Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)

Use of Trust Vectors for CyberCraft and the Limits of Usable Data History for Trust Vectors

Michael Stevens and Paul D. Williams, PhD

E-mail: {michael.stevens@afit.edu, paul.williams@afit.edu}

Abstract

The CyberCraft initiative is designing a system of agents to provide command and control of future Air Force information systems.[1] *In such a distributed environment, trust between agents becomes pertinent to provide a combatant commander an accurate picture of the network by merging conflicting data and giving the commander confidence in the control of their network. The Trust Vector model breaks from traditional trust models as it provides a range of trust values for data, enabling the system to describe partial trust in data rather than binary values of trust or no trust, and from that build confidence intervals in fused data. The Trust Vector model builds trust based on current and historical data, but does not address the limits of utility of the historical data. This research*[2] *provides implementors of the Trust Vector model input as to how much historical data should be stored to balance accuracy of trust with limits of storage.*[3]

1. Overview

On 7 December 2005, Secretary of the Air Force Michael Wynne and Chief of Staff of the Air Force T. Michael Moseley stated "The mission of the United States Air Force is to deliver sovereign options for the defense of the United States of America and its global interests – to fly and fight in Air, Space, and Cyberspace" [1]. The CyberCraft Initiative is creating vehicles for the Cyberspace domain, as aircraft operate in the domain of the air and spacecraft exploit the domain of space, to enable commanders to defend Air Force information systems in the same manner that aircraft protect our nation's airspace. CyberCraft agents will be deployed on Blue networks to provide commanders management of their network and a fused picture of the health of the network. CyberCraft agents are resident on host computers, and have the ability to load different payloads to accomplish different missions, from loading a policy that changes settings on the host computer to comply with a specific INFOCON level to reporting on network traffic received by the host computer [2].

With faster networks and rapid computer processing, a human's reaction time is too slow to prevent network attacks. The SQL Slammer worm infected 75,000 hosts on the internet, the majority within 10 minutes. The SQL Slammer worm instances on the internet doubled every 8.5 seconds[3], which is far faster than a human can defend against. CyberCraft agents will possess some decision making capabilities to identify and defend against network attacks faster than a human could react. For a commander to incorporate the CyberCraft agents into the decision making process or to give responsibility for the defense of the network to the automated decision making agents, that commander must have some level of trust in the data provided by that agent, and some level of confidence in the ability of the CyberCraft agent to successfully and accurately perform the defense mission.

2. Background

Historically, trust between computers or software agents is defined in Boolean terms; either an agent completely trusts another agent or it does not trust the other agent. Dr Indrajit Ray and Dr Sudip Chakraborty were funded by the Air Force Research Laboratory (AFRL) and the Federal Aviation Administration (FAA) to develop a new model of trust that accounts for differing degrees of trust. If an agent has a track record of 70% good data, a user of that data (automated or human) would like to assign more merit to new data from that agent than the one with a 50% track record, but not completely trust the data. [4].

In order to describe a range of trust between two agents, Dr Ray and Dr Chakraborty developed the Trust Vector model, where trust is defined as a vector with three components, experience (past performance of remote agent), knowledge (known ability of remote agent), and recommen-

1 The authors would like to acknowledge the support of AFOSR and AFRL for this research

2 This work is drawn from the master's thesis of the first author

3 The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government

dations (from other agents regarding their trust in the remote agent). By their admission, these three components are not the only components that can be used to determine trust. The range of trust spans from complete trust (represented as +1) to no trust (0) to complete distrust (-1). Distrust differs from no trust in that distrust indicates a level of confidence that the information is incorrect rather than total uncertainty about the veracity of the information.

This Trust Vector model also incorporates a degradation function to illustrate that trust (or distrust) will lessen over time to approach 0 or no trust. Modelling trust between humans, as time passes the trust one agent has in the other slowly degrades as the remote agent's abilities may have changed, or the data provided becomes stale.

3. Goal, Hypothesis, and Results

AFRL and the FAA funded the Trust Vector research for applications like the CyberCraft initiative, Air Traffic Control systems, and other distributed systems. As the CyberCraft Initiative seeks to move decision making for network defense out of the hands of humans, the humans employing these CyberCraft agents need a measure of confidence that the correct decision will be made. The Trust Vector Model seeks to quantify the trust in the data and in the agents' decision making abilities to give the human warfighters the confidence that the system will operate as expected.

