# Detection of Virtual Environments and Low Interaction Honeypots

S. Mukkamala, K. Yendrapalli, R. Basnet, M. K. Shankarapani, A. H. Sung
Department of Computer Science
Institute for Complex Additive Systems Analysis
New Mexico Tech
(srinivas|krishna|rbasnet|madhu|sung@cs.nmt.edu)

*Abstract* – ***This paper focuses on the detection of virtual environments and low interaction honeypots by using a feature set that is built using traditional system and network level finger printing mechanisms. Earlier work in the area has been mostly based on the system level detection. The results aim at bringing out the limitations in the current honeypot technology.***

***This paper also describes the results concerning the robustness and generalization capabilities of kernel methods in detecting honeypots using system and network finger printing data. We use traditional support vector machines (SVM), biased support vector machine (BSVM) and leave-one-out model selection for support vector machines (looms) for model selection. We also evaluate the impact of kernel type and parameter values on the accuracy of a support vector machine (SVM) performing honeypot classification. Through a variety of comparative experiments, it is found that SVM performs the best for data sent on the same network; BSVM performs the best for data sent from a remote network.***

## 1. INTRODUCTION

One of the purposes of a honeypot is to lure the attacker into interacting with the honeypot and gather information about emerging threats or attack vectors so that the organization's defenses can be updated. New tools can be discovered, attack patterns can be determined, and the very motives of the attackers can be studied [1, 2].

Being able to detect honeypots is important to malicious users as well as security professionals. The stealthy-ness of a honeypot is an important factor to consider in an organization's overall security strategy but more importantly honeypot developers have few tools with which to test their products.

Earlier work on detection of honeypots has focused on detecting them at system level by examining simple features such as system calls or installed software [3]. The work presented in this paper concentrates on network level detection. The fact that low interaction honeypots do not implement a complete feature set (which a real system does) and also that emulated environments have a significant software overhead when multiple virtual machines are running on a single physical machine have been the key features in carrying out the experiments.

A technique called service exercising was implemented based on low interaction honeypots having an incomplete feature set and TCP/IP finger printing techniques in combination with learning machines are implemented to detect a benign and honeypot systems. One of the key features used in TCP/IP finger printing is timing analysis, technique that sends a stream of ICMP echo requests to the target and then measures how quickly the nodes can reply. The results obtained show how the two groups (honeypots and real systems) can be clearly distinguished.

The paper is constructed as follows: the first section is this introduction; section 2 provides an insight to network level detection of honeypots. Section 3 describes methodologies used for detection and data collection. Models generated by Biased Support Vector Machine using leave-one-out model for support vector machines (looms) is given in section 4. A brief introduction to model selection using SVMs for detecting honeypots is given in section 6. In section 6, we analyze classification accuracies of SVMs using ROC curves. Section 7 presents the results and discussion. Summary and Conclusions are given in section 8.

## II. NETWORK LEVEL DETECTION OF HONEYPOTS

Previous efforts to detect honeypots have focused on system level features such as installed software, detecting kernel modules, detecting virtual environments, and performing timing analysis of system functions [3]. While successful, these techniques require access to the local system. Use of these techniques in a networked environment requires a user account or some other way to execute arbitrary code. A faster, more versatile method of network based honeypot detection is needed [4].

**Network based honeypot features:** An ideal honeypot will mirror a real system exactly and is thus difficult to detect but unfortunately existing honeypot technology is far from ideal. In general there are

several high level "features" that honeypots possess but real production systems do not:

- There should be no network activity on the honeypot
- All interactions with the honeypot are logged extensively
- Bandwidth is often restricted to prevent a compromised honeypot from damaging other networks
- Low interaction honeypots do not implement a full feature set
- Emulated environments have multiple virtual machines running on a single physical machine or have significant software overhead when compared to real systems

The first feature is hard to detect without long term monitoring of the honeypot's local network traffic. It is worth noting that the only way to detect a ideal or "pure" honeypot at the network level is to monitor local traffic and even then there is a danger for a high false positive rate.

