One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses

Katherine A. Heller Krysta M. Svore Angelos D. Keromytis Salvatore J. Stolfo
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10025
{heller,kmsvore,angelos,sal}@cs.columbia.edu

Abstract

We present a new Host-based Intrusion Detection System (IDS) that monitors accesses to the Microsoft Windows Registry using Registry Anomaly Detection (RAD). Our system uses a one class Support Vector Machine (OCSVM) to detect anomalous registry behavior by training on a dataset of normal registry accesses. It then uses this model to detect outliers in new (unclassified) data generated from the same system. Given the success of OCSVMs in other applications, we apply them to the Windows Registry anomaly detection problem. We compare our system to the RAD system using the Probabilistic Anomaly Detection (PAD) algorithm on the same dataset. Surprisingly, we find that PAD outperforms our OCSVM system due to properties of the hierarchical prior incorporated in the PAD algorithm. In the future, these properties may be used to develop an improved kernel and increase the performance of the OCSVM system.

1. Introduction

One of the most popular and most often attacked operating systems is Microsoft Windows. Malicious software is often run on the host machine to inflict attacks on the system. Several methods can be used to combat malicious attacks, such as virus scanners and security patches. However, these methods are not able to combat unknown attacks, so frequent updates of the virus signatures and security patches must be made.

An alternative to these methods is a Host-based Intrusion Detection System (IDS). Host-based IDS systems detect intrusions on a host system by monitoring system accesses. Most IDS systems utilize signature based algorithms that rely on knowing the attacks and their signatures, which limits their ability to detect unknown attack methods. Alternatively, "behavior-blocking" technology aims to detect and stop malicious activities using a set of signature-based descriptions of good behavior, i.e. what is expected of program or system execution. To improve performance, data mining techniques have recently been applied to IDS systems [20, 22] to automatically learn models of "good behavior" and "bad behavior" by observing a system under normal operation. In this paper, we describe a new approach based on anomaly detection, utilizing a method that trains on normal data and looks for anomalous behavior that deviates from the normal model [11, 12, 13]. This method can better identify unknown attacks. Previous work using IDS systems has been done using system call analysis [14, 15, 17, 19, 24] and network intrusion detection [13, 18, 21].

We use the Registry Anomaly Detection (RAD) system to monitor Windows registry queries [9]. During normal computer activity, a certain set of registry keys are typically accessed by Windows programs. Users tend to use certain programs regularly, so registry activity is fairly regular and thus provides a good platform to detect anomalous behavior. We apply an OCSVM algorithm to the RAD system to detect anomalous activity in the windows registry. Although OCSVMs have previously been applied successfully to other anomaly detection problems, they have never before been used to detect anomalous accesses to the Windows registry. The OCSVM builds a model from training on normal data and then classifies test data as either normal or attack based on its geometrical deviation from the normal training data [23]. We present our results of the RAD system using the OCSVM algorithm and demonstrate its abilities to detect anomalous behavior with several different kernels. We also compare our system with work done on the RAD system using the Probabilistic Anomaly Detection (PAD) algorithm [14, 9]. PAD outperforms the OCSVM

system due to the use of the estimator developed by Friedman and Singer [16]. This estimator uses a Dirichlet-based hierarchical prior to smooth the distribution and account for the likelihoods of unobserved elements in sparse data sets by adjusting their probability mass based on the number of values seen during training. An understanding of the differences between these two models and the reasons for differences in detection performance may help to construct a more discriminative kernel, and is critical to the development of effective anomaly detection systems in the future.

