A Theoretical Model for Probabilistically Based Detection and Mitigation of Malware Using Self Organizing Taxonomies

Gregory Vert
Department of Computer Information Systems
Texas A&M Central Texas
(206) 409-1434
greg.vert@ct.tamus.edu

Anitha Chennamaneni
Texas A&M University Central Texas
(254)519-5463
anitha.chennamaneni@ct.tamus.edu

S.S. Iyengar
FIU School of Computing and Information Sciences
Florida International University
Miami, Florida
Iyengar@cis.fiu.edu

Abstract – Initial identification of attacks on computer systems is crucial to defending against them. The wide range of malware attacks available makes detection and defense a difficult prospect when looking at software vulnerabilities only. However, if the data space is reduced to only look at response from hardware based system state variables, it is possible to detect a wide range of unknown malware, simply by looking at how malware affects state variables describing hardware operation. In this paper, we present a developing theoretical model for the detection of unknown malware by probabilistic time series based modeling. Our model lends itself to a new method for responding to an attack based on fuzzy similarity matrices that lead to dynamic classification of unknown malware using self-organizing taxonomies. Such methods can be made adaptive to reflect general trends in system state variable data over time.

Key-Words: - Taxonomies, Probabilistic, Self Organizing, Computer Security, Complex Systems

I. INTRODUCTION

With the proliferation of malware, increasingly more sophisticated approaches have been needed for its mitigation. It has been estimated that about 10-20 new viruses appear daily [1].

Many information resources are available that will notify users of new security holes on a subscription basis [2,3,4,5]. There are also several security databases that will let users browse the vulnerabilities for various software packages [6,7]. Companies such as Symantec, Security Focus and CERT keep large databases of known attacks [6,7,11,12]. Symantec has over 50,000 entries for known internet security related threats [13]. With the proliferation of new viruses daily, these databases will soon become unwieldy. Yet, with all of the above, the malware problem only seems to be growing in size instead of going away. Part of the reason for this is that security, even with all the information about threat and vulnerability is still a very manual operation in response and mitigation. Few systems currently have the ability to mount an automated mitigation to malware activity partly because of the wide range of software vulnerabilities malware attempts to exploit.

II. PROBLEM FORMULATION

There has been research attempting to classify different types of attacks, from Unix specific vulnerabilities [8,9] to network attack assessment [10]. This research has been important and useful, but their classification has focused on a specific class of attacks. Even more key classification is that classification often is focused on software vulnerabilities. This has lead to a high dimension data space that describes the key attributes of an attack. This is often referred to as the needle in the haystack problem. What has been needed is a method to reduce the high dimension data space of software vulnerabilities to a lower dimension equivalency. Such a reduction can be found by looking at state variables describing the operation of software and malware on the underlying hardware. This is a new approach developed in the model theory presented in subsequent sections.

III. PROBLEM SOLUTION

Attack databases, such as Security Focus' Vulnerability Database [6] and NIST's ICAT [7], list information about reported attacks. Once an attack is known, it often becomes clear about how to mitigate its effects the next the malware operates. The problem is trying to figure out and anticipate the unknown malware that next arrives. With the exploding proliferation of malware anticipation of the unknown is a steep challenge. The key is to find a method to for dynamic classification of unknown malware. Dynamic classification implies self organization in classification schemes. Very little work has been done in this area because of the question of how to create classification instances and sub instances in an automated fashion. One approach that is part of our model, describes a methodology that can potentially be used to classify unknown attacks and subsequently respond and mitigate their threat with the use of self organizing taxonomies.

Hierarchies are a good paradigm for organizing data. Taxonomy is an important tool in organizing data. Taxonomy makes it easier to navigate, access and maintain data. However, the construction of taxonomies manually is time consuming, cumbersome, expensive and inefficient. Besides, in a dynamic setting, where change is a constant, taxonomy needs to adapt constantly. The first part of our model proposed a good taxonomy for characterizing security hardware vulnerabilities that is unique, distinctly different from other taxonomies found in the extant literature which focuses on software vulnerabilities. As stated previously focus on hardware and its state variables is believed to reduce the dimension of the data space describing malware operation against software. Given the expensive and unmountable nature of manual intervention, later work presented in this paper may have to found a way to construct the taxonomy in a completely automated fashion.

