Reasoning about Complementary Intrusion Evidence

Blind Review

***Abstract*— This paper presents techniques to integrate and reason about complementary intrusion evidence such as intrusion alerts generated by intrusion detection systems (IDSs) and reports by system monitoring or vulnerability scanning tools. To facilitate the modeling of intrusion evidence, this paper classifies intrusion evidence into either *event-based evidence* or *state-based evidence*. Event-based evidence refers to observations (or detections) of intrusive *actions* (e.g., IDS alerts), while state-based evidence refers to observations of the *effects* of intrusions on system states. Based on the interdependency between event-based and state-based evidence, this paper develops techniques to automatically integrate complementary evidence into Bayesian networks, and reason about uncertain or unknown intrusion evidence based on verified evidence. The experimental results in this paper demonstrate the potential of the proposed techniques. In particular, additional observations by system monitoring or vulnerability scanning tools can potentially reduce the false alert rate and increase the confidence in alerts corresponding to successful attacks.**

I. Introduction

It is well-known that current intrusion detection systems (IDSs) produce large numbers of alerts, including both actual and false alerts. The high volume and the low quality of intrusion alerts (i.e., missed attacks and false alerts) make it a very challenging task for human users or intrusion response systems to understand the alerts and take appropriate actions.

Several alert correlation techniques have been proposed to facilitate the analysis of intrusion alerts, including those based on the similarity between alert attributes [9], [13], [29], [32], previously known (or partially known) attack scenarios [14], [15], and prerequisites and consequences of known attacks [10], [23]. However, most of these correlation methods focus on IDS alerts, overlooking other intrusion evidence provided by system monitoring tools (e.g., anti-virus software) and vulnerability scanning tools (e.g., Nmap [4]). Since none of the above methods can perfectly construct attack scenarios due to the imperfection of the IDSs, it is desirable to include additional, complementary intrusion evidence to further improve the performance of intrusion analysis.

Several researchers recently investigated ways to consider multiple information sources during intrusion analysis [22], [25]. A formal model named M2D2 was proposed to represent data relevant to alert correlation, including characteristics of monitored systems, properties of security tools, and observed events [22]. Though quite useful for alert correlation, M2D2 does not provide a specific mechanism to automatically reason about information provided by multiple sources. Another mission-impact-based method [25] reasons about the relevance of alerts by fusing alerts with the targets' topology and vulnerabilities, and ranks alerts based on their relationships with critical resources and users' interests. Though the mission-impact based method can automate the analysis of intrusion alerts, the construction of a mission-impact based model requires substantial human intervention, and the constructed model is highly dependent on the monitored systems. Thus, it is desirable to seek other effective mechanisms that can handle complementary intrusion evidence automatically.

In this paper, we develop techniques to automatically integrate and reason about complementary intrusion evidence, including IDS alerts, reports from system monitoring or vulnerability scanning tools, and human observations.

Our approach is based on the interdependency between attacks and system states. That is, an attack may need certain system states to be successful, and will modify the system states as a result. However, IDS alerts, which represent detected attacks, are uncertain due to the imperfection of current IDSs. To reason about uncertain IDS alerts, our approach automatically builds Bayesian networks that consist of variables representing IDS alerts and system states. With additional, complementary evidence about system states provided by system monitoring tools, vulnerability scanning tools, and human observations, we can then make further inference about uncertain IDS alerts. As a result, we can increase our confidence in alerts corresponding to successful attacks, and at the same time reduce the confidence in false alerts.

The main contribution of this paper is a reasoning framework for complementary intrusion evidence. To our best knowledge, this is the first attempt to *automatically* integrate and reason about complementary intrusion evidence such as IDS alerts and vulnerability scanning reports. In addition, we also perform a series of experiments to validate our approach and gain further insights into the problem. The experimental results demonstrate the potential of the proposed approach as well as the effectiveness of our techniques.

The rest of this paper is organized as follows. The next section describes our techniques to integrate and reason about complementary intrusion evidence. Section III presents the results of our initial experiments. Section IV discusses related work. Section V concludes this paper and points out some future research directions. The appendix includes additional details about our experiments.

II. Reasoning Framework

In this section, we present our techniques to reason about complementary intrusion evidence, including IDS alerts and reports from system monitoring tools or vulnerability scanning tools. In the following, we first describe our representation of intrusion evidence, and then present the framework to reason about complementary intrusion evidence using Bayesian networks.

A. Modeling Intrusion Evidence

We classify intrusion evidence into two categories: *event-based evidence* and *state-based evidence*. Event-based evidence refers to observations (or detections) of attacks. For example, an IDS alert of a buffer overflow attack against a web server is event-based evidence. State-based evidence refers to observations of the *effect* of attacks on system states. For example, the existence of a rootkit[1] on a machine is state-based evidence indicating that the machine has been compromised.

1) System Attributes and State-Based Evidence: We follow [7], [28] to represent system states (e.g., vulnerabilities, attacker access privileges, and network connectivities) as *system attributes* (or simply *attributes*), each of which is a boolean variable representing whether the system is in a certain state or not. For example, we may use `RootkitInstalled = True` to represent that a rootkit is installed on the system of concern. Notation-wise, we use a system attribute directly to represent that it is True, and use its negation to represent that it is False. For example, we may use `RootPrivilege` to represent that an attacker has acquired root privilege on the system, and use its negation ¬`RootPrivilege` if not. There may be implication relationships between attributes. For example, `RootPrivilege` implies `FileTransferPrivilege`, which indicates that an attacker having the root privilege also has the privilege to transfer files from/to the system. Note that such a representation can be extended to include variables to provide more flexibility. For example, we may use `RootPrivilege (x)` to represent the attacker has acquired root privilege on host `x`. However, for simplicity, we do not do so in this paper.

State-based evidence consists of observations on system attributes related to possible attacks. They may be collected by vulnerability scanning tools (e.g., Nmap [4], XScan [33]), system monitoring tools (e.g., anti-virus software), or through human observations. Such system state information may be changed during running time, and such changes may be detected by monitoring/scanning tools. We refer to the change of an attribute as an *attribute alteration*. Since these attribute alterations are potentially related to attacks, the time information of them is also important for intrusion analysis. The *timestamp* of an attribute alteration is the time when the alteration is detected or inferred. Such a timestamp can be stored together with each attribute alteration.

For convenience, we refer to the probability for a system attribute to be True as the *confidence* in the attribute. When a system attribute is in negation form, the confidence in the attribute is the probability that the negation form is True. For example, the confidence in ¬`sshd_running` is the probability that ¬`sshd_running` is True. Compared with IDS alerts, reports by scanning/monitoring tools are more reliable due to the verifiable nature of most system attributes. We can assume the confidence in a verifiable attribute is 1. However, some other system attributes cannot be checked easily. For example, we may not know whether an attacker has acquired a user's password by looking over his/her shoulder. In addition, some system attributes are difficult to check due to the security policy on the target system or performance reasons. In such cases, unless we have any further knowledge or evidence about the attribute, we assume the confidence in such an attribute is 0.5. Intuitively, this represents the lack of information about the state of the attribute.

2) Event-Based Evidence: Typical sources of event-based evidence include event logs, IDS alert logs, network traffic logs, system call logs, etc. Different kinds of logs provide event-based evidence on the system in different granularities and toward different aspects of the system. In this paper, the only event-based evidence we consider is IDS alert, which is in a coarser granularity but semantically richer compared with other types of system logs. We will use IDS alerts and event-based evidence interchangeably in the rest of the paper. Our representation of IDS alerts is closely related to our model of attacks. Thus, we first introduce our representation of attacks before discussing IDS alerts.

Similar to [7], [28], we model an attack as an atomic transformation that establishes a set of system attributes called *postcondition*, given a logical condition called *precondition* over system attributes. Intuitively, if the precondition of an attack is satisfied, the attack can then transform the system into the state specified by its postcondition. Given a certain privilege, an attacker may exploit some vulnerabilities of a system to launch an attack, which may introduce further vulnerabilities into the system, or give more privileges to the attacker. For example, an attack `sshd_buffer_overflow` may have `sshd_running`∧`sshd_vulnerable` as the precondition, and {`root_access, `¬`sshd_running`} as the postcondition. In other words, an `sshd_buffer_overflow` attack requires that the victim system runs a vulnerable `sshd` daemon, and as the result of this attack, the attacker gains root access privilege and the `sshd` daemon stops running.

IDS alerts represent potentially detected attacks. Thus, we can model IDS alerts in a similar way to attacks. However, IDS alerts are not exactly the attacks launched toward the target due to the imperfection of current IDSs. On the one hand, an IDS may report a false alert when it mistakes a normal operation for an attack. On the other hand, an IDS may raise no alert about an actual attack if the IDS does not recognize it. One goal of this paper is to use the additional information provided by state-based intrusion evidence to enhance our confidence in alerts representing successful attacks and at the same time reduce our confidence in false alerts. Moreover, we would like to make reasonable hypotheses about attacks possibly missed by the IDSs based on complementary evidence, and thus make the re-constructed attack scenario more consistent and closer to the reality.

To facilitate the reasoning about IDS alerts, we use the true alert rate of each attack to represent its quantitative property.

[1] A rootkit is a collection of tools (programs) that a hacker uses to mask intrusion and obtain administrator-level access to a computer or computer network (`http://searchsecurity.techtarget.com`).

The *true alert rate of an attack type T*, denoted *TAR(T)*, is the probability for a corresponding alert to represent an actual type *T* attack. The true alert rate of each type of attack can be gathered by analyzing historical data. It represents our prior knowledge about IDS alerts based on previous experience. One may observe that true alert rates are not stable since they are dependent on not only the quality of the IDSs, but also the attack frequency and background activities in a specific system. However, in the later part of this paper, we will see that our reasoning approach is still useful despite these unreliable true alert rates, because it reduces the uncertainty of intrusion evidence when additional verified evidence is considered. In some sense, *TAR(T)* is the *belief* that a type *T* alert is a real instance of attack, and our reasoning framework is to increase or decrease our belief in alerts based on complementary intrusion evidence.

Similar to the confidence in a system attribute, we refer to the probability that an IDS alert corresponds to a *successful attack* as the *confidence* in the alert.

We summarize our prior knowledge about IDS alerts and attacks below:

- An IDS alert *e* of attack type *T* has the probability *TAR(T)* to be a real attack;
- A real attack *E* has probability 1 to be successful when its precondition is satisfied by the system attributes before the attack happens;
- A real attack *E* has probability 0 to be successful if its precondition is not satisfied by the system attributes before the attack happens;
- The attributes in the postcondition of a successful attack *E* are True after the attack happens.