Goal: Our research examines the application of the Trust Vector model to the CyberCraft initiative and identify challenges and future areas of research with the Trust Vector model.

Hypothesis: We believe that there is a threshold for the utility of historical data for the Trust Vector model, after which the cost of storing the data far outweighs the benefit provided by the data

Results Our results support our hypothesis in that as data ages, the benefit provided by the older data is diminished to the point where the contribution to the current trust level is so small that keeping the data is not worth the storage cost. We do not attempt to identify a specific point to discard data, but provide a model which can provide recommendations based on the implementation of the Trust Vector model

4. Trust Vectors

A trust vector is valid in a specific context; one agent may have multiple trust vectors for another agent in different contexts, denoting that it trusts the data from the other agent differently depending on the context of the data. The experience component ($_AE_B^c$ represents the Experience component for the trust relationship between truster A and trustee B in context c) is based on previous data received from the remote agent and the veracity of that data. The knowledge component ($_AK_B^c$) is based on the knowledge one agent has about the abilities of the other agent in that context. The recommendation component ($_\Psi R_B^c$ where Ψ represents a group of recommenders) is the sum of the recommendations of other agents on the trustworthiness of an agent in that context weighted by the trust the receiving agent in the recommendation context of the other agents. Trust vector values degrade over time, to reflect the value of current information [5].

Each component of a trust vector ranges in value from -1 to +1. To produce a single value for trust, a trust policy is applied to the trust vector. The trust policy has the same components as the trust vector, and each component of the trust policy represents the associated weight placed on the components of the trust vector, thus W_e, W_k, W_r represent the weights of the experience component, knowledge component, and recommendation component respectively. $W_e, W_k, W_r \in [0, 1]$ and $W_e + W_k + W_r = 1$. By multiplying the value of each component of the trust vector by the corresponding weight and summing the product

$$W_e \cdot_A E_B^c + W_k \cdot_A K_B^c + W_r \cdot_\Psi R_B^c \in [-1, +1]$$

a single value between -1 and +1 is produced, where -1 indicates total distrust of the remote agent, 0 indicates lack of trust, but not distrust of the remote agent, and +1 indicates complete trust.

4.1. Components

4.1.1. Recommendation Component The recommendations component is updated through querying other agents about their trust with the remote agent. Each other agent's recommendation is tempered by the local agent's trust in the recommending agent in the context of accepting recommendations. If a recommending agent has a record of making poor recommendations, then the local agent will trust that recommending agent's recommendation less than the recommendation from an agent that has a better record for accurate recommendations.

4.1.2. Knowledge Component The knowledge component represents the local agent's knowledge about the remote agent's abilities. An example of this is two remote agents each have a software package to scan a network for vulnerabilities. If one agent's package is known to have a high rate of false positives (or false negatives), then the value of the knowledge component for the context of scanning for vulnerabilities would be less than the agent who has the more accurate scanner. As different scanners have different false positive rates for the different vulnerabilities, a more granular approach would be to have a trust vector for the local agent's trust in the remote agent for detecting each vulnerability, but this could lead to scalability issues with the amount of data being stored for multiple trust vectors.

4.1.3. Experience Component The experience component is based on the past performance of the remote agent in the given context. Each event where the performance of the remote agent is evaluated is given a value of trust positive (+) or trust negative (-). This collection of events is then divided into by time into intervals. The values of the events in each interval are summed to produce a single value for that interval. If the interval from t0 to t1 has four events, three trust positive and one trust negative, the value of that interval would be +2. Each interval is weighted based on the number of intervals and the position of each interval. The weight for each interval is calculated by the formula $w_i = \frac{i}{S}$ where $S = \frac{n(n+1)}{2}$.

Older intervals will be weighted less than more recent intervals. The length of an interval is arbitrary, and the number of events that occur in an interval may not be constant, but the weighted value of each interval will be normalized between -1 and +1 by dividing each weighted value by the number of events in the interval.

4.2. Trust degradation over time

As mentioned in the background, the Trust Vector model has a degradation function to calculate the current value of trust based on a past value. Figure 1 shows how the trust value of a trust relationship approaches 0 or no trust as time increases.

