

Prediction and Visualization of Malware Propagation on Large Networks

Sanjeeb Nanda

SDS International Inc., Advanced Technologies Division
3403 Technological Avenue, Suite 7
Orlando, FL 32817
407-282-4432
snanda@sdslink.com

Keywords:

Malware, large networks, attack graphs, propagation, prediction

ABSTRACT: *The rapid growth of the Internet over the years has triggered an explosion in the number of networked applications leveraging its capabilities. Unfortunately, many of them are intentionally designed to burden or destroy the capabilities of their peers and their supporting network infrastructure. Hence, considerable commercial development has been focused on detecting, quarantining and annihilating these malicious applications. However, the enormity of the Internet poses a formidable challenge to such efforts, with some of the salient hurdles being; the visualization of the state of the network from a bird's eye view to obtain a synopsis of the scope of an attack, as well as detailed views from proximity to exhibit the damage rendered with clarity; forecasting the network vulnerabilities that might be exploited by such applications for their propagation; and deriving solutions for the application of countermeasures to defend against attacks. In response to these needs, this paper presents methods for the visual analysis of large cyber attack graphs that embodies the algorithms to analyze an attack and display its proliferation and anticipated growth.*

1. Introduction

An attack graph can be expansive in several respects. One is the sheer number of edges representing potential exploits originating from a system that target others. In this respect, the network is a directed multigraph admitting multiple directed edges between a given pair of nodes, where each node represents a system. Therefore, to simplify the problem of displaying such a graph without congesting the display, techniques are needed to selectively exclude exploits from display without compromising the informative value of the visualization. To achieve this, we may consider the following methods.

- Typically an observer is only interested in attack paths on a network. Therefore it may suffice to display only edges that represent exploits rather than the connections between all pairs of systems. An alternative is to display edges that do not represent exploits using lesser visual emphasis. This may be accomplished using a combination of methods such as the use of decreased value of alpha in a 32-bit RGBA color to render such lines with a transparent effect, or the use of lighter shades of gray or thinner lines in OpenGL.
- An attack is meaningful only if it targets an open port. Hence, it may suffice to display only those edges that

target such vulnerable ports. For instance, if an exploit targets an unopened port then the user may not require the directed edge corresponding to that exploit to be displayed.

2. Visualization Criteria

Users may also choose to visualize events that have been selectively filtered from an alert log using one or more choices in a set of criteria designed to facilitate the visualization of key aspects of exploits on a given network. Some of the candidate criteria for selection that may be integrated into the filtering module are as follows.

2.1. Attacks with the highest severity on each system

An observer may be interested only in attacks on a system with the highest severity. Thus, for a given origin and destination node it may suffice to display only that edge between them that corresponds to the attack with the highest severity. The observer may then imply that exploits with lesser severity may exist from the same source to the same destination. For example, the popular rules developed by Sourcefire's Vulnerability Research Team (VRT) are each assigned a security identifier (SID) in the rule file having the extension *.rules*. Additionally,

each rule is a member of a class that is assigned a unique priority level ranging from 1 (highest) to 4 (lowest) indicating its level of severity in the file *classification.config*. Figure 1 displays a sample set of classes with their corresponding priorities. Thus all rules that are members of a given class have the same priority. Note that, spaces have been added after the commas in the class specifications to improve their readability at the expense of the correctness of syntax.

```
class: attempted-user, Attempted User Privilege Gain, 1
class: successful-dos, Denial of Service, 2
class: unknown, Unknown Traffic, 3
class: tcp-connection, A TCP connection was detected, 4
```

Figure 1. Sample class types in classification.config

The priority levels associated with the classes specified by Sourcefire VRT have coarse granularity and are therefore simpler to visualize using corresponding color codes. On the other hand, finer granularity of severity can be obtained using the Common Vulnerability Scoring System (CVSS) at the National Vulnerability Database (NVD) at the URL <http://nvd.nist.gov/nvd.cfm> with values ranging from 0 through 10. For example, CVE-1999-0166 identifies the vulnerability where NFS allows users to change to the parent directory to access directories other than those exported by the file system. This vulnerability has a score of 3.3 with a qualitative severity of “low.” In contrast, CVE-1999-0080, which identifies the vulnerability where a FTP server allows root access via the “site exec” command, has a score of 10 with a qualitative severity of “high.” Fortunately, many of the rules defined by Sourcefire VRT in their corresponding *.rules* file have their corresponding CVE number specified alongside wherever applicable. Thus, the visualization application can be designed to use either categorization scheme indicated by the user to identify the severity of an exploit.

