Using Bayesian Networks for Cyber Security Analysis

Peng Xie*, Jason H Li*, Xinming Ou†, Peng Liu‡, Renato Levy*
*Intelligent Automation Inc. Rockville, MD, USA, email:{pxie, jli, rlevy@i-a-i.com}
†Kansas State University, Manhattan, KS, USA, email:xou@ksu.edu
‡ Penn State University,University Park, PA, USA,email: pliu@ist.psu.edu

Abstract

Capturing the uncertain aspects in cyber security is important for security analysis in enterprise networks. However, there has been insufficient effort in studying what modeling approaches correctly capture such uncertainty, and how to construct the models to make them useful in practice. In this paper, we present our work on justifying uncertainty modeling for cyber security, and initial evidence indicating that it is a useful approach. Our work is centered around near real-time security analysis such as intrusion response. We need to know what is really *happening, the scope and severity level, possible consequences, and potential countermeasures. We report our current efforts on identifying the important types of uncertainty and on using Bayesian networks to capture them for enhanced security analysis. We build an example Bayesian network based on a current security graph model, justify our modeling approach through attack semantics and experimental study, and show that the resulting Bayesian network is not sensitive to parameter perturbation.*

1 Introduction

To carry out enterprise security analysis, graphical models capturing relationships among vulnerabilities and exploits have become the main-stream approach [3], [13], [18], [21]. An attack graph illustrates possible multi-stage attacks in an enterprise network, typically by presenting the logical causality relations among multiple privileges and configuration settings. Such logical relations are *deterministic*: the bad things will certainly happen in their worst forms as long as all the prerequisites are satisfied, and no bad things will happen if such conditions do not hold. While it is important to understand such logical relations, the deterministic nature has limited their use in practical network defense, especially when the graphical models are to be used in real-time intrusion response.

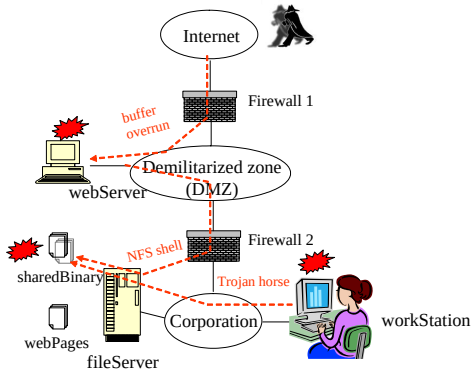


Fig. 1. An example attack scenario.

Let us look at an example, shown in Fig. 1, which is taken from Ref. [21]. Suppose the following potential attack paths are discovered after analyzing the configuration. An attacker first compromises `webServer` by remotely exploiting vulnerability `CVE-2002-0392` to get local access on the server. Since `webServer` is allowed to access `fileServer` through the NFS protocol, he can then try to modify data on the file server. There are two ways to achieve this. If there are vulnerabilities in the NFS service daemons, he can try to exploit them and get local access on the machine; or if the NFS export table is not set up appropriately, he can modify files on the server through the NFS protocol by using programs like NFS Shell[1]. Once he can modify files on the file server, the attacker can install a Trojan-horse program in the executable binaries on `fileServer` that is mounted by machine `workStation`. The attacker can now wait for an innocent user on `workStation` to execute it and obtain control on the machine. A portion of the corresponding attack graph is shown in Figure 2.

The node p_4 and its parents p_1, p_2, p_3 express the

1. A program that provides user-level access to an NFS server (ftp://ftp.cs.vu.nl/pub/leendert/nfsshell.tar.gz)

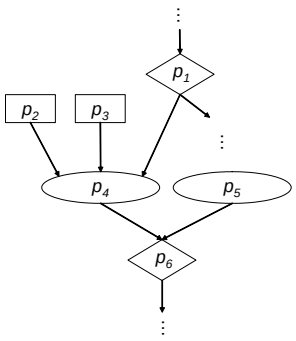


Fig. 2. A portion of the example graph.

causality relation in the NFS Shell attack: if an attacker compromises the web server (p_1), the web server can access the file server through the NFS protocol (p_2), and the file server exports a partition to the web server (p_3), then the attacker will be able to launch an NFS Shell attack to access files on the file server (p_4). Suppose we want to use this piece of information in real-time security analysis. When we suspect the web server has been compromised, with how much confidence can we say that the files on the file server have been compromised? The answer is far less certain than the deterministic logic can provide. How can we know whether the attacker has chosen to launch this attack? Even if he did so, how can we know that the attack has succeeded? Moreover, how can we account for the real-time observations that may be relevant? For example, a file system integrity checker such as Tripwire [16] may report that certain files have been modified. How shall we update our belief about possible attacks given this observation?

Real-time security analysis is a far more imprecise process than deterministic reasoning. We do not know the attacker's choices, thus there is the uncertainty from unknown attacker behaviors. Cyber attacks are not always guaranteed to succeed, thus there is the uncertainty from the imperfect nature of exploits. The defender's observations on potential attack activities are limited, and as a result we have the uncertainty from false positives and false negatives of intrusion detection system (IDS) sensors. Nevertheless, the logical causality encoded in a deterministic attack graph is invaluable to understand security events, and will be useful for building practical network defense tools if we can *appropriately* account for the uncertainty inherent in the reasoning process.

Recent years have seen a number of attempts at using Bayesian networks to model such uncertainty in security analysis [2], [10], [11], [12]. A Bayesian network (BN) is a graphical representation of cause-and-effect relationships within a problem domain. More formally, a Bayesian network is a Directed Acyclic Graph (DAG) in which: the nodes represent variables of interest (propositions); the directed links represent the causal influence among the variables; the strength of an influence is represented by conditional probability tables (CPT). For example, if we imagine that the graph structure in Figure 2 is a Bayesian network, then node p_4 could have the following CPT associated with it.

