

Automatic Discovery of Attack Messages and Pre- and Post-Conditions for Attack Graph Generation

Marco Carvalho and Choh Man Teng
Institute for Human and Machine Cognition
Pensacola, FL 32502, USA
mcarvalho@ihmc.us
cmteng@ihmc.us

Abstract: Network attack graphs are directed graph-representations of possible attack paths and vulnerabilities in a computer network. Each attack path is a sequence of steps taken by an attacker to achieve one or more goals in the target system. While there are some variations in the representations of the graph proposed by different researchers, typically the edges represent possible actions (or exploits) available to an attacker, and vertices represent the possible states for the system and applications. Attack graphs are often manually created or, less often, automatically generated from a set of attack models and detailed information about the network topology and its applications. There have been several proposals for the automatic identification and representation of attack models, but they all rely on some prerequisite knowledge of the pre- and post-conditions for the different attack steps. A pre-condition may include requirements such as “attacker must have root privileges”, while a post-condition defines the state of the system after an action is taken. In this paper we propose algorithms for the automatic identification of likely pre- and post-conditions that can be used for the generation of attack graphs. Our approach extracts such candidate conditions from observational data. By monitoring low-level events on multiple network nodes, in correlation with detected anomalies or attacks, our approach can automatically and unobtrusively identify the attributes of interest for the attack model required for attack graph generation. The paper provides a brief review of the requirements for automatic attack graph generation, and describes our proposed approach in detail. We also present preliminary simulation results for the automatic discovery of attack messages and their pre- and post-conditions, in a simplified fully connected network environment.

Keywords: attack graphs, network security, damage detection, graphical models

1. Introduction

As computer networks become larger and more complex, they become increasingly difficult to monitor and secure. Attackers exploit vulnerabilities in a system to perform unauthorized operations ranging from the prankish to the malicious.

Network vulnerabilities and attack signatures are typically hand coded or extracted from vulnerability scanner reports (for example Nessus 2009; Retina 2009; GFI LANguard 2009) and vulnerability databases (for example SecurityFocus 2009; OSVDB 2009; NVD 2009). Based primarily on this available vulnerability information, computer systems and networks are tested and analyzed to identify their specific weaknesses, such that appropriate safeguards or risk control measures can be devised. One common way to represent vulnerabilities specific to a particular network is to construct an *attack graph* of the network. An attack graph captures the sequences of attack steps that will lead to a security breach in a network, taking into account its configuration, vulnerabilities, and topology.

Traditionally, attack graphs are manually constructed and analyzed by security experts, but such graphs can also be automatically generated using for instance model checking methods, provided the prerequisites and consequences, called the *pre-* and *post-conditions*, of each attack step are known and encoded a priori (Ammann et al 2002; Sheyner et al 2002; Jajodia et al 2005; Lippmann et al 2006).

The identification of the pre- and post-conditions is a key bottleneck in the usability and effectiveness of attack graphs. The process is prone to errors and omissions, time consuming, and cannot respond to novel attacks that have not been pre-encoded. In addition, some devices, such as vulnerability scanners, actively probe the network for security flaws; they are intrusive and can interfere with the normal operation of the network. We developed a method to automatically identify attack messages and their pre- and post-conditions from observing ordinary network traffic data. This method can learn novel attack signatures automatically and unintrusively.

2. Attack Graphs

An attack graph represents all possible attack paths tailored to the specification of a network. An example of an attack graph is shown in Figure 1. Under this representation, each node encodes an atomic attack step that can be taken given the state of the network at that point. Each path from a root node to a leaf node represents a sequence of attack steps that constitute a successful attack.

Attack graphs can be automatically generated given the inputs described in Section 2.1. We will not describe the algorithms for generating attack graphs, as our focus is not on the generation of these graphs but on the automatic learning of the inputs to such algorithms. Selected attack graph generation algorithms can be found in (Ammann et al 2002; Sheyner et al 2002; Jajodia et al 2005; Lippmann et al 2006).