2.2. Attacks above a threshold severity

The severity of certain attacks may be low enough to be inconsequential to the observer. Hence the observer may desire to filter out attacks that are below a certain threshold. Filtering using this criterion has the added advantage of reducing the number of vertices (representing systems) and the edges incident to them (representing attacks) that have to simultaneously be displayed. This can substantially reduce cluttering while providing the observer a very concise picture of the meaningful exploits that affect the network at any given time. Figure 2 displays a hypothetical network with edges colored to represent attacks on the systems that equal or

exceed a given threshold. At top left, the edges colored green represent attacks that equal or exceed the lowest priority (i.e., 4); at top right, yellow for attacks that equal or exceed priority 3; at bottom left, orange for those that equal or exceed priority 2; at bottom right, red for those that equal 1. Finally, the links to untargeted systems are shown in black. Now, when a higher priority level is selected as the threshold, the number of qualifying directed edges decreases and vice-versa. Note that, the color spectrum in Figure 2 run from left to right – green, yellow, orange and red. Also, each exploit path appears lighter than other edges when viewed in grayscale.

The nodes of the network shown in Figure 2 are real 3D models representing gateways and routers constructed using 3DMax, and are directly usable by the proposed OpenGL application for rendering in virtual 3D space. These may be seen with greater clarity by zooming in into the document. Similarly, the colored directed edges of the network correspond to the physical path taken by packets such as that yielded by an IP trace, and are easily rendered using simple 3D line and color primitives in OpenGL.

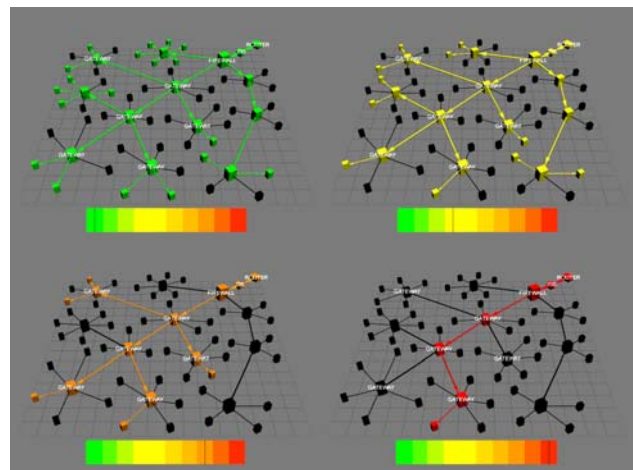


Figure 2. A hypothetical network showing exploits with increasing severity between systems.

2.3. Visualizing only those edges that qualify into a given category of exploits

An observer may have interest in only those events that belong to a given category of exploits. For example, a user may be only interested in exploits by viruses using illegal attachments. Sourcefire VRT rules for Snort are partitioned into the following categories, each corresponding to a type of protocol, application or action. Note that rules within a given category may not necessarily be in the same class. Nevertheless, such

grouping may be used to display only those exploits in a given category of interest.