B. Basic Reasoning framework

In normal situations, a system should stay in a legitimate state. Starting from a legitimate system state, an attacker may launch a sequence of attacks to get the system into some intermediate states, and finally into the attacker's objective state. It is easy to see that there exist causal relationships among attacks and system attributes. Our approach is to use these causal relationships to reason about complementary IDS alerts and system attributes reported by scanning/monitoring tools. Specifically, we organize IDS alerts and system attributes into Bayesian networks [17] based on those causal relationships, and use these Bayesian networks to reason about complementary intrusion evidence.

1) Network Structure: To identify and represent these causal relationships, we integrate IDS alerts with system attributes based on the preconditions and postconditions of attacks. Specifically, we place IDS alerts, available system attributes, and system attributes possibly modified by the corresponding attacks into a directed graph, which we call an *alert-attribute network*.

Each node in such a graph is a binary variable representing either an IDS alert or a system attribute. For brevity, we refer to a node representing an IDS alert (or a system attribute) directly as an IDS alert (or a system attribute). When a node represents a system attribute, it can denote either a piece of state-based evidence (e.g., scan report), or a hypothesized attribute alteration caused by an IDS alert. A node denotes a hypothesized attribute when the attribute is in the postcondition of the attack corresponding to an IDS alert. Each node is timestamped. The timestamp of an alert node is the time when the corresponding activities take place, while the timestamp of an attribute node is the time when the attribute alteration is observed or inferred.

All edges in the graph are directed. An edge from an alert node to an attribute node represents that the corresponding attack changes the system attribute into this new state. An edge from an attribute node to an alert node represents that the attribute is a part of the precondition of the corresponding attack. An edge from an attribute node to another attribute node represents that the first attribute implies the second attribute. There are no edges that connect two alert nodes together directly.

We construct such a graph starting with the initial system state, which is represented in the graph as a set of attribute nodes corresponding to the initial attributes. As time goes by, new IDS alerts and system monitoring reports are raised. When a new IDS alert is reported, a corresponding alert node is added into the graph only if the alert's precondition is evaluated to be True given the attributes presented in the graph by the time. Also, edges are added from the latest attribute nodes corresponding to the attributes in the alert's precondition to the newly generated alert node (to represent the causal relationships). To serve the same purpose, edges from the alert node to its postcondition attribute nodes are also established when they are created. For each attribute node in the alert's postcondition, if nodes related to the same attribute already exist in the graph, which could either be caused by some previous alerts or reported by system monitoring tools, an edge from the latest such node to the new node is added to represent the implication relationship. By doing so, each attribute node in the graph represents the accumulative effects on the attribute of all the prior related alerts. Thus, when an attribute is part of the precondition of an alert, only the latest attribute node before this alert is connected to the alert node.

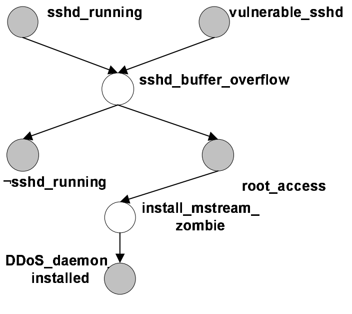


Fig. 1. A Bayesian network built from intrusion evidence

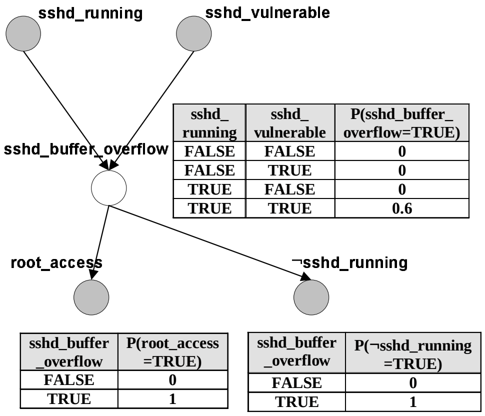
Figure 1 shows an example alert-attribute network, which is constructed as discussed above. The gray nodes represent initial or updated system attributes, and the white nodes represent IDS alerts. Initially, both attribute sshd_running and `vulnerable_sshd` are set to True. For simplicity, we do not show the initial system attributes that are not involved in the precondition or postcondition of the corresponding attacks. Alert `sshd_buffer_overflow` indicates an attempt to compromise the system through the vulnerable sshd. The precondition of `sshd_buffer_overflow` is `sshd_running`$\wedge$`vulnerable_sshd`, and the postcondition is $\{$`¬sshd_running, root_access`$\}$. Thus, this attempt can be successful since its precondition is satisfied in the system state. As a result, this attack introduces two attribute alterations: `¬sshd_running` and `root_access`. In other words, the attacker stops the sshd daemon and gains root access to the system. As shown in Figure 1, the attacker then installs a mstream zombie program, changing the attribute `DDoS_daemon_installed` from False to True.

2) Conditional Probabilities: A Bayesian network is a directed acyclic graph (DAG), where each directed edge represents a causal relationship between the two ends of the edge, and each node stores a conditional probability table describing the statistical relationships between the node and its parent nodes [17].

Based on the construction of the alert-attribute network, it is easy to see that a graph constructed in that way is acyclic. Indeed, all the edges are from previously existing nodes to newly added nodes, and thus will not result in any cycle. From our discussion above, the causal relationships among the nodes in an alert-attribute network are obvious. Now we discuss how to determine each node's conditional probability table so that the alert-attribute network becomes a Bayesian network.

When an IDS alert e is reported, the probability for the alert e to be a real attack is $TAR(e)$. The variable e being True represents that the corresponding attack is successful. We assume an attack will succeed if its precondition is satisfied. Thus, the probability of e being True is the true alert rate of the corresponding IDS alert when its precondition is satisfied, or 0 otherwise. Since an attack's precondition is a logic formula of system attributes, the conditional probability of an alert node can be easily derived. The conditional probability table associated with node `sshd_buffer_overflow` in Figure 2 shows such an example, where we assume *`TAR`*`(sshd_buffer_overflow)` = 0.6. Note that the probability of an IDS alert variable being False under these preconditions can be easily computed from the above probabilities. Thus, we do not include them here.

Conditional probability tables associated with system attributes are even simpler to compute. Indeed, if an IDS alert e represents a successful attack, all the system attributes in its postcondition should turn to True. Otherwise, the system attributes that are False before the IDS alert should remain False. If two attribute nodes of the same attribute are connected together with an edge representing implication relationship, and the earlier one is True, the latter one should also be True. Thus, the conditional probability of a system attribute a being True would be 1 if at least one of its parent variables (either alert nodes or attribute nodes) is True, and 0 if all its parent variables are False (unless it is reported by system scanning/monitoring tools). The tables associated with `root_access` and `¬sshd_running` show examples of such conditional probabilities. Similar to the above example, we only show the probabilities for the attributes to be True, from which the probabilities for the attributes to be False can be easily computed.



sshd_ running	sshd_ vulnerable	P(sshd_buffer_ overflow=TRUE)
FALSE	FALSE	0
FALSE	TRUE	0
TRUE	FALSE	0
TRUE	TRUE	0.6

sshd_buffer _overflow	P(root_access =TRUE)
FALSE	0
TRUE	1

sshd_buffer _overflow	P(¬sshd_running =TRUE)
FALSE	0
TRUE	1

Fig. 2. Conditional probability tables in an alert-attribute network

3) Reasoning about Intrusion Evidence: The Bayesian networks constructed in this way offer an excellent opportunity to reason about the uncertain intrusion evidence, particularly the IDS alerts. We call those attributes with a confidence value of 1 the verified attributes. The report of such verified attributes are observations of facts. When new verified attributes are reported by system monitoring/scanning tools, we can use these observations to re-compute the confidence values in the related previous objects in the network with Bayesian inference. Take the Bayesian network shown in Figure 2 as an example. We may be uncertain about an IDS alert reporting a buffer overflow attack against `sshd`, since the IDS has reported the same type of alerts incorrectly in the past. However, if by scanning the system we find that `sshd` is not running properly after the IDS reports this alert, we can then update the confidence in $\neg sshd\_running$ to be 1. Thus, we are more certain about the alert, which caused the attribute alteration. Though human users would do the same reasoning, placing these evidence into Bayesian networks offers additional benefits, since such a reasoning process can then be performed automatically and systematically. Also such reasoning could become too difficult for human users when dealing with very complicated scenarios.

It is easy to see that the more verified state-based evidence we have, the better judgment we can make by reasoning about the uncertain IDS alerts and system states. This suggests that we should monitor the system closer and possibly scan the system more frequently, as system monitoring tools and vulnerability scanning tools usually generate evidence with high confidence value. However, such monitoring and scanning are often expensive and may hurt the other applications by consuming resources. Thus, it is important to determine the right balance for system monitoring and scanning activities.

Nevertheless, this problem is out of the scope of this paper. We leave it for future consideration.

4) Merging Attribute Nodes: As discussed earlier, there may be edges between attribute nodes corresponding to the same attribute, which represent implication relationships between them. We observe that in certain cases, such attribute nodes can be merged without affecting the reasoning about intrusion evidence in alert-attribute networks. This observation is reflected by Lemma 1, which is presented next. For the sake of presentation, if two attribute nodes A and B are connected with edge (A, B), we refer to the action of removing node A with all its outgoing edges and redirecting all its incoming edges to node B as *merging* A *into* B.

Lemma 1: Consider two attribute nodes A and B corresponding to the same attribute and connected by an edge (A, B). If either there is no other outgoing edge from node A or A is instantiated (verified), merging A into B does not change the probability of any other node when reasoning about intrusion evidence.

Proof: The proof is divided into two steps. The first step is to prove that merging the two nodes will not affect other nodes in the downward reasoning in the Bayesian network. The second step is to prove that such a merge will not affect the posterior probability values of other nodes in the upward reasoning (belief updating) in the Bayesian network.

As shown in Figure 3, we assume node A's parent nodes are $X_1, X_2, \ldots, X_m$, node B's parent nodes are $Y_1, Y_2, \ldots, Y_n$ and A, and (A,B) is the only outgoing edge from A.

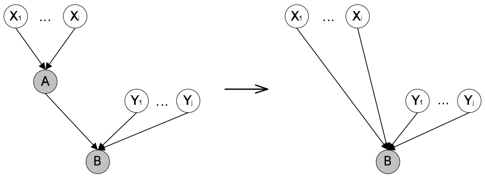


Fig. 3. Merging two attribute nodes

Since A and B are both attribute nodes, A is True if any of X_1, X_2, …, X_m is True, and B is True if any of A, Y_1, Y_2, …, Y_n is True. Thus, B is True if any of A, X_1, X_2, …, X_m, Y_1, Y_2, …, Y_n is True. After merging A into B, B's parent nodes are X_1, X_2, …, X_m, Y_1, Y_2,…, Y_n, and B is True if any of these nodes is True, which is exactly the same logic equation as before the merging. Thus, in the downward reasoning process, the probability value of any node other than A in the network remains the same as before merging A into B.