2. The Windows Registry and the RAD system

The Windows registry is a database that stores configuration settings for programs, security information, user profiles, and many other system parameters. The registry consists of entries, which are called registry keys, and their associated values. Programs query the registry for information by accessing a specific registry key. Each registry query has five components: the name of the process, the type of query, an associated key, the result, and the success status of the query. The process may be an attack or normal process. Each record in both our test dataset and training dataset contains all five of these entries. A sample record entry appears as:

Process: `EXPLORER.EXE`
Query: `OpenKey`
Key: `HKCR\CLSID\B41DB860-8EE4-11D2-9906-E49FADC173CA\shellex\MayChange DefaultMenu`
Response: `SUCCESS`
ResultValue: `NOTFOUND`

The Registry Anomaly Detection (RAD) system has three parts: an audit sensor, a model generator, and an anomaly detector. Each registry access is either stored as a record in the training set or sent to the detector for analysis by the audit sensor. The model generator develops a model of normal behavior from the training dataset, and the anomaly detector uses this model to classify new registry accesses as normal or anomalous.

The Registry Anomaly Detection (RAD) system utilizes the five raw features given above, such that the algorithm used for anomaly detection classifies each entry as either normal or attack according to these feature values. The process is the name of the process querying the registry. The query is the type of access being sent to the registry. The key is the key currently being accessed. The response is the outcome of the query. The value of the accessed key is the result value. For more detailed information on RAD and the Windows registry, refer to [9].

3. The PAD Algorithm

The Probabilistic Anomaly Detection (PAD) algorithm, developed by Eskin [14, 9], trains a model over normal data features. It is essentially density estimation, where the estimation of a density function $p(x)$ over normal data allows the definition of anomalies as data elements that occur with low probability. The detection of low probability data (or events) are represented as consistency checks over the normal data, where a record is labeled anomalous if it fails any one of these tests.

First and second order consistency checks are applied. First order consistency checks verify that a value is consistent with observed values of that feature in the normal data set. It computes the likelihood of an observation of a given feature, $P(X_i)$, where X_i are the feature variables. Second order consistency checks determine the conditional probability of a feature value given another feature value, denoted by $P(X_i|X_j)$, where X_i and X_j are the feature variables.

One way to compute these probabilities would be to estimate a multinomial that computes the ratio of the counts of a given element to the total counts. However, this results in a biased estimator when there is a sparse data set. Instead, the estimator given by Friedman and Singer is used to determine these probability distributions [16]. Let N be the total number of observations, N_i be the number of observations of symbol i, α be the "pseudo count" that is added to the count of each observed symbol, k^0 be the number of observed symbols, and L be the total number of possible symbols. Then the probability for an observed element i is given by:

$$P(X = i) = \frac{N_i + \alpha}{k^0\alpha + N}C \tag{1}$$

and the probability for an unobserved element i is:

$$P(X = i) = \frac{1}{L - k^0}(1 - C) \tag{2}$$

where C, the scaling factor, accounts for the likelihood of observing a previously observed element versus an unobserved element. In [16], they compute C as:

$$C = (\Sigma_{k=k^0}^{L} \frac{k^0\alpha + N}{k\alpha + N} m_k)(\Sigma_{k \geq k^0} m_k)^{-1} \tag{3}$$

where $m_k = P(S = k)\frac{k!}{k=k^0}\frac{\Gamma(k\alpha)}{\Gamma(k\alpha+N)}$ and $P(S = k)$ is a prior probability associated with the size of the subset of elements in the alphabet that have non-zero probability.

In PAD, however, the above computation of C is too costly, so a heuristic method is used, where C is given by:

$$C = \frac{N}{N + L - k^0} \tag{4}$$

They normalize the consistency check to account for the number of possible outcomes of L by considering if P is the probability estimated from the consistency check, then they report $log(P/(1/L)) = log(P) + log(L)$.

Since there are five feature values for each record in the RAD system, there are 5 first order consistency checks and 20 second order consistency checks. A record is labeled anomalous if any of the 25 consistency checks is below a given threshold. This method labels every record in the dataset as normal or anomalous. To improve the detection rate, pairs of features are examined since a record may have a set of feature values that are inconsistent even though all single feature values are consistent for that record. Most attacks effect a large number of records.

