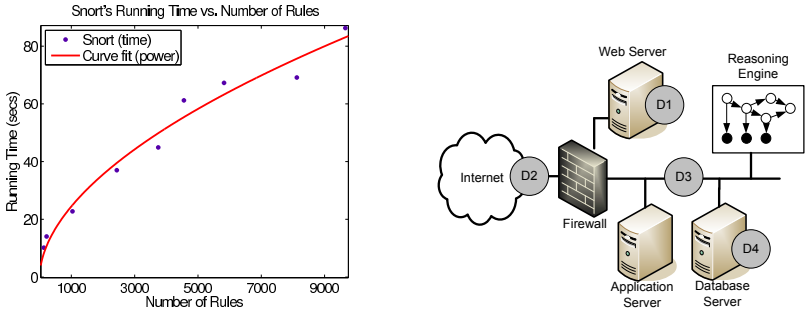
to detect MSAs, it would be desirable to detect the security state of various components in an enterprise distributed system—the outward facing services as well as those placed inside the periphery. Further, the security state needs to be inferred from the alerts provided by intrusion detection sensors (henceforth, shortened as "sensors") deployed in various parts of the system.

In the context of MSAs against distributed systems, this is challenging because sensors are designed and deployed without consideration for assimilating inputs from multiple detectors to determine how an MSA is spreading through the protected system. Prior work has shown that it is possible to determine the choice and placement of sensors in a systematic manner and at runtime, perform inferencing based on alerts from the sensors to determine the security state of the protected system components [7][1]. In achieving this, the solutions have performed characterization of individual sensors prior to deployment, in terms of their capability to detect specific attack step goals. At runtime, inferencing has been performed on the basis of the evidence—the alerts from the sensors—to determine the unobservable variables—the security state of the different components of the protected system. The sensors may be either network-based sensors, which observe incoming or outgoing network traffic, or host-based sensors, which observe activities within a host.

However, no existing solution has handled the various sources of dynamism that are to be expected in large-scale protected systems deployed in enterprise settings. The underlying protected system itself changes with time, with the addition or deletion of hosts, ports, software applications, or changes in connectivity between hosts. A static solution is likely to miss new attacks possible in the changed configuration of the protected system as well as throw off false alarms for attack steps that are just not possible under the changed configuration. The nature of attacks may also change with time or the anticipated frequencies of attack paths may turn out to be not completely accurate based on attack traces observed at runtime. Existing solutions cannot update their "beliefs" in an efficient manner and are therefore likely to be less accurate. Finally, when the compromise of a critical asset appears imminent, fast reconfiguration of existing sensors (such as, turning on some rules) may be needed to increase the certainty about the security state of the critical asset. Our contribution in this paper is to show how the choice and placement of sensors can be updated through incremental processing when the above kinds of dynamism occur.

The solution we propose in this paper called **D***istributed* **I***ntrusion and* **A***ttack* **D***etection* **S***ystem (DIADS)* is to have a central inferencing engine, which has a model of MSAs as attack graphs. DIADS creates a Bayesian Network (BN) out of an attack graph and observable (or evidence) nodes in the attack graph are mapped from sensor alerts. It receives inputs from the sensors and performs inferencing to determine whether a rechoosing or replacement of sensors is needed. Further, it can reconfigure existing sensors, by turning on or off rules or event

[1] In this paper, we will refer to the distributed enterprise system that is being protected as the *protected system* and the set of sensors embedded in various components of the protected system as the *sensor system*.



(a) Time to match a packet (b) Diagram for DIADS

Fig. 1: (a) Results from curve fitting the data points from the Snort experiment. (b) General block diagram of the proposed DIADS. A wrapper (software) is used to allow communication from the sensors (circles labeled D1 to D4) and firewall to the reasoning engine and viceversa (only for sensors).

definitions based on the changed circumstances. Thus, the inferencing engine has a two-way communication path with the sensors. DIADS determines changes to the protected system by parsing changes to firewall rules at network points as well as at individual hosts and updates the BN accordingly. If on the basis of current evidence, it determines that a critical asset (also synonymously referred to as a crown jewel) will imminently be compromised, it determines what further sensors close to the asset should be chosen, or equivalently, what further rules in an already active sensor should be turned on.

One may think that a perfectly acceptable, and a much simpler, solution is to activate all the available sensors and turn on all the available rules at any sensor. Thus, there will be no reason to react to dynamic changes of the three types mentioned above. However, this will impose too high an overhead on the protected system in terms of the amount of computational resources that will be required and the frequency of false alerts that will be generated. For example, we determine empirically that for the popular Snort IDS [19] turning on the default set of rules will cause it to potentially take 85 seconds to match a single packet (corresponding to 9700 rules in Figure 1a). Therefore there is the motivation to dynamically reconfigure the sensors according to the activity observed in the network.

To sum up, in this paper we make the following contributions:

1. We design a distributed intrusion detection system that can choose and place sensors in a distributed system to increase the certainty of knowledge about the security state of the critical assets in the system.
2. We imbue our solution with the ability to evolve with changes to the protected system as well as the kinds of attacks seen in the system.
3. Through domain-specific optimizations, we make our reasoning engine fast enough that it can perform reconfiguration of existing sensors while a multi-stage attack (MSA) is coursing through the protected system.

We structure the remainder of this paper as follows. In Section 2 we review previous work on distributed intrusion detection systems (DIDS), MSA, and probabilistic approaches to intrusion detection. Section 3 states the problem studied and the threat model considered. Section 4 presents the proposed DIADS framework to detect MSAs and to reconfigure detection sensors, including a description of the different components and algorithms used. In Section 5 we provide a description of the experiments performed along with the results. Finally, Section 6 provides conclusions and future work.

2 Related Work

There has been previous work on developing and proposing DIDSs. Early examples of these systems are [20], [17], [26], and [21]. A starting point for DIDSs is the collaboration between Lawrence Livermore National Labs, U.S. Air Force and other organizations [20]. It represented the first attempt to physically distribute the detection mechanism, while centralizing the analysis phase in a single component, running a rule-based system.