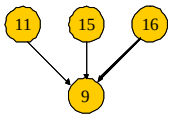
p_1	p_2	p_3	p_4
T	T	T	0.8
otherwise			0

Essentially this CPT indicates that if all of p_4's parents are true, the probability of p_4 being true is 0.8; in all other cases the probability is 0 (p_4 is false).

Bayesian Network is a powerful tool for real-time security analysis **if a BN model can be built that reflects reality.** However, it is not trivial to construct a Bayesian Network from an attack graph.

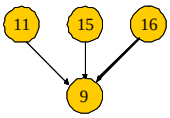
First, it is difficult to model the uncertainty inherited in security analysis. For example, we know that due to the uncertainty from the attacker's choice, p_4 may not become true after all of p_1, p_2, and p_3 are true simply because the attacker did not choose to launch the attack. There may be other reasons why p_4 does not become true after all its parents are true — for example, the attacker may have chosen to launch the attack but the attack failed due to the difficult nature of the exploit. Such uncertainty will have to be encoded in the same CPT associated with p_4. Thus the CPT number 0.8 will have a number of contributing factors in it, which makes the generation and maintenance of the CPT parameters a difficult task. For example, when we see the same attack activity in other parts of the network, we may want to increase the likelihood that an attacker may choose to use this attack. But in the unmodified graph structure there is no easy way to separate this attacker-choice uncertainty from the other factors in the CPT number of 0.8. As a result this type of correlation cannot be conducted elegantly. This is just one example problem we have discovered in the literature on building BN models from attack graphs for security analysis. We believe a more disciplined BN construction methodology is needed.

Second, cyber security analysis, unlike other more well-behaved problem domains, does not naturally lend itself to statistical analysis. In general, we do not have the ground truths in real traces from which we can learn the large number of CPT parameters, and the attackers are constantly adapting. As a result, the CPT parameters need to be produced from often vague and subjective judgments. However, it is infeasible to ask a human expert to assign every CPT parameter for every BN model. The vast majority of these numbers need to be computed automatically from various sources that reflect various types of uncertainty in cyber security. A BN model that modularizes and separates the various types of uncertainty will make this process easier. Since those numbers are imprecise in nature, the results of BN analysis should not



11	15	16	9
T	T	T	1
otherwise			0

Fig. 3. Attack structure and CPT at node 9.



11	15	16	9
T	T	T	0.8
otherwise			0

Fig. 4. Attack structure and modified CPT that captures uncertainty.

be too sensitive to CPT parameters.

While previous studies have proposed various ways of building BN models from attack graphs, there are a number of potential problems in the current approaches. In this paper, we present a BN modeling approach that we believe possesses the following properties:

1) The graphical structure shall modularize: it should separate various types of uncertainty and avoid mingling different types of uncertainty in the same CPT.
2) The majority of CPT parameters shall be computed automatically from realistic data sources.
3) The BN model should not be too sensitive to perturbation on the CPT parameters.

How to build a BN model for practical security analysis is a non-trivial problem. Extensive research must be done to justify the BN modeling approach and to study its applicability in real-world security analysis.

2 Capturing uncertainty in security analysis

In this section we provide a taxonomy of uncertainty in cyber security, describe what we believe the best way to capture them in a BN model, and explain how they can be used in real-time security analysis.

2.1 Uncertainty in attack structure

Figure 3 shows another portion of the full attack graph. Let us look at the following nodes. 1) Node 11: The attacker obtains network access to `webServer` on tcp/80; 2) Node 15: The program `httpd` is a service running on `webServer` as user `apache`, listening on tcp/80; 3) Node 16: The vulnerability `CAN-2002-0392` exists in the `httpd` program on `webServer`; and 4) Node 9: The attacker obtains code execution privilege on `webServer`. The relationship of these nodes is simple: "nodes 11, 15, and 16 altogether enable node 9". Hence, we can obtain the basic attack structure, as shown in Fig. 3. The logic AND can be represented using Bayesian network techniques via the conditional probability table (CPT) stored at node 9.

Essentially, attacks can only happen by obeying both of the two mandates: 1) **Physical path:** attacks can only occur by following network connectivity and reachability; this is the physical limit for attacks. 2) **Attack structure:** attacks can only happen by exploiting some vulnerability, with pre-conditions enabling the attacks and post-conditions as the consequence (effect). Careful inspection reveals that almost all attack graphs to date embed the physical path and attack structure information in the models, though the graph generation algorithms themselves may or may not have considered doing so explicitly. Furthermore, it is noted that while the physical paths are obviously network specific, the attack structure can be abstract knowledge without encoding any network specific information (*e.g.* particular hosts). Therefore, the abstract knowledge can be modeled and managed independent of specific networks.

The attack structures contain inherent uncertainty, since most attacks do not have 100% guarantee of success. Given that nodes 11, 15, and 16 are all true, is it absolutely the case that node 9 is achieved by an attacker? More generally, knowing that there is a vulnerability in a network service accessible to an attacker, can the attacker absolutely obtain privilege on the server? In reality, the answer is often mixed. For example, National Vulnerability Database (NVD) [1] publishes a large number of software vulnerabilities, many of which are categorized as remote vulnerabilities that can cause privilege escalation. But undoubtedly there are variations in the difficulty of exploit among those vulnerabilities. For a particular vulnerability, such as `CAN-2002-0392` in the example, we may know that a working exploit is already publicly available and it works most but not all of the time. Given that, maybe we should change the CPT accordingly, as shown in Fig. 4.