The PAD algorithm takes time $O(v^2R^2)$, where v is the number of unique record values for each record component and R is the number of record components. The space required to run the algorithm is $O(vR^2)$.

4. One Class Support Vector Machine (OCSVM)

Instead of using PAD for model generation and anomaly detection, we apply an algorithm based on the one class SVM algorithm given in [23]. Previously, OCSVMs have not been used in Host-based anomaly detection systems. The OCSVM code was developed by [10] and has been modified to compute kernel entries dynamically due to memory limitations. The OCSVM algorithm maps input data into a high dimensional feature space (via a kernel) and iteratively finds the maximal margin hyperplane which best separates the training data from the origin. The OCSVM may be viewed as a regular two-class SVM where all the training data lies in the first class, and the origin is taken as the only member of the second class. Thus, the hyperplane (or linear decision boundary) corresponds to the classification rule:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \tag{5}$$

where $\mathbf{w}$ is the normal vector and b is a bias term. The OCSVM solves an optimization problem to find the rule f with maximal geometric margin. We can use this classification rule to assign a label to a test example $\mathbf{x}$. If $f(\mathbf{x}) < 0$ we label $\mathbf{x}$ as an anomaly, otherwise it is labeled normal. In practice there is a trade-off between maximizing the distance of the hyperplane from the origin and the number of training data points contained in the region separated from the origin by the hyperplane.

4.1. Kernels

Solving the OCSVM optimization problem is equivalent to solving the dual quadratic programming problem:

$$\min_{\alpha} \frac{1}{2} \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) \tag{6}$$

subject to the constraints

$$0 \leq \alpha_i \leq \frac{1}{\nu l} \tag{7}$$

and

$$\sum_i \alpha_i = 1 \tag{8}$$

where α_i is a lagrange multiplier (or "weight" on example i such that vectors associated with non-zero weights are called "support vectors" and solely determine the optimal hyperplane), ν is a parameter that controls the trade-off between maximizing the distance of the hyperplane from the origin and the number of data points contained by the hyperplane, l is the number of points in the training dataset, and $K(x_i, x_j)$ is the kernel function. By using the kernel function to project input vectors into a feature space, we allow for nonlinear decision boundaries. Given a feature map:

$$\phi : X \rightarrow \mathbb{R}^N \tag{9}$$

where ϕ maps training vectors from input space X to a high-dimensional feature space, we can define the kernel function as:

$$K(x, y) = \langle \phi(x), \phi(y) \rangle \tag{10}$$

Feature vectors need not be computed explicitly, and in fact it greatly improves computational efficiency to directly compute kernel values $K(x, y)$. We used three common kernels in our experiments:

Linear kernel: $K(x, y) = (x \cdot y)$
Polynomial kernel: $K(x, y) = (x \cdot y + 1)^d$, where d is the degree of the polynomial
Gaussian kernel: $K(x, y) = e^{-\|x-y\|^2/(2\sigma^2)}$, where σ^2 is the variance

Our OCSVM algorithm uses sequential minimal optimization to solve the quadratic programming problem, and therefore takes time $O(dL^3)$, where d is the number of dimensions and L is the number of records in the training dataset. Typically, since we are mapping into a high dimensional feature space d exceeds R^2 from the PAD complexity. Also for large training sets L^3 will significantly exceed v^2, thereby causing the OCSVM algorithm to be a much

more computationally expensive algorithm than PAD. An open question remains as to how we can make the OCSVM system in high bandwidth real time environments work well and efficiently. All feature values for every example must be read into memory, so the required space is $O(d(L+T))$, where T is the number of records in the test dataset. Although this is more space efficient than PAD, we compute our kernel values dynamically in order to conserve memory, resulting in the added d term to our time complexity. If we did not do this the memory needed to run this algorithm would would be $O(d(L+T)^2)$ which is far too large to fit in memory on a standard computer for large training sets (which are inherent to the windows anomaly detection problem).