Another distributed IDS is EMERALD [17]. It is a signature- and anomaly-based distributed IDS with statistical analysis capabilities (rule-based and Bayesian inference). The communication among sensors and monitors is structured in a hierarchy. NetSTAT [26] is a network-based IDS modeling intrusions as state transition diagrams and the target network as hypergraphs. By using both models, the system prioritizes which network events to monitor. AAFID [21] is a distributed framework based on software agents to collect and analyze data and used as a platform to develop intrusion detection techniques. An interesting policy-based proposal based on the popular Bro IDS [15] was presented in [6], using intrusion detection sensors in a distributed, collaborative manner.

Unfortunately there has not been much discussion about DIDS in the last few years so the impact of more complex distributed systems on the detection capabilities of IDS as well as the evolution of MSAs has been somewhat neglected. Previous work has primarily concentrated on increasing the accuracy of IDSs by improving their true positive (TP) rate on single step attacks. Additionally, it does not consider the dynamic nature of the protected system, one of our focus areas.

Previous work has considered MSAs [9], [10] but within a limited scope. [9] proposes an offline-method to correlate alerts using an attack graph, to improve detection rate, while reducing false positive (FP) and false negative (FN) rates. It is a rule-based method and does not consider a probabilistic approach. [10] presents a formal conceptual model based on Interval Temporal Logic (ITL) to express the temporal properties of MSAs.

A principal component for our framework is an attack graph, from which to create a corresponding Bayesian network. An example of previous work on using attack graphs for intrusion detection is found in [4]. Other works have previously focused on using attack graphs to evaluate (offline) the vulnerability state of the computer system [27].

Bayesian networks have been used for intrusion detection, examples include [7] and [5]. [7] models the potential attacks to a target network using a Bayesian network to determine (off-line) a set of detectors to protect the network. [5] presents a method based on Dynamic Bayesian networks to include the temporal properties of attacks in a distributed system.

Alert correlation is an area related to intrusion detection, that has received the attention of the research community. Schemes in this area can be classified under two basic groups: schemes that require patterns of actual attacks and/or alert interdependencies, and schemes that do not. Members of the first group include [11], [12], and [13]. Our proposed framework, can be classified as part of the first group. The second group of correlation schemes works without any specific knowledge of attacks. Examples include [25], [18].

In [11], the authors present a formal framework for alert correlation that constructs attack graphs by correlating individual alerts on the basis of the prerequisites and consequences manually associated to each alert. [12] presents techniques to learn attack strategies from correlated attack graphs. The basic idea is to compute how similar different attack graphs are by using error tolerant subgraph isomorphism detection. In [13] the authors built on the results from the previous two papers, integrating two alert correlation methods: correlation based on prerequisites and consequences of attacks and those based on similarity between alert attribute values. They used the results to hypothesize and reason about single attacks possibly missed by the IDSs. There are several similarities between their approach and ours. We both represent attack scenarios as graphs, assume attack steps are usually not isolated but rather part of an MSA. Still, there are also several differences between their alert correlation approach and ours. In a nutshell, our approach is adaptive, provides a larger visibility of the target network, follows a probabilistic model, and works online, while theirs is not.

3 Problem Statement and Threat Model

In this paper, we answer two fundamental questions:
(1) How to update the configuration of sensors in an enterprise distributed system (i.e., one with many hosts and software applications and hence attack injection points) based on updated information that is obtained after the protected system and the sensor system have been deployed.
(2) When the imminent threat to a critical asset(s) is high, how to reconfigure existing sensors (such as, by activating new rules) to increase confidence in the estimate of the security state of the critical asset(s).

In terms of the model for the protected system, all the components fall target network under a single administrative domain and therefore, there is complete trust between the owners of the different assets.

The profile of the attackers includes highly motivated individuals that might have an economical incentive to compromise the distributed system. Attackers follow a multi-step approach to compromise a resource or acquire data. It could

start with some reconnaissance, followed by exploitation of different hosts or services in the target network. This description also fits the cases where attack sources are botnets and malware, that does not include human intervention. We do not address intruders who steal data by physically connecting to a host (for example, an insider's attack using a USB memory stick).

In our framework, one or more *critical assets* are identified in the protected system by the system owner and become the main protection objective of our DIADS framework. Each critical asset is represented in the BN as a leaf node. An example of a critical asset is a database that contains personally identifiable information (PII). The above statement does not preclude having sensors that detect attacks at other assets (such as, at a network ingress point), but our inferencing uses such sensors to provide evidence of attacks leading up to a potential compromise of the critical assets. Also, our DIADS framework is not attempting to create better intrusion detection sensors; rather it is seeking to use existing sensors intelligently to obtain a better estimate of the security state of critical assets in an enterprise distributed system.

We consider only multi-stage attacks (MSAs) to distributed systems. An important example is an MSA to a three-tier system (web / application logic / database) which might allow an attacker to launch HTTP-based attacks to ultimately reach the database and the information stored in it.

4 DIADS Framework

In this document we propose a distributed intrusion detection framework that includes two components: (1) a probabilistic reasoning engine and (2) a network of detection sensors to detect various stages of MSAs, as shown in Figure 1b. The second component comprises off-the-shelf sensors, augmented with a standard wrapper that allows the sensor to send alerts to the reasoning engine and receive commands back from the reasoning engine. The architecture is able to alert intrusion events related to potential MSAs and determine if any critical asset has been compromised, or is under imminent likelihood of being compromised based on current evidence of the spread of the attack. It also allows for reconfiguration of sensors according to changes to the protected system that is being monitored by the DIADS. Through this architecture, the DIADS can reduce the number of false positives that it would report if it were independently considering each step of the MSA. A block diagram of the proposed architecture is shown in Figure 2.

The reasoning engine represents different possible MSAs as a single Bayesian network, which is updated according to events reported by the detection sensors and the changing network configuration. The probabilistic engine can also request more information from sensors when necessary. The reasoning engine can estimate the security state of the critical assets given partial information about multi-stage attacks and from imperfect or noisy sensors.