**Service exercising**: one method to detect a honeypot is to test or "exercise" the services it provides. Some environments (especially low interaction honeypots) do not implement a full feature set and by selecting uncommon features or operations we may be able to determine if we are working with a legitimate system or a part of the network defenses.

**Timing analysis of ICMP ECHO requests**: detection technique builds on a simple observation: most honeypot software responds slower to ICMP ECHO (ping) requests compared to non-emulated systems. This is illustrated in figure 1 below, which shows the response time of several virtual machines and a real Windows 2000 and Redhat Linux system. The first three systems are virtual environments (Microsoft Virtual PC, Honeyd, and VMWare) emulating a single Microsoft Windows 2000 system and the last two systems are a real Microsoft Windows 2000 machine and a RedHat Linux machine for comparison. While VPC responds the slowest there is a clear separation between the real systems and the emulated or virtual systems. By simply creating a "threshold" at a delay of $4.4 \times 10^{-4}$ seconds we can separate the benign systems from the honeypots.

While a precise explanation for this separation is infeasible without intimate knowledge of the inner workings of the virtual machines we can propose a reasonable hypothesis. Because Virtual PC and VMWare both sit on top of a complete guest operating system (in this case Windows 2000) the network data must at the very least go through the link layer before being passed to the virtual system and it is quite probable that it passes trough the full TCP/IP stack on the guest system [4].

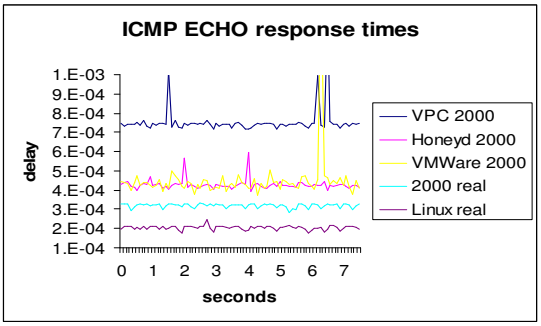


**Figure 1**: ICMP ECHO response times

In effect this doubles the delay from the operating system. Other delays could be introduced when multiple virtual machines are present on a single guest operating system and the guest operating system must route packets between several processes. This is one of the few features that distinguish virtual machines from real systems.

**TCP/IP finger printing:** active finger printing is used to collect the data for analysis. For each of the TCP/IP connection, 49 various quantitative and qualitative features were extracted. The list of the features is given in tabel 1.

**Table 1.** TCP/IP Features Used for Analysis

| S.No | Feature Extracted |
|---|---|
| 1 | sent_packets |
| 2 | received_packets |
| 3 | total_packets |
| 4 | sent_bytes |
| 5 | average_sent_bytes |
| 6 | received_bytes |
| 7 | average_received_bytes |
| 8 | total_bytes |
| 9 | average_bytes |
| 10 | sent_ttl |
| 11 | received_ttl |
| 12 | average_received_ttl |
| 13 | total_ttl |
| 14 | average_ttl |
| 15 | sent_tcp_header_length |
| 16 | average_sent_tcp_header_length |
| 17 | received_tcp_header_length |
| 18 | average_received_tcp_header_length |
| 19 | total_tcp_header_length |
| 20 | average_tcp_header_length |
| 21 | sent_ack_flags |
| 22 | average_sent_ack_flags |
| 23 | received_ack_flags |
| 24 | total_ack_flags |
| 25 | average_ack_flags |
| 26 | sent_push_flags |
| 27 | average_sent_push_flags |

| | |
|---|---|
| 28 | received_push_flags |
| 29 | average_received_push_flags |
| 30 | total_push_flags |
| 31 | average_push_flags |
| 32 | sent_syn_flags |
| 33 | average_sent_syn_flags |
| 34 | received_syn_flags |
| 35 | average_received_syn_flags |
| 36 | total_syn_flags |
| 37 | average_syn_flags |
| 38 | sent_fin_flags |
| 39 | average_sent_fin_flags |
| 40 | received_fin_flags |
| 41 | average_received_fin_flags |
| 42 | total_fin_flags |
| 43 | average_fin_flags |
| 44 | sent_window_size |
| 45 | average_sent_window_size |
| 46 | received_window_size |
| 47 | average_received_window_size |
| 48 | total_window_size |
| 49 | average_window_size |