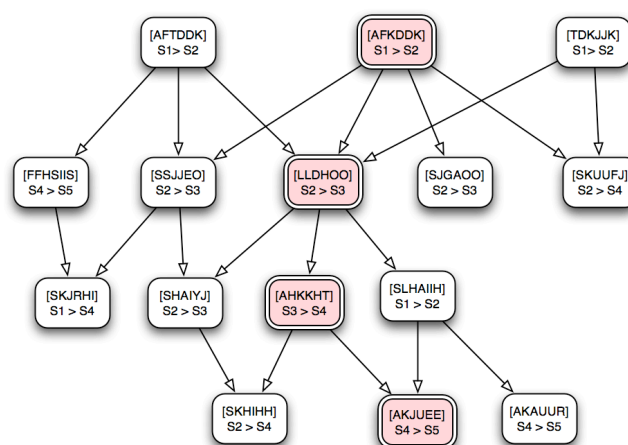


Figure 1. Example of an Attack Graph. The marked nodes constitute the steps in a specific attack sequence. In each node, the first line denotes the action taken, encoded as a message, and the second line shows the sender and receiver of the message.

Once an attack graph is generated, it can undergo detailed analysis to identify security loopholes, characterize the risks involved, and develop counteracting measures. For instance, we may be interested in a minimal set of operations that, if prevented, can block all attacks (as captured in the attack graph), or the optimal placement of intrusion detection units. Patches and reconfigurations of the network may be recommended, and given additional information on the probability of each attack step and the severity of each kind of failure, the implementation of these recommendations may be prioritized. Analysis of attack graphs can also support the estimation of the attack surface of a software system (Howard et al 2005).

2.1 Inputs for Automatically Generating Attack Graphs

To construct an attack graph, the following pieces of information are needed.

- Network vulnerabilities, typically obtained from security experts or vulnerability databases and scanners.
- Host configurations and services running on each host.
- Reachability relation describing the topology of the network and how the host machines are connected.
- Initial state of the attacker, including its privilege on the network hosts.

We will concentrate on the first item, the set of network vulnerabilities, and in particular, on the automatic identification of conditions that enable a successful attack. Current vulnerability analysis studies heavily rely on the use of automated scanners and attack tools or known vulnerability information combined from multiple databases using a common terminology

Common Vulnerabilities and Exposures (CVE 2009). While effective as a means to identify weaknesses against known attacks, the conventional approach is very limited against previously unseen attacks or even small variations of known attacks.

We instead pursue a method for automatically learning the pre- and post-conditions from the network traffic data. This method can respond to novel attacks by inducing attack signatures from patterns of network breaches and non-breaches. Moreover, we will rely only on observational traffic data, coupled with information about the network topology and host configurations. This avoids active probing of the network (the way vulnerability scanners gather information), which can be intrusive and disruptive for operational networks.

2.2 Pre- and Post-Conditions

As mentioned above, the construction of attack graphs assumes the set of vulnerabilities is known in advance. These vulnerabilities give rise to the attack steps and sequences potentially available to an intruder. An atomic attack step is defined by its pre- and post-conditions. The *pre-conditions* of an attack step are the prerequisites that must be satisfied for the attack step to be successful. The *post-conditions* are the consequences that will result once this attack step has been executed. Both types of conditions are defined in terms of the state of the network and the state of the attacker. By comparing the pre- and post-conditions of an attack step, one can deduce the changes brought about by this step.

3. Learning Attack Messages and Their Pre- and Post-Conditions

We aim to learn the pre- and post-conditions of atomic attack steps from observational network traffic data, augmented with information about the topology and configuration of the network. We will identify these conditions by first learning to distinguish attack steps from non-attack steps, and then extracting the pre- and post-conditions from the changes they induce on the system.