The reasoning engine also collects background information about the distributed system so the model can be updated. As a starting point, we should consider the network and policy configurations stored in a firewall. The firewall

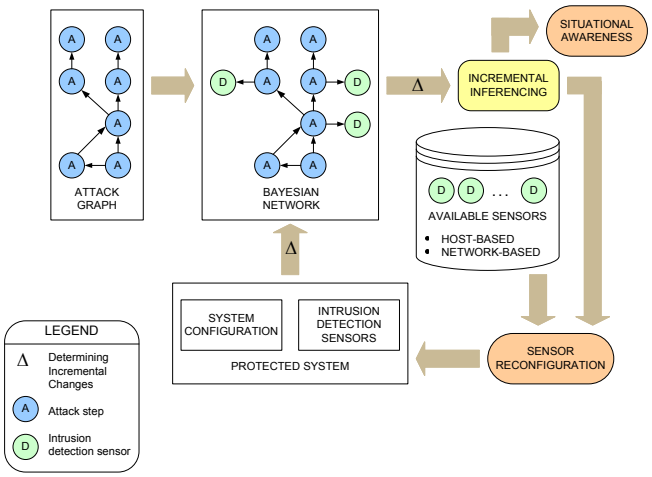


Fig. 2: Diagram of the proposed framework, providing details on the components of the reasoning engine.

can be at a network ingress-egress point as well as at individual hosts. The firewall configuration indicates which components are allowed to communicate with which components and thus has an important determining effect on the structure of the attack graph, and consequently, on the structure of the BN.

4.1 Probabilistic Reasoning Engine

To build our reasoning engine, we use Bayesian Network (BN), which is a popular probabilistic graphical model. It is a macro-language, representing joint distributions compactly by using a set of local relationships between random variables and specified by a graph. A key point is that the missing edges in the graph imply the conditional independence between the corresponding nodes. BN captures the characteristic in real-world data of *locality of influence*, the idea that most variables are influenced by only a few others. [7] shows the implications of this.

Bayesian networks combine graph theory with statistical techniques to model MSA scenarios. In our framework, we use an attack graph to create the structure of the BN, a directed acyclical graph. Each node in the graph represents a vulnerability, more specifically, a 3-tuple: *host* × *port* × *vulnerability* existing in the target network. This means that the service running on that host and listening on that port has that vulnerability. The edges between nodes represent the direct precondition relationship between the attack steps. The BN also includes nodes to represent intrusion detection sensors. An edge $A \rightarrow D$ from an attack step node to a sensor node represents the possibility of the sensor detecting that attack step, with the CPT quantifying the accuracy and precision of the detection. Each node is parametrized by a set of probability values and represented as a *conditional probability tables (CPT)*. Proposed in previous work [7] and also

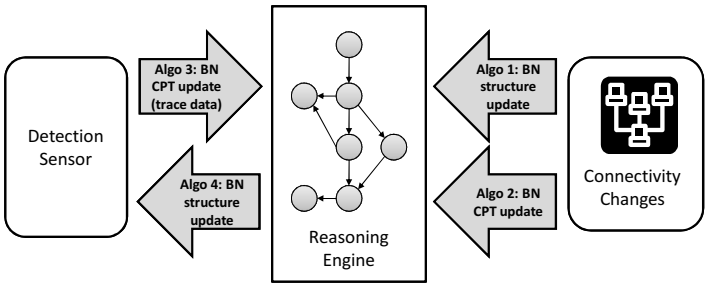


Fig. 3: The framework uses four algorithms, three to update the reasoning engine and one to reconfigure the detection sensors.

suggested by [5], the Bayesian network representation can unify the information available from multiple sensors, in order to determine if an MSA is occurring.

There are several benefits of using Bayesian networks. First, it can be a more appropriate representation of reality than deterministic approaches, accounting for several sources of uncertainty—noisy sensors, unknown intentions of the adversary affecting the path of the MSA, and unknown difficulty of transitioning from one attack step node to the next. A potential drawback of probabilistic models is the combinatorial explosion faced when computing a joint probability distribution. In our work, we address this issue by using the Noisy-OR model [16] to represent the CPTs. Further details are provided in section 4.5. Our DIADS framework is composed of four algorithms, which are schematically shown in Figure 3. Pseudo-code for algorithms 1, 2, and 4 are provided in the Appendix.

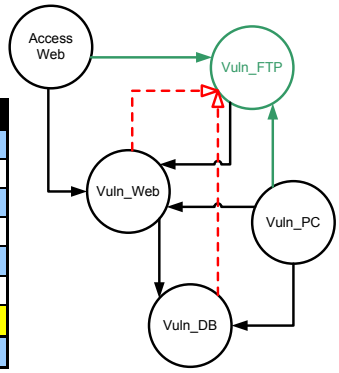
4.2 Algorithm 1: BN update to structure based on Firewall rule changes

The algorithm produces a list of nodes and edges that should be added to (V_a, E_a) or deleted from (V_d, E_d) the Bayesian network to represent changes to the protected system. We use changes to firewall rules as a proxy for the changes to the protected system. The firewalls can be at a network ingress-egress point or at individual hosts in the system.

The message passed from the Firewall to the reasoning engine has the following structure: $message = < number, srcIPaddr, destIPaddr,\ portnumber,\ action,\ ruletype >$ where *number* refers to the order of the rule in the firewall table. $srcIPaddr$ and $destIPaddr$ are the IP addresses for the source and destination of communication; *portnumber* is the TCP or UDP port number (16-bits in IPv4); *action* is one of three options: allow, deny or drop; and *ruletype* refers to the change made to the rule table: adding a new rule, modifying an existing rule or deleting an existing rule. For the purposes of our experiments, we only considered firewall rule tables composed of *allow* rules followed by a *denyall* rule. So effectively, the rule table creates a policy where allowed communication is explicitly defined and everything else not defined, is denied.