3.1 Attack Attributes: The first step in taxonomy development was to come up with a naming notation for hardware based state variables. We have developed a list of attributes in previous work to describe hardware state variables and relate them in a fashion that would support taxonomic trees development. An example of the notation is:

Network.Protocol.TCP.InPorts

For attributes that have multiple values, we separate the values by commas, and define ranges using the "En dash" character (–). For example, a TCP port scan that targets ports 25 (SMTP), 80 (HTTP), and 1024 through 6000 would be defined as:

Network.Protocol.TCP.InPorts = 25, 80, 1024–6000

In previous work our list of state variables included 22 separate hardware state variable that are thought to change upon the execution of malware on the hardware. . For the sake of brevity this list is not included in this paper.

*3.2 Node Actionable Response Rules (NARRs):*Complex systems theory makes a general supposition that complex behaviors can be created through the composition and action of a series of smaller, simpler rules. In this model actionable rules are associated with state variables mentioned in section 3.1, and are simple in form. Our taxonomy is built by clustering state variables into taxonomic nodes with simple actionable rules to mitigate the abnormal operation of a variable. As a taxonomy is processed with current system state information to classify malware operation, these simple rules at each node in the tree are applied to system operation if abnormal system operation has been detected. These are referred to as Node Actionable Response Rules (NARR) and will be integrated into our taxonomic model later in this paper. For example a rule in the Network taxonomy (Figure 1.) might be:

Network.TCP.InPort n= High: NARR = block n

or for the CPU

CPU Usage = High: NARR = Kill process with high use

These action rules are progressively applied to the system as processing drops through layers of the taxonomies. After each application system operation is evaluated to see if it has returned to normal. If not, NARR's are applied as processing and classification continues deeper into the taxonomy. Effectively, this model starts trying to mitigate or counter attack malware operation. More will be presented about this in section 4.

*3.3 Attack Taxonomies:*As stated in section 3.1, the first type of taxonomy developed is based on common attributes of attacks that originated through a remote connection across a network as shown in figure 1.

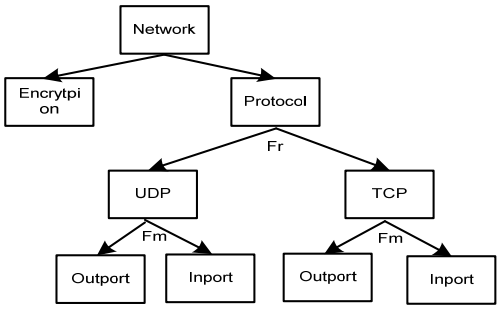


Figure 1: Network Attribute Taxonomy

Figure 1 presents the *network attribute taxonomy*. In our hierarchy, child nodes inherit all the attributes and descriptive properties of their parent nodes, as well as having node specific attributes. The network attributes specified in the tree help to define attacks based on the protocol, bandwidth, and action characteristics of the attack.

In our research we also found an entire category of attacks on files and file systems as mentioned above. The *file system taxonomy* (Figure 2) was developed to structure and organize this data into a taxonomic model. The attributes in this tree define what files on the victim's machine are created, changed, and deleted.

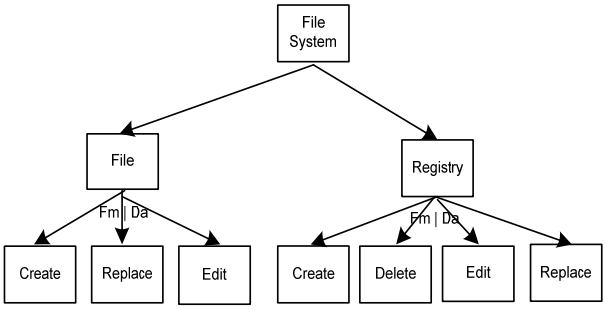


Figure 2: File System Attribute Taxonomy

Another category of attacks are based on system exploits. Figure 3 presents the *exploit attribute taxonomy* which was referenced in section 3.1. The exploit tree defines the vulnerability that an attacker may use on a victim's machine. This taxonomy models common programming errors, improper configurations, and user errors.