Learning by observation in this case requires some feedback from the nodes to identify that an attack has taken place in some part of the system. Furthermore, we expect that the overall system is able to sustain some levels of localized damage, possibly recovering to a normal state. This assumption is often used, for instance, in distributed tactical systems where nodes are capable of identifying and reporting 'damage' while maintaining an overall operation of the system (Carvalho 2009; Liu et al 2010). In some cases, nodes and applications deemed compromised can be removed from the system or re-initialized from a read-only image, providing valuable information that can be used for the runtime creation of defense mechanisms to improve the overall robustness of the system.

3.1 Method

Our method consists of two parts. First, we learn the steps that are part of an attack; second, we identify the pre- and post-conditions of each attack step from the changes in the state of the network associated with each attack step.

We will assume each step, attack or non-attack, can be generically denoted by a message delivered from the sender host to the receiver host. On receipt of a message, the receiver may ignore the message if certain execution conditions are not satisfied or it may accept the message and carry out the request specified in the message. The steps in an attack sequence do not arrive consecutively in a time block; typically attack steps are interleaved with a large number of non-attack steps.

Intuitively, messages that contribute to a successful attack occur more frequently than "normal" in the "short" time period leading up to a network failure. Our method looks for such regularities by making use of the probabilities and conditional probabilities of the occurrences of messages in the stream of network traffic.

In this context, normality is captured by the baseline rate of the occurrence of each message. The frequent occurrence of a message in general, or even the frequent occurrence of a message in the short period right before each failure, is not a sufficient indicator that the message contributes to the failure. It may be that the message is just associated with common services and requests.

However, if a message occurs more frequently shortly before a failure than it normally does in general, this is reasonable evidence to think that this message may have a role in the failure. In more concrete terms, if the conditional probability of the occurrence of a message, given that a failure occurred within a time period t after this message, is higher (taking into account statistical variance) than the baseline probability of the occurrence of this message in general when there is no attack within a time period t after this message, we will consider this message a candidate member of an attack sequence.

Once we have learned the attack messages, we can learn the pre- and post-conditions of these messages by considering the sets of accepted and rejected messages corresponding to each step and the states of the network before and after the occurrence of each type of message. This can be accomplished, for the pre-conditions of a message, by constructing a data set consisting of the system attributes before instances of this particular message were received by a node, and building a classifier to distinguish between those system attributes that correspond to the messages that were accepted, and those that were rejected. Similarly, the post-conditions can be learned by constructing a classifier from a data set of system states after instances of this message have been received (and either accepted or rejected).

3.2 Problem Setting

We assume a number of conditions on the network. Each attack step consists of one message, and all potentially relevant system states can be captured by a vector of sender and receiver node attributes. An attack consists of a sequence of attack messages, and an attack is successful if all messages in the attack sequence have been accepted. Attack messages and non-attack messages may interleave, but the whole attack sequence must be completed within a reasonably short time frame.

We do not know exactly when and whether a node has been partially compromised on accepting a message that is part of an attack sequence. We assume that the only observable evidence of attack occurs when an attack sequence is completed successfully, in which case the system fails in some tangible way (for example, the node detects damage or simply crashes). For the system, we assume that the configuration of the nodes and the connectivity between the nodes are both known.

4. Experimental Evaluation

As a first approximation, we consider a fully connected topology with 9 nodes. Each node plays the role of one of three types of servers: core servers, application servers and client machines. Core servers represent machines that provide basic infrastructural services such as DNS and authentication. Application servers are responsible for responding to requests from client nodes. Client machines are the users of services provided by the core and application servers.

Our simulation environment consisted of 3 core server nodes, 3 application nodes, and 3 clients. Nodes generated and sent messages following profiles that define different message and target distribution models (Table 1). At the beginning of each simulation run, one of the clients assumed the profile of an attacker, and others may also become attackers as they were compromised and taken over.