No.	Source	Destination	Action
1	Any	Web:80	Allow
2	Web	DB:3306	Allow
3	Web	DB:22	Allow
4	DB	Web:22	Allow
5	PC	DB:3306	Allow
6	PC	DB:22	Allow
7	Any	FTP:21	Allow
8	Any	Any	Deny

(a) Firewall rule table



(b) Bayesian network

Fig. 4: Impact of changes to a firewall rule. A new rule (No.7) in the firewall table changes the topology of the Bayesian network. Two of the four new edges, shown as dashed lines, will be removed by the algorithm since they lead to a cycle. A BN node is actually host × port × vulnerability, but here for simplicity, we have a single vulnerability per service (i.e. per host × port).

The algorithm can be divided into four parts: how to select the nodes and edges to be added, if the rule has type *add* (lines 1 to 11); how to select the nodes and edges to be deleted, if the rule has type *delete* (lines 13 to 29); checking for the resulting changes to the BN to not introduce cycles and to confirm that the resulting nodes are part of a path to the nodes representing the critical assets (lines 31 to 37); and finally, the converting the *destIPaddr:port* nodes into their corresponding *address:port:vulnerability* nodes in the BN.

When a rule has type *add* or *delete*, the algorithm checks if the source and destination addresses are new to the BN or already exist. If a node exists, then the edges shared with its parents (line 4) or its children (line 7) should be included to the set of edges to add (E_a). Also, the edge explicitly defined by the rule is included in (E_a). If a node is new, then it should be added to the set of nodes to add (V_a). A similar approach (but with opposite results) is used for case when a rule has type *delete*.

The algorithm then checks the nodes and edges in the resulting BN by running *Depth First Search* (DFS) to determine if the nodes have a path to the critical assets. If the nodes do not, then they are pruned. DFS also checks if the addition creates any cycles and if so, the back edges are deleted. The first is an important optimization focusing the attention of DIADS to the critical assets and limiting the growth of the BN.

Finally, the algorithm transforms the nodes in the sets V_a and V_d to nodes in the BN. It does this by doing a lookup in a matrix R that maps the *host* × *port* to the vulnerability. It acquires the raw data for this from the *National Vulnerability Database* (NVD) [23], a public repository of vulnerability management data.

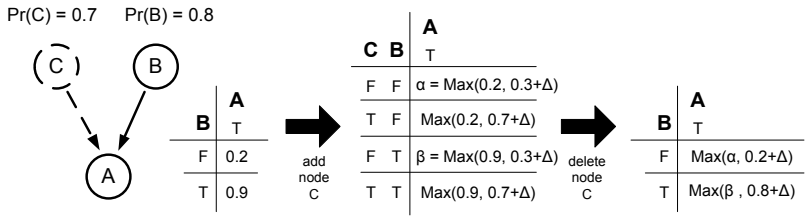


Fig. 5: Example for algorithm 02: initialization of BN CPT. To add a new parent (C) to an existing node (A), we create the marginal probability $Pr(C)$ from its CVSS (base metric) value and use it to update the new CPT of A.

As an example, consider a distributed system connected to the Internet, with three computers: a web server (accessible from the Internet), a database server and a desktop computer. The database server and the desktop computer are connected to the same subnet, while the web server is connected to a separate subnet (DMZ). All computers are protected by a network-based firewall and the rule table is shown in Figure 4a. A Bayesian network can be built from the table, as shown in Figure 4b. The critical asset is the database server and for simplicity purposes, we have assumed one existing vulnerability per host.

If the rule `any -- > FTP:21 allow` is now added to the network firewall because a new FTP server has been deployed and connected to the DMZ network, the resulting Bayesian network is shown in Figure 4b. A new node, `Vuln_FTP`, is added and will have five edges. Four are inbound, created from the added rule and one outbound, from rule No. 1 in the table. The inbound edges from nodes `Vuln_Web` and `Vuln_DB` are not included in the final Bayesian network as they make the graph cyclical.

4.3 Algorithm 2: Initialization of BN CPTs based on firewall changes

Algorithm 2 produces a list of CPTs for the changed nodes, i.e., nodes for which there is an increase or reduction in the number of parents of the nodes, according to the output from Algorithm 1. To update the CPT, we use the base metric value of the $CVSS$ score [3] of the node (corresponding to a vulnerability) to be added or removed and divide it by 10 to use it as its marginal probability value. Then if the resulting CPT is for an existing node, we take $max(newProb(v_i) + \Delta, oldProb(v_i))$. Figure 5 shows an example of how we use the formula.

In figure 5, first a new parent node C is added to an existing node A in the BN. We take the base metric score (7) of the vulnerability corresponding to node C and divide it by 10. Then use the formula $max(Prob(C)+\Delta, oldProb(A|$previous evidence$))$ to create the new CPT. In our experiments, we use $\Delta = 0.05$. Figure 5 also shows the CPT when node C is later removed. The base metric score of the other parent node (B) is used to update the CPT.

4.4 Algorithm 3: BN update of CPT based on incremental trace data

The alerts received by the reasoning engine from the individual sensors are used to update the CPTs in the Bayesian network in an incremental manner. To achieve this, this algorithm uses the set of alerts received during a window of time and the matrix R, that maps the existing vulnerabilities in the system to their corresponding hosts and ports. The output of the algorithm is the set of CPTs with the updated values.