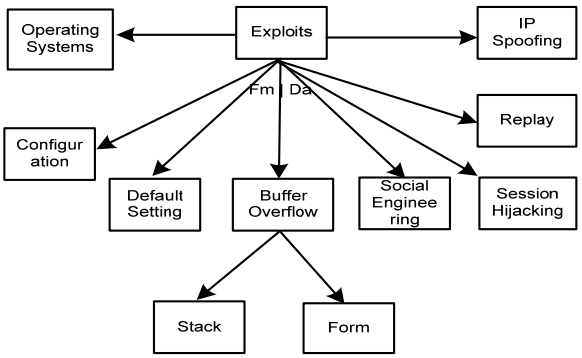


Figure 3: Exploit Attribute Taxonomy

Finally, attacks exist that use services and drivers to gain elevated system privileges [14]. The *kernel taxonomy*, mentioned above, is presented in Figure 4 and shows the types of attacks that are possible using kernel privileges.

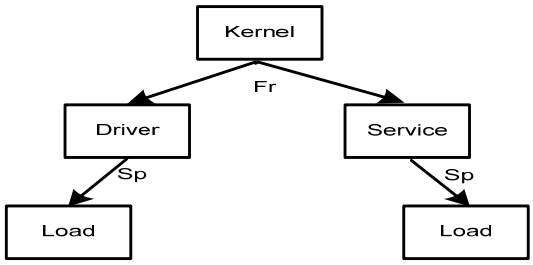
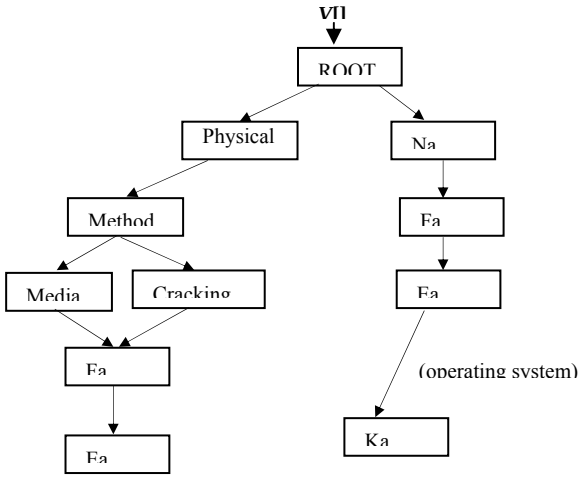


Figure 4: Kernel Attribute Taxonomy

Our initial effort was to consolidate all of the above hard state variable based taxonomies into a single taxonomy produces what might be referred to as a taxonomic graph. This taxonomy is shown in Figure 5, where each box represents the sub taxonomies presented in Figure 1 through Figure 4.



Ka – Kernel attribute tree
Ea – Exploit attribute tree
Na – Network attribute tree
Fa – File attribute tree

Figure 5: Consolidated Taxonomic Graph

In Figure 5 all leaf nodes are connected to the next subtaxonomic tree's root except for the right subtree connection from Ea to Ka. In this case, only the "operating system" node of the Exploit subtree is connected to the root of the Kernel subtree.

Input vector ***V*** is an *n*-dimensional feature vector whose attributes describe the current system data variable data used to populated the sub taxonomies. It might reflect normal operation or it might reflect malware operation. This vector contains the same attributes as those used in the subtrees when selecting and moving to the next child node. At this point in the development of our research several thing were realized:

i) an attack can actually branch to two or more child nodes in a subtree or two or more subtrees in the consolidated taxonomic graph. The reason is that attacks are typically multi-pronged in their approaches.

ii) the structure of the taxonomic graph could be different between attacks and needs to be able somehow to dynamically self organize.