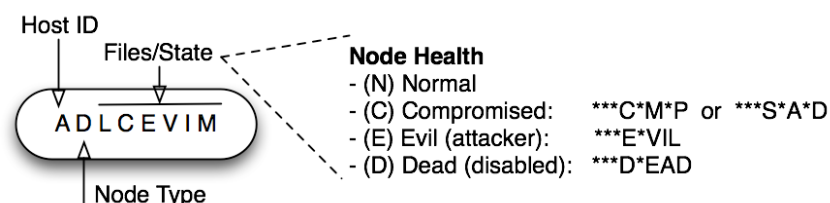


Figure 2. The State of a node showing the ground-truth for the node health status

Each node was represented by a vector of 8 symbols (as illustrated in Figure 2). There were 12 possible letters for each symbol. The first and second symbols were fixed and represented, respectively, the ID of the node and its Type (core, application, or client). The remaining six symbols represented the state of the node, which changed in response to messages.

For the purpose of this work, the attacker’s goal is to disable at least one node in the network, which consists of changing its health state to (D), as illustrated in Figure 2. A natural extension of this problem, which we will not cover in this paper, is the scenario where different nodes have different values, and an attacker has a particular interest in one given node in the network.

Table 1: Node Profiles and their message and target distributions

Profile	Message Distribution	Target Distribution
Core Node		
App. Node		
Client Node		
Attacker		

While every message changes the state of the network (at least at some level), most are innocuous to its health while others are designed for changing the state to a vulnerable state that will eventually lead to a disabled state, which is the goal of the attacker. The latter messages are the attack messages.

The goal of our proposed approach is to identify the attack messages, and the pre- and post-conditions required for the messages to successfully lead a target node to a disabled state. Only when a node becomes disabled is the system aware of the problem. All messages are

logged when exchanged between nodes. An example of the logged information is shown in Figure 3, for a single transaction between two nodes.

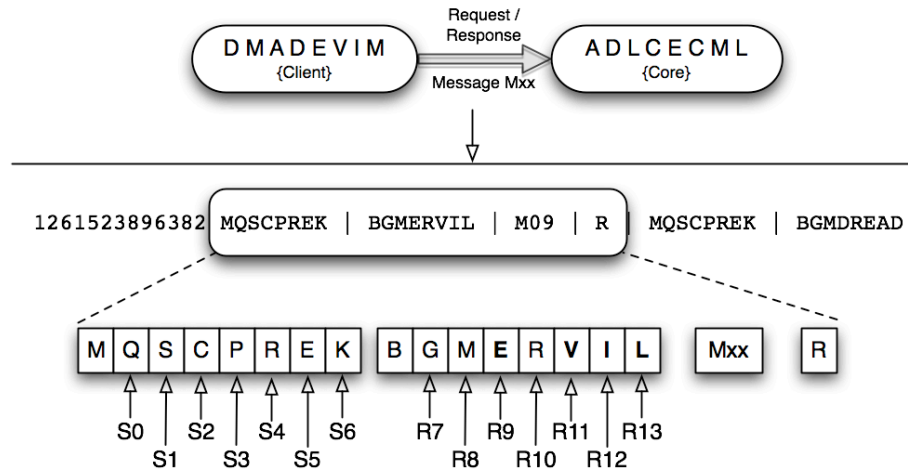


Figure 3. Example of a message in our simulation environment.

In Figure 3, the first column contains the time stamp of the message (for the node that is logging it), followed by the pre-state of the sender (MQSCPREK), the pre-state of the target node (BGMERVIL), the type of message (M09), a flag indicating if the message was accepted or rejected (A/R), and the posterior state of both the sender and the target nodes.

4.1 Attack Message Models

We defined twelve types of messages that can be exchanged between nodes. Four of those messages can be combined to build an attack sequence that may disable a target. The messages are identified as M01 through M12, and messages M09 through M12 are the attack messages.