The algorithm uses a popular and powerful model known as Noisy-OR [16] to represent each CPT. Noisy-OR allows us to specify the CPT of a node with n parents, using with $n+1$ parameters as opposed to 2^n for binary nodes. This prevents the exponential growth experienced by the CPT of a node when the number of parents (n) is large. The Noisy-OR model assumes that effect of each parent on the CPT of the edge to the child node (v_i) is independent from that of the other parents and is sufficient to produce the effect (represented by the child node) in the absence of all other parents. An additional parent node is added to capture all other causes that were not modeled explicitly. The marginal probability of this node is $1-p_0$. Then the CPT can be built with the following formula, where C represents a combination of the values for the parents of the child node:

$$Prob(v_i|C) = 1-(1-p_0) \prod_{A=parent(v_i)\in C} \Big(\frac{1-Prob(v_i|A=T, \text{Others} = \text{F})}{1-p_0}\Big) \quad (1)$$

4.5 Algorithm 4: Update choice of sensors based on runtime inference

The final algorithm of our framework is used to reconfigure the detection sensors. This includes adding and removing sensors, as well as reconfiguring existing ones. The high level objective is to reduce the uncertainty of knowing if the critical asset has been achieved or not. The algorithm works by looking at the alerts received and uses them as evidence to compute the posterior probability of each Bayesian network node that corresponds to the critical asset.

The first step of the algorithm (line 1) is to compute the posterior probability for the critical asset, given the evidence received from the currently enabled sensors in the system. If the value is larger than a threshold (determined by the system administrator), this is taken as indication that the critical asset is likely to be compromised and therefore greater certainty is needed in the determination of the security state. Therefore, the algorithm measures (lines 3 and 4) the impact of candidate sensors, which are close to the detected alerts and the critical asset. A radius can be set a priori in terms of the number of edges away from a particular node to determine the candidate set of sensors. Previous work [7] has shown that the effect of a sensor on a Bayesian network node fades beyond 2-3 hops and thus this restriction appears reasonable.

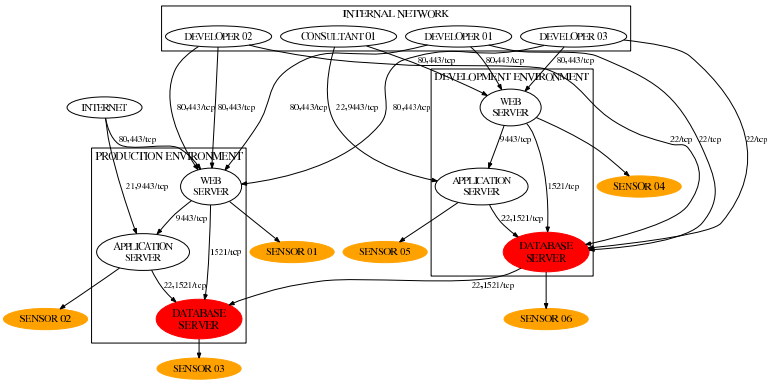


Fig. 6: Connectivity graph for testing scenario, showing the TCP ports enabled for communication between different hosts. The shaded nodes represent the critical asset (databases) in the protected system.

The algorithm determines a new set of detectors by using the *Fully Polynomial Time Approximation Scheme* (FPTAS) presented in [8] for the problem of determining the placement of intrusion detection sensors. The same cost bound is maintained which will prevent the algorithm from blissfully adding new sensors. This problem has been mapped to the *0-1* Knapsack problem for which a dynamic programming solution (FPTAS) exists that runs in *pseudo-polynomial* time (running time scales up as the solution approaches the optimal). The algorithm finishes by comparing the set of current detectors with the new set. The delta between the sets indicates the set of detectors to be disabled or enabled, which is output by the algorithm.

5 Experiments and Results

5.1 Experimental Setup

For our experiments, we used attacks against a real-world distributed system which is part of an NSF Center at our university and serves content and simulation tools for an engineering domain for thousands of users. The system includes fifteen hosts that include two environments, one for production and another for development of applications and staging prior to moving them to the production environment. Each environment includes a web server, an application server and a database server. A team of developers' and consultants' computers have access to subsets of both environments. Communication between all hosts is controlled by firewall rules at each host. The corresponding connectivity graph is shown in Figure 6.

In our experiments, the database servers are the critical assets to protect. A strong motivation to pick the databases is their role to store critical information for the organization. We created a Bayesian Network (BN) from the distributed

system by first generating a list of the vulnerabilities found by the OpenVAS [14] vulnerability scanner on servers and client machines. Each vulnerability was then mapped to a node in the BN by associating it to the host and service(port) where the vulnerability was found. Finally, the nodes were connected according to the connectivity information for the distributed system. The BN had 345 nodes and 1948 edges. We then pruned the BN to only include high risk vulnerabilities, according to OpenVAS, as these ones are the primary vectors used by attackers to compromise systems. The final BN had 90 nodes and 582 edges.

We provide comparative results between DIADS (our algorithms presented in this paper) and a static and heuristic-driven choice of sensors. All results are presented as *Receiver Operating Characteristics* or *ROC curves* [22]. The curve is a graphical plot of the tradeoff between true positive rate (TPR, detection rate) and the false positive rate (FPR, false alerts) for a detector. The different points in the ROC curves are generated by varying the threshold for the probability value for the BN nodes corresponding to the critical assets.

We had a total of 18 possible sensors; 3 sensors for each of the web server, application server, and database server, in both the development and the production environments. They are all generic sensors with signatures customized to detect the class of attack into which the corresponding (vulnerability) node can be categorized. For all experiments, for both baseline and DIADS, we constrain the algorithms to pick a set of 6 from 18 possible detectors.

It is important to note that DIADS' goal is to improve the performance of a set of detectors, by considering temporal information (i.e. when detectors are sending alerts about a progressing attack or when changes occur to the distributed system). For our experiments, we defined detectors with adequate but not perfect performance (in terms of TP and FP). It is not our goal to improve the performance of individual detectors.

5.2 Experiment 1: Dynamic Reconfiguration of Detection Sensor

The first experiment compared the performance between a dynamic reconfiguration of sensors and an static set of sensors, all close to the database servers. The static setup follows the intuitive decision of turning on all the sensors at the critical assets, in this case the database servers. To test both setups, we use an attack scenario that had the following progress: the attack started from the Internet, compromised the production web server, from where to compromise the applications server and then elevate permissions, and finally compromise the database server. Further details for all attack scenarios and the Bayesian network used in all experiments, are provided in [2].