- | | | |
|-------------------|-----------------|---------------|
| ▪ Attack response | ▪ Miscellaneous | ▪ Shell code |
| ▪ Backdoor | ▪ Multimedia | ▪ SMTP |
| ▪ Bad traffic | ▪ MySQL | ▪ SNMP |
| ▪ Chat | ▪ NetBIOS | ▪ SQL |
| ▪ DNS | ▪ NNTP | ▪ Telnet |
| ▪ DOS | ▪ Oracle | ▪ TFTP |
| ▪ Experimental | ▪ Other IDS | ▪ Virus |
| ▪ Exploit | ▪ P2P | ▪ Web attacks |
| ▪ Finger | ▪ Policy | ▪ Web CGI |
| ▪ Ftp | ▪ POP2 | ▪ Web client |
| ▪ ICMP | ▪ POP3 | ▪ Coldfusion |
| ▪ ICMP Info | ▪ Pornography | ▪ FrontPage |
| ▪ IMAP | ▪ RPC | ▪ Misc. Web |
| ▪ Informational | ▪ RServices | ▪ PHP |
| ▪ Local | ▪ Scan | ▪ X11 |

2.4. The propagation of exploits using traced paths

One of the details that an observer may want to visualize is the propagation of attacks on a given network along the actual paths used by the IP packets. Using this, an observer can gauge the extent and speed of propagation of attacks over a desired span of time. This is accomplished by filtering events in the alert log based on the time at which they were captured. For instance, consider the two sample events shown in Figure 3 that were generated using Sourcefire VRT rules. The timestamp of each event is shown in bold to facilitate their identification. We note that the first event was recorded on 03/11 at 22:58:06.420120 while the latter was on the same day as well, but approximately a second later at 22:58:07.391217.

```

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Miscellaneous activity] [Priority: 3]
03/11-22:58:06.420120
80:32:20:0:4:0 → 4:0:4:0:0:0 type:0x800 len:0x4A
207.69.188.186 → 4.235.66.45 ICMP TTL: 56
TOS:0x0 ID:34453 IpLen:20 DgmLen:60
Type:0 Code:0 ID:768 Seq:256 ECHO REPLY

[**] [1:1620:5] BAD TRAFFIC Non-Standard IP [**]
[Classification: Non-standard protocol or event]
[Priority: 2]
03/11-22:58:07.391217
80:32:20:0:4:0 → 4:0:4:0:0:0 type:0x800 len:0x4EB
65.187.160.60:7951 → 4.235.66.45:1026 UDP TTL: 120
TOS:0x0 ID:49366 IpLen:20 DgmLen:1245
Len: 1217

```

Figure 3. Timestamps of events in a Snort generated alert log.

It is then straightforward to visualize the propagation of exploits by progressively retrieving events that were captured within an increasing window of time and cumulatively displaying them. Suppose that $\{t_0, t_1, t_2, \dots, t_N\}$ is a sequence of uniformly distributed points in time with t_0 and t_N being the starting and ending times within which events are to be visualized. Then we may first display all events that were captured within the time interval (t_0, t_1) followed by the display of all events captured within the interval (t_0, t_2) or earlier, followed by (t_0, t_3) or earlier, and so on. This will result in the display of the propagation of exploits over each time interval. Furthermore, this idea can be applied onto subsets of events that have been already filtered by their severity or category. Figure 4 contains a sequence of three frames displaying the hypothetical propagation of attacks over a network over time, where the color of each edge (u, v) corresponds to the attack with highest severity from node u to v at that specific time, with severity levels indicated by color – red is level 1 (highest), orange is 2, yellow is 3 and green is 4 (lowest). For instance, if an edge (u, v) is colored green (corresponding to the severity level 4), we can assume that attacks from u to v have occurred with a severity level of 4 only. Note that, the time interval is advanced using a slider bar control as shown at the bottom of each frame. As the time interval in Figure 4 is advanced, an increasing number of edges are shown colored and, in some cases, edges that were previously identified as a low risk (4) or no risk, actually transition to a higher risk level.