IV APPROACH

As mentioned when examining the above taxonomy in figure 5, it was realized that it has limitations, among them are *that the static nature of the taxonomy may limit its ability to detect unknown types of malware operation on the system*. What was needed was a taxonomic model that had the following properties:

i) the taxonomy algorithm can self organize

ii) the method of malware operation detection could be probabilistic and quantitative

iii) the method should be adaptive

Borrowing on the work developed, it was realized that the approach of looking at hardware and state variables describing hardware operation was a good approach because it limited the data space that needed to be evaluated for malware operation. In other words there can be numerous types of vulnerabilities in software but the fundamental operation of hardware was probably going to have a typical type of signature for *classes* of vulnerabilities thus making detection simpler if it was focused on hardware response to malware. This relationship can be mapped as the following:

$$|h_r| < |s_v|$$

where:
$|h_r|$ is the cardinality of hardware malware state variable response for a given attack
$|s_v|$ is the cardinality of software vulnerabilities response for a given attack

In other words, we felt that it is simpler to monitor for malware activity by watching for state changes in hardware state variable rather than anticipate every possible way software can operate when malware is affecting it. What has been developed based on this concept is based on time series analysis, standard deviation over time for hardware state variables and affinity analysis in the creation of self organizing taxonomies.

*4.1 Probabilistic Malware Detection:*The first component of the theoretical model argues that standard deviations of a state variable (s_v) (same as a hardware state variable) over time is an indication of malware in operation, ergo, malware starting to operate. An example of this would be cpu usage might typically have an average (μ) utilization of 75% and 38.2% of the time falls within .05 standard deviation of $\mu = .75$.

$$38.2\% = \mu \pm .05\sigma$$

For a given hardware state variable (h_s) a time series of standard deviations can be calculated with a time window such that the series can produce σ as it changes over time. Note the times series is expected to produce a normal distribution unless there is a malware attack in progress. This could look like the following

t_1	t_2	t_3	t_4	t_5 …….. t_n
σ				
1.2	1.4	1.1	1.2	1.25……

where:
t_n - is σ at time n

and:

the calculation of σ is done for h_s (e.g. cpu usage) using a system specified time slice that ranges in the form of:

$$t_n \pm t_s$$

where:
t_s - is a specified time slice e.g. 60 seconds

In the above, the if it is 1:10pm, the sampling time slice would run for example from 1:09 – 1:10. During which perhaps every six seconds the cpu usage would be collected and a standard deviation calculated over this population of 10 samples.

Using the above concept of sampling to calculate standard deviation of given state variable over a time window, for all s_v's defined in a system to following would be an examples of many state variables being sampled and the standard deviation for a time slice then being stored::

t_1	t_2	t_3	t_4	t_5 …….. t_n
s_{vi} σ				
1.2	1.4	1.1	1.2	1.25……
s_{vj} σ				
6	4.5	5	5	5.7……
s_{vz} σ				
z_1	z_2	z_3	z_4	z_5……

In the above example the calculated σ stay relatively constant around a similar values for a given s_{vi} with minor predictable jitter.

Using the above time series during malware initiation and activity e.g. cpu usage, the time series model would look very different. If cpu usage had a μ of 50% ± .05σ and malware operation drove the usage to 97% the calculated value over the time window hypothetically will change. As an example:

t_1	t_2	t_3	t_4	t_5 …….. t_n
σ				
1.2	1.4	1.1	5	1.25……

could be what is observed indicating initialization of malware operation at t_4. From the previous table where multiple hardware state variables were being watched we can imagine the following pattern where the malwares original signature initiates in the cpu and them at the next time moment moves into driving disk usage up. Such an example might look like the following:

t_1	t_2	t_3	t_4	t_5 …….. t_n
$s_{v\ spu\ utilization}$ σ				
1.2	1.4	1.1	5	5……
$s_{v\ disk\ usage}$ σ				
6	4.5	5	5	9……

What the above demonstrates is the following:

i) t_4 shows a change in its σ calculation for cpu usage

ii) t_5 shows a continued elevated σ for the $s_{v\ cpu\ usage}$ and there is now elevation in the σ for $s_{v\ disk\ usage}$.

The above is the expected time based probabilistic quantitative signature of malware initiation and operation for only two hardware state variable. This method could be scaled to many state variables creating a sophisticated detection mechanism in a relatively lower dimension data space.