There exist already standardized metrics on the exploit difficulty of vulnerabilities. For example, CVSS [19], [23] is a standard for specifying vulnerability attributes. The base metric of *Access Complexity (AC)* describes the complexity of exploiting the vulnerability and can take the values of "high", "medium", or "low". This metric indicates the success likelihood of an exploit when all the necessary pre-conditions are met and an attacker launches the exploit. The AC metric is part of the Basic metrics in CVSS which are already maintained by NVD for every reported vulnerability. Hence we can use this existing data source to derive the CPT parameter (Table 1). Another relevant CVSS metric is the *Exploitability (E)* metric from the Temporal category. This metric describes the current state of exploit and can take the values of "unproven", "proof-of-concept", "functional", or "high". The E metric may change over time when new exploits are published or new attack data are collected. This metric would also be useful to derive the CPT parameter — a vulnerability with "high" exploitability is more likely to yield a successful attack than a "proof-of-concept" exploitability. However, NVD currently does not maintain any Temporal metric, including the E metric. These CVSS metrics are good

TABLE 1. CVSS AC metrics and success likelihoods

AC metric	Success Likelihood
high	25%
medium	75%
low	85%

TABLE 2. Discrete probability levels

Name	Value
certain	100%
probable	85%
expected	75%
unlikely	25%
improbable	15%
impossible	0%

sources to derive the CPT parameters for attack structures that involve exploiting software vulnerabilities. We can specify a function that maps the vulnerability's AC and/or E metric to the CPT parameter of the corresponding node in the BN, like node 9 above. We currently only use the AC metric in Table 1.

For attack structures that are not about exploiting software vulnerabilities, we can specify the likelihood of success directly in the attack-structure knowledge base. We also use discrete levels similar to those found in CVSS metrics, as shown in Table 2. For example, the attack structure that leads to node p_4 in Figure 2 is an NFS Shell attack. For this specific type of exploit we can estimate its success likelihood when all the preconditions are met. Here the number 75% will be used as the CPT parameter. We believe that providing these numbers in such a discrete manner is reasonable, since the numbers are already imprecise: what is the difference between "75%" and "80%" to a human? In Section 4 we demonstrate that the resulting BN is not sensitive to input parameter purturbation, further justifying the use of discrete levels in deriving CPT parameters.

2.2 Uncertainty about attacker actions

This is the unique and perhaps the biggest uncertainty in real-time security analysis. Suppose for the simple attack structure (as shown in Fig. 3 and 4) we have used CVSS to derive the success likelihood of the attack. Then can we use that number as the CPT parameter? If what we want to know is "what **could** happen" then the answer is yes. This is the typical kind of questions asked during pre-deployment planning phase, and the Bayesian network model can sufficiently answer them. However, the above node structure is not sufficient for real-time analysis. In real-time analysis, even when all the prerequisites become true, there may not be an attacker there. For analogy, if the door is open to a potential attacker, the attack may not happen until an attacker approaches the door. Since what we care about in real-time analysis is "what's **really** happening", the key difference from pre-deployment planning is that we need to model whether an attack is happening or not. This is the unique uncertainty inherent in real-time analysis. To this end, we introduce a new kind of node in our Bayesian network models, called the attack action node (AAN).

An attack action node is introduced as an additional parent node for those important attacks (Fig. 5). An AAN is an artifact introduced by the modeler for better modeling power and clarity. "AAN is true" means the attacker action is present, and other prerequisites become effective. "AAN is false" means no attacker action is present. This will "block" all other prerequisites from being effective. In other words, the CPT at node 9 will have a zero probability for all rows where AAN is false, as shown below.

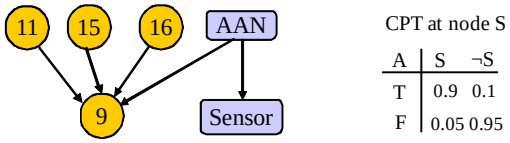
11	15	16	AAN	9
T	T	T	T	0.8
otherwise				0

Not all attack nodes need an AAN; typically only those "important" nodes in Bayesian network models should be equipped with AAN nodes. For example, those first (or very early) stepping stones in multi-step attacks should have AAN nodes associated with them to indicate whether or not attacks are ongoing. As another example, an AAN is not necessary when a privilege does not need an attacker to take any action, *e.g.* a privilege that can be "naturally" obtained as a result of NFS file-sharing semantics.

The next question is: how can one obtain information about the AAN states? Knowing whether or not the attack is ongoing will greatly help the subsequent reasoning process. First, the CPT at the AAN node represents the prior likelihood of an attack. This number can be set globally by the user. For example, if this type of attack has been seen recently, the user may decide to increase the AAN node's prior likelihood for all such attacks to indicate an increased threat level. Second, many security monitoring systems can provide evidence of possible attack activities and these observations indicate an increased posterior likelihood of the attack. To model this correlation we introduce a sensor node as the child of an AAN whenever a sensor is available that can report potential attack activity of this sort. In our Bayesian network models, which focus on high-level reasoning rather than low-level data processing, the sensor node can hardly be a physical tell-tale sensor. Most likely it captures aggregated results from low-level sensors (e.g. IDS sensors), which indicate the presence of certain attacks. The reliability of the sensor node is reflected in its local CPT, with false positive and false negative probabilities explicitly expressed (Fig. 5). This is one example of the local observation model to be discussed in the next section.

2.3 Uncertainty about alerts

It is well known that alerts coming from intrusion detection systems tend to have some amount of false positives. In this work, we will not model raw alerts



CPT at node S

A	S	¬S
T	0.9	0.1
F	0.05	0.95

Fig. 5. Uncertainty related to attacker action.