The degradation function described by Ray and Chakraborty is $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})\Delta t)^{2k}}$ where $\mathbf{v}(T_{t_i})$ is the trust value of a trust vector (T_{t_i}) at time t_i and $\mathbf{v}(T_{t_n})$ is the degraded value at time t_n, and $\Delta t = t_n - t_i$ and k is an integer greater or equal to 1. The value for k is arbitrary and determines the rate of decline.

In analyzing Ray and Chakraborty's equation, we saw that higher initial values of trust degrade faster than lower initial values, which is counter intuitive (Figure 2). Inverting the $v(T_{t_i})$ term (trust value at time t_i) from the exponent of e changes the equation to $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$,

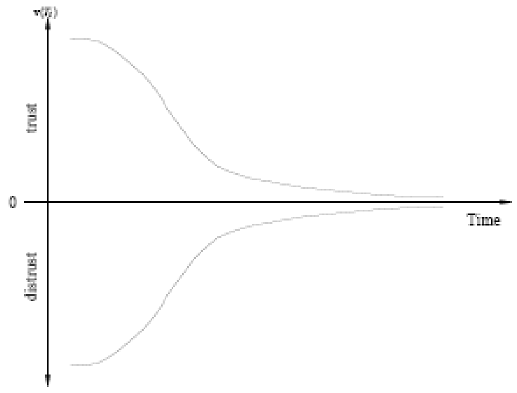

Figure 1. Trust degradation as time increases

which degrades all trust values at an equal rate as shown in Figure 3 (k is arbitrarily set to 2 to match Figure 2).

In section 5.2 we will cover the utility of keeping past trust values.

4.3. Vulnerability Assessment Scenario

To illustrate how the three components and trust work together we will consider a scenario where three agents are scanning the same network for vulnerabilities.

- Agent 1 has a high false positive rate when scanning for a particular vulnerability.
- Agents 2 and 3 are accurate when scanning for the same vulnerability.

All agents have two trust vectors for each of the other two agents; one vector for the remote agent's ability to

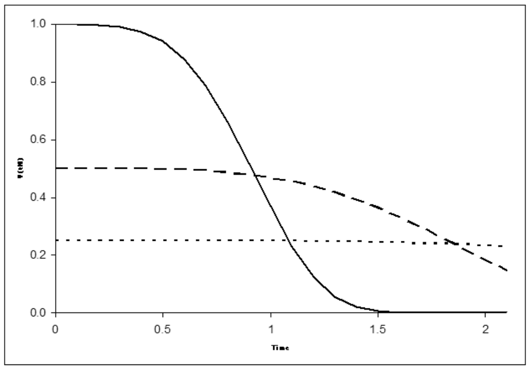

Figure 2. Higher values degrade faster using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})\Delta t)^{2k}}$

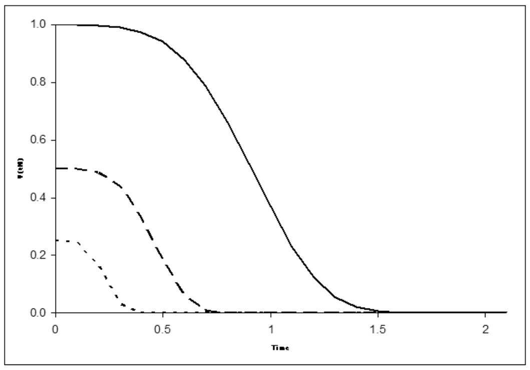

Figure 3. All values degrade at an equal rate using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$

scan for the vulnerability, and another vector for the remote agent's ability to provide accurate recommendations.

Each agent also has two trust vector to themselves; one for their trust in their scanning ability, and the other for their ability to provide accurate recommendations. These vectors enable the agent to self identify itself as possibly producing bad data if the value of the trust vector to itself drops below a certain level. If Agent 1 receives recommendations from the other agents that Agent 1 is reporting systems vulnerable when they are not(due to the high false positive rate), Agent 1's trust vector to itself for its scanning ability will decrease in value. As the value of the trust vector drops, Agent 1 will have less trust in its own data, acknowledging that it may be compromised or have a problem with its scanning software.