## III. METHODOLOGY AND DATA COLLECTION

Based on the observations presented in section 2 a technique was developed to classify a node as a honeypot or a benign system. Streams of network requests are sent to the suspect node to collect the data (49 features described in tabel1) and then the delay between the request and the response time is also measured. A monitor attached to a network tap captures the raw packets passing between the scanner and the target and extracts the timing information from the data link layer headers (in most cases the data link layer will be Ethernet) and the features described in table 1. The network layout is presented in Figure 2 below [4].

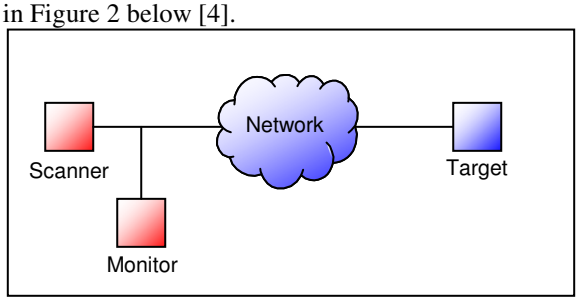


**Figure 2**: Base case network layout

This technique has the potential to be very sensitive to network noise. The further the target gets from the host the more hops the packets must take, each one potentially adding delay. To determine if this technique is still valid in a real world scenario the honeypot will be tested from two distances from the scanner:

- Local: on the local Ethernet, no more than two hops from the target to the scanner
- Wireless: on a wireless router attached to the local network. Several hops between the target and the scanner

The honeypot technology scanned ranges from single virtual machines to three simultaneous virtual machines on a single physical host. The systems being evaluated are:

- Microsoft Virtual PC emulating a single Windows 2000 machine
- Microsoft Virtual PC emulating three Windows 2000 machines
- VMWare emulating a single Windows 2000 machine
- A real (benign) Windows 2000 machine
- A real Redhat Linux machine

### A. Live Network Testing

To make data collection more realistic and to avoid false alarms the honeypots are deployed in a real production network and are scattered amongst different physical LANs. The detection scans are performed both at the local level and through a wireless router to confuse the situation even more. Figure 3 below shows the logical layout of the test case network [4]. Data is collected at multiple time intervals [1 sec, 2 sec, 5 sec, 10 sec, ].

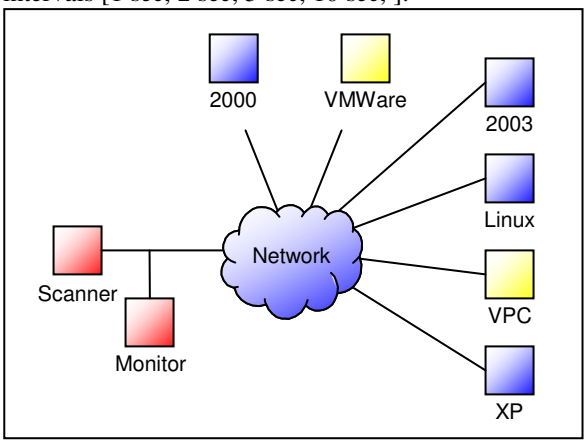


**Figure 3:** Test case network layout

Four virtual machine honeypots are compared to four real production systems. The systems being compared are:

- Microsoft Virtual PC emulating three Windows 2000 machines
- VMWare emulating a single Windows 2000 machine
- A real Windows 2000 machine
- A real Windows 2003 machine
- A real Windows XP machine
- A real Redhat Linux machine

## IV. BIASED SUPPORT VECTOR MACHINES

Biased support vector machine (BSVM), a decomposition method for support vector machines (SVM) for large classification problems [5,6]. BSVM uses a decomposition method to solve a bound-constrained SVM formulation. BSVM Uses a simple working set selection which leads to faster convergences for difficult cases and a bounded SVM formulation and a projected gradient optimization solver which allow BSVM to quickly and stably identify support vectors. Leave-one-out model selection for biased support vector machines (BSVM) is used for automatic model selection [7].