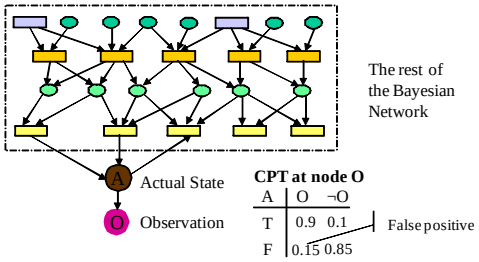


Fig. 6. Local observation model.

directly. Instead, we will only input relevant correlated alerts that can help high-level reasoning. Nonetheless, the correlated alerts may still contain a fair amount of false positives. If for some reason we know that the correlated alerts come from a high-fidelity alert correlation process, we may impose high confidence level upon them. Lower confidence will be put otherwise.

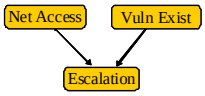
We propose to use a *local observation model (LOM)* to model such uncertainty about alerts. As shown in Fig. 6, for states that can be inferred from imperfect sensors, we introduce a pair of nodes: the `ActualState` node and the `Observation` node. The `ActualState` node is not observable itself. The `Observation` node is a direct child of the `ActualState` node, and the `Observation` node provides observations to infer the true state of the `ActualState` node. Suppose both the `ActualState` node and the `Observation` node are binary, and the CPT associated with the `Observation` node represents how the `ActualState` node will affect the `Observation` node. In this CPT, a false positive probability is inherently included.

A concrete evidence about node `Observation` will change the posterior probability of node `ActualState` by computing P(ActualState|Observation=True). This kind of "backward" computation is routine in hidden Markov models (HMM), and Bayesian networks can naturally execute such kind of inference. Further, such computations can be executed in some fairly efficient manner [15][22].

2.4 Modularized CPT computation

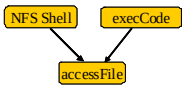
There are well-studied BN modeling techniques that can modularize various sources of uncertainty in the computation of CPT parameters. We provide two examples that may be directly applied to cyber security analysis.

The first example is called "Noisy-And", and it extends from the deterministic AND logic. With deterministic AND (see Fig. 7), node `Escalation` will become True



Net Access	Vulnerability Exist	P(Escalation)	P(¬Escalation)
F	F	0.08=0.2×0.4	0.92
F	T	0.4	0.6
T	F	0.2	0.8
T	T	1	0

Fig. 7. Noisy-And example.



NFS Shell	execCode implies file access	P(accessFile)	P(¬accessFile)
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.7	0.3
T	T	0.94	0.06=0.2×0.3

Fig. 8. Noisy-Or example.

only when both its parents `NetAccess` and `VulnExist` are True. This says that `Escalation` will never happen otherwise. A Noisy-And however does not imply that a child is definitely false if one of the parents is false.

To model the "leaky" chances that `Escalation` may still happen without requiring all of the parents to be True, "leaky" parameters are introduced. In particular, each parent has an associated (enabling) influence to the child that is represented by a probability. For example, suppose the vulnerability scanner does not report any findings (`VulnExist` is False). In practice, however, there could be zero-day vulnerabilities in a piece of software. Let 0.2 represent the likelihood of the existence of zero-day vulnerability in the software under concern. In other words, 0.2 is the leaky chance for the vulnerability to be actually true (though reported False). Hence, the leaky parameter $P(E|\neg V) = 0.2$ represents the likelihood that vulnerability scanner misses a true vulnerability. We can define another leaky parameter $P(E|\neg N)$ = 0.4, which could mean the likelihood the attacker is able to circumvent the firewall to gain network access, even when attack-graph analysis shows that there is no network path.

The second example is called "Noisy-Or" and it extends from the deterministic OR logic. With deterministic OR (see Fig. 8), node `accessFile` is True as long as one or more of its parents become True. A Noisy-Or logic however does not imply that a child is definitely true if one of the parents is true.

As in Noisy-And, the leaky parameters are introduced to model the "leaky" chances that `accessFile` may not always be True when one or more of its parents is True. For example, in Fig. 8, let $P(\neg accessFile|NFSShell) = 0.3$, which means that `NFSShell` being True does not necessarily imply that `accessFile` is True; there is still a 30% chance that `accessFile` will not happen. Similarly, we can define another (inhibiting) leaky parameter as $P(\neg accessFile|execCode) = 0.2$.

It is noted that in "Noisy-And" and "Noisy-Or" logic, the leaky parameter is defined separately and indepen-

dently. This independence assumption simplifies the specification of the parameters and enables simple and efficient calculation of the probability distribution. As shown in Fig. 7 and Fig. 8, the CPTs at nodes `Escalation` and `accessFile` have 8 entries. However, just *two* leaky parameters are specified; all other entries can be easily computed from these parameters. For instance, $P(E|N = F, V = F) = 0.2 \times 0.4 = 0.08$. Further, the independent assessment of the leaky parameters is more intuitive for human experts, since humans are known to perform relatively better in a "case-by-case" manner. It would be an extremely daunting task for human experts to assess situations considering different combinations of multiple factors, which is exponential with the number of parents and also non-intuitive. The independence assumption alleviates the difficulty.

2.5 Summary

Our BN modeling approach separates three important types of uncertainty in real-time security analysis: the uncertainty on attack success, the uncertainty of attacker choice, and the uncertainty from imperfect IDS sensors. This enables computing CPT parameters from existing data sources such as NVD/CVSS. The more advanced BN modeling techniques such as Noisy-And and Noisy-Or can further modularize the sources of uncertainty within a CPT. Our BN modeling approach satisfies the first two desirable properties described in Section 1.

3 Implementation

In this section we describe how to build a Bayesian network from an attack graph tool. We use the MulVAL attack graph toolkit [21] for our implementation, but the approach can extend to other attack graphs with similar semantics [8], [13], [26]. The MulVAL reasoning system can incorporate CVSS metrics from NVD data sources and output the AC metric. We use the same example as in Section 1 to describe how we derive the BN structure from the attack graph.