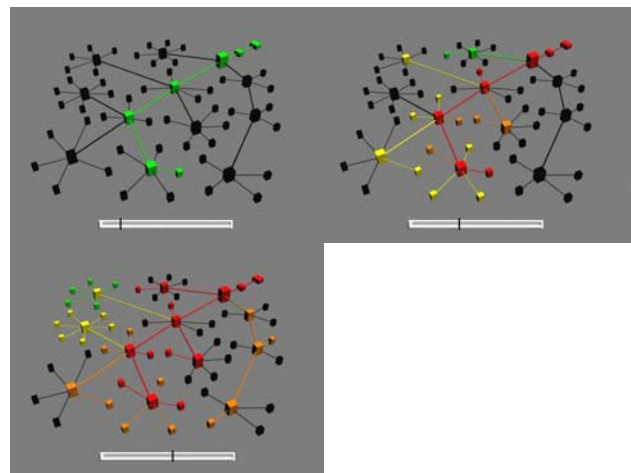


Figure 4. Propagation of exploits over time.

Equally important to an observer may be the visualization of the propagation of attacks on a network along the logical paths used by the IP packets. That is, only the source and destination nodes of an attack are relevant to the observer. While this is simpler to render, the effects can be visually messy with the potential for the display of multitudes of overlapping edges. One way to avoid this

problem is to display only attacks captured within its corresponding interval of time as illustrated in Figure 5. In this case, the first frame displays a compromised system that is the source of an attack. The next shows all the attacks that were observed to be originating from that system to various targets within the corresponding interval of time. Similarly, the last frame displays all the attacks emanating from the systems compromised in the previous interval of time.

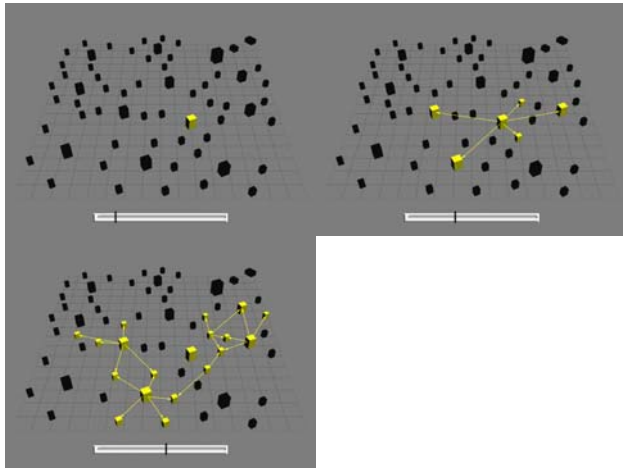


Figure 5. Propagation of exploits over time on logical connections.

2.5. Visualizing the compromised status of systems

A user may not only be interested in visualizing attacks on the systems in a network, but also in distinguishing systems that have been compromised by such attacks. This is easily determined to be true if the result of AND-ing the subnet mask of that network and the source IP address of the exploit equals the result of AND-ing the same subnet mask and the destination IP address of that exploit. For example, if the source and destination IP addresses of an exploit are 192.168.225.359 and 192.168.225.201 respectively, and the subnet mask of that network is 255.255.255.0, then we know that the system with the address 192.168.225.359 has been compromised. This is based on the assumption that any system within a given network – in this case a class C network – is initially in a unaffected state and its role as the source of an exploit can only mean that, its has been compromised. Compromised systems may be colored suitably (such as red) to discern them from those that are vulnerable (yellow) and those that are in an unaffected stated (green).

2.6. Visualizing the aggressiveness of systems

Systems that have been already compromised may display varying levels of aggressiveness expressed by the

numbers and severity of attacks they initiate on their peers. This variation may result due to factors such as differences in system configuration that prevent malware on the compromised systems from accessing the resources necessary for initiating specific attacks. For instance, the number of attacks originating from a system with compromised email is proportional to the number of entries in its corresponding address book. Aggressiveness may be visualized concisely by the out-degree of each node and the colors of the corresponding directed edges. While displaying the actual number of directed edges originating from each system to indicate the number of attacks each has initiated is useful, it can lead to substantial cluttering. A visually appealing alternative is to color the node representing such systems in a manner that is representative of its out-degree. For instance, red may represent systems with out-degrees in the highest interval, followed by orange, yellow and green.