In this experiment, a set of alerts are generated for the first three steps of the attack scenario. This set serves as evidence and is provided to the reasoning engine for DIADS to recompute the set of sensors. As shown in Figure 7, the dynamic reconfiguration setup outperforms the static configuration of sensors. The area under the continuous line (dynamic) is greater than the area under the dotted line (static) by 16% ($Area_{DIADS} = 0.7810$ and $Area_{baseline} = 0.6728$).

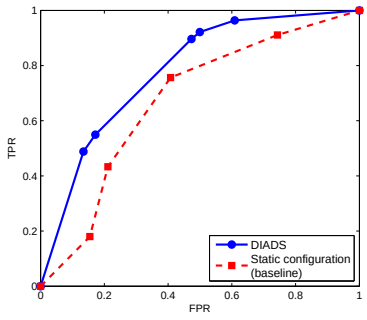


Fig. 7: Performance comparison between dynamic configuration of DIDS and a set of detectors monitoring only DB servers.

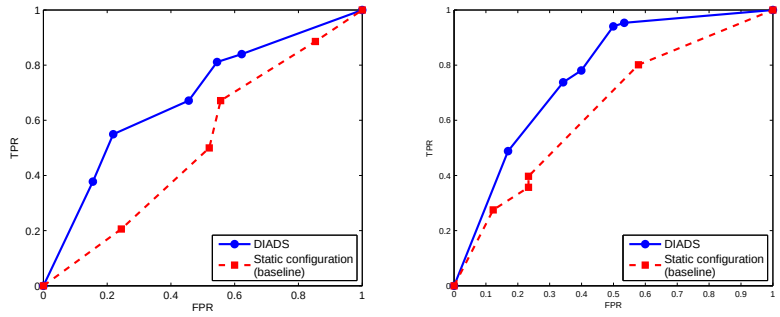
This also means, the dynamic setup provides a higher detection rate at points when both setups have the same false alarm rate.

A notable point is that the difference between both setups is not large. This should be expected as the static setup is concentrated around the database servers (the critical asset and final setup in the attack scenario) while the dynamic setup is scattered around the protected system.

5.3 Experiment 2: Dynamism from Firewall Rules Changes

Experiment 2 tested the performance of the dynamic and static setups as changes were made to the firewall rule table of the protected system. We considered two real scenarios: (1) removing from the system a host belonging to a developer and (2) adding a direct communication path is created from a consulant's host to the database server, in the development environment (in this case, the consultant determined some changes to the database schema had to be tried out in the development environment prior to unveiling it on production). For the static configuration, one sensor was deployed on each host in both development and production environments.

For the first firewall change where a developer's host was removed, we tested both setups using an attack scenario starting from another developer's host. This represents the increasingly common client-side attacks. The attack starts as the developer visits a malicious website that installs some malware on the host. Then permissions are elevated thanks to another existing vulnerability in the developer's host. Then a vulnerability in the database server (production) is exploited and finally another vulnerability is used to access the data in the database. For the second firewall change where a direct communication path is created, we used a different attack scenario. The attack starts from another developer's host that also visits a malicious website and malware is installed in the host. Then a vulnerability in the web server (development) is exploited, after which the application server and finally the database server, all part of the development environment, are compromised.



(a) Removing a host (b) Opening ports to DB server

Fig. 8: Impact on topology changes. (a) Removing a host (developer) from network. (b) Allowing direct access between the consultant box and the DB development server.

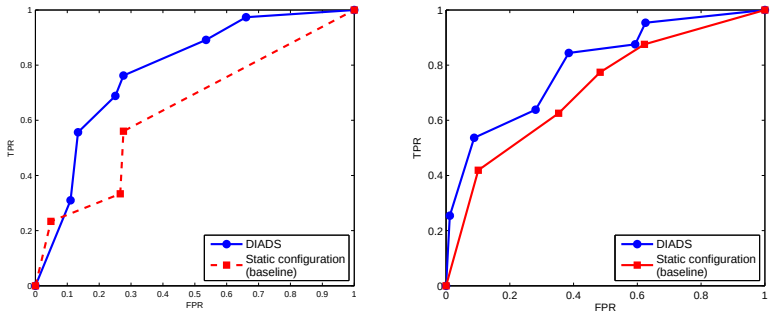
For DIADS, the BN was modified based on the firewall rule changes and the dynamic programming algorithm picked the set of detectors after receiving the alerts at the start of the attack - the starting point being the same as in the static case.

Results from this experiment are shown in Figures 8a and 8b. The dynamic reconfiguration setup performs better under both attack scenarios than the static configuration. The area under the curve is greater by 32.7% ($Area_{DIADS} = 0.6809$ and $Area_{baseline} = 0.5132$) in the scenario when a host is removed and 20% ($Area_{DIADS} = 0.7659$ and $Area_{baseline} = 0.6383$) in the scenario when a direct access is set up between a consultant box and the DB development server. We consider the most interesting result to be in Figure 8b, where both setups show similar performance at the start. Both lines in the ROC curve have similar slopes, which is expected as the dynamic and static setups share 4 out of the 6 initial sensors. But as the alerts from the first three attack steps are provided to the reasoning engine in the dynamic setup, three sensors are reconfigured. This is the cause of the difference in performance, as shown in the ROC cuve.

5.4 Experiment 3: Dynamism with Attack Spreading

The goal of this experiment was to see if DIADS can reconfigure sensors on the fly as an attack spreads through the protected system. We used two different attack scenarios: (1) one starting from the Internet and (2) another starting from the internal network, a developer's host. An attack starting from the internal network usually requires less steps to reach the critical asset than attacks starting from the Internet. The static configuration picks sensors as in the earlier experiment 2 (one for each host).