4.2 Taxonomic Self Organization based on σ and Fuzzy Similarity Matrices: In the previous section, the model demonstrates the potential ability to detect malware operation in a probabilistic and time based dynamic fashion. Thus, when the time series σ state variable analysis detects suspicious changes in system system, our model suggests a way malware can be mitigated using the response rules mentioned previously for the state variables and self organizing taxonomies.

The second component of the model argues for self organization of the state variables found in the previously presented taxonomies so that the response rules (NARR) can be applied to counter the attack. This can be done also quantitatively and adaptively by the creation of a similarity matrix. In this model the state variables from the previous static taxonomies, no longer are owned by a sub tree. Instead they are clustered by association into taxonomic nodes forming the various levels of the taxonomic graph. This is self organization and can be accomplished via the use of similarity analysis of state variables and their σ changes flagging malware operation.

A similarity matrix is used often in fuzzy set theory to indicate that some variable has a degree of relation to another variable in a set [16]. This theory organizes data into sets by the degree of relation. Set membership traditionally is denoted by a characteristic function of the form:

$$u(n) = \begin{Bmatrix} 0 \mid & \text{if n} \neg \text{member of set} \\ 1 \mid & \text{if n is full member, crisp} \mid \\ [0..1] \rrbracket & \text{membership function} \end{Bmatrix}$$

In section 4.1, the detection of malware operation was done with the use of state variable changes in the above time series detection of malware operation using state variable σ. This can then be utilized in conjunction with fuzzy similarity based matrices to create self organizing taxonomies.

The first step in this process is to dynamically build (train) a similarity matrix based on the degree that a change in one state variables σ correlates with a change in another state variables σ. Using the fuzzy function above, this results in the following logic:

i) $\Delta s_i \rightarrow \Delta s_j \mid u(s_i,s_j) = 1$

ii) $\Delta s_i \neg \rightarrow \Delta s_j \mid u(s_i,s_j) = 0$

iii) ii) $\Delta s_i \approx \rightarrow \Delta s_j \mid u(s_i,s_j) = [0,,1]$

In more basic terms the above means that if a change in s_{vi} *always* results in a change in s_{vj} then the similarity matrix value is 1, if is *sometimes* results in a change then it is results in a similarity matrix value of:

$$0 < u(s_i,s_j) < 1$$

where:

$u(s_i,s_j)$ = can be calculated based on the percentage of times when s_{vi} changes that there is a corresponding change in s_{vj}

Building on the above foundation, the similarity matrix for $s_{cpu\ usage}$ and $s_{disk\ usage}$ might look like the following:

Table I
A sample similarity matrix for state variables where similarity is calculated base on the percent of times an change in one state variable results in a change in another variable

	s_{v1}	$s_{disk\ usage\ 2}$	$s_{cpu\ usage\ 3}$	s_{v4}	s_{v5}	s_{v6}	s_{vj}
s_{v1}	1	0	0	.1	0	0	0
$s_{cpu\ usage\ 2}$	.86	.9	1	.9	.82	.6	.5
$s_{disk\ usage\ 3}$	.81	1	.2	.7	.3	0	0
s_{v4}	.7	0	.8	1	0	0	0
s_{v5}	…						
s_{v6}	…						
s_{vi}	…						

Table I is a partially populated table of state variable similarity values. In the above table, state variables always have a value of 1 as membership with them selves. The above does illustrate a few curious findings:

i) $\Delta s_{cpu\ usage} \approx \rightarrow \Delta s_{disk\ usage} = .9\ (\%) \therefore s_{disk\ usage} R\ s_{cpu\ usage = strong}$

ii) i) $\Delta s_{disk\ usage} \approx \rightarrow \Delta s_{cpu\ usage} = .2 \therefore s_{disk\ usage} R\ s_{cpu\ usage = weak}$

The above illustrates the fact that relationships (R) in similarity are *not symmetric*. From the above table, an increase in cpu usage usually leads to an increase in disk usage (.9), however, the opposite is not true (.2) that an increase in disk usage would result in an increase in cpu usage.

The values in the table are calculated based on *u(n)* and reflect the % of times a change in one state variable results in a change of another state variable.