Note that, the use of node coloring and the meaning of a color vary with the visualization criterion chosen by the user. Thus, a green colored node indicates an unaffected system when the criterion is to display compromised nodes, while that same color indicates a system with relatively low aggressiveness compared to its peers in terms of the number of attacks it has initiated.

To enhance situational awareness of the exploits and vulnerabilities affecting a network, a visualization application must be capable of rendering models of the various systems, gateways, routers, firewalls, intrusion detection and prevention systems, and other salient components of interest that constitute it. Furthermore, the connectivity between these components must be displayed with fidelity and their depiction reasonably representative of their physical layout. While 2D top-down views accurately depict scale and layout, they are not innately suited for human cognition that is geared towards observing and appreciating 3D environments and objects. For this reason, 3D renderings are a more natural and efficient choice for conveying information [8].

3. Predicting Propagation

While the illustration of past or current characteristics of attacks on a network is important, it is nevertheless only a stepping stone for answering the more vital question of which resources are likely to be targeted next. The answer to that question is a decisive factor that enables network administrators to make decisions regarding quarantining or decoupling systems, or groups of systems, that are highly susceptible to attack. Such predictions can be made using a Markov model as described next.

A Markov model for estimating the future state of an attack samples the state of the attack at discrete intervals

over a given span of time. Assuming that the number of states is finite, we derive the conditional probability of the attack being in a certain state in a given sample, subject to it being in another state in the previous sample. Note that the two states may not necessarily differ. The set of transitional probabilities for each pair of states for the current and previous samples is then used to determine the probability of the attack being in any of those finite states in the future. This model has been extensively studied and used to characterize and predict the behavior of processes such as motorists and adversaries on the battlefield [2] [3] [7]. We briefly describe this model and the proposed algorithm next.

Let x_t be the state of a potential attack at an instance of time t . The sequence of states $\{x_0, x_1, x_2, \dots, x_t, x_{t+1}, x_{t+2}, \dots\}$ of such an attack may be viewed as a stochastic process, where we can predict the occurrence of any state in the future with a probability only. Furthermore, it is intuitive to assume that the probability of occurrence of a certain state in the future is dependent only on the current state as opposed to any past state. This assumption is the basis for a Markov chain model that may be stated formally as follows. Let $S = \{s_0, s_1, s_2, \dots, s_M\}$ be the finite set of values characterizing an attack's state. Then the probability that the state of the attack x_t at time t having the value s_i is given by the conditional probability dependent on the state of the attack x_{t-1} at time $t-1$ having the value s_j . This may be denoted as $\Pr(x_t = s_i | x_{t-1} = s_j)$ and is called the transitional probability between state s_j to s_i .

Let us assume that if an attack is in state s_j at time t the probability of it being in state s_i at time $t+1$ is p_{ij} for all instances of t . Then Q represents the transition matrix of the Markov chain, where entry (i, j) i.e., in row i and column j , gives the probability p_{ij} .

$$Q = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1\ m-1} & p_{1\ m} \\ p_{21} & p_{22} & \dots & p_{2\ m-1} & p_{2\ m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{m-1\ 1} & p_{m-1\ 2} & \dots & p_{m-1\ m-1} & p_{m-1\ m} \\ p_{m\ 1} & p_{m\ 2} & \dots & p_{m\ m-1} & p_{m\ m} \end{bmatrix}$$

In Q , each probability $p_{ij} \geq 0$. Also the sum of the probabilities in each row must equal 1 for the reason that from a given state with probability p_{ij} the state of the attack can transition to at most m states in S . If an attack has the transition matrix Q , its probability of being in state s_i at time $t+2$ given that it is in state s_j at time t is given by entry (i, j) in the matrix Q^2 . It is easy to see why, since a transition from state s_j at time t to state s_i at time $t+2$ can transition through any of m states at time $t+1$.