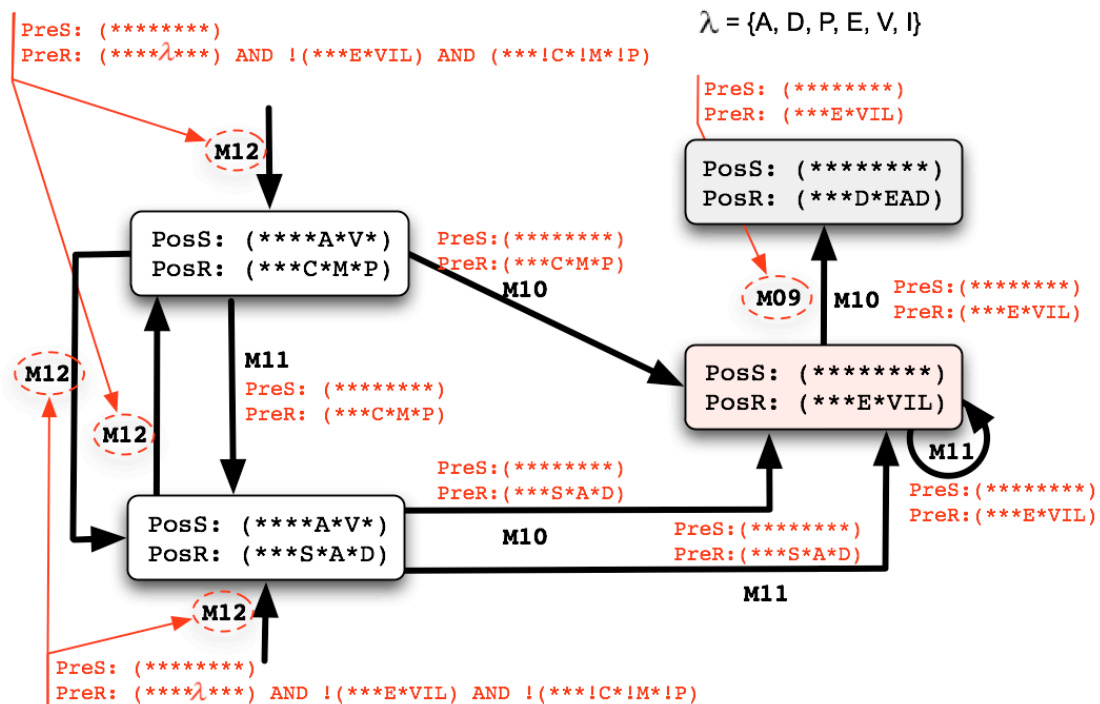


Figure 4. State transition diagram for network nodes

Figure 4 shows the state transition diagram for the network nodes. The vertices describe the posterior states of the sender (posS) and receiver (posR) after the receiver node accepts and processes an attack message sent from the sender node. The attack messages (and only the attack messages) are shown as edges in the diagram. Associated with each edge is the set of required preconditions for the sender (preS) and the receiver (preR) of the message.

The diagram was used for generating the data in our simulation. It also serves as the ground truth against which our results will be compared. However, note that the learning algorithm itself has no information about the diagram, and relies solely on the determination that a node became DEAD (i.e. was disabled) and the logged messages to identify the messages involved in the attack, and the pre- and post conditions of such messages.

4.2 Node Profiles and Message Distribution Models

The interactions between nodes in a real network are usually very complex, relying on a mixture of protocols and communication patterns that are platform, application and context dependent. In our simulation scenario we created three generalized profiles to illustrate the different classes of hosts. Each profile defines the behavior of the node in three dimensions:

- *Message Type Distribution:* A fixed set of messages M01-M12 is assumed for our simulations. The message type distribution defines the probability of a node choosing a specific message from the fixed set for transmission. Most nodes in the simulation favor non-attack messages, while the attacker (and nodes that eventually become compromised and converted to attacking nodes) favors a distribution that is heavier for the attack messages.
- *Message Rate Distribution:* The rate of message generation (and distribution) is defined as a random variable that is characteristic to different types of nodes. Core service nodes tend to follow a more stable pattern for message distributions while application servers and clients tend to have a higher variance on the message distribution rates.
- *Node Interaction Profile:* The node interaction profile defines the distribution of peers that a node will likely contact during a given session. Core servers are more likely to contact other core services providing basic capabilities such as DNS and authentication. Client nodes, on the other hand, are more likely to contact application servers and possibly other clients (peer-to-peer applications). Each class of nodes is associated with a pre-defined node-interaction profile.