4.3 Taxonomic Creation and Application to Offensive Mitigation of Malware Threat: The values in table 1 meet the adaptivity criteria of our model in that it can be calculated and recalculated dynamically as the system operates. The above values then become the basis for the creation of the self organizing taxonomy once the time series σ analysis of state variables suggests the need to create a taxonomy with NARR rules in it.

Reiterating, from the previous discussion about the Node Actionable Response Rules (NARR), each particular state variable can have a very simple rule associated with it, and the state variables around found in the static taxonomic nodes. For instance:

NARR 1: if cpu usage > σ r1 = kill largest process

NARR 2: if disk usage > .5 σ r2 = don't allow deletion of files (e.g. malware could be doing a recursive delete of the file system)

where:
NARRn - node actional rule
r_n – the specific action to apply for the rule

The similarity matrix shows how to construct dynamically a self organizing taxonomy with the NARR rules associated with each state variable. In this method, state variables are not statically assigned to nodes as they were in the previous section. Instead state variables are clustered into nodes at various levels in the tree based on their similarity relations. NARR's are also clustered with their state variables at various nodes in the self organizing taxonomy. The following is the clustering algorithm that builds the taxonomy:

//create the root node of the taxonomy
if s_{vi} change > system set σ for s_{vi}
gather all s_v's from the similarity matrix > .9 from the row s_{vi}
place them in the taxonomic root node with their NARR's and response rules

//create the children nodes, stepping for similarity
//values classes n < level < m, e.g. .9top - .8bottom
// level 1,…
for 0 < similarity values < 1
calculate level boundaries for top and bottom values of the class level

create new child node$_i$
gather all sv$_j$ values in the row sv$_i$ that fall within current level top and bottom ranges

assign all collected NARR for sv$_{j's}$ to node level$_i$

The above algorithm, given the similarity matrix in table 1, might have the following structure upon completion. The initiation of the trees construction starts when the time series probabilistic analysis of a given $S_{cpu\ usage2}$ σ > system threshold σ (triggering tree construction):

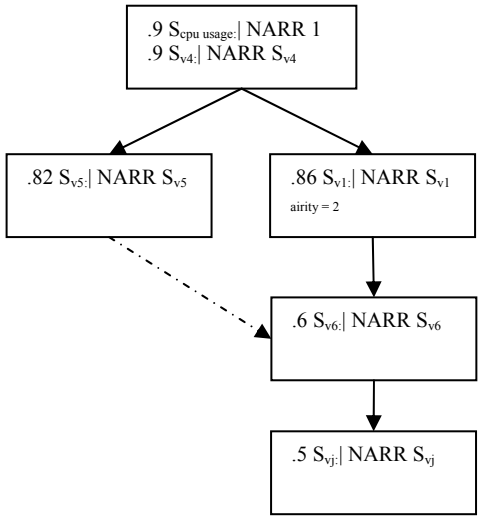


Figure 7. Sample taxonomy constructed from similarity matrix (airity = 2)

The above tree may be augmented with what informally we are calling airity. For the sake of this research, airity is defined to be:

Definition: Airity is the number of child nodes at the same level in a taxonomy where the level is a similarity range and that range is sub divided mathematically into sub ranges and NARR response layers. All child nodes at a given level have a sub class NARR set applied, system measured and then the next airity of a level in the tree is applied.

Of note in figure 7, is the use of the airity concept at level 2 in the tree to create two nodes. Airity divides a levels range and used to partition the response rules such that the rules first applied have the highest similarity value. In this case the level 2 node with the values ranges .85 < similarity < .9, the right most child node would be applied first in the follow sections algorithm .

A matter of further research is how does a node at one level link to the child node of a sibling's at the save level in the tree. This is shown with a dashed arrow in Figure 7.

Additionally multiple state variables may have abnormal σ's from the time series probabilistic analysis method developed earlier. It is a matter of further research about if new stand alone taxonomies should be created for these case or if they are hooked somehow into an existing taxonomy using the magnitude σ's to determine where and how trees are joined together.