While scanning the network agents 1, 2 and 3 scan the same box (Figure 4). Agent 1 reports that the box is susceptible to the vulnerability, and Agents 2 and 3 report the box is not vulnerable.

- Agent 1 would record this event as trust negative (i) for its trust vectors for Agents 2 and 3's scanning abilities.
- Agent 2 to Agent 1 = -, and to Agent 3 = +
- Agent 3 to Agent 1 = -, and to Agent 2 = +

Over time, as more computers are scanned, the history of events is populated and the experience components of the trust vectors for scanning can be calculated. If Agent 3 needed to make a determination as to whether or not the scanned box had the vulnerability, it would modify each result by its trust value for the agent that supplied each result. After modifying each result, Agent 3 would then combine the three results to produce a single value which represents its confidence that the target machine is vulnerable or is not vulnerable.

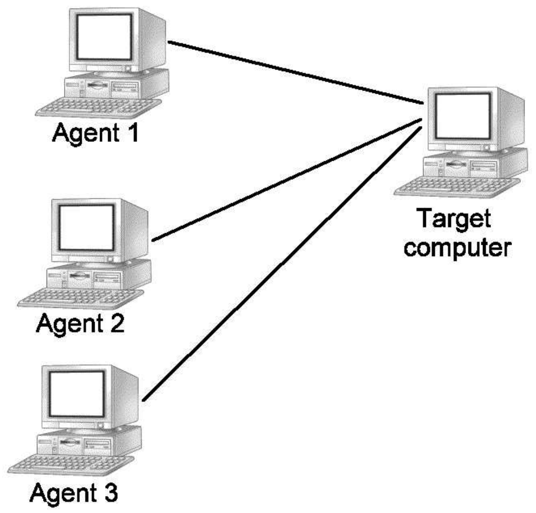

Figure 4. Scenario of Agents 1, 2, and 3 scanning the same computer

If Agent 1 receives data about a target computer that it did not scan, it would be able to make a determination as to the state of that computer based off of the trust it has in the computers reporting the data.

An advantage of this approach of using multiple machines and multiple scanning techniques is that if a compromised computer is configured to hide vulnerabilities from a particular scanner, as other scanners detect the vulnerability the trust vector system will note that there is a problem with the scanner that is being spoofed (in addition to identifying the compromised computer).

4.4. Problem Statement

As storage will be a factor with multiple trust vectors across a large number of agents, how much history should be stored? At what point does the weight of the interval diminish the value of the interval to the point that it is not worth storing the data?

5. Analysis

To determine how much historical data should be stored, we first analyzed how much the historical data contributes to the current trust value. Historical data is used to calculate the value of the experience component (which in turn is used to calculate the trust value of the trust vector), and in the current value of a previous trust vector.

5.1. Utility of historical data for experience component

As stated in section 4.1.3, the experience component is calculated by weighting the values of the intervals of the event history and summing the products of the weights and the values of the intervals. Table 1 shows the weight associated with the oldest interval kept for a given number of intervals.

As shown in Table 1, if 10 intervals of data are stored for the calculation of the experience component, the oldest interval will count for less than 2% of the total value of the experience component.

The following scenario illustrates the the difference between keeping 11 intervals of historical data vs. only keeping 10 intervals (Figure 5). If the events in the oldest interval are all trust-negative, and all events since the oldest interval were trust-positive, the normalized value of interval 1 (the oldest interval) is -1 and the normalized values of the rest of the intervals are +1. The difference between summing the weighted values for the 11 intervals versus the ten most recent intervals is 0.03 (twice the weight of the oldest interval). An alternative scenario is that the oldest interval's value is still -1 and the rest of the intervals' values are 0. The difference in this case is 0.015, equivalent to the weight of the oldest interval. The maximum difference between the keeping 11 intervals and keeping 10 intervals occurs when the eleventh interval's value was one extreme and the rest of the intervals were the other extreme.