We defined four profiles, as shown in Table 1. For each profile, the first column shows the probability distribution for the different kinds of messages generated, and the second column shows the probability of sending the generated message to each class of target nodes. Core nodes, for instance, only sent messages to other core nodes, while the other profiles had a more balanced (75%/25%) distribution between Application and Core Servers as targets. The message rate distribution is not shown in Table 1.

All nodes independently generate messages and identify targets following their current profile. Attacker nodes, as illustrated in Table 1, are more likely to generate attack messages (M09-M12), while other (non-attacker) nodes are more likely to generate other types of messages (M01-M08).

Once a node reaches the health state of E (**E*VIL), it becomes an attacker itself while continuing to be vulnerable to other attackers. As illustrated in Figure 4, in the E state, the node is more likely to move to health state D, becoming disabled. The algorithm has no prior knowledge about the state diagram in Figure 4, or the combinations of symbols that map to each state.

The exchange of messages between nodes eventually leads a node to a DEAD state, at which point the node signals that it has been disabled. Once a node is disabled, the attacker has achieved its objective and all nodes are reset to a new randomized state, and a new episode of the simulation can be started. The experimental setup enables us to learn about the attack messages and the pre- and post-conditions from observational data.

4.3 Attack Window

An attack window defines the area of interest for messages that are likely to have participated in the attack. When an attack succeeds, a node becomes disabled and the current episode is completed. It is assumed that messages involved in an attack are more likely to have occurred within a window of time (or interaction) right before the node becomes disabled.

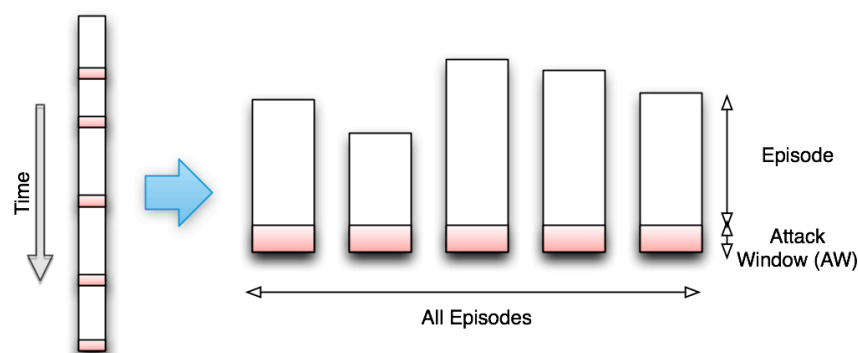


Figure 5. Splitting a sequential message log into multiple episodes

As illustrated in Figure 5, the sequential log of a node that becomes the victim of several successful attacks is shown on the left. After each completed attack, the system resets to a clean state (randomized to minimize the chance of a replay of the attack) and continues to operate. The episodes are of varying length due to the stochastic nature of the network model.

After each successful attack, an episode is extracted. An episode always ends with a node becoming disabled. Across multiple episodes, messages that show a (statistically significant) higher probability of occurring within the attack window than occurring elsewhere in the episode are assumed to be part of the attack sequence. However, this does not provide any information about the possible pre- and post-conditions that enable an attack, a task which we will tackle in the next section.

The width of the attack window was arbitrarily fixed to 35 messages, and our experiments showed that the choice was reasonable and gave fairly good results. Note that the size of the window does not affect the complexity of the algorithm. The message log is scanned once to tally the frequencies of each message type in the attack window and not in the attack window. However, using too small of an attack window would result in missing some attack messages that occur outside the window, and using too large of a window would dilute the difference between the frequencies of an attack message in and out of the window.

In our problem setting (see Section 3.2) we assume most message sequences would be completed within some reasonably short period of time, the length of which would be a good value for the window size. While we do not explore this issue here further, we could, without much increase in complexity, tally the frequencies of the messages in time blocks of the basic window size, and perform the comparison of the message frequencies using one or more blocks, creating multiples of the basic window size.