3.1 Adding new nodes

Attack Action Node (AAN). As discussed in section 2.2, we need to introduce the AAN node to model the existence of an attacker actively exploiting the system. Thus, for some nodes in the graph model that represent consequences of an attack, we may put an AAN node as its parent. The attack's post-condition will become true only if all its pre-conditions are met *and* the AAN node is true. This changes the graph's semantics from "what could happen" to "what has happened". A separate AAN is used for each selected attack node, rather than sharing a single AAN node among multiple attack nodes. This is because the attacker may choose one of many possible attack paths.

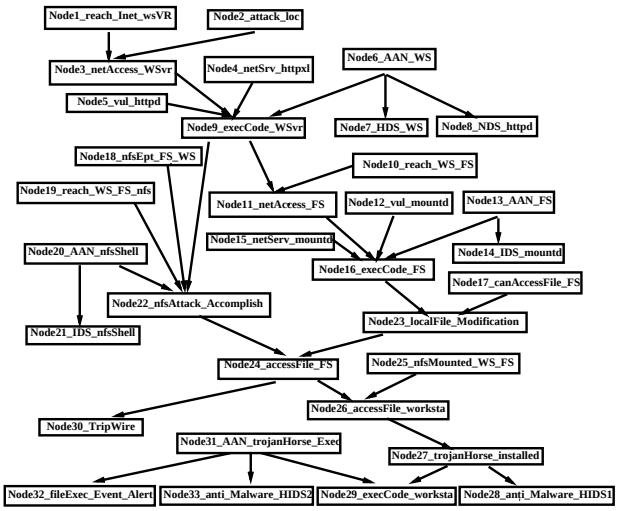


Fig. 9. An example Bayesian network model.

Local Observation Model (LOM). Section 2.3 introduces the notion of local observation model that can be used to incorporate the various detectors used in cyber security, such as IDS. In real-time security analysis, there are methods to monitor and detect potential security threats. For example, an IDS could be used to detect the existence of an attacker and a file system monitor such as Tripwire can be used to detect file-system modifications. However, the observation or detector may not always be accurate. In our LOM, a new node is introduced to model the inaccurate observation (detectors), and an arc from the actual state to the observation state represents the fact that the observation is influenced by the actual state.

Fig. 9 shows the generated BN structure for the attack scenario shown in Fig. 1. Using conditional probabilities (and Noisy-And and Noisy-Or semantics), there is no distinction between AND/OR nodes any more. Node 22 indicates that the NFS Shell attack against the file server has been accomplished by the attacker. Obviously, if there is no attacker, no attack can be accomplished. So we introduced an AAN node for node 22. When an attacker is present and sending the file server an NFS shell exploit packet, this action could be detected by a network-based IDS, such as Snort. Thus we introduced an LOM observation node 21 so that whenever the IDS reports a suspected NFS exploit packet, this node will be true. If the attacker successfully modified the files through the exploit, node 24 will be true. In this case, a Tripwire monitor could report a suspicious file modification. Thus we added an LOM observation node *Tripwire* to capture this event.

3.2 Determining the CPT tables

Each node in a Bayesian network needs to be associated with a CPT which is the probability distribution of the node's possible states conditioned on the parents' states. For a node with no parent (root nodes), the CPT is

the node's prior probability distribution. By adopting the Noisy-Or and Noisy-And techniques discussed in Section 2, the CPT computation can be reduced to obtaining certain likelihood values associated with individual conditions, such as the difficulty level of exploiting a vulnerability, the likelihood that a piece of software contains a zero-day vulnerability, *etc*. Many of them can be derived from currently-available data sources, such as the National Vulnerability Database, which provides various metrics for security vulnerabilities in the CVSS format. In our implementation, we assign the CPT values based on the values from Table 2. We use the exact values corresponding to the likelihood such as `certain` and `impossible` and use the middle values of the ranges corresponding to other likelihood. For example, we use value 20% to reflect the likelihood for `unlikely`.

However, there are still CPT entries that would rely on human experts to fill in, such as the false positive and false negative rates for IDS detectors, the a priori likelihood for attacks, etc. We can have the experts specify those conditional probabilities using the discrete values given in Table 2. In this work, we assign those levels manually based on our understanding of the security problems.

4 Experimental Results

Although the CPT tables of a BN-based security analysis tool are often determined by human experts, the effectiveness of the BN-based security analysis tool must be evaluated in an *objective* way.

4.1 Evaluation Methodology

In our experiments, we want to check if the BN-based tool can help the security administrator in security analysis. We evaluate the performance of a BN-based tool by comparing its outputs with a Referee's. The Referee knows all the ground truth and is absolutely correct. In our experiments, the ground truth is a fully ordered sequence of events; the Referee has complete and perfect knowledge on which events are malicious and which events are legitimate. The order of the events is determined by the start time of the events. For simplicity, we assume that each event will be instantly finished. In this way, we ignore the differences among event durations. The BN-based tool can only access the information generated by IDS sensors, which is already readily available to the security administrator. Note that in our experiments, the BN-based tool actually sees a distorted version of the ground truth since the sequence of the events witnessed by each IDS sensor may be different from the ground truth due to the sensor's false positive, false negative and detection latency. Moreover, the errors in firewall rules and (Nessus) vulnerability reports also contribute to distort the ground truth.

In our experiments, we ask the BN-based tool two questions at proper time points:

(Q1) Which machines are very likely to have been compromised?

(Q2) Which exploits have happened but not been detected yet? What alerts are missing?

These two questions are typically asked by the security administrator. We evaluate the BN-based tool by comparing its answers with the ground truth.