Hence the probability of that attack being in state s_i at time $t+2$ must equal,

$$\sum_{k=1}^m p_{ik} p_{kj} \cdot \text{This, of course equals entry } (i, j) \text{ in the matrix } Q^2.$$

Therefore, in general the probability of that attack being in state s_i at time $t+\delta$ given that it is in state s_j at time t is given by entry (i, j) in the matrix Q^δ . Then given the state probability vector $w = (p_1, p_2, \dots, p_m)$ where p_j represents the probability of that attack being in state s_j at time t , wQ^δ gives the probability of that attack being in any of the states at time $t+\delta$. But since the state of the attack is known to be in some state s_j at time t , w is a unit vector with 1 at index j , and 0 elsewhere.

We now identify some of the prospective candidates for defining the state of an attack. We may define a state as a tuple (x_1, x_2, \dots, x_L) where each x_i is an attribute with a value in its corresponding domain A_i . For example, we may choose A_1 to be the set of operating systems, with version, known to be targeted by it; A_2 to be the set of logical ports on which attacks can occur; A_3 to be the set of address values of the systems within a subnet mask, e.g., 1-255 in a class C network; A_4 to be the class of the system e.g., server, router/gateway or workstation; and finally, A_5 to be the set of exploitable applications. These are just a few choices for attributes that may be used to define the state of an attack.

The choice of the aforementioned attributes is based on the following reasoning. The modus operandi of each attack follows a pattern prescribed by the architect of the corresponding malware. Hence, a malware is likely to display characteristic behavior with respect to its choices of operating systems (attribute domain A_1), logical ports (attribute domain A_2), set of address values within a subnet (attribute domain A_3), the class of the system (attribute domain A_4) and the exploitable applications (attribute domain A_5) on which to initiate or propagate attacks. Furthermore, it is also likely to exhibit discernable traits in its transition from one state to another. That is its probability of attacking a system with state $(b_1, b_2, b_3, b_4, b_5)$ when having already compromised a system with state $(a_1, a_2, a_3, a_4, a_5)$ where a_1, b_1 represent the class of the systems, a_2, b_2 represent the operating systems, a_3, b_3 the IP addresses, a_4, b_4 the logical ports, and a_5, b_5 the targeted applications.

Matrix computations can be expensive with $N^2(2N-1)$ multiplication and addition operations required to multiply two $N \times N$ matrices. A matrix used for our modeling is typically quite large. For instance, if we construct a Markov model comprised of states defined by just four attributes, each with 10 values, we would have

10^4 possible states. Then, Q would be of size $10^4 \times 10^4$, requiring 1.99999×10^{12} operations to derive Q^2 . Note that, contemporary system processors running at over 3 GHz deliver approximately 10^{10} floating point operations per second and would require over 3 minutes to compute Q^2 . In practice, we are likely to have far more attributes and larger sets of discrete values as the domain for each attribute. The problem is further exacerbated by the fact that Q^2 only provides the probabilities of outcomes at the next instance of time. Hence, the need to glimpse the likelihood of outcomes further out magnifies the computational load significantly. For instance, if the state of a potential attack were sampled with a periodicity of 1 second to obtain the probabilities of outcomes after 1 minute, we would need to compute Q^{60} . A brute-force approach that iteratively computes $Q^2, Q^3, Q^4, \dots, Q^{60}$ will then take over three hours using a single processor. Needless to say, this is a prohibitive value. However, one can easily optimize the process by observing that $Q^4 = (Q^2)^2, Q^8 = (Q^4)^2, Q^{16} = (Q^8)^2$, and so on. Thus the computation of Q^{60} can be achieved in just 8 steps by computing $Q^2, Q^4, Q^8, Q^{16}, Q^{32}, Q^{48} = Q^{16}Q^{32}, Q^{56} = Q^8Q^{48}$, and finally, $Q^{60} = Q^4Q^{56}$. Although this method offers considerable computational relief, it is still unacceptable because the time required for computing the solution exceeds the span of time within which the solution is desired. We overcome this challenge in the following manner.