The results are presented in Figures 9a and 9b for the two attack scenarios. In the attack starting from the Internet, the static setup shows a lower false



(a) Attack from the Internet (b) Attack from the internal network

Fig. 9: Comparison between our dynamic technique and a static setup for two attacks scenarios. The dynamic reconfiguration technique allows to reconfigure the detection sensors as alerts from the initial steps of the attacks are received.

alerts rate than the dynamic setup. But as evidence is provided, the ROC curve shows that the dynamic setup improves its performance. The curve shows the importance of taking into account the alerts from the initial stages of the attack to improve the performance of detection system. The improvement over the static setup, in terms of the area under the curve, is 23% ($Area_{DIADS} = 0.7845$ and $Area_{baseline} = 0.6366$). During the experiments, 4 of the 6 original sensors are replaced by the reasoning engine.

For the attack starting from internal network, the ROC curve in Figure 9b shows a similar performance between both setups. Three of the six sensors selected for the static setup are on the attack path and are quite accurate. Therefore, though DIADS outperforms the static setup, the advantage is not very large (11% where $Area_{DIADS} = 0.7964$ and $Area_{baseline} = 0.7128$). This experiment shows promise that inferencing in BN can be done fast enough relative to the speed of attacks. Of course, further experimentation is needed with a variety of attacks (and attack speeds).

6 Conclusions and Future Work

Current attacks to distributed systems involve multiple steps, with the ultimate goal of compromising a critical asset such as a database where important data is stored for an organization. In this paper, we presented a distributed intrusion detection system called DIADS that picks and places sensors in a protected system, decreasing the uncertainty inherent in estimating the security state of the critical assets in the system. DIADS has the ability to evolve when changes are made to the topology of the protected system and with further evidence coming in the form of alers while the deployed system is operational.

Future work will include experimenting further with the size of the Bayesian network. We consider we made reasonable assumptions when pruning the Bayesian network, such as only including high risk vulnerabilities as nodes. Still, as the size of the CPTs for the nodes in the Bayesian network grows exponentially in terms of the number of nodes' parents, we would like to answer the question of whether inferencing can be done fast enough. Another area to explore is the impact of evasion techniques or attacks directly targeted against DIADS. If an attacker has complete knowledge of our model, she might launch attacks to falsely cause reconfiguration of our sensors away from the attack paths.

Acknowledgment

The work described in this paper was conducted under partial funding by Northrop Grumman Information Systems under the Northrop Grumman Cybersecurity Research Consortium. We acknowledge the help of Dr. Kenneth Brancik and Dr. Donald Steiner of Northrop Grumman in formulating the problem and identifying how the solution integrates in an enterprise security architecture.

References

1. Acohido, B.: Hackers breach Heartland Payment credit card system. USA Today. (Jan. 2009)
2. Addendum: Secure Configuration of Intrusion Detection Sensors. http://sites.google.com/site/securecomm11msa/
3. Forum of Incident Response and Security Teams: Common Vulnerability Scoring System (CVSS), http://www.first.org/cvss/
4. Foo, B., Wu, Y., Mao, Y., Bagchi, S., Spafford, E.: ADEPTS: Adaptive Intrusion Response Using Attack Graphs in an E-Commerce Environment. In: International Conference on Dependable Systems and Networks, pp. 508–517. IEEE Computer Society (2005)
5. Frigault, M., Wang, L., Singhal, A., Jajodia, S.: Measuring network security using dynamic bayesian network. In: 4th ACM workshop on Quality of protection, pp. 23–30. ACM, New York (2008)
6. Kreibich, C., Sommer, R.: Policy-controlled Event Management for Distributed Intrusion Detection. In: 4th Int. Workshop on Distributed EventBased Systems (2005)
7. Modelo-Howard, G., Bagchi, S., Lebanon, G.: Determining Placement of Intrusion Detectors for a Distributed Application through Bayesian Network Modeling. In: International symposium on Recent Advances in Intrusion Detection, pp. 271–290. Springer-Verlag, Berlin (2008)
8. Modelo-Howard, G., Bagchi, S., Lebanon, G.: Approximation Algorithms for Determining Placement of Intrusion Detectors. CERIAS Tech Report 2011-01. (2011)
9. Noel, S., Robertson, E., Jajodia, S.: Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances. In: 20th Annual Computer Security Applications Conference, pp. 350–359. IEEE Computer Society, New York (2004)

10. Nowicka, E., Zawada, M.: Modeling Temporal Properties of Multi-event Attack Signatures in Interval Temporal Logic. In: IEEE / IST Workshop on Monitoring, Attack Detection and Mitigation. (2006)
11. Ning, P., Cui, Y., Reeves, D.: Constructing attack scenarios through correlation of intrusion alerts. In: 9th ACM Conf. Computer and Communications Security, pp. 245–254. ACM Press, New York (2002)
12. Ning, P., Xu, D.: Learning attack strategies from intrusion alerts. In: 10th ACM Conf. Computer and Communications Security, pp. 200–209. ACM Press, New York (2003)
13. Ning, P., Xu, D., Healey, C., St. Amant, R.: Building Attack Scenarios through Integration of Complementary Alert Correlation Method. In: Network and Distributed System Security Symposium (2004)
14. OpenVAS. The Open Vulnerability Assessment System. `http://www.openvas.org`
15. Paxson, V.: Bro: a system for detecting network intruders in real-time. J. Comp. Net. 31, pp. 2435–2463. (1999)
16. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco (1988)
17. Porras, P., Neumann, P.: EMERALD: Event monitoring enabling responses to anomalous live disturbances. In: 20th National Information Systems Security Conference, pp. 353–365 (1997)
18. Qin, X., Lee, W.: Statistical causality analysis of infosec alert data. In: 6th Inter. Symp. Recent Advances in Intrusion Detection, pp. 73–93. Springer (2003)
19. Roesch, M.: Snort: Lightweight Intrusion Detection for Networks. In: 13th Conference on Systems Administration, pp. 229–238. USENIX (1999)
20. Snapp, S., et al.: DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype. In: 14th National Computer Security Conferenc, pp. 167–176 (1991)
21. Spafford, E., Zamboni, D.: Intrusion detection using autonomous agents. J. Comp. Net. 34, 547–570 (2000)
22. Swets, J.: The Relative Operating Characteristic in Psychology. In: Science 182, pp. 990-1000 (1973)
23. U.S. Department of Commerce. National Vulnerability Database. `http://nvd.nist.gov/`
24. U.S. Department of Health & Human Services: Health Information Privacy: Breaches Affecting 500 or More Individuals, http://www.hhs.gov/ocr/privacy/hipaa/administrative/breachnotificationrule/postedbreaches.html
25. Valdes, A., Skinner, K.: Probabilistic Alert Correlation. In: 4th Inter. Symp. Recent Advances in Intrusion Detection, pp. 54–68. Springer (2001)
26. Vigna, G., Kemmerer, R.: NetSTAT: A Network-based Intrusion Detection System. J. Comp. Sec. 7, 37–71 (1999)
27. Wing, J.: Scenario graphs applied to network security. In: Qian, Y., Tipper, D., Krishnamurthy, P., Joshi, J. (eds.) Information Assurance: Dependability and Security in Networked Systems. Morgan Kaufmann, San Francisco (2007)