TABLE 3. Pre-Deployment Ground Truth

Label	Vulnerability
Node 1	reachability (Internet, webService, TCP,80)
Node 4	networkServiceInfo(webServer, httpd,tcp,80,apache)
Node 5	VulExists(webServer,'CAN-2002-0392',httpd,remoteExploit, privEscalation
Node 10	reacability(webserver,fileserver,rpc,100005
Node 12	vulExists(fileserver,vulID,mountd,remoteExploit,privEscalation)
Node 15	networkServiceInfo(fileServer,mountd,rpc,100005,root)
Node 17	canAccessFile(fileServer,root,write,'/export')
Node 18	nfsExportInfo(fileServer,'/export',write,webServer)
Node 19	reachability(webserver,fileServer,nfsProtocol,nfsPort)
Node 25	nfsMounted(workstation,'/usr/local/share',fileServer,'/export',read)

TABLE 4. Good events and attack events

Event ID	Event
Event 1	Mallory (i.e., the attacker) sends probing packet #B1 (after TCP 3-way handshake) to port 80 of webServer, but packet #B1 fails.
Event 2	Good packet #G1 gets into port 80 of webServer.
Event 3	Good packet #G2 gets into port 80 of webServer.
Event 4	Mallory sends probing packet #B2 to webServer, but packet #B2 fails.
Event 5	Good packet #G3 gets into port 80 of webServer.
Event 6	Good packet #G4 gets into port 80 of webServer.
Event 7	Good packet #G5 gets into port 80 of webServer.
Event 8	Mallory sends probing packet #B3 to webServer; packet #B3 succeeds.
Event 9	Mallory sends probing packet #B4 to the RPC port of fileServer, but packet #B4 fails.
Event 10	Good packet #G6 gets into the RPC port of fileServer.
Event 11	Mallory sends probing packet #B5 to the rpc port of fileServer; packet #B5 succeeds. The network is now in the state specified by Node 23.
Event 12	Good packet #G7 gets into the RPC port of fileServer.
Event 13	Good packet #G8 gets into the RPC port of fileServer.
Event 14	Good packet #G9 gets into the RPC port of fileServer.
Event 15	Binary file X in directory "/export" is changed by a good user.
Event 16	Binary file X in directory "/export" is changed by another good user.
Event 17	Mallory changes file X in directory "/export" to install a Trojan horse.
Event 18	Binary file Y in directory "/export" is changed by a good user.
Event 19	File X, the Trojan horse, is executed by admin. The Trojan horse executes code on workStation with root privilege.
Event 20	Binary file Y in directory "/export" is changed by another good user.
Event 21	File Y is executed by a regular user.
Event 22	Binary file Z in directory "/export" is changed by another good user.
Event 23	File Z is executed by a regular user.

4.2 Experiment Settings

In our experiments, we adopt the attack scenario shown in Fig. 1 and the corresponding BN shown in Fig. 9. We have two types of ground truth: Pre-Deployment Ground Truth, which addresses the pre-deployment vulnerabilities, and Post-Deployment Ground Truth, which focuses on the post-deployment attack events. The Pre-Deployment Ground Truth is shown in Table 3. Note that Node 18 is a false vulnerability report which is mistakenly reported by an imperfect vulnerability scanner.

The Post-Deployment Ground Truth includes two types of events: attack events and good events. In our settings, each experiment will involve different alert events, but all the experiments will in fact have the same sequence of interleaved attack events and good events. The good events and attack events adopted in our experiments are listed in Table 4. In our experiments, we adopt nine alerts and one false negative as shown in Table 5. Note that AE4 is a false positive alert.

TABLE 5. Alert events

Label	Semantics
AE 1	against Event 1: saying that packet #B1 matches a signature compromising webServer.
AE2	against Event 8: saying that packet #B3 matches a signature compromising webServer.
AE3	against Event 8 and #B3. However, due to detection latency, this alert is raised after Event 13.
FN1	False Negative against Event 11: the sensor did not raise any alert about #B5.
AE4	false positive: saying that webServer runs a malicious NSF shell.
AE5	against event 15: saying that file X in directory "/export" is changed.
AE6	against event 16: saying that file X in directory "/export" is changed.
AE7	against event 17: saying that file X in directory "/export" is changed.
AE8	against event 17: saying that file X is a Trojan horse.
AE9	against event 19: saying that Trojan horse is being executed.

4.3 Simulation Experiments

Summary of all results: We have run six simulation experiments. In each experiment, we adopt a different sequence of ground truth and alert events. In other words, effects of imperfect IDS on the BN-based tool in these experiments are different.

Through the six experiments, we compare the answers from the BN-based tool to questions Q1 and Q2 with the ones from the Referee. Even though the BN-based tool does not always give the perfect answers, most of the answers given by the BN-based tool is reasonably close to the ground truth. Moreover, with more and more ground truth revealed, the answers from the BN-based tool are more and more closer to the truth. Furthermore, the BN-based tool can effectively infer the missed false negative alarm and mitigate the disturbance caused by an imperfect IDS.

Experiment 1: In this experiment, we use the following complete sequence of events: E1 → AE1 (report, AI1) → E2 → E3 → E4 → E5 → E6 → E7 → E8 → AE2 (do not report, AI2) → AE3 (do not enter, AI3) → E9 → E10 → E11 → FN1 (node 14, do not report, AI4) → E12 → E13 → AE3 (report, AI5) → E14 → E15 → AE5 (report, AI6) → E16 → AE6 → E17 → AE7 (do not report) → AE8 (report, AI7) → E18 → E19 → AE9 (report, AI8) → E20 → E21 → E22 → E23.

In this sequence, → represents the absolute time order between events. Command "report" means that a new evidence is visible to the BN-based tool. Command "do not report" means that no new evidence visible to the BN-based tool. "AI1" (Answer It) represents the first timepoint when the Referee asks the BN-based tool to answer Q1 and Q2. "AI2" represents the second answer-it timepoint, etc.