From this, the maximum difference between keeping x intervals and keeping x-1 intervals would be twice the weight of the oldest interval when x intervals are kept. From Table 1, the weight at 8 intervals is 2.8%, so to keep 7 intervals versus 8 intervals could lead to a 5.6% difference

Number of Intervals	$S = \frac{n(n+1)}{2}$	Weight of Oldest Interval
1	1	100%
2	3	33.3%
3	6	16.7%
4	10	10.0%
5	15	6.7%
6	21	4.8%
7	28	3.6%
8	26	2.8%
9	45	2.2%
10	55	1.8%
11	66	1.5%
12	78	1.3%

Table 1. Decreasing value of Oldest Interval

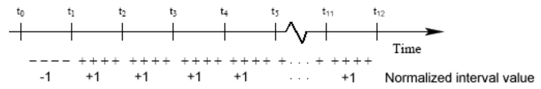

Figure 5. Calculation of experience component

in the value of the experience component. Each application that uses the trust vector system have different needs for the granularity of trust, and Table 1 contains the information to determine the number of intervals needed to meet the level of granularity. The experience component is only one part of the trust vector system, and the impact it has on determining trust is governed by the trust policy. If the trust policy is set where the experience component if 50% of the trust value, then the 5.6% difference in the value of the experience component leads to a 2.8% difference in the trust value.

5.1.1. OS Fingerprinting Scenario To illustrate the effect of keeping different amounts of historical experience data on the trust value, the following deterministic scenario was developed. A group of five agents are fingerprinting a single target machine that runs Windows and can run Linux as a virtual machine. The agents scan the target machine every minute and exchange new recommendations every five minutes. The Trust Policy Vector is Experience = 40%, Knowledge = 25%, and Recommendations = 35%. This Trust Policy Vector was chosen to place more weight on the experience of the agents, followed by slightly less weight on the recommendations. This will cause discrepancies between agents to affect the trust values faster, enabling quicker detection of deviations between agents. Four of the agents will detect the virtual machine and report the OS as being Linux (while Linux is running), and the fifth agent will continue to report the OS as being Windows (this could be due to a different scanning package that the fifth machine is using that is not able to detect the virtual machine, or is able to see past the virtual machine to the host OS). The difference in the reports from the fifth machine and the other four machines will lead to the trust between the four machines that detected Linux and the fifth machine to diminish, while the trust between the four machines that detected Linux will increase. This scenario shows the difference in trust when the target machine switches from Windows to the Linux virtual machine at 9:00 AM and switches back to Windows at 9:30 AM (when all five machines again agree upon the detected OS, leading to increase in trust amongst the agents).

Figure 6 shows the differences in the Trust Value of the agents reporting the Linux virtual machine with regard to

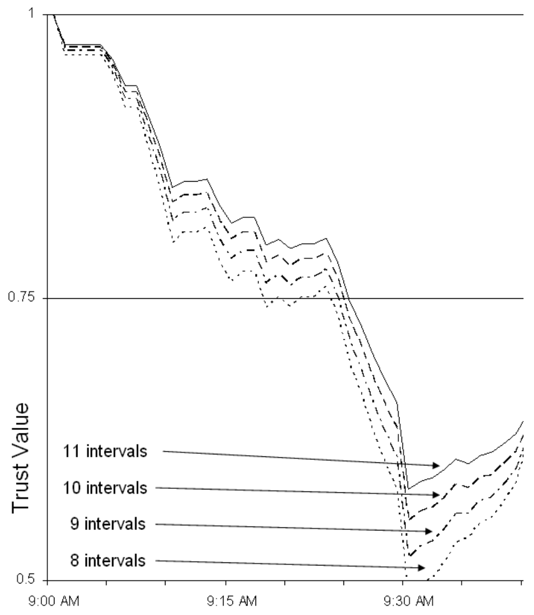

Figure 6. Trust Value for different experience intervals kept

the agent reporting Windows OS[4] [5]. The graph shows that as the number of intervals kept decreases, the Trust Value changes faster and drops farther when discrepancies occur, as there is less historical data to counterbalance current data. The difference between 8 intervals kept and 9 intervals kept is greater than the difference between 10 intervals kept and 11 intervals kept (shown in Table 2). The maximum difference of the Trust Values between runs with different number of intervals kept is consistent with the mathematical analysis in Table 1. The maximum difference between keeping 9 and 10 intervals of data in the scenario was 0.0321, which is slightly more than 1.6% of the range of Trust Values (-1 to 1). The 1.6% difference in trust values is less than the 1.8% of the experience component listed as the value of the oldest interval when 10 intervals are kept (which is to be expected, as the experience component only accounted for 35% of the Trust Value).