When computing the posterior probability value of a node M in the network after there is additional verified evidence E, the posterior probability can be computed as

$$P(M|E) = \frac{P(M,E)}{P(E)}.$$

In this equation, $P(E)$ and $P(M,E)$ are derived from margining out all the other variables in the joint probability density function $Prob(S)$, where S is the set of all the nodes (X_1, X_2, $\cdots$, Y_1, $\cdots$, M, $\cdots$) in the Bayesian network.

Because A and B's probability values solely depend on X_i and Y_j, given a set of input $(A, B, \{X_i\}, \{Y_i\}, \cdots)$, the probability value $Prob(A, B, \{X_i\}, \{Y_j\}, \cdots)$ either equals to 0 as A and B cannot be True given $(\{X_i\}, \{Y_j\})$, or equals to $Prob(\{X_i\}, \{Y_j\}, \cdots)$ as A and B are determined to be True given $(\{X_i\}, \{Y_j\})$. Thus, the result of margining out A and B from $Prob(S)$ before merging A into B

$$\sum_B \sum_A Prob(A, B, \{X_i\}, \{Y_j\}, \cdots)$$

equals to

$$\sum_{B=True} Prob(\{X_i\}, \{Y_j\}, \cdots). \tag{1}$$

Similarly, margining out B from the joint probability density function of all nodes after the merge can also be represented as

$$\sum_{B=True} Prob'(\{X_i\}, \{Y_j\}, \cdots), \tag{2}$$

where $Prob'(\{X_i\}, \{Y_j\}, \cdots)$ is the joint probability density function of the rest of the nodes in the merged network. Because A and B are solely dependent on $\{X_i\}$ and $\{Y_j\}$, and B's conditional probability table over $(\{X_i\}, \{Y_j\})$ does not change after the merge, formula 1 equals to formula 2. Thus, the posterior probability of any other node in the network remains the same as before merging A into B. ■

With Lemma 1, we can recursively merge attribute nodes that satisfy the condition specified in Lemma 1 to reduce the complexity of the network structure without affecting the reasoning result.

C. Alert Aggregation and Abstraction

The reasoning framework can greatly reduce the number of false alerts, and provides a method to combine multiple observations in intrusion analysis. However, in reality, IDSs often generate a large number of alerts for the same attack during a short period of time. Such alerts may be due to repeated attack attempts, or false alerts triggered by similar and repeated normal operations. For example, Snort [27] generated 24 "SNMP public access udp" alerts in our experiment without raising any other alert during a period of 10 minutes. As a result, these alerts share the same parent nodes and child nodes in the Bayesian network. This introduces two problems. First, a child node of the 24 "SNMP public access UDP" alerts has a conditional probability table with 2^{24} entries. Having so many entries makes it difficult to take advantage of existing Bayesian network tools, though it is possible to reduce the storage overhead by computing the conditional probability table on the fly. Second, when we use verified evidence discovered later to reason about these 24 alerts, the effect of the additional evidence will spread over these 24 alerts, since we do not know which of the 24 alerts indeed contributes to the modification of system attributes or later attacks.

In practice, when there are multiple consecutive alerts of the same type of attack, we usually do not care which one is the actual successful attack, but whether at least one of them is successful and changes the system state. Thus, a natural approach to addressing the above problem is to aggregate such alerts together into one single node, which represents "at least one of the component alerts corresponds to a successful attack". Specifically, we aggregate the alert nodes that have the same attack type, parent nodes, and children nodes into one aggregated alert node.

The conditional probability table of an aggregated alert node can still be computed similarly. However, we need to use aggregated true alert rate TAR_a, which represents the probability that at least one of its component alerts corresponds to an actual attack. Given n component alerts for attack type T that are merged into one aggregated alert, the aggregated true alert rate $TAR_a(T)$ can be computed as

$$TAR_a(T) = 1 - (1 - TAR(T))^n.$$

IDSs usually raise different alerts for similar attacks, or variations of the same attack. For example, Snort has more than 100 WEB-IIS related alerts, and many of them are exploiting the same unicode vulnerability and have the same impact. In many cases we do not care about the subtle difference between these alert variations, but only want to know if any of them is a successful attack. Thus, we may consider these alerts as the same type of alerts in a coarser granularity. To do so, we abstract alert variations into one common alert and apply alert aggregation. Specifically, we replace the attack type of each IDS alert with an abstract attack type, and follow the same procedure as for alert aggregation. Note that this abstraction requires human knowledge about the alerts and attack types.

Since different variations of the alerts being aggregated may have different true alert rates, we need to adjust the computation of the aggregated true alert rate slightly. The true alert rate is computed as below:

$$TAR_a(T) = 1 - \prod_{i=1}^{n}(1 - TAR(Type_{a_i})),$$

where $a_1, a_2, ..., a_n$ are the alerts to be aggregated, $Type_{a_i}$ is the attack type of alert a_i, and T is the aggregated attack type.

D. Hypothesizing about Missed Attacks

With alert aggregation and abstraction, our model can handle a larger number of alerts generated by IDSs. However, the model still cannot deal with missed attacks. When there is a missed attack, the effect of the attack on the system will not be reflected in the alert-attribute network, and some later alerts corresponding to successful attacks may be considered False. Thus, the alert-attribute network generated by the model may not reflect the reality when there are missed attacks. In other words, the current model only works when there are no missed attacks. (Note that this is a common problem shared by almost all alert correlation methods.) Because none of the current IDSs can guarantee to detect all attacks, it is necessary to improve the reason framework to deal with missed attacks.

We observe that when successful attacks are missed by IDSs, it is still possible for the system monitoring tools to catch the impact of the attacks on the system states. In other words, we may observe "unexpected" attribute alterations caused by the missed attacks. Such cases essentially cause *inconsistency* in the alert-attribute networks, where an attribute alteration is reported by system monitoring tools but there are no alert nodes in the network leading to the alteration.

We propose to hypothesize about missed attacks based on the above inconsistencies in alert-attribute networks. Inconsistencies are almost always caused by missed attacks: An "unexpected" attribute alteration causing the inconsistencies can either be directly caused by some successful attack missed by IDSs, or by a detected attack that does not appear in the network because its precondition is not satisfied in the network without the missed successful attacks. The only exception is that it could be caused by false alerts if the monotonicity property of attacks does not hold for some particular types of attacks, that is, a successful attack disables other attacks' preconditions. According to [28], this kind of attacks are very rare. We can always recognize such attacks and pay additional attention in the investigation when they are involved.

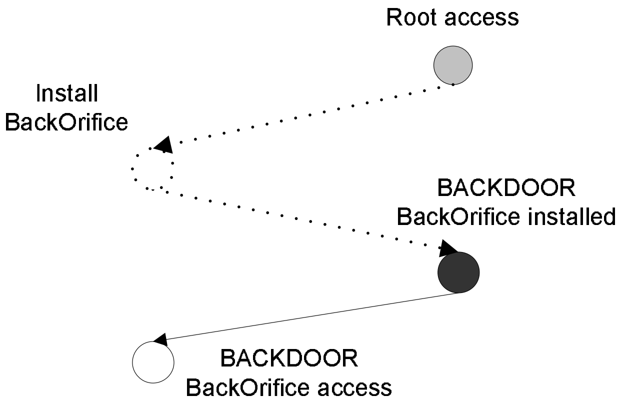


Fig. 4. An example of hypothesized attack

Figure 4 shows an example to hypothesize about possibly missed attacks to resolve such inconsistency. As we can see in the figure, when the system monitoring tool reports the fact that the backdoor "BackOrifice" was found in the local system, the system adds node "BACKDOOR BackOrifice installed" to the graph immediately, which activates the precondition of the later alert "BACKDOOR BackOrifice access". However, there is no previous node possibly causing the "BackOrifice installed" attribute set to True. To fill in this gap, we look up the graph structure for established attributes and attacks, the knowledge base for possible attacks that can cause this attribute alteration, and the log of previously dropped alerts for possible related attacks. According to the above information, we make a hypothesis of a possibly missed attack "Install BackOrifice" linking the attribute nodes "Root access" and "BackOrifice installed". The hypothesized node and edges are presented with dotted lines in the figure.

A hypothesis of a possibly missed attack infers that

- the attack has happened,
- the attack has been missed by IDSs, and
- the attack is successful.

Thus, as the three properties are independent from each other, the probability of a hypothesized attack being a correct hypothesis of a successful attack is $P_{hypothesis} = P_{happened} \cdot P_{missed} \cdot P_{successful}$, where $P_{happened}$ is the probability for the attack to have happened, P_{missed} is the probability for the IDS to miss the attack, and $P_{successful}$ is the probability for the attack to succeed if it happens. From our previous discussion, the success of an attack is determined by whether its precondition is satisfied by the system attributes. Thus, a hypothesis will have a probability of $P_{happened} \cdot P_{missed}$ if its precondition is satisfied by the system attributes. Then, the conditional probability table of a hypothesis node over the attributes in the attack's precondition is similar to a normal alert node's probability table except that the non-zero value of the node in the conditional probability table is $P_{happened} \cdot P_{missed}$ instead of *TAR*.

P_{missed} is the prior knowledge (or the belief) of human experts about known attack types, which can be determined based on historical data and experience. However, P_{happen} solely depends on the attacker's knowledge and personal preference, which is unpredictable. There is no way that we can have a fixed value of this probability. To be conservative in identifying missed attacks, we assume all the hypothesized attacks have happened in the Bayesian reasoning. Thus, we use the value P_{missed} instead in the conditional probability table. Accordingly, we refer to the probability of a hypothesized attack computed from the Bayesian network with this conditional probability table (and other verified evidence) the *confidence* in the hypothesized attack. Although this confidence value has a different meaning from that in those normal alert nodes, it still shows which hypothesis is more expectable given the available evidence.

We add the the hypothesized attacks with the corresponding conditional probability tables into the alert-attribute network. From the earlier discussion, we can see that such a hypothesis is made and placed into the alert-attribute network only if the attack is possible given the system state at the time. However, there may be later evidence showing that some attribute in the pre-/post-condition of the hypothesized attack is not valid, and such evidence will affect the confidence in the hypothesized attacks via belief update process in Bayesian inference. Thus, with the Bayesian network inference, we can always keep the hypotheses consistent with our observations in the system. For example, we may find negative evidence against a hypothesis, and the Bayesian inference process may update the probability of the hypothesized attack to 0, implying that the hypothesis cannot be a successful attack.

Validation is necessary for all hypotheses. From the above discussion about Bayesian inference about the hypotheses, we can see that the validation process is embedded in the Bayesian inference process. There are other ways to further validate the hypothesized attacks, such as those pointed out by [24]. However, these are out of the scope of this paper.

E. Scaling Up

The reader may have observed that as more IDS alerts are reported, the Bayesian network will grow larger and larger. Though by periodically scanning the system and gathering evidence about attacks, we may verify earlier alerts to be either successful or not, there will still be a number of unverifiable alerts. This has a severe impact on intrusion analysis. Indeed, both exact and approximate inferences in a Bayesian network upon partially observed evidence have been proved to be NP-hard [8], [12]. It is very expensive, and even infeasible, to make inferences upon new evidence if the Bayesian network is very large and complex.