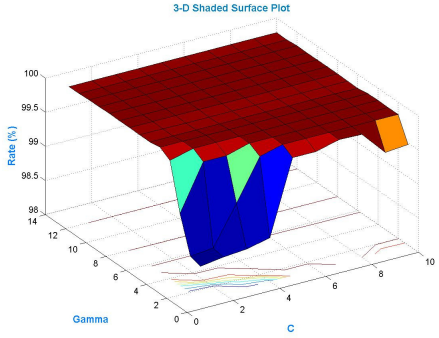


**Figure 4**: BSVM model for Local to Local [100000 sec]

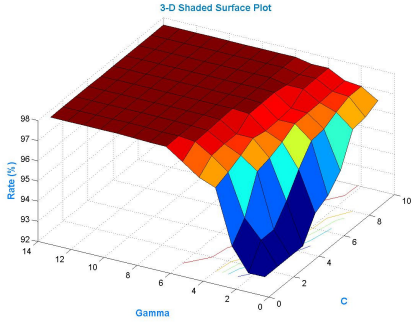


**Figure 5**: BSVM model for Remote to Local [.1 sec]

Models generated for TCP/IP data using leave-one-out model for support vector machines (looms) are given in figures 4 to 5. BSVM model generated for data collected from sending packets to the honeynet from the same network using a time interval of .1 sec is given in figure 4. BSVM model generated for data collected from sending packets to the honeynet from a remote network using a time interval of .1 sec is given in figure 5.

## V. MODEL SELECTION SVMs

In any predictive learning task, such as classification, both a model and a parameter estimation method should be selected in order to achieve a high level of performance of the learning machine. Recent approaches allow a wide class of models of varying complexity to be chosen. Then the task of learning amounts to selecting the sought-after model of optimal complexity and estimating parameters from training data [8,9].

Within the SVMs approach, usually parameters to be chosen are (i) the penalty term $C$ which determines the trade-off between the complexity of the decision function and the number of training examples misclassified; (ii) the mapping function $\Phi$; and (iii) the kernel function such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. In the case of RBF kernel, the width, which implicitly defines the high dimensional feature space, is the other parameter to be selected [10,11].

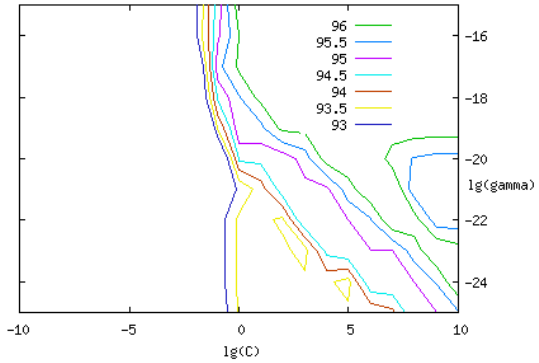


**Figure 6**: SVM model for Local to Local [2 sec]

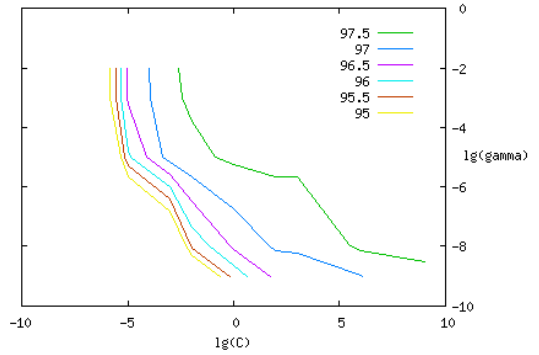


**Figure 7**: SVM model for Remote to Local [100000 sec]

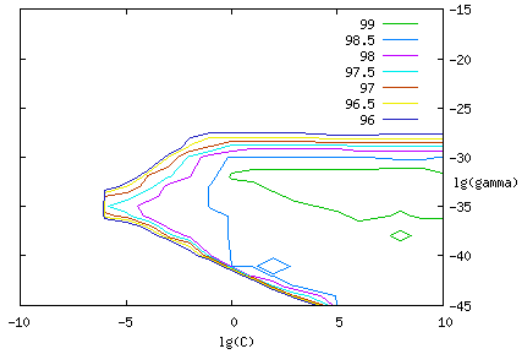


**Figure 8**: SVM model for Remote to Remote [1 sec]

We performed a grid search using 10-fold cross validation for each of the five faults in our data set