At any given time, we know the state s in which a potential attack exists at time t . Assuming that there are N states, we also know the transitional probabilities to every other state from s given by the vector $p = (p_1, p_2, \dots, p_N)$. Then what we need is to iteratively compute the following state probability vector:

$$p^{t+1} = p^t Q, \text{ for } t = 1, 2, \dots \quad [E1]$$

Assuming that we use the aforementioned, 3 GHz, system processor, each iteration of [E1] will require 0.01999 seconds. Thus, for example, the solution to the previously mentioned problem that required the computation of probabilities of outcomes after 1 minute can be generated in 1.1994 seconds. Without a doubt, this mechanism yields solutions far more expeditiously, and is therefore pragmatic for deployment in operational tools. We describe this approach in Figure 6.

EstimateProbabilities()

```
{
  // Initialize the number of observed occurrences
  // of all possible state transitions to zero.
  Step A:
  for all possible  $s_i$  and  $s_j$  do
  {  $sum(s_i, s_j) \leftarrow 0; sum(s_i) \leftarrow 0;$ 
   $past\_state \leftarrow 0;$ 
```

```
// Tabulate the states of the attack across time.
```

Step B:

```
for time  $t \leftarrow 1$  to  $N$  do
{
  if ( $current\_state = s_i$ ) and ( $past\_state = s_j$ )
  {
     $sum(s_i, s_j) \leftarrow sum(s_i, s_j) + 1;$ 
     $sum(s_j) \leftarrow sum(s_j) + 1;$ 
  }
}
```

```
// Compute the state transition probabilities.
```

Step C:

```
for all possible  $s_i$  and  $s_j$  do
{  $Pr(s_i, s_j) \leftarrow sum(s_i, s_j)/sum(s_j);$ 
Construct  $Q = [q_{ij}]$ , with  $q_{ij} = Pr(s_i, s_j);$ 
```

```
// Compute the state probability vector after time
//  $\Delta T$ . Let sampling frequency be  $f$  Hz, and the
// initial state probability (unit vector) be  $w$ .
```

Step D:

```
for  $t \leftarrow 0$  to  $\Delta T/f$  do
{
   $w \leftarrow wQ;$ 
}
}
```

Figure 6. The Markov chain computation across multiple transitions.

4. Conclusion

The algorithm we have described for computing the likelihood of states in the future is fairly straightforward. However, the real challenge in its implementation lies in the choice of the various attribute domains that constitute the states.

5. References

- [1] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217 – 224, 2002.
- [2] T. Kumagai, Y. Sakaguchi, M. Okuwa and M. Akamatsu, "Prediction of Driving Behavior through Probabilistic Inference," *Proceedings of the Eighth International Conference on Engineering Applications of Neural Networks*, pp. 117 – 123, 2003.
- [3] S. Nanda and J. Weeks, "Contemporary Models for Path Prediction of Dynamic Entities", *Proceedings*

of the Spring 2006 Simulation Interoperability Workshop, vol. 2, pp. 479 – 487, 2006.

- [4] Z. Nikoloski and N. Deo, “The Game of Cops and Robbers on Graphs: A Model for Quarantining Cyber Attacks,” *Congressus Numerantium*, vol. 162, pp. 193 – 215, 2003.
- [5] Z. Nikoloski and N. Deo, “On the Complexity of Quarantining Cyber Attacks,” *Discrete Applied Mathematics*, (2006), under review.
- [6] S. Noel, S. Jajodia, “Managing Attack Graph Complexity through Visual Hierarchical Aggregation,” *Proceedings of the ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 109 – 118, 2004.
- [7] V. Pavlovic, J. M. Rehg, T-J. Cham, and K. P. Murphy, “A dynamic Bayesian network approach to figure tracking using learned dynamic models”, *International Conference on Computer Vision*, pp. 94 – 101, 1999.
- [8] M. St. John, M. B. Cowen, H. S. Smallman and H. M. Oonk: “The use of 2-D and 3-D displays for shape understanding vs. relative position tasks”, *Human Factors*, Vol. 43, pp. 79 – 98, 2001.
- [9] E. R. Tufte, “The Visual Display of Quantitative Information,” Graphics Press LLC 2nd ed., 1990.