One possible solution is to rebuild Bayesian networks when the previous ones grow too large. This can avoid intractable Bayesian networks. However, the effect of the evidence accumulated in the previous Bayesian networks will be lost, especially the system attributes that have been reasoned about using other evidence but not yet verified. As a result, information collected in an earlier Bayesian network cannot be fully carried over to the new one.

To make a trade-off between the accumulated information and the network size, we propose to use a sliding window to process and reduce the Bayesian networks. Specifically, we use a time window to decide what evidence to keep in the Bayesian network as well as what to remove. When new alerts or scanning results are reported, we slide the window so that the front of the window advances to the most recent evidence. Some old evidence may move out of the window, and be removed from the Bayesian network. IDS alerts can be simply removed from the network. However, for system attributes, the last version before the end of the window will be used as the initial system state in the updated Bayesian network.

Note that the effect of the removed evidence is still kept in the Bayesian network. When a Bayesian network is first constructed, all the probabilities of the nodes are computed from the prior probabilities. As old nodes are removed, previously internal nodes become the root nodes of the updated Bayesian network. These new root nodes use the previously updated probabilities as their prior probabilities for later inferences. As a result, the effect of earlier evidence is retained by the updated Bayesian network.

One may point out that sliding windows give attackers an opportunity to defeat our technique. That is, an attacker may slow down his/her attacks so that the related attacks are not effectively considered since they do not appear in the same Bayesian network. However, even if an attacker slows down the attacks, the effect of each successful attack step is still captured by its postcondition in a Bayesian network, if the attack is detected. Thus, we can still reason about an individual alert if its postcondition is verified. Moreover, if an attacker has to slow down his/her actions to avoid being detected, our technique has already deterred attacks.

The size of the sliding window is critical to the effectiveness of the Bayesian networks. If the window size is too small (e.g., shorter than the time interval between two consecutive system scans), some IDS alerts may be discarded before we can use related evidence to reason about them. Certainly, such a Bayesian network cannot be too large due to the difficulty in computing with large Bayesian networks. Thus, we should balance the computational cost and the risk of losing information, and decide the window size. We will study this problem in our future work.

III. Experimental Results

We have performed a series of experiments to evaluate the effectiveness of the proposed techniques. In our experiments, we connected three PCs through a hub in an isolated network. For convenience, we refer to them as *attacker*, *victim*, and *IDS*. We launched attacks from the attacker against the victim, while monitoring the attacks on the IDS.

We use Snort version 1.9.1 [27] as the IDS sensor. We also use Nessus [3] and XScan [33] as the vulnerability scanning tools. We evaluate our techniques with five attack scenarios, which we refer to as Scenario 0 to Scenario 4. The goals of these attack scenarios vary from modifying the web pages on the target to converting the target machine into a part of attacker's own distributed network. Some attack scenarios target MS Windows systems, while the others target Linux systems. Accordingly, the victim runs either Windows or Linux, depending on the attack scenarios. We run TripWire [6] (for MS Windows) and Samhain [5] (for Linux) on the victim as the file system integrity monitoring tools. We also run Trojan horse scanning tools Tauscan [30] (for MS Windows) and chkrootkit 0.43 [1] (for Linux) on the victim as additional system scanning tools. We developed a program to automatically generate alert-attribute networks from the IDS alerts and the reports of these scanning tools, and then use JavaBayes [2] to make inference using these networks.

To simulate the real-world system administration, we configure the file system integrity monitoring tools (Tripwire and Samhain) to monitor important files and directories only, i.e., system configurations files, service configuration files, and the main webpage files.

To mimic an operational network, we also inject background traffic into the network during our experiments. We randomly select one of the training datasets (the training dataset on Monday in the third week) in the 1999 DARPA intrusion detection evaluation datasets [21] as the background traffic in the experiments, as it is attack free. This background traffic triggers 325 alerts in Snort, which are all false alerts. All the other alerts reported by Snort are real alerts.

In the rest of this section, we first present the analysis of Scenario 0 in detail, and then summarize the results of all five attack scenarios. Additional details of the other four attack scenarios are included in the Appendix.

A. Analysis of Scenario 0

1) Details of Scenario 0: In this attack scenario, the attacker exploits the remote buffer overflow vulnerability in some old versions of Serv-U ftp server to get administrative access. The victim machine is a Windows box running a vulnerable Serv-U 5.0 ftp server with default public anonymous access. At the same time, the victim also runs Norton antivirus with file system real-time protection. When the system attempts to access a file containing known virus or backdoor, the file system real-time protection will quarantine the file.

The attack scenario includes five steps:

1) remote buffer overflow attack against the Serv-U,
2) attempt to install BackOrifice on the victim, which was quarantined by Norton anti-virus,
3) kill the Norton anti-virus process with system process tools through the remote administrative shell,
4) install the BackOrifice again (successful), and
5) changing the web page through BackOrifice.

The initial system attributes include

- Serv-U 5.0 on port 21,
- anonymous ftp access,
- Norton Anti-virus with file system real-time protection, and
- http on port 80.

During the attack process, Snort reported the following 2 alerts:

- One `FTP command overflow attempt` alert
- One `BACKDOOR BackOrifice access` alert

Norton also logged that BackOrifice was found in the file system and quarantined successfully during the attack period. In the end, Tripwire logged and reported the modification to the web page file and the system logged that Norton anti-virus was shut down.

2) Reasoning about Intrusion Evidence: Our alert-attribute network generation tool generated the network shown in Figure 5 based on the above information and the prior probabilities and attack type information, which are included in the appendix,.

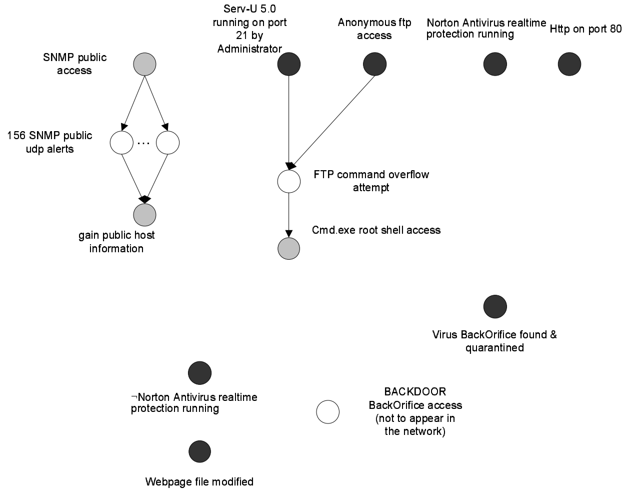

Serv-U 5.0 running on port 21 by Administrator
Anonymous ftp access
Norton Antivirus realtime protection running
Http on port 80
SNMP public access
156 SNMP public udp alerts
...
FTP command overflow attempt
Cmd.exe root shell access
gain public host information
Virus BackOrifice found & quarantined
¬Norton Antivirus realtime protection running
BACKDOOR BackOrifice access (not to appear in the network)
Webpage file modified

Fig. 5. Initial alert-attribute network

To distinguish between different types of nodes in a Bayesian network, we use white nodes to denote IDS alerts, gray nodes to denote unverified system attributes, and black nodes to denote verified system attributes. The relative vertical position of nodes in the graph represents the relative time order among nodes.

Note that Figure 5 includes 156 “`SNMP public access udp`” alerts, which results in 2^{156} entries in the conditional probability table of `gain public information`. Computing with such a conditional probability table is out of JavaBayes’ handling capacity. However, after alert aggregation, the 156 nodes are aggregated into a single node and thus can be handled easily by JavaBayes.

Now let us look at possible missed attacks. There are several obvious inconsistencies in Figure 5. There are no detected alerts causing the verified attributes “`Norton Anti-virus not running`”, “`Virus BackOrifice found` & `quarantined`”, and “`Webpage file modified`”. Based on our knowledge about attacks, “`Shut down Norton Anti-virus via cmd.exe shell`” and “`Install BackOrifice`” are the only possible hypotheses that can fill in the first two gaps. For the attribute “`Webpage file modified`”, it could be done through remote control via either cmd.exe shell or BackOrifice access. The first option implies hypothesized remote control via cmd.exe, while the second one implies hypothesized installation of BackOrifice after Norton was shut down. These hypotheses lead to a new alert-attribute network in Figure 6.

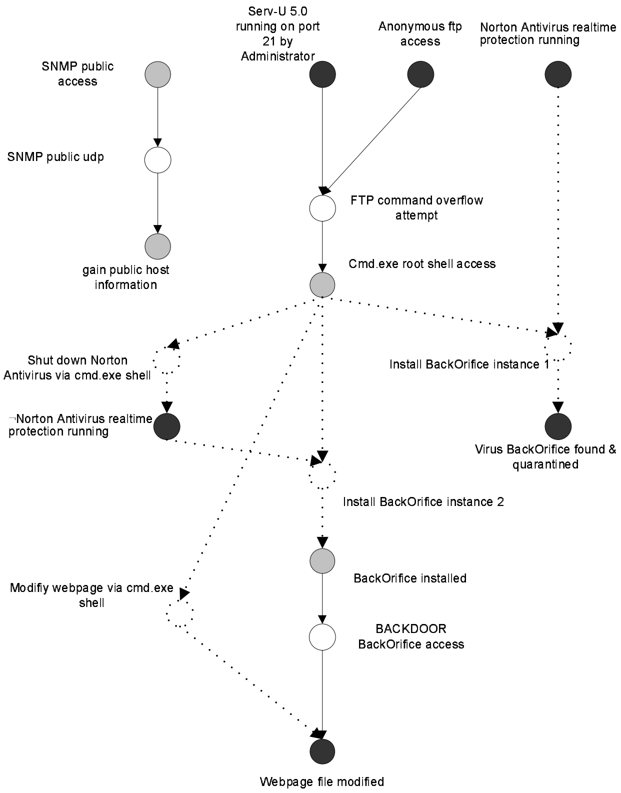

Serv-U 5.0 running on port 21 by Administrator
Anonymous ftp access
Norton Antivirus realtime protection running
SNMP public access
SNMP public udp
FTP command overflow attempt
gain public host information
Cmd.exe root shell access
Shut down Norton Antivirus via cmd.exe shell
Install BackOrifice instance 1
Norton Antivirus realtime protection running
Virus BackOrifice found & quarantined
Install BackOrifice instance 2
BackOrifice installed
Modifiy webpage via cmd.exe shell
BACKDOOR BackOrifice access
Webpage file modified

Fig. 6. Updated alert-attribute network

In Figure 6, the dotted nodes and edges denote hypothesized attacks and corresponding causal relationships. Conditional probability table of each node can be generated automatically given the network structure and prior probability values. Then JavaBayes generates updated confidence values of each node in this Bayesian network. The confidence values of the related alerts before and after reasoning are shown in Table I. We can see significant increases in the confidence values of successful attacks; however, all the false alerts have either decreased or unchanged confidence.