5. Experiments and Results

The one class SVM system we develop detects abnormal accesses to the Windows registry. The training and testing datasets were developed from real usage of the Windows system, and each experiment took one to two weeks to run on a 1.5GHZ Pentium IV dual processor. The training data we used was collected on Windows NT 4.0 and consists of approximately 500,000 attack-free records. These attack-free records are labeled normal and consist of operating system programs and typical Windows programs. The test data consists of approximately 300,000 records of which approximately 2,000 are labeled attacks. Possible attacks include aimrecover, browslist, setuptrojan, and other publicly available attacks [1, 2, 3, 4, 5, 6, 7, 8].

We obtained kernels from binary feature vectors by mapping each record into a feature space such that there is one dimension for every unique entry for each of the five given record values. This means that a particular record has the value 1 in the dimensions which correspond to each of its five specific record entries, and the value 0 for every other dimension in feature space. We then computed linear kernels, second order polynomial kernels, and gaussian kernels using these feature vectors for each record.

We also computed kernels from frequency-based feature vectors such that for any given record, each feature corresponds to the number of occurences of the corresponding record component in the training set. For example, if the second component of a record occurs three times in the training set, the second feature value for that record is three. We then used these frequency-based feature vectors to compute linear and polynomial kernels.

To evaluate the system's accuracy, two statistics have been computed: detection rate and false positive rate. The detection rate is the percentage of attack records that have been correctly identified. The false positive rate is the percentage of normal records that have been mislabeled as anomalous. The threshold is the value that determines if a record is normal or attack. Table 1 includes a sample of the varying thresholds and their effects on the detection rate and false positive rate.

Threshold	False Positive Rate (%)	Detection Rate (%)
-1.08307	0.790142	0.373533
-1.08233	0.828005	0.480256
-1.07139	1.54441	0.533618
-0.968913	1.65734	1.17396
-0.798767	3.58736	3.89541
-0.79858	3.63784	5.60299
-0.798347	3.68999	6.77695
-0.767411	3.72054	6.83031
-0.746663	4.35691	7.47065
-0.746616	4.63025	8.00427
-0.71255	8.34283	20.9712
-0.712503	8.75201	22.0918

Table 1. The effects of varying the threshold on the false positive rate and the detection rate.

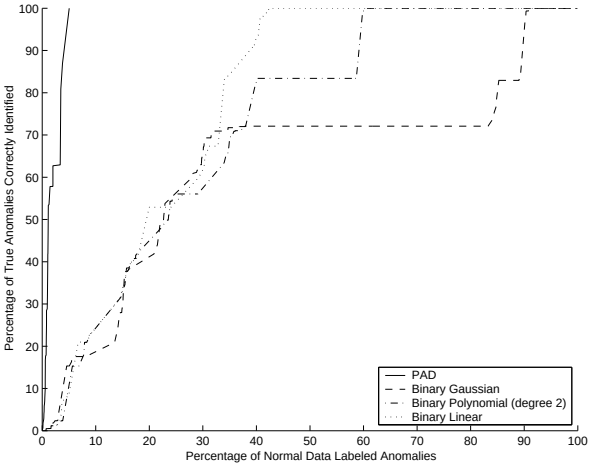

Figure 1. ROC curve for the kernels using binary feature vectors (false positives versus true positives).

We can measure the performance of the one class SVM on our test data by plotting its Receiver Operator Characteristic (ROC) curve. The ROC curve plots the percentage of false positives (normal records labeled as attacks) versus the percentage of true positives. As the discriminant threshold increases, more records are labeled as attacks. Random classification results in 50% of the area lying under the curve, while perfect classification results in 100% of the area lying under the curve. Results from our one class SVM system are shown with the results of the PAD system on the same dataset in Figures 1 and 2. Figure 1 is the ROC curve for the linear and polynomial kernels using binary feature