The results of Experiment 1: As shown in Table 6, "WEB" denotes webServer; "FS" denotes fileServer; "WS" denotes workStation; "NFS" denotes "NFS shell". In this table, each column represents (a) a timepoint when the BN-based tool is asked to answer Q1 and Q2, and (b) the corresponding answers given by the Referee and the BN-based tool at that specific timepoint.

The results from Experiment 1 show that the BN-based tool can help the security administrator to find the most likely compromised machine. As shown in Table 6, the BN-based tool gives a reasonably correct answer at timepoint AI5 when alert AE3 is reported. At this timepoint, the BN-based tool tells the likelihood of webServer being compromised is 89.92%. At timepoint AI6 when alert AE5 is visible, the BN-based tool shows the likelihood of webServer being compromised is 92.73%. Therefore, the BN-based tool are more confident about this conclusion. Moreover, the BN-based tool shows the likelihood of file server being compromised is 53.04% at timepoint AI6, a correct conclusion though a weak one. At timepoint AI7, BN's answer is more useful. When a Trojan horse is detected, the BN-based tool is able to tell which one is the real *cause* of the Trojan horse. As shown in Table 6, the BN-based tool derives that the likelihood of the NFS shell attack being the real cause is 57.53% while the likelihood of the *mountd* attack being the real cause is 68.93% which is much higher. Hence, the BN-based tool suggests that the *mountd* attack is the real cause, which is a correct conclusion. At timepoint AI8 when true alert AE9 is detected, the Referee can conclude that `workStation` has been compromised. It can be seen that the BN-based tool also gives the perfect answer (100% WS). Other probability values remain the same as those at timepoint AI7.

The BN-based tool also detects the missed alert event. In this experiment, the BN-based tool can tell that HIDS-Alert is a missed alert. As shown in Table 6, the likelihood that HIDS-Alert is true is 85.06% even though no alert is reported. Therefore, the BN-based tool can help the security administrator to infer that the HIDS-Alert is most likely missed. Nevertheless, we found that the BN-based tool gives a wrong answer at timepoint AI7 by saying that the likelihood of workStation being compromised is 60%, which is a weak false alarm.

Results of the Other Experiments: The other experiments are similar to Experiment 1 except for the following differences. (a) Experiment 2 lets false alert FN1 have substantial detection latency, i.e., it is raised after Event 18. The results show that the BN-based tool can generate useful conclusions in most timepoints. (b) Experiment 3 lets the alert AE8 have detection latency and the results show that the BN-based tool performs consistently well in identifying delayed alerts. (c) Experiment 4 shows that the BN-based tool can leverage additional types of evidence beyond alerts and vulnerability reports. The ability to use extra evidence in a handy way is a major advantage of the BN-based tool. In this experiment, the results show that correlation evidence such as AE5 and the event that file X is executed on `workStation` increases the likelihood of `workStation` being compromised. (d) Experiment 5 shows that the BN-based tool has strong capability in mitigating the disturbance generated by false positives: the BN-based tool can give a correct answer even in the presence of a false alert. In this experiment, we first let the false alert, AE4 be raised, and true alert AE5 be raised later. The

TABLE 6. The results of Experiment 1

Q1:	AI1	AI2	AI3	AI4	AI5	AI6	AI7	AI8
Referee's	None	WEB	WEB	WEB; FS	WEB; FS	WEB; FS	WEB; FS; Trojan	WEB; FS; Trojan; WS
BN's	83.13% WEB 85.06% HIDS_ALERT	83.13% WEB 85.06% HIDS_ALERT	83.13% WEB 85.06% HIDS_ALERT	83.13% WEB 41.14% FS	89.92% WEB 43.10% FS	92.73% WEB 53.04% FS; 51.39% WS 38.47% Trojan	97.23% WEB; 68.93% FS; 60% WS; 100% Trojan; 57.53% NFS shell; 68.93% FS mountd	+100.0WS

BN-based tool says that the likelihood of fileServer being compromised is 74.8%, while the likelihood of NFS Shell attack being enforced is 61.19%, which is much lower. As confirmed by the Referee, the fileServer is the correct answer. (e) Experiment 6 shows the perfect case. In this experiment, there are no false positives and false negatives, and only AE2 has insignificant latency. The results show that the BN-based tool performs extremely well in this case.

4.4 Sensitivity Analysis

In order to make the BN-based tool practical in real-time security analysis, one question must be answered: how sensitive is the BN-based tool to its CPT tables? Since the CPT tables are generated based on human expertise, they cannot be the absolute truth; instead, they are only approximate to the truth in the real world. Therefore, the BN-based tool must be robust against reasonable (small) changes on its CPT tables. In other words, the quality of the answer given by the BN-based tool should not be significantly affected by a slight changes on its CPT tables. The most desirable sensitivity analysis should be holistic, i.e., the combined effects of all related parameters should be considered in the analysis. However, such sensitivity analysis method is extremely difficult to develop. In our work, we analyze the sensitivity of the BN-based tool in an isolated way, i.e., if there are multiple parameters related to the answers, we only consider the effect of one parameter at a time while keeping others constant.

In our experiments, we have carried out extensive sensitivity analysis experiments by using the sensitivity analysis tool, called SamIam [5] to check the sensitivity of the answers from the BN-based tool to its related parameters. When the BN-based tool generates an answer with some probability, we use SamIam to reversely check the effects of related CPT tables, i.e., we change the probability associated with an answer by 5%, 10% and 15% and check the required changes on the related CPT tables.

We find out that in all these experiments, the changes in CPT tables can only result in at most the same amount of change on the answers reported by the BN-based tool. For example, in order to generate +5% change on the answer at time AI1 (83.13%), the **minimum** change required on parameters is 5%. The same holds for a -5% change.

Therefore, the BN-based tool in our experiments is not sensitive to the CPT tables in the sense that changes in CPT tables are not amplified on the answers given by the BN-based tool.