TABLE I
CONFIDENCE VALUES BEFORE AND AFTER THE REASONING

alert name	before reasoning	after reasoning	relative increase
FTP command overflow attempts	0.3	1	233.33%
BACKDOOR BackOrifice access	0.3	0.6	100%
Shut down Norton Anti-virus via cmd.exe shell (Hypothesized)	N/A	1	N/A
Install BackOrifice Instance 1 (Hypothesized)	N/A	0	N/A
Install BackOrifice Instance 2 (Hypothesized)	N/A	1	N/A
Modify web page via cmd.exe shell (Hypothesized)	N/A	1	N/A
156 individual SNMP public access udp	0.075	0.075	0%
aggregated SNMP public access udp	0.5	0.5	0%
other 169 alerts	0.25	0	-100%

We also find some interesting observations in Table I. The confidence values in three of the hypothesized nodes turned into 1, and two of them are the two options to resolve the same inconsistency. As we have discussed in Section II-B.2, unless a hypothesis is the only option to solve the inconsistency, a confidence value of 1 for a hypothesized attack does not mean that the attack must have happened. Instead, it implies that *if* that attack has happened, it must be successful. Thus, although the confidence values for the two hypothesized nodes are both 1, it does not mean that both attacks must have happened. However, comparing the probability of the path from the initial verified attributes to the later verified attribute (by multiplying the probabilities of all the intermediate nodes along the path), we find that the one through “`Modify web page via cmd.exe`” has a greater probability than the other one. Although it is not what exactly happened in our experiment, it shows that both methods can achieve the goal of modifying web page without being detected, and modifying through established remote cmd.exe shell is simpler and easier compared to the other option, which requires several extra attack steps. Also, the probability of a hypothesized node being 0 means either it is not missed by the IDS, or it is a failed attack attempt.

3) Using Confidence for Intrusion Detection: With the reasoning framework for intrusion evidence, we are able to associate a quantitative measure (i.e., confidence) with each IDS alert. It is natural to think about using the alert confidence values to improve the performance of intrusion detection. In addition, we want to see how additional complementary

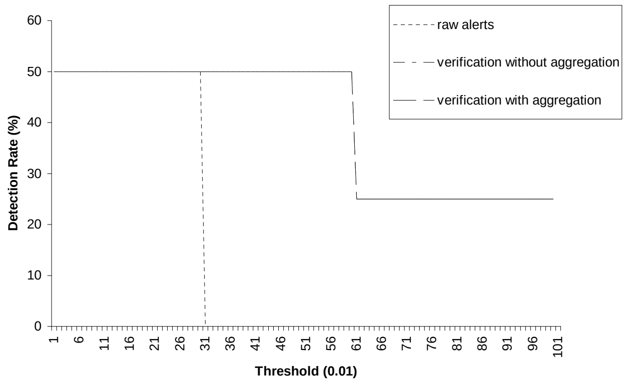


Fig. 7. Detection rate vs threshold (Scenario 0)

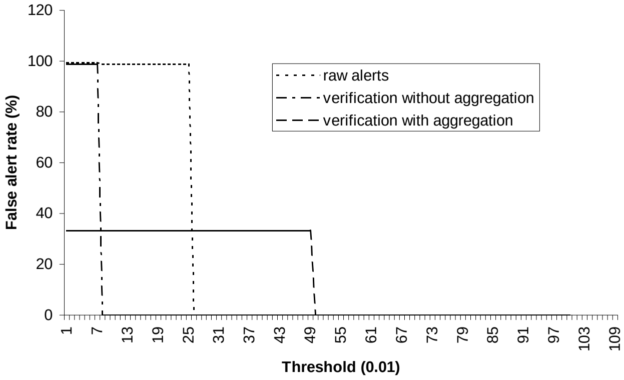


Fig. 8. False alert rate vs threshold (Scenario 0)

evidence (e.g., verified system attributes) helps in this process.

In our experiments, we used a confidence threshold to determine whether an IDS alert is a successful attack or not. Specifically, if the confidence in an alert is greater than or equal to the threshold, we accept the alert. Otherwise, we simply drop it. We change the threshold value between 0 and 1, and collect the detection rates and false alert rates. To compare the results in different situations, we repeated the above process in two cases: (1) without alert aggregation and abstraction, (2) with alert aggregation and abstraction. The performance graphs for the five attack scenarios are very similar.

In our evaluation, we abuse the notions of detection rate and false alert rate to represent the *detection rate of successful attacks* and *false alert and failed attack rate*, respectively. Figure 7 and 8 show the detection rate curves and false alert rate curves w.r.t. different thresholds in all cases for one of our scenarios. Since the meaning of the confidence in a hypothesized attack is different from that in an IDS alert, we do not consider hypothesized attacks in this evaluation.

From the two figures, we can see that the Bayesian reasoning with verified evidence can significantly increase the detection rates and decrease the false alert rates in a large threshold space by adjusting the confidence values of the alerts based on complementary evidence. Please note that because the largest confidence value of alerts before reasoning is 0.3, the curve of raw alerts only continues to 0.3 on the x axis in Figure 8. No alerts can be detected with a threshold larger than 0.3 before the reasoning. After the reasoning, the detection range is also greatly increased, which provides more flexibility for making security policies. If we further consider the fact that after the reasoning, we know for sure that three of the four hypothesized attacks must have happened, the framework's ability to improve detection performance is actually more than what is shown in the figure. The reason the result with aggregation has a higher false alert rate when the probability threshold is over 0.3 is because that the 156 false "`SNMP public access UDP`" alerts are aggregated into one single alert and its probability is greatly increased. Also, the detection rate shown in the figure does not consider the hypothesized attacks, where 2 out of the 4 hypothesized attacks are actual successful attacks missed by the IDS, and 1 of the other 2 hypothesized attacks is an actual failed attack attempt.

B. Summary of Experimental Results

In the following, we summarize the results obtained from all the five attack scenarios. We first discuss the impact of the proposed techniques on alerts, and then describe the results about hypothesized attacks.

We use a simple metric named confidence ratio to examine the usefulness of the proposed techniques in reasoning about IDS alerts. Specifically, a *confidence ratio* is the ratio between the average confidence of alerts corresponding to successful attacks and the average confidence of the other alerts (i.e., false alerts and alerts corresponding to failed attack attempts).

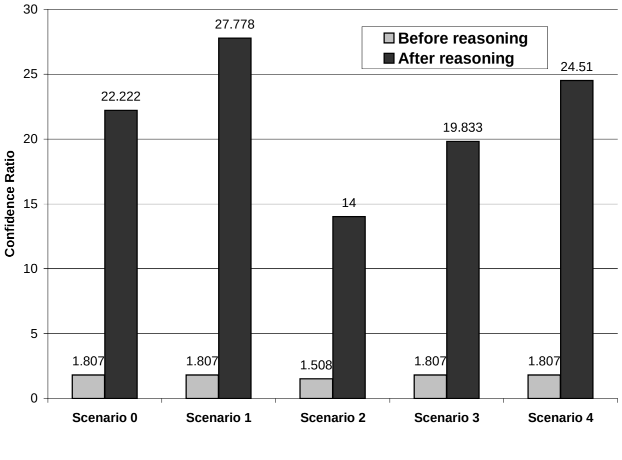


Fig. 9. Confidence ratio before and after the reasoning

Figure 9 shows the confidence ratios in all five attack scenarios before and after using the proposed techniques. (We have discussed Scenario 0 in the previous subsection; details of the other scenarios can be found in the appendix.) These results indicate that with the proposed techniques, the average confidence in alerts of successful attacks are greatly increased compared with the average confidence in the other alerts (false alerts and alerts for failed attack attempts). In fact, the average confidence in the other alerts either remain the same or decrease.

We totally made ten hypotheses during the reasoning in the analysis of the five attack scenarios. Table II shows the accuracy of these hypotheses in these attack scenarios, respectively.

TABLE II
ACCURACY OF HYPOTHESES IN THE EXPERIMENTS

Scenario	Accuracy
0	75%
1	100%
2	50%
3	100%
4	50%

In the experiments, six out of the ten hypothesized attacks are actual successful attacks missed by Snort, and one out of the other 4 hypotheses is an actual failed attack attempt. Among the seven real attacks, we have definite confidence that four of them must have happened from the alert-attribute network. The result shows that with sufficient local system evidence, our model is efficient and effective in discovering some missed attacks.

IV. RELATED WORK

The techniques closest to ours are M2D2 [22] and the mission-impact-based correlation method [25], which have been briefly discussed in the introduction. All these methods, including the techniques proposed in this paper, attempt to correlate intrusion evidence from multiple sources. However, M2D2 is intended to provide a formal model to represent intrusion related information, while the mission-impact-based method requires substantial human involvement in the specification of correlation models. In contrast, our method can automatically construct Bayesian networks of IDS alerts and other complementary intrusion evidence based on the knowledge of individual attacks, and harness the rich results developed for reasoning about uncertain events.

Another approach was proposed in [24] to make hypotheses about missed attacks based on the pre/post-conditions of known attacks. Our approach differs in that the hypotheses made in our model is based on not only the pre/post-conditions of known attacks but also the available system states.

The techniques proposed in [10], [23], [31] are also based on modeling individual attacks, similar to ours. However, these approaches only focus on IDS alerts, but do not take advantage of other information sources. Our approach can potentially get more concrete analysis results due to the additional, complementary information considered in our model.

There are other alert correlation techniques. The techniques in [9], [13], [29], [32] correlate alerts on the basis of the similarities between the alert attributes. The Tivoli approach correlates alerts based on the observation that some alerts usually occur in sequence [15]. The alert clustering techniques in [19], [20] use conceptual clustering and generalization hierarchy to aggregate alerts into clusters. It is proposed in [26] to use time series analysis to discover potential causality between alerts without specifically modeling attacks. Alert correlation may also be performed by matching attack scenarios specified by attack languages. Examples of such languages include STATL [16], LAMBDA [11], and JIGSAW [31]. These methods use mechanisms different from ours to correlate alerts, and are potentially complementary to our approach. It may be possible to improve some of these approaches to support complementary intrusion evidence. However, we do not consider it in this paper.

Our approach is also related to the recent results on vulnerability analysis (e.g., [7], [18], [28]). In particular, the methods in [7], [28] also model system state as system attributes, and attacks atomic transformation that establish postconditions given the attacks' preconditions. However, our approach is aimed at reasoning about intrusion evidence rather than finding out possible sequences of attacks.