[11]. First, we achieved the search of parameters $C$ and $\gamma$ in a coarse scale and then we carried through a fine tuning into the five detection faults proper space. Model selection results obtained through grid search are given in figures 6 to 8 for Local to Local, Remote to Local and Remote to Remote, respectively.

## VI. ROC CURVES

The Receiver Operating Characteristic (ROC) curves are generated by considering the rate at which true positives accumulate versus the rate at which false positives accumulate with each one corresponding, respectively, to the vertical axis and the horizontal axis in Figures 9 to 11.

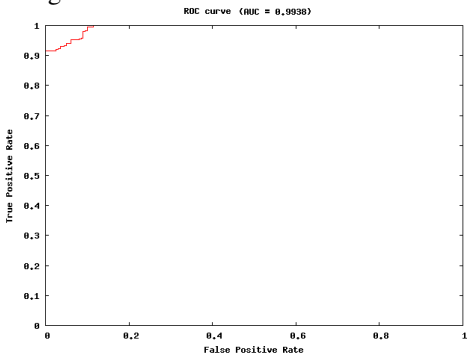


**Figure 9**: SVM accuracy for Local to Local [2 sec]

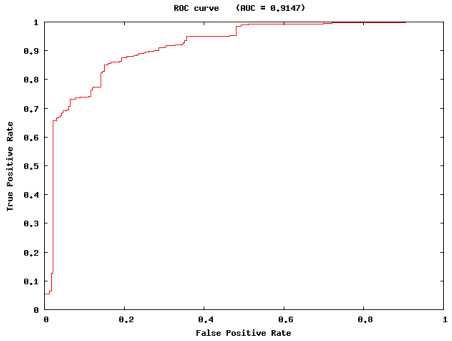


**Figure 10**: SVM accuracy for Remote to Local [5 sec]

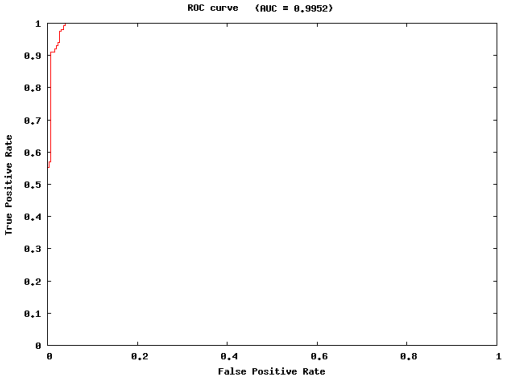


**Figure 11**: SVM accuracy for Remote to Remote [1 sec]

The point (0,1) is the perfect classifier, since it classifies all positive cases and negative cases correctly. Thus an ideal system will initiate by identifying all the positive examples and so the curve will rise to (0,1) immediately, having a zero rate of false positives, and then continue along to (1,1).

Detection rates and false alarms are evaluated for the hooneypot dataset described in section 2 and the obtained results are used to form the ROC curves. Figures 9 to 11 show the ROC curves of the detection models by benign machines and honeypots. In each of these ROC plots, the x-axis is the false alarm rate, calculated as the percentage of benign machines detected as honeypots; the y-axis is the detection rate, calculated as the percentage of honeypots detected. A data point in the upper left corner corresponds to optimal high performance, i.e, high detection rate with low false alarm rate [12].

## VII. RESULTS

The service exercising results are given in table 2 below. A check mark (✓) denotes the feature or command was present and a cross (×) denotes the feature could not be found. The "real systems" column is the combined results for both the Windows 2003 server as well as the Red Hat server. The servers performed identically.