4.4 Application of the taxonomy to Threat Mitigation: Once the similarity based, self organized taxonomy has been created, it has in theory has the ability to start mitigation efforts against the malware through progressive application of state variable NARR rules as deeper and deeper nodes in the tree are procesed. The idea is to *only apply enough NARR rules to return the systems operation to normal as measured by σ*. Thus the general method for this is to:

i*) apply a nodes, NARR's*

ii) measure system response

iii) if normal, abort further NARR application, otherwise proceed to next child node and iterate

The pseudo code algorithm for this might be of the form:

// general algorithm for application of the nodes
//NARR rules
for root node to child node$_j$
execute NARR rules for taxonomic node$_j$
measure system response
if system response (σ) normal
exit
else
proceed to next child node on same level or next child node
iterate

Interestingly the above model should create various organizations of state variables in different nodes over time as they respond to different types of unknown malware attacks. This is where it is believed that the model can detect and respond to previously unknown malware. The summing multiple taxonomies for unknown and mitigated software over time has the potential to create elaborate, self adaptive, system specific powerful taxonomies..

V CONCLUSION

The wide range of malware attacks available makes detection and defense a difficult prospect when looking at software vulnerabilities. However, if the data space is reduced to only look at response from hardware based system state variables, the data space can be reduced and offers a possibility to detect a wide range of unknown malware, simply by looking at how it affects state variables describing hardware operation.

Identification of state variable behavior and mal-behavior can be done probabilistically, with time series based monitoring of standard deviations for a given state variable. In conjunction, this information, when questionable state operation data is found σ, coupled with a similarity matrix, the model offers a method for

dynamic creation of malware classification taxonomies. Borrowing from complex systems behavior, nodes in the taxonomy can have very simple rules associated with them that can create complex responses to threat and offer the possibility of a system that has the capacity to mitigate malware operation in an automated and adaptive fashion.

This initial work is being further refined and developed. It is at a theoretical point currently with many further research questions to be investigated. The future includes building a small prototype to determine how well the model actually works an to refine its theory of operation.

References:

[1] Ducklin, Paul. The ABC of Computer Security. Retrieved April 12, 2003, from http://www. sophos.com/virusinfo/whitepapers/abc.html

[2] Symantec Corporation. Security Response. Retrieved March 15, 2003, from http:// securityresponse.symantec.com/

[3] SecurityFocus. What is BugTraq? Retrieved March 15, 2003, from http://www. securityfocus.com/popups/forums/bugtraq/intro.shtml

[4] NTBugTraq. NTBugTrack Home. Retrieved March 16, 2003, from http://ntbugtraq.ntadvice. com/

[5] SANS Institute. Computer Security Education and Information Security Training. Retrieved March 20, 2003, from http://www.sans.org/

[6] SecurityFocus. Vulns Archive. Retrieved March 12, 2003, from http://www.securityfocus.com/ bid

[7] National Institute of Standards and Technology. ICAT Metabase. Retrieved March 13, 2003, from http://icat.nist.gov/icat.cfm

[8] Taimur Aslam. A Taxonomy of Security Faults in the Unix Operating System. Master's Thesis, Purdue University, Department of Computer Sciences, August 1995

[9] M. Bishop. A taxonomy of unix system and network vulnerabilities. Technical Report CSE-9510, Department of Computer Science, University of California at Davis, May 1995.

[10] Shostack, Adam and Scott Blake. Towards a Taxonomy of Network Security Assessment Techniques, July 1999. Retrieved March 29, 2003, from htttp://razor.bindview.com/publish/ papers/taxonomy.html

[11] CERT. CERT® Advisory CA-2003-07 Remote Buffer Overflow in Sendmail. Retrieved April 2, 2003, from http://www.cert.org/advisories/ CA-2003-07.html

[12] Symantec Corporation. Backdoor.FTP_Ana.D. Retrieved April 13, 2003, from http:// securityresponse.symantec.com/avcenter/venc/ data/backdoor.ftp_ana.d.html

[13] Symantec Corporation. Security Response. Retrieved April 21, 2003, from http:// securityresponse.symantec.com/avcenter/search.html

[14] SANS Institute, Knark: Linux Kernel Sub-version. Retrieved April 24, 2003, from http://www.sans.org/resources/idfaq/knark.php

[16] Yen, John, Langari, Reza. *Fuzzy Logic, Intelligence, Control and Information*, Prentice Hall, 1999.