In the application of the Trust Vector Model, the decision of how many intervals are to be kept will depend on the trade-off between storage of each interval and the need for granularity of the Trust Values. The storage cost of each interval can be calculated by $E \times B \times C \times A$ where

- $E =$ Events per Interval
- $B =$ Bytes per event
- $C =$ Number of contexts
- $A =$ Number of agents

This scenario uses four events per interval, but this is arbitrary. If the time period of the interval is fixed but the occurrence of a trust event is stochastic, then the number of events per interval could vary depending on the rate of events generated.

The Trust Vector Model proposed by Ray and Chakraborty uses boolean values for each trust event, so every event is either trust-positive or trust-negative, but there are many contexts where a range of values for the trust event would be more applicable. One example of this is the recommendations. The modified Trust Vector Model that is used in this research uses floating point values between -1 and 1 to represent each event for recommendations, as it makes more sense to represent an agent's trust in a recommendation as the absolute difference between that recommendation and the agent's trust value for the same context rather than a boolean value of agreeing with the recommendation and not agreeing with the recommendation. This scenario used one byte for every event.

This scenario only used two contexts, OS Fingerprinting and Recommendations, but it is conceivable that scanning a network for vulnerabilities could have a different context for every vulnerability scanned (or the ability of a network scanner could be abstracted into a single context).

This scenario used five agents, so each agent would keep one Trust Vector per context for all other agents plus a Trust Vector to itself for each context. Thus each interval would cost 40 bytes to store per agent ($4 \times 1 \times 2 \times 5 = 40$).

5.2. Utility of previous trust vectors

Previous trust values are used in conjunction with current trust values to produce a composite trust score. If an agent had produced poor data in the past but is now producing what appears to be good data, the current trust level

Number of Intervals	Maximum difference of Trust Values
8 to 9	0.0357
9 to 10	0.0321
10 to 11	0.0287

Table 2. Decreasing value of Oldest Interval

4 This scenario was based on only the current calculated Trust Values, and not previously calculated Trust Values with the degradation function.

5 This scenario also only concentrates on the discrepancy between reports between agents. A real world application of OS fingerprinting should incorporate some method to account for Virtual Machines or dual boot machines. This will be covered in the Future Work section.

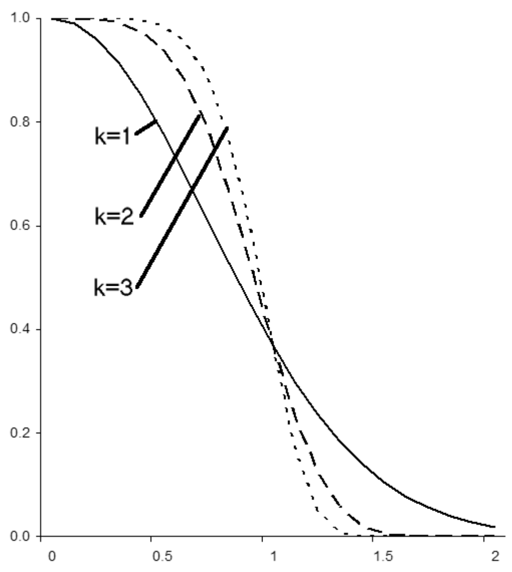

Figure 7. Degradation of Trust Value of 1 at different rates

is moderated with historical knowledge of the previous performance. Another scenario has an agent that has not been heard from for a while, but is now providing data again (e.g. the reporting machine was turned off and has since been rebooted). The degraded trust value is used to estimate the value of the new data, as it would be imprudent to accept the data at the previous trust level as it is unknown what has happened to that agent in the interim.

Figure 7 shows the degradation of the trust values based on different decay rates. In the equation $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})^{-1}\Delta t)^{2k}}$, k is an arbitrary integer value for the decay rate of the trust value. As k increases, the trust value decays slowly at first and then rapidly decays, but we see that for $k > 1$ the value approaches 0 between about $t_{1.25}$ and $t_{1.5}$. Even with $k = 1$ we see that by t_2 the decayed value is approaching 0. With higher values of k the decay approaches zero between t_1 and $t_{1.25}$.