5 Related work

Bayesian network techniques have been applied to intrusion detection systems [4], [17], [25]. Our application of BN is at a different level. Our work *makes use* of the output of intrusion detectors and incorporates it into a holistic security analysis framework. Our BN model does not deal with low-level system events such as raw IP packets, system calls, *etc.*, which have already been taken care of by various types of intrusion detectors.

Frigault *et al.* [10], [11] study how to use Bayesian Network and attack graphs to measure network security risk. Their work focuses on the pre-deployment planning phase in the sense that the security metrics produced by the BN reflect the inherent risk in a network. Our BN model address a wider range of security anlysis, most importantly the problem of real-time situation awareness which must account for various types of run-time observations like IDS alerts to answer the question of "what is really going on".

Attack graphs have been widely studied in the context of enterprise security management, intrusion detection and response, and security metrics [12], [13], [9], [14], [20], [21], [27]. Our work further extends the utility of attack graphs by constructing a well-founded Bayesian network model that enables reasoning with uncertainty for situation-awareness security analysis. Dantu and Kolan also construct Bayesian networks based on attack graphs for risk management [6]. However, our Bayesian network model is fundamentally different from theirs. Dantu and Kolan use Bayesian networks to model the attacker's behaviors, whereas we use Bayesian networks to model uncertainties inherent in the causality relationships among system conditions in an attack graph as well as run-time observations. Our Bayesian networks model does not use any attacker profiles.

Tang [24] applies Dempster-Shafter (DS) theory [7] in fault-diagnosis for overlay networks. The focus of our work is on analyzing cyber attacks which has very different characteristics than faults, due to the exitence of malicious players.

6 Conclusions

Graphical models are important tools for analyzing security events in enterprise networks. Although it may seem

straightforward to combine attack graphs and Bayesian networks, one should not simply juxtapose these two and think that they will work nicely together. By doing this, either the real semantics and inference power of Bayesian networks are not fully utilized, or Bayesian networks are used in an inappropriate manner. As pointed out in previous sections, the key to using Bayesian networks correctly is to identify and represent relevant uncertainties.

In this paper, we have built an example Bayesian network model to capture uncertain relationships, and experimental results show that using Bayesian networks may bring in new opportunities for improved enterprise security analysis. To the best of our knowledge, our work is the first effort that investigates systematic approaches to combining attack graphs and Bayesian networks.

7 Acknowledgement

This work was partially supported by Army Research Office, contract W911NF-07-C-0101. Xinming Ou was partially supported by U.S. NSF under Grant No.0716665, and U.S. AFOSR under contract FA9550-09-1-0138. Peng Liu was supported by ARO MURI on Computer-aided Human Centric Cyber Situation Awareness, AFOSR MURI FA9550-07-1-0527, NSF CNS-0905131, NSF CNS-0916469, and AFRL FA8750-08-C-0137.

References

[1] NVD CVSS national vulnerability database cvss support. http://nvd.nist.gov/cvss.cfm, April 2008.

[2] Magnus Almgren, Ulf Lindqvist, and Erland Jonsson. A multi-sensor model to improve automated attack detection. In *RAID 2008*. RAID, September 2008.

[3] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS 2002*, Washington, DC, 2002.

[4] Pablo Garcia Bringas. Intensive use of Bayesian belief networks for the unified, flexible and adaptable analysis of misuses and anomalies in network intrusion detection and prevention systems. In *18th International Conference on Database and Expert Systems Applications*, 2007.

[5] Hei Chan and Adnan Darwiche. When do numbers really matters. *Journal of Artificial Intelligence Research*, pages 265–287, 2002.

[6] Ram Dantu and Prakash Kolan. Risk management using behavior based bayesian networks. In *IEEE International Conference on Intelligence and Security Informatics*, May 2005.

[7] A.P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Ann. Statistics*, 28:325–339, 1967.

[8] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *CCS'07*, 2007.

[9] Bingrui Foo, Yu-Sung Wu, Yu-Chun Mao, Saurabh Bagchi, and Eugene Spafford. Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment. In *DSN2005*, June 2005.

[10] Marcel Frigault and Lingyu Wang. Measuring network security using bayesian network-based attack graphs. In *STPSA'08*, 2008.

[11] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, 2008.

[12] Saurabh Bagchi Gaspar Modelo-Howard and Guy Lebanon. Determining placement of intrusion detectors for a distributed application through bayesian network modeling. In *RAID 2008*. RAID, September 2008.

[13] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *ACSAC 2006*, Miami Beach, Florida, December 2006.

[14] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.

[15] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[16] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *CCS'94*, 1994.

[17] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian event classification for intrusion detection. In *ACSAC'03*, December 2003.

[18] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and restoring defense in depth using attack graphs. In *MILCOM 2006*, Washington, DC, U.S.A., October 2006.

[19] Peter Mell, Karen Scarfone, and Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. FIRST'07, June 2007.

[20] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004.

[21] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *CCS 2006*, pages 336–345, 2006.

[22] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1999.

[23] Mike Schiffman, Gerhard Eschelbeck, David Ahmad, Andrew Wright, and Sasha Romanosky. *CVSS: A Common Vulnerability Scoring System*. National Infrastructure Advisory Council (NIAC), 2004.

[24] Yongning Tang and Ehab AI-Shaer. Sharing end-user negative symptoms for improving overlay network dependability. In *DSN2009*, June 2009.

[25] Alfonso Valdes and Keith Skinner. Adaptive, model-based monitoring for cyber attack detection. In *RAID'00*, 2000.

[26] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29:3812–3824, November 2006.

[27] Leevar Williams, Richard Lippmann, and Kyle Ingols. Garnet: A graphical attack graph and reachability network evaluation tool. In *VizSEC'08*, 2008.