**Table 2**: Service features tested

| Service | Feature / command | Real systems | Honeyd |
|---|---|---|---|
| HTTP | GET | ✓ | ✓ |
| | OPTINOS | ✓ | × |
| | HEAD | ✓ | × |
| | TRACE | ✓ | × |
| FTP | USER | ✓ | ✓ |
| | PASS | ✓ | ✓ |
| | MODE | ✓ | × |
| | RETR | ✓ | × |
| SMTP | HELO | ✓ | ✓ |
| | MAIL | ✓ | ✓ |
| | DATA | ✓ | × |
| | VRFY | ✓ | × |
| | ETRN | ✓ | × |

From these results it is clear that while Honeyd implements the basic functionality of a service it falls short when one tries to actually use the service.

### A. Timing analysis

The honeynet (five honeypots and two benign systems) is scanned from both the local wired Ethernet network and from a wireless remote access point. After the base case had been established the honeypots were distributed around the local campus network and the two scans were repeated. Figure 12 below shows the cumulative average of the response times for the wired base case (scanning the entire subnet from the same Ethernet switch). The cumulative average was chosen because it shows how the node's response times converge. The VMWare based virtual machine experienced some early delays

but even then we can see that it is beginning to converge back to a response time close to its initial value.

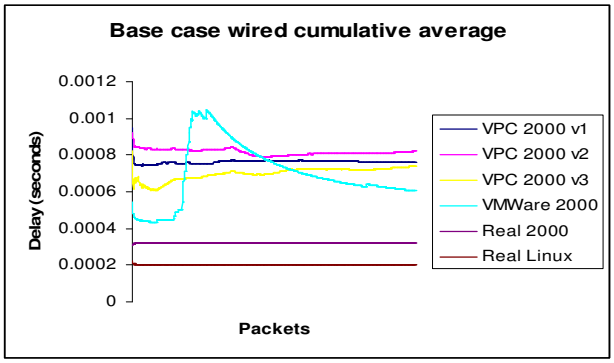


**Figure 12:** Wired cumulative average

There is a clear separation between all of the virtual machines and the real machines. By setting a threshold of $3.7\times10^{-4}$ (all nodes that respond slower than the threshold are honeypots) it is possible to clearly separate the two groups.

B. Finger Printing

In our experiments, we perform 2-class classification using different kernel methods. The (training and testing) data set contains 11982 randomly generated points from the data set representing the 2 classes. The set of 5092 training data and 6890 testing data are divided in to 2 classes: benign systems and honeypots. Same training and test datasets were used for all the experiments. Table 3 summarizes the overall classification accuracy of SVMs, BSVMs and Looms for data sent from the local network to the honeynet. Table 4 summarizes the overall accuracy of SVMs, BSVMs and Looms for data sent from the remote network to the honeynet. Table 5 summarizes the overall accuracy of SVMs, BSVMs and Looms for data sent from the remote network to the remote honeynet.

**Table 3:** Classification accuracies for Local to Local

| Data Collection Time Intervals | SVM | BSVM | Looms |
|---|---|---|---|
| 1 sec | **99.79** | 97.84 | 98.07 |
| 2 sec | **99.38** | 96.93 | 96.93 |
| 5 sec | **97.12** | 94.958 | 94.96 |
| 10 sec | 93.55 | **96.09** | 88.27 |
| 0.1 sec | **99.95** | 99.80 | 99.78 |
| 0.5 sec | 98.55 | 98.51 | **100.00** |

**Table 4:** Classification accuracies for Remote to Local

| Data Collection Time Intervals | SVM | BSVM | Looms |
|---|---|---|---|
| 1 sec | 96.40 | 95.498 | **96.51** |
| 2 sec | 92.70 | **93.27** | 92.68 |
| 5 sec | **91.47** | 88.81 | 85.60 |
| 10 sec | 86.50 | **88.45** | 77.01 |
| 0.1 sec | 97.64 | 97.63 | **97.73** |
| 0.5 sec | **97.55** | 97.44 | 97.44 |