Appendix: Algorithms

Algorithm 1 BN-Structure-Update $(message, A)$

Input: message m = ($number$, $srcIPaddr$, $destIPaddr$, $portnumber$, $action$, $ruletype$) . This input represents an addition, change, or deletion of a firewall rule; Adjacency matrix representation of Bayesian network $BNet = (V, E)$ consists of a $|V|x|V|$ matrix $A = (a_{ij})$ such that $a_{ij} = 1$ if $(i, j) \in E$ otherwise $a_{ij} = 0$

Output: V_a = set of nodes to add, V_d = set of nodes to delete, E_a = set of edges to add, E_d = set of edges to delete

1: //case when a rule is added
2: **if** $ruletype$ = add **then**
3: **if** $srcIPaddr : *$ in A **then**
4: add all $(parents(srcIPaddr : *), srcIPaddr : *)$ to E_a
5: **end if**
6: **if** $destIPaddr : port$ in A **then**
7: add all $(destIPaddr : port, children(destIPaddr : port))$ to E_a
8: **else**
9: add $E_a \leftarrow (srcIPaddr : *, destIPaddr : port)$
10: **end if**
11: **end if**
12: // case when a rule is deleted
13: **if** $ruletype$ = delete **then**
14: add $E_d \leftarrow (srcIPaddr : *, destIPaddr : port)$
15: **if** $srcIPaddr : *$ in A **then**
16: **if** not$parents(srcIPaddr : *)$ **then**
17: add $V_d \leftarrow srcIPaddr : *$
18: **else**
19: add all $(parents(srcIPaddr : *), srcIPaddr : *)$ to E_d
20: **end if**
21: **end if**
22: **if** $destIPaddr : port$ in A **then**
23: **if** not$children(destIPaddr : port)$ **then**
24: add $V_d \leftarrow destIPaddr : port$
25: **else**
26: add all $(destIPaddr : port, children(destIPaddr : port))$ to E_d
27: **end if**
28: **end if**
29: **end if**
30: // check if new edge creates a path to the end goal and if node creates a cycle
31: **for all** $address : port \in V \cup V_a$ **do**
32: run DFS from $address : port$
33: **if** not$(address : port \rightsquigarrow V_{CA})$ **then**
34: remove $address : port$ from V_a
35: **end if**
36: add $backedges$ to E_d
37: **end for**
38: // convert address:port node to address:port:vulnerability node
39: **for all** $address : port \in V_a$ **do**
40: **if** $vulnerability(address : port) \in NVD$ **then**
41: update $address : port$ to $address : port : vulnerability(v_i)$ in V_a and E_a
42: **else**
43: remove $address : port$ from V_a
44: **end if**
45: **end for**
46: **for all** $address : port \in V_d$ **do**
47: search BNET and replace for corresponding $address : port : vulnerablity(v_i)$
48: **end for**
49: return V_a, V_d, E_a, E_d

Algorithm 2 BNet-CPT-Update (V_a, V_d, E_a, E_d)

Input: V_a = set of nodes to add, V_d = set of nodes to delete, E_a = set of edges to add, E_d = set of edges to delete
Output: S_{CPT} = set of CPTs to update

for all $v_i \in V_a$ **do**
  new $Prob(v_i) = CVSS(v_i)/10$
  add each $outedge(v_i) \in E_a$
  for all $children(v_i)$ **do**
    update CPT using $max(newProb(v_i) + \Delta, oldProb(v_i))$
  end for
end for
for all $(v_i, v_j) \in E_a$ **do**
  new $Prob(v_i) = CVSS(v_i)/10$
  add each $(v_i, v_j) \in E_a$
  for all $children(v_i)$ **do**
    update CPT using $max(newProb(v_i) + \Delta, oldProb(v_i))$
  end for
end for
for all $v_i \in V_d$ **do**
  new $Prob(v_i) = CVSS(v_i)/10$
  remove all $inedge(v_i)$ and $outedge(v_i)$
  for all $children(v_i)$ **do**
    update CPT using $max(newProb(v_i) + \Delta, oldProb(v_i))$
  end for
end for
for all $(v_i, v_j) \in E_d$ **do**
  new $Prob(v_i) = CVSS(v_i)/10$
  remove all $(v_i, v_j) \in E_d$
  for all v_j **do**
    update CPT using $max(newProb(v_i) + \Delta, oldProb(v_i))$
  end for
end for

Algorithm 3 Sensor-Reconfiguration $(E, Detectors_{existing})$

Input: $E = evidence$, represented by set of alerts received; $Detectors_{existing}$ = set of detectors currently enabled
Output: set of nodes to enable/disable. Nodes correspond to $< address, port, vulnerability >$ tuple so can be mapped to a detection sensor

compute $a = Prob(critical\ asset\ |E)$
if $a > threshold$ **then**
  Create set of candidate sensors close to E and *critical asset*
  Run $FPTAS(BN)$
end if
$Detectors_{disable} = |Detectors_{existing} - Detectors_{FPTAS}|$
return $Detectors_{FPTAS}, Detectors_{disable}$