Except for extremely small (e.g. 1) and extremely large (e.g. the size of the whole episode) attack windows, messages that show a difference in frequency in and out of an attack window of any arbitrary size would be a reasonable candidate attack message. Further heuristics for identifying an appropriate window size were not explored as part of this work.

4.4 Learning Pre- and Post-Conditions

After we have identified the attack messages, we proceeded to learn their pre- and post-conditions. For each of the identified attack messages, we assembled two data sets, one for learning the pre-conditions and one for learning the post-conditions. Each instance in the pre-condition data set for message M_i consists of the state of the sender node before M_i was processed (S0-S6 in Figure 3), the state of the target node before M_i was processed (R7-R13 in Figure 3), and whether the message was accepted or rejected (A or R in Figure 3).

The post-condition data sets were similarly structured, using the states of the sender and target nodes after each message has been processed, except that S0 and R7 were excluded. These two attributes denote respectively the types of node (core server, application server, or client) of the sender and target, and cannot be reasonably changed by the messages.

A decision tree classification algorithm (weka J48) (Hall et al 2009) was used to construct decision trees to classify the data sets into two classes, accepted and rejected. The structure of the trees tells us the system state attributes that are relevant for distinguishing between the accepted and rejected messages. More specifically, the attributes that were used for splitting are possibly relevant attributes; the set of attribute-values belonging to the same path from the root to a leaf in a decision tree indicates a possible conjunctive grouping of the relevant attribute-values; and the set of different paths from the root to the leaves constitutes disjunctive conditions. These groupings are likely to correspond to the pre- and post-conditions of the messages.

5. Experimental Results and Analysis

Given the detailed discussion of the experimental setup in the previous section, the results of the experiments are fairly straightforward to describe. The goal is to learn from the simulated network traffic log the identity of the attack messages (and non-attack messages) and their associated pre- and post-conditions.

5.1 Learning Attack Messages

Of the 12 message types, 4 are attack messages: M09, M10, M11 and M12, and their pre- and post-conditions are given in Figure 4. We logged 1000 episodes, each episode starting from a random initial network state and ending when an attack sequence has been successfully completed. Using the attack window method described above, all and only the four true attack messages were tagged as such. In other words, the probability of each of these attack messages occurring within the attack window was greater than the probability of the message occurring elsewhere in the episode. The differences were statistically significant using a t-test at 0.05 significance level.

5.2 Learning Pre- and Post-Conditions

For each of the data sets assembled as described in Section 4.4, we constructed a decision tree using J48's default parameters. The attributes used for splitting and their values in each case are shown in Table 2.

Table 2: Attributes used for splitting and their values for accepted messages in the decision trees built using the state vectors before and after a type of message was processed. For uniformity the "State After" entries are also shown with 7 attributes, even though the first ones (S0 and R7) were not used in the specification of post-conditions.

	Sender (State Before)	Target (State Before)	Sender (State After)	Target (State After)
M09		(**E*VI*)		(**D*E*D)
M10		(**S*A*D) or (**C*M*P) or (***VIL)		(**D*E*D) or (***VIL)
M11		(**S*A*D) or (**C*M*P) or (***VIL)		(**S*A*D) or (***VIL)
M12		(***{ADPEVI}***) and !(***VIL*)	(***A*V*)	(**C*M*P) or (**S***D)

By comparing Table 2 to Figure 4, we can verify that the attributes and attribute values identified by the classifiers are all part of the corresponding pre- and post-conditions of the messages. In most cases, the conditions identified are missing one of the attribute-values; for instance, the pre-condition for message M09 should be, for the target node, **E*VIL, but only three of the four attribute-values (E, V and I) were identified.

We believe one of the reasons for this behavior is that the data we logged were relatively sparse, compared to the number of attributes in the state vectors (14) compounded with the number of possible values for each attribute (12). Thus, there are not enough cases (and variety) in our simulated data to require the full discriminating power of all the components in the pre- and post-conditions. We are working to resolve this issue.