V. CONCLUSION AND FUTURE WORK

In this paper, we developed a method to integrate and reason about complementary intrusion evidence, including IDS alerts, reports of system monitoring or vulnerability scanning tools, and even human observations. By using the interdependency between attacks and system states, we combine IDS alerts and attributes representing modifications of system states into Bayesian networks, which are then used to infer about uncertain IDS alerts based on additional observations of system states. We further proposed to refine these Bayesian networks through alert aggregation and abstraction, so that we can focus on the reasoning about existences of successful attacks and use complementary intrusion evidence more effectively. We also proposed to use sliding windows to provide a trade-off between the intractability of reasoning with large Bayesian networks and the ability to integrate and reason about IDS alerts and other evidence. Our initial experimental results have demonstrated the potential of the proposed techniques.

A limitation of our approach is that it reasons about successful attacks, but cannot handle attack attempts in the same way. In other words, with additional evidence such as a verified attribute, our approach will increase the confidence in alerts corresponding to successful attacks, but decrease the confidence in those representing failed attack attempts. This feature certainly restricts the applicability of our approach. Another limitation is that our model cannot reason about attacks which has no effect on the local system, i.e., probes and scans. Indeed, most attackers need to gather certain information via network to launch attacks. For example, a probe to some specific ports may be necessary for attackers to gain related information to launch some corresponding exploits. However, the effect of such information gathering activities is on the remote attackers' side, which cannot be predicted and be used as preconditions of attacks. Information can be gathered in multiple ways other than network scans, e.g., chatting with a careless administrator or wire-tapping the telephone. Thus, such attacks will not appear in the alert-attribute Bayesian network so that the reasoning will not affect and be affected by such alerts. Information of such alerts is usually useful for human administrators in analyzing the attacker's intentions and strategies in reality. For example, people may have a higher belief on alerts of follow-up attacks after monitored probes on some special ports. However, modeling this observation brings risk of being distracted by forged traffic from attackers.

This paper is only the starting point of our effort to integrate and reason about complementary intrusion evidence. In our future work, we will investigate additional techniques to improve the performance. In particular, we will study the use of dynamical Bayesian networks in processing streams of IDS alerts and other intrusion evidence, investigate approaches to handling attacks missed by IDSs, and perform experiments with large sets of intrusion evidence.

REFERENCES

[1] checkrootkit. http://www.checkrootkit.org. Accessed on Feb. 4, 2004.

[2] Javabayes. http://www-2.cs.cmu.edu/~javabayes/Home/. Accessed on Oct 10, 2003.

[3] Nessus. http://www.nessus.org. Accessed on Feb. 4, 2004.

[4] Nmap – network exploration tool and security scanner. http://www.insecure.org/nmap/.

[5] Samhain. http://la-samhna.de/samhain/. Accessed on April 4, 2004.

[6] Tripwire. http://www.tripwire.com. Accessed on Feb. 4, 2004.

[7] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, November 2002.

[8] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[9] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.

[10] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[11] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Proc. of Recent Advances in Intrusion Detection (RAID 2000)*, pages 197–216, September 2000.

[12] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[13] O. Dain and R.K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 231–235, June 2001.

[14] O. Dain and R.K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pages 1–13, November 2001.

[15] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, LNCS 2212, pages 85 – 103, 2001.

[16] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[17] F.V. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Information Science. Springer, 2001.

[18] S. Jha, O. Sheyner, and J.M. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th Computer Security Foundation Workshop*, June 2002.

[19] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 12–21, December 2001.

[20] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *The 8th ACM International Conference on Knowledge Discovery and Data Mining*, July 2002.

[21] MIT Lincoln Lab. 1999 DARPA intrusion detection scenario specific datasets. http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html, 1999.

[22] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[23] P. Ning, Y. Cui, and D. S Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.

[24] P. Ning, D. Xu, C. Healey, and R. St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, February 2004.

[25] P.A. Porras, M.W. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, 2002.

[26] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, September 2003.

[27] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA conference*, 1999.

[28] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.

[29] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[30] Tauscan. http://www.agnitum.com/products/tauscan/.

[31] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31 – 38. ACM Press, September 2000.

[32] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 54–68, 2001.

[33] X-scan. http://www.xfocus.org.

APPENDIX

ADDITIONAL ATTACK SCENARIOS USED IN OUR EXPERIMENTS

Results about Scenario 0 have been discussed in the main text. Here we give further details about the experimental results for the remaining scenarios.

A. Scenario 1

In Scenario 1, we simulated a common script kid activity, which exploits a common vulnerability to get certain privilege, and modify the remote server's web page. In this particular scenario, we exploited the format string vulnerability of wu-ftpd 2.6.0 on a RedHat linux 6.2 server to get remote root access. The attack scenario includes two steps:

1) A remote format string attack toward the wu-ftpd, and
2) replacing the remote server's web page with a "Gotcha" web page via the remote root shell access gained after the previous attack.

Snort raised the following alert(s):

- 1 FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux

Note that no alerts were raised for the remote root shell access. The file system monitoring tool (Samhain) generated

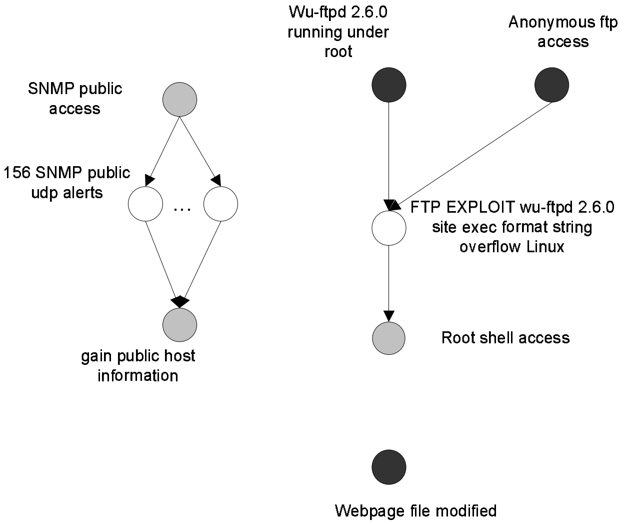

Wu-ftpd 2.6.0 running under root
Anonymous ftp access
SNMP public access
156 SNMP public udp alerts
...
FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux
gain public host information
Root shell access
Webpage file modified

Fig. 10. Initial alert-attribute network

alert for the web page modification since we configured the threshold on the times of modifications on those files to be 1.

Initial vulnerability scan showed that the system was running a vulnerable wu-ftpd 2.6.0 on port 21 with anonymous access open. The result alert-attribute network before making hypotheses is as shown in Figure 10.

Based on the observation of the inconsistency in the alert-attribute network shown in Figure 10, together with the attack type knowledge, the only possible hypothesis to fill in the inconsistency is that some remote control via the root shell caused the web page modification. Figure 11 shows the updated alert-attribute network with a hypothesized attack.

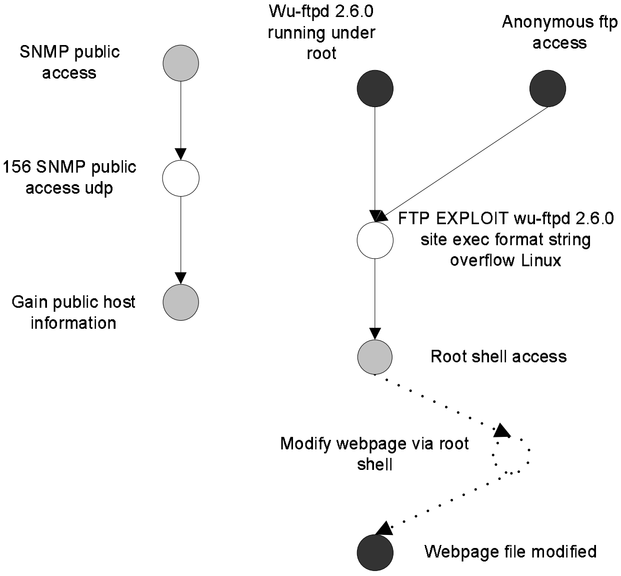

Wu-ftpd 2.6.0 running under root
Anonymous ftp access
SNMP public access
156 SNMP public access udp
FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux
Gain public host information
Root shell access
Modify webpage via root shell
Webpage file modified

Fig. 11. Alert-attribute network after hypotheses

We use dotted nodes and edges to denote hypothesized nodes and relationships in this new figure of alert-attribute network.

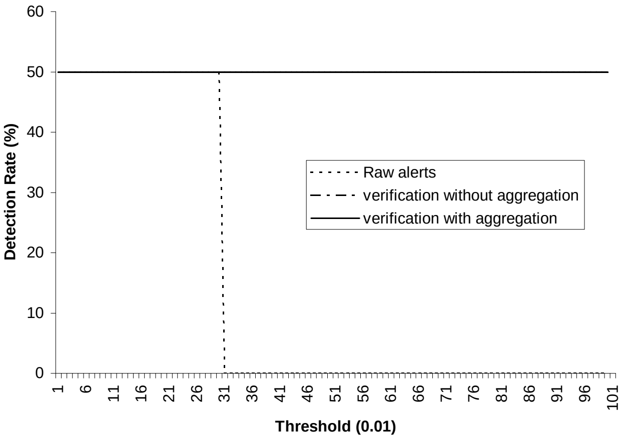


Fig. 12. Detection rate VS. threshold (Scenario 1)

According to the prior probability values and attack type information included in appendix B, our program generated the Bayesian network from the evidence log automatically and the reasoning result using JavaBayes is shown in table III. Table III also shows the relative increase of the confidence values of the alerts.

TABLE III
CONFIDENCE VALUES BEFORE AND AFTER THE REASONING

alert name	before reasoning	after reasoning	relative increase
156 SNMP public access udp (156)	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
FTP SITE EXEC format string attempt linux	0.3	1.0	333.33%
Remote control via root shell (hypothesized)	N/A	1.0	N/A
Other alerts	0.25	0	−100%

The confidence in the only hypothesized attack “Remote control via root shell” turned to 1 after the inference, which indicates that it would be successful and missed by the snort if it happened.

When using confidence threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 12 and Figure 13.