**Table 5:** Classification accuracies for Remote to Remote

| Data Collection Time Intervals | SVM | BSVM | Looms |
|---|---|---|---|
| 1 sec | 99.52 | 99.45 | **99.64** |
| 2 sec | 96.36 | **98.91** | 97.81 |
| 5 sec | 91.85 | **100.00** | **100.00** |
| 10 sec | 95.96 | **100.00** | 94.44 |
| 0.1 sec | 97.12 | **97.90** | 97.12 |
| 0.5 sec | **99.71** | 99.08 | 99.54 |

## VIII. DISCUSSION & CONCLUSIONS

Detecting honeypots by performing timing analysis is heavily dependent on network topology as well as the similarity between the systems being scanned. A slight change in software (for example changing the operating system) has a huge effect on the accuracy of this technique. However, this technique has the advantages of not requiring local system access as well as being able to compare many systems simultaneously. Correctly selecting the threshold to split the two groups is the largest weakness of this technique.

These techniques vary greatly in their aggressiveness but are still fairly stealthy. Service exercising is arguably the stealthiest technique with regards to bandwidth consumed or packets sent. Timing analysis, however, requires a lot of information to be sent and received in order to get an accurate result. When compared to local system level detection these techniques consume very little bandwidth.

Using finger printing data Kernel methods easily achieve high detection accuracy (higher than 95%). SVM performs the best on local network; BSVM performs the best for data sent from a remote network.

Model selection results using Leave-one-out model selection for support vector machines (looms) based on BSVM are presented in (Figures 4 and 5). A grid search for honeypot detection using SVM (Figures 6 to 8) which seeks the optimal values of the constraint penalty for method solution and the kernel width $(C,\gamma)$ has been performed. We demonstrate that the ability with which SVMs can classify honeypots is highly dependent upon both the kernel type and the parameter settings.

ACKNOWLEDGEMENTS

Support for this research received from ICASA (Institute for Complex Additive Systems Analysis, a division of New Mexico Tech), a DOD IASP, and an NSF SFS Capacity Building grants are gratefully acknowledged. We would also like to acknowledge many insightful discussions with Patrick Chavez and Rajeev Veeraghattam that helped clarify our ideas.

REFERENCES

1. Know Your Enemy: Honeynets. The Honeynet Project's Know Your Enemy Series 2005
2. Curran, K., et al., "Monitoring hacker activity with a Honeynet," *International Journal of Network Management,* 2005. **15**(2): p. 123-134.
3. Holz, T., F. Ravnal, "Detecting Honeypots and other syspicious enviroments," *Proceedings of the 2005 IEEE, Workshop on Information Assurance and Security, 2005.*
4. P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, A. H. Sung, "Network Based Detection of Virtual Environments and Low Interaction Honeypots," *Proceedings of the 2006 IEEE SMC, Workshop on Information Assurance.*
5. C. W. Hsu, C. J. Lin, "A comparison on methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, 13, pp. 415-425, 2002.
6. C. H. Chan, I. King, "Using Biased Support Vector Machine to Improve Retrieval Result in Image Retrieval with Self-organizing Map," *Proceedings of 11th International Conference, ICONIP. Lecture Notes in Computer Science 3316 Springer*, ISBN 3-540-23931-6, pp. 714-719, 2004.
7. J. H. Lee, C. J. Lin, "Automatic model selection for support vector machines," *Technical report*, Department of Computer Science and Information Engineering, National Taiwan University, 2000.
8. O. Chapelle, V. Vapnik, "Model selection for support vector machines," *Advances in Neural Information Processing Systems 12*, 1999.
9. V. Cherkassy, "Model complexity control and statistical learning theory," Journal of natural computing 1, pp. 109–133, 2002.
10. N. Cristianini, J. S. Taylor, "Support Vector Machines and Other Kernel-based Learning Algorithms," *Technical Report*, Cambridge University Press, 2000.
11. C. C. Chang, C. J. Lin, "LIBSVM: a library for support vector machines," *Technical Report*, Department of Computer Science and Information Engineering, National Taiwan University, 2001.
12. J. P. Egan, "Signal detection theory and ROC analysis," New York: Academic Press, 1975.