Time periods are arbitrary, as the time difference between t_0 and t_1 could be 10 seconds or 2 months. We suggest that the retention of previous trust values be based on the value of k used and the granularity needed for the trust model, but that previous trust values when $k = 1$ are discarded no later than t_2 and when $k > 1$ trust values are discarded by $t_{1.5}$.

6. Future Work

This research is focusing on the application of the Trust Vector model to the CyberCraft initiative, to include the balancing the number of agents that check each other with the overhead of storage and network traffic that larger trust groups would cause. As the CyberCraft initiative will eventually span hundreds of thousands of computers, scalability of the Trust Vector model will become an issue. Another scalability issue is mentioned in section 4.1.2, where the trade off between the granularity of multiple trust vectors for specific tasks must be balanced with the storage and computational complexity requirements to support several trust vectors. Future research will address the culminating point between the granularity of trust contexts and the storage, computational power, and bandwidth needed to support multiple trust vectors for different contexts between agents

As alluded to in section 5.1.1, another area that this research will address in future work is the use of Trust Vectors with differing views that do not necessarily conflict. An example of this would be OS Fingerprinting as in the scenario of 5.1.1, where one agent reports a target system as being a Mac OS, another agent reports the same system as a Windows OS (but not specifically XP or 2000), and a third machine reports the machine as a Windows XP SP2, how does the model combine these inputs to produce a best guess at the target machine's OS with an associated confidence. Further research would entail how to accommodate data that changes with time, such as a dual booting system or virtual machines.

Ambiguity in the reported data is another area this research is looking into. An example of this would be if an agent was fingerprinting a target system, and the data received from the scan was not conclusive, e.g. the target OS had some Linux characteristics, but not enough to definitely label the target system as Linux. Another example of the above is if the agent suspects that it is compromised, and is being fed inaccurate data. Data sent from these agents to other agents should reflect the ambiguity in the data held by the reporting agent. We are also looking at integrating Markovian Decision Trees into the Trust Vector model to supplement building a trust vector with little data.

Another possible application of Trust Vectors is the use of Trust Vectors with dynamic routing of network traffic. This could include using the rate of dropped packets for the experience component, the available bandwidth of a link (as well as graph theory with network topology) as the knowledge component, and abilities of neighboring routers to reach a distant end for the recommendation component to determine the utility of a link to reach a distant end.

7. Conclusion

As the impact of the oldest interval decreases, even with a 50% weight of the experience component on the trust value, keeping 9 intervals instead of 10 intervals leads to less than a 2% maximum difference, which is below the level of granularity needed for most applications. When CyberCraft is deployed across thousands of computers and each agent holds multiple trust vectors per remote agent, keeping one less byte of data per trust vector may be significant.

The utility of previous Trust Values decreases as time increases, and section 5.2 shows that by t_2 (two time periods since the trust value was last calculated) previous trust values have decayed to the point that they are too small to impact the current value of trust. This conclusion is valid using the modified equation $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})^{-1}\Delta t)^{2k}}$, as Ray and Chakraborty's equation will keep the value of smaller initial trust longer (an initial trust value of .25 will decay to .05 at approximately $t_{4.5}$ at $k = 2$).

In summary, implementors of the Trust Vector algorithm should identify the points where the contribution of historical data is too small to impact the current trust value, and conserve storage by expunging data older than these points. This research provides a model which can assist in the identification of these points.

References

[1] M. Wynne and T. M. Mosely, “Letter to airmen: Mission statement.” http://www.af.mil/library/viewpoints/jvp.asp?id=192, December 2005.

[2] P. W. Phister, Jr., D. Fayette, and E. Krzysaik, “Cybercraft: Concept linking network control warfare principles with the cyber domain in an urban opertional environment.” Air Force Research Labratory, Information Directorate, 2004.

[3] I. Dubrawsky, “Effects of worms on internet routing stability.” http://www.securityfocus.com/infocus/1702, June 2003.

[4] I. Ray and S. Chakraborty, “Cyber trust research: Motivation (in a nutshell).” http://www.cs.colostate.edu/ indrajit/Security/trust/index.htm.

[5] I. Ray and S. Chakraborty, “A vector model of trust for developing trustworthy systems,” in *ESORICS*, 2004.