B. *Scenario 2*

Scenario 2 exploits the unicode vulnerability of MS IIS 5.0. The victim machine was a Windows box configured to be running a vulnerable IIS 5.0. The initial system vulnerability scan showed that there existed `IIS 5.0 unicode` vulnerability on the victim. The attacks scenario includes two steps:

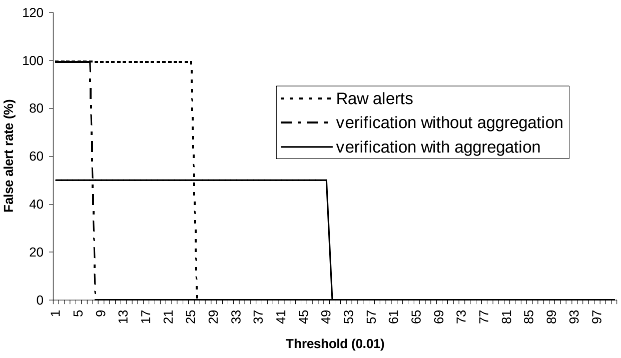


Fig. 13. False alert rate VS. threshold (Scenario 1)

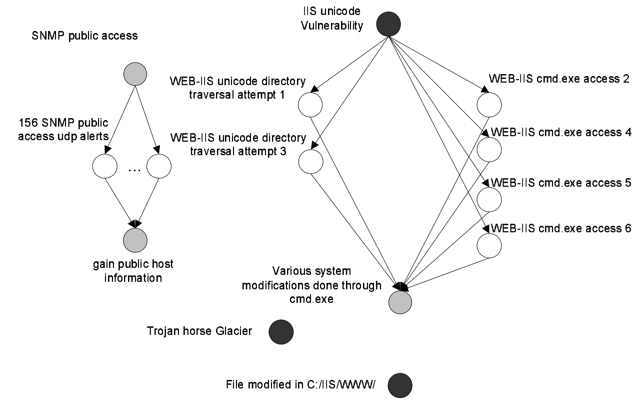


Fig. 14. Initial alert-attribute network

1) Exploiting the unicode vulnerability to download and install a Trojan horse named `Glacier` to the victim
2) Monitoring/controlling the victim file system remotely through the Trojan horse. The activities include replacing the web page

When the above attacks were launched without background traffic, Snort generated the following 6 alerts:

- 2 WEB-IIS unicode directory traversal attempt alerts, and
- 4 WEB-IIS cmd.exe access alerts.

Both the WEB-IIS unicode directory traversal and the WEB-IIS cmd.exe access alerts indicate web attacks exploiting IIS 5.0's unicode vulnerability, thorough which the attacker can execute commands remotely thorough local host's cmd.exe program and cause various system modifications. They can actually be taken as Snort's reports on the same type of attacks. Thus, in the later analysis we aggregate them together into a single node. The postconditions of such attacks are highly dependent on the detailed content of the message sent by the attacker. As snort does not detect and distinguish those details, we define for this type of attacks a general postcondition "`Various system modifications done through cmd.exe`", which could imply any system modifications possibly done through cmd.exe.

Snort failed to detect the attempts of installing the Glacier backdoor and the remote control accesses through the Glacier backdoor. However, on the local system side, Tauscan did report in real time that Trojan horse Glacier was found in the system immediately after the IDS reported WEB-IIS alerts. Then Tripwire logged the modifications to the files in the IIS's web page directories a few minutes later after that.

Again, before aggregation and making any hypotheses about missed attacks, we have an alert-attribute network with inconsistencies as shown in Figure 14.

For the Tauscan's report of Trojan horse Glacier found, this attribute may only be implied by the "`Various system modification done`" caused by the IIS unicode exploits, thus we can have a hypothesized implication relationship between them, which is denoted by a dotted empty arrow in the figure. For the modifications in web page files, according to the system state and knowledge base (attack type info) we have, there are two options to achieve it on the victim: It could either be caused by some missed WEB-IIS unicode exploit, or by the remote control via the Glacier Trojan horse. Due to the first option, a hypothesized IIS unicode exploit alert node is added to the network to link the initial "`IIS unicode vulnerability`" node and the web page file modified node. Due to the second option, a hypothesized alert node "`Remote control via Glacier Trojan horse`" is added to link the "`Trojan horse Glacier found`" and the "`File modified in c:/IIS/WWW/`" node. With all these hypotheses and necessary alert aggregations/abstractions the complete network is shown in Figure 15.

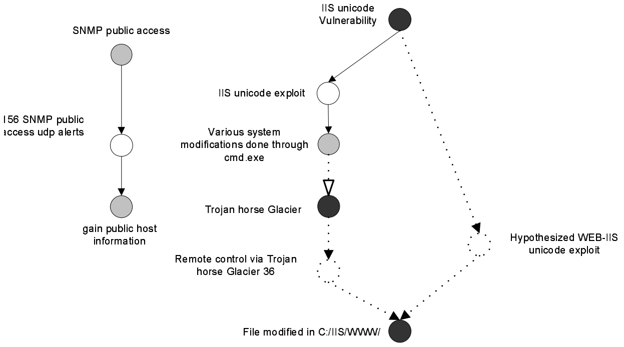


Fig. 15. Updated alert-attribute network

The probabilistic confidence values of alerts before and after the analysis are shown in table IV.

In the reasoning, we assumed the remote control via Trojan horse Glacier to have a much larger probability to be missed by snort, compared with the IIS unicode exploit. This results in a higher confidence that the remote control via Glacier actually succeeded and was missed by snort, compared with the other hypothesis.

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 16 and Figure 17.

TABLE IV
CONFIDENCE VALUES BEFORE AND AFTER THE REASONING

alert name	before reasoning	after reasoning	relative increase
2 individual WEB-IIS directory traversal attempts	0.25	0.50394	101.58%
4 individual WEB-IIS cmd.exe access	0.25	0.50394	101.58%
Aggregated WEB-IIS unicode exploit	0.822	1	21.6%
Remote control via Trojan horse Glacier (hypothesized)	N/A	0.8	N/A
IIS unicode exploit (hypothesized)	N/A	0.4	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
Other 169 alerts	0.25	0	−100%

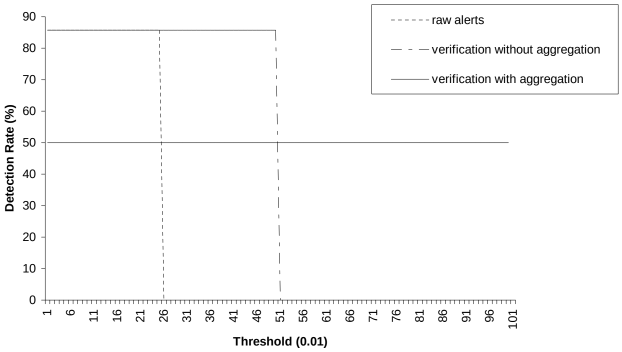


Fig. 16. Detection rate VS. threshold (Scenario 2)

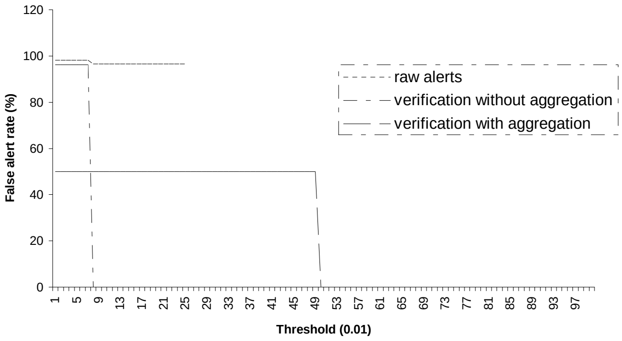


Fig. 17. False alert rate VS. threshold (Scenario 2)

Note that the reason for the rate after alert aggregation and abstraction being smaller than before aggregation is that the true alerts are aggregated into one single alert. Thus, the total numbers of alerts are different for the two sets of results.

C. Scenario 3

This attack scenario was a popular one on the Internet in the fall 2002, cause we have read a number of victim reports on the Internet and one of the machines in our lab was unfortunately one of them. The attack exploits the remote buffer overflow vulnerability of several older versions of Serv-U ftp server to get administrative access, and installs IRC DCC bot on the victim server to make it part of the attacker's public distribution network.

We configured the victim machine to be a Windows box running a vulnerable Serv-U 5.0 ftp server with default public anonymous access. At the same time, the victim was also running snort to log intrusion activities, and Tripwire to monitor the local file system.

The attack scenario includes four steps:

1) Two remote buffer overflow attack attempts toward the Serv-U 5.0 server, one of which failed while the other succeeded.
2) Downloading and installing the IRC DCC bot on the victim through the remote root shell.
3) Cleaning the attack trace logged by snort after noticing the existence of snort in the system process list..
4) Starting another ftp server on port 28021.

The initial system scan reported finding vulnerable Serv-U 5.0 ftp server running on port 21 with public anonymous access.

When the attacks were launched without background traffic, snort reported two "FTP command overflow attempt" alerts and the system monitoring tools reported the following two observations:

- File modifications found on "c:/snort/log/alerts.ids" and "c:/servu/ServUDaemon.ini" reported by Tripwire.
- IRC DCC bot running on port 6666 and Serv-U 5.0 ftp server running on port 28021 reported by later system port scan.

Thus, when combined with the false alerts generated by the background traffic, we have the alert-attribute network shown as in Figure 18.

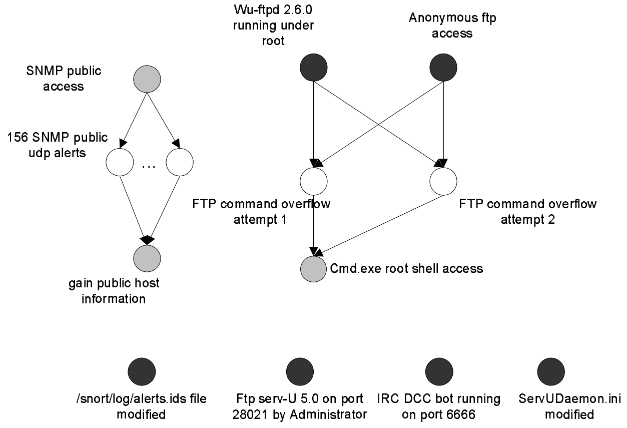

Wu-ftpd 2.6.0 running under root
Anonymous ftp access
SNMP public access
156 SNMP public udp alerts
...
FTP command overflow attempt 1
FTP command overflow attempt 2
Cmd.exe root shell access
gain public host information
/snort/log/alerts.ids file modified
Ftp serv-U 5.0 on port 28021 by Administrator
IRC DCC bot running on port 6666
ServUDaemon.ini modified

Fig. 18. Initial alert-attribute network

After aggregation and making hypotheses, the final alert-attribute network is shown in Figure 19.

The confidence values of the related alerts before and after reasoning is shown in table V.