6. Conclusions

In this paper we introduced an approach for the automatic identification, from observational data, of attack messages and their pre- and post-conditions required for a successful attack. While still a work in progress, our initial results are encouraging and we believe an optimized version of the proposed approach could greatly increase the utility of attack graphs as security and analysis tools, with the added capability of learning potential attack messages and attack pre- and post-conditions from observational data, one of the key bottlenecks for automated attack graph discovery can be overcome.

As a continuation of this effort, we will explore the effects of the attack window and the required number of episodes for learning new attack conditions. We will also continue our work on partially connected topologies, better representing enterprise networks, improve our capability to a larger number of states and evaluate our approach with real attack data collected on honeynets.

Acknowledgement

The work presented in this paper has been partially sponsored by the U.S. Army Research Laboratory, under Cooperative Agreement No. W911NF-07-2-0022, CFDA No. 12.630.

References

- Adam, Nabil R. and Worthmann, John C. (1989) "Security-Control Methods for Statistical Databases: A Comparative Study", *ACM Comput. Surv.*, 21:4(515-556).
- Ammann, Paul, Wijesekera, Duminda and Kaushik, Saket (2002) "Scalable, Graph-Based Network Vulnerability Analysis", *Proceedings of the ACM Conference on Computer and Communications Security*, pages 217-224.
- Carvalho, M. (2009) "A Distributed Reinforcement Learning Approach to Mission Survivability in Tactical MANETs", *Cyber Security and Information Intelligence Research Workshop, CSIIRW 2009*, Oak Ridge National Laboratory, April 2009.
- CVE (2009) *Common Vulnerabilities and Exposures*, <http://cve.mitre.org/>. Retrieved on Dec. 28th, 2009.
- GFI LANguard (2009) <http://www.gfi.com/languard/>. Retrieved on Dec. 28th, 2009.
- Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer Bernhard, Reutemann, Peter and Witten, Ian H. (2009) "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, Volume 11, Issue 1.
- Howard, Michael, Pincus, Jon and Wing, Jeannette M. (2005) "Measuring Relative Attack Surfaces", *Computer Security in the 21st Century*, pages 109-137, Springer.
- Jajodia, Sushil, Noel, Steven and O'Berry, Brian (2005) "Topological Analysis of Network Attack Vulnerability", *Managing Cyber Threats: Issues, Approaches and Challenges*, pages 247--266, Springer.
- Liu, P., Jia, X., Zhang, S., Ziong, X., Jhi, Y., Bai, K. and Li, J. (2010) "Cross-Layer Damage Assessment for Cyber Situational Awareness", *Cyber Situational Awareness, Advances in Information Security*, 46 Springer, 2010.
- Lippmann, Richard, Ingols, Kyle, Scott, Chris, Piwowarski, Keith, Kratkiewicz, Kendra, Artz, Mike and Cunningham, Robert (2006) "Validating and Restoring Defense in Depth Using Attack Graphs", *Proceedings of the IEEE Military Communications Conference*, pages 1-10.
- NVD (2009) *The National Vulnerability Database*, <http://nvd.nist.gov/>. Retrieved on Dec. 28th, 2009.

Nessus (2009) *Nessus Vulnerability Scanner*, <http://www.nessus.org/>. Retrieved on Dec. 28th, 2009.

OSVDB (2009) *The Open Source Vulnerability Database*, <http://osvdb.org/>. Retrieved on Dec. 28th, 2009.

Retina (2009) *Retina Vulnerability Scanner*, <http://www.eeye.com>. Retrieved on Dec. 28th, 2009.

SecurityFocus (2009) <http://www.securityfocus.com/>. Retrieved on Dec. 28th, 2009.

Sheyner, Oleg, Haines, Joshua, Jha, Somesh, Lippmann, Richard and Wing, Jeannette M. (2002) "Automated Generation and Analysis of Attack Graphs", *Proceedings of the IEEE Symposium on Security and Privacy*, pages 273-284.