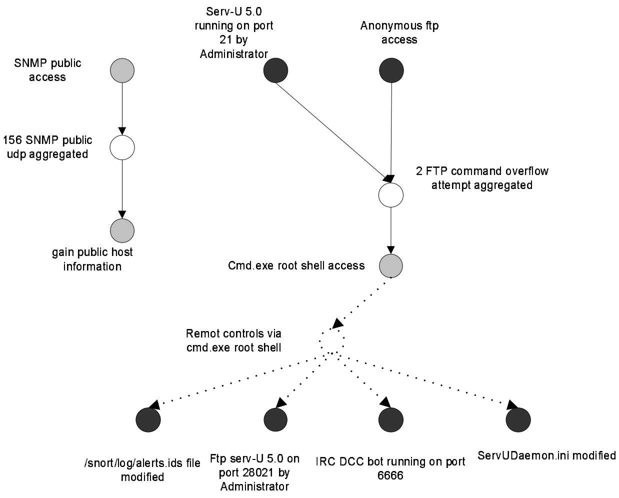

Serv-U 5.0 running on port 21 by Administrator
Anonymous ftp access
SNMP public access
156 SNMP public udp aggregated
2 FTP command overflow attempt aggregated
gain public host information
Cmd.exe root shell access
Remot controls via cmd.exe root shell
/snort/log/alerts.ids file modified
Ftp serv-U 5.0 on port 28021 by Administrator
IRC DCC bot running on port 6666
ServUDaemon.ini modified

Fig. 19. Updated alert-attribute network

TABLE V
CONFIDENCE VALUES BEFORE AND AFTER THE REASONING

alert name	before reasoning	after reasoning	relative increase
2 individual FTP command overflow attempts	0.3	0.714	40.06%
Aggregated FTP command overflow attack	0.51	1	96.08%
Remote control via cmd.exe root shell (hypothesized)	N/A	1	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
other 169 alerts	0.25	0	−100%

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 20 and Figure 21.

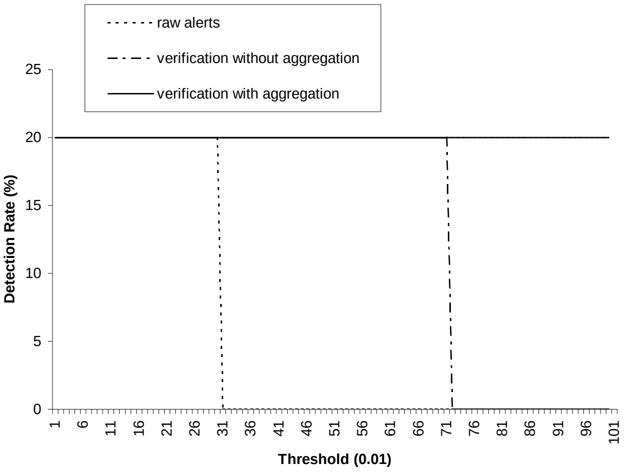


Fig. 20. Detection rate VS. threshold (Scenario 3)

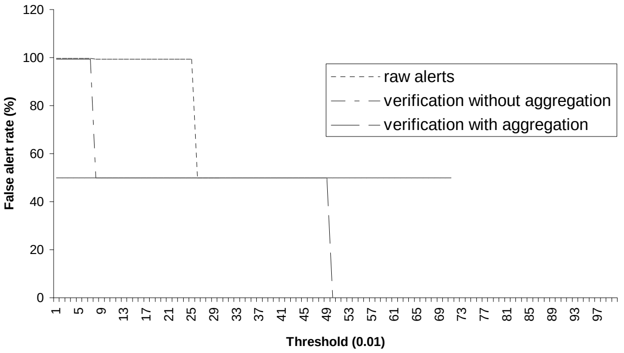


Fig. 21. False alert rate VS. threshold (Scenario 3)

D. Scenario 4

This attack scenario studies the attacks on a target with multiple vulnerabilities. The victim machine was configured to be running both a vulnerable Serv-U ftp service and a vulnerable IIS 5.0. Initial system scan showed that vulnerable Serv-U 5.0 is running on port 21 with public anonymous access, and IIS is vulnerable to unicode attacks.

We exploited the ftp vulnerability to attack the victim machine. The attack scenario includes 2 steps:

1) Remote buffer overflow attack to the Serv-U ftp and get remote root shell.
2) Modifying the web page file on the remote system.

During the attack, snort reported one “FTP command overflow attempt” alert, while Tripwire logged and reported the modification of web page file.

Thus, when combined with the false alerts generated by the background traffic, we have the alert-attribute network shown as in Figure 22.

After aggregation and making hypotheses, the final alert-attribute network is shown in Figure 23.

The confidence values of the related alerts before and after reasoning are shown in table VI.

From the comparison in table VI, we can see that although multiple vulnerabilities introduce multiple choices when making hypotheses and we can not have definite confidence in the hypotheses. The comparison result of the hypothesized attacks shows that when both preconditions are satisfied, the attack with a higher missing rate gains a higher confidence from the reasoning, which means that they are more expectable in reality.

When using probability threshold to decide whether an alert denotes a successful attack, the experiment yields the detection rate and false alert rate curves as shown in Figure 24 and Figure 25.

Table VII specifies the preconditions and postcondition of the attacks used in our experiments.

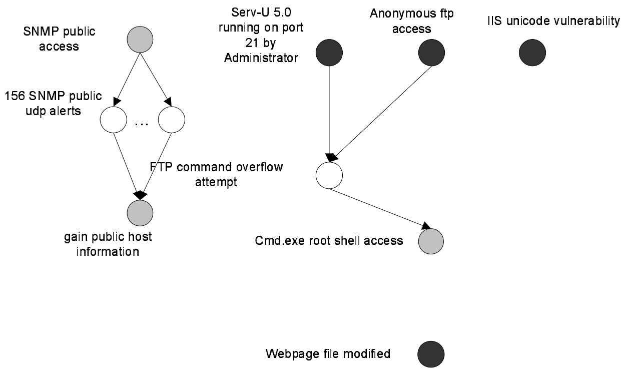

SNMP public access
156 SNMP public udp alerts
...
gain public host information
Serv-U 5.0 running on port 21 by Administrator
Anonymous ftp access
IIS unicode vulnerability
FTP command overflow attempt
Cmd.exe root shell access
Webpage file modified

Fig. 22. Initial alert-attribute network

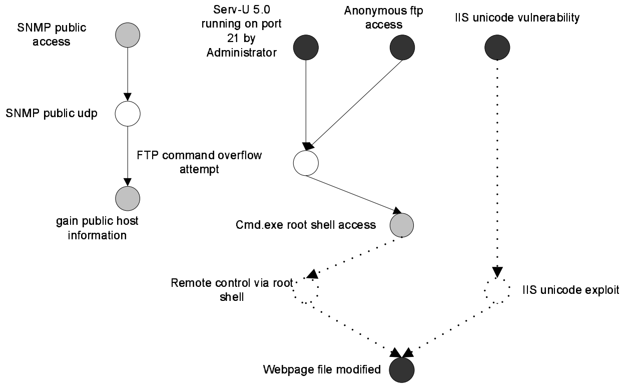

SNMP public access
SNMP public udp
gain public host information
Serv-U 5.0 running on port 21 by Administrator
Anonymous ftp access
IIS unicode vulnerability
FTP command overflow attempt
Cmd.exe root shell access
Remote control via root shell
IIS unicode exploit
Webpage file modified

Fig. 23. Updated alert-attribute network

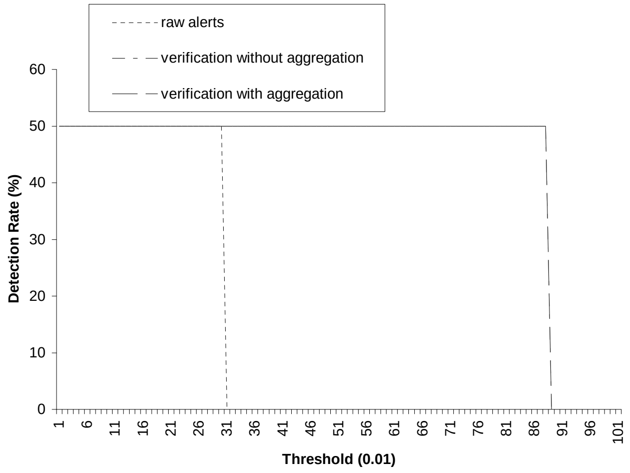


Fig. 24. Detection rate VS. threshold (Scenario 4)

Table VIII shows the prior probabilities about the attacks we used in our experiments.

TABLE VI

CONFIDENCE VALUES BEFORE AND AFTER THE REASONING

alert name	before reasoning	after reasoning	relative increase
FTP command overflow attempts	0.3	0.88235	194.12%
Remote control via cmd.exe root shell (hypothesized)	N/A	0.88235	N/A
IIS unicode exploit	N/A	0.29412	N/A
156 individual SNMP public access udp	0.075	0.075	0
Aggregated SNMP public access udp	0.5	0.5	0
other 169 alerts	0.25	0	-100%

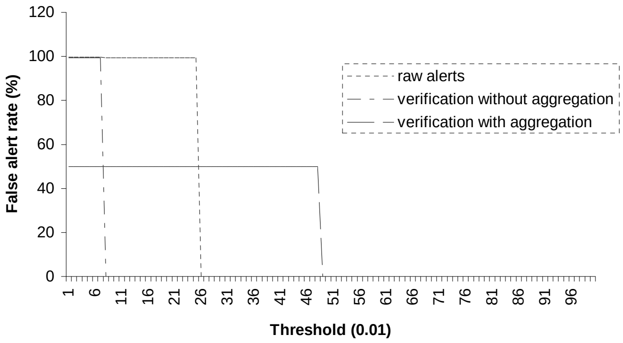


Fig. 25. False alert rate VS. threshold (Scenario 4)

TABLE VII

PRECONDITIONS AND POSTCONDITIONS OF THE ATTACKS IN OUR EXPERIMENTS

attack	precondition	postcondition
WEB-IIS cmd.exe access	IIS unicode vulnerability	{gain cmd.exe access}
WEB-IIS directory traversal attempt	IIS unicode vulnerability	{gain cmd.exe access}
FTP command overflow attempt	(Serv-U 5.0) $\wedge$(anonymous access)	{cmd.exe root shell access}
Modify web page / shutdown Norton Antivirus	root access	{web page modified / $\neg$Norton Antivirus running}
Install BackOrifice	(user access $\wedge$ $\neg$Norton Antivirus running)	{BackOrifice installed}
Install BackOrifice	(Norton Antivirus running)	{Virus BackOrifice quarantined}
FTP SITE EXEC format string attempt	(vulnerable wu-ftpd version $\leq$ 2.6.2)$\wedge$ (anonymous ftp access)	{Root shell access}
SNMP public access udp	SNMP public access	{gain public host information}
WEB-CGI redirect access	vulnerable ColdFusion / ClusterCATS	{gain account information}
ATTACK RESPONSE Invalid URL		{}

TABLE VIII

PRIOR PROBABILITIES OF THE ATTACK TYPES IN OUR EXPERIMENTS

attack	true alert rate	missing rate
WEB-IIS cmd.exe access	0.5	0.2
WEB-IIS directory traversal attempt	0.5	0.2
BACKDOOR BackOrifice access	0.6	0.5
FTP command overflow attempt	0.6	0.5
FTP SITE EXEC format string attempt	0.6	0.5
Remote control via cmd.exe root shell	N/A	1
Install BackOrifice	N/A	1
Modify web page via cmd.exe shell	N/A	1
Remote control via Trojan horse Glacier	N/A	1
Remote control via root shell	N/A	1
SNMP public access udp	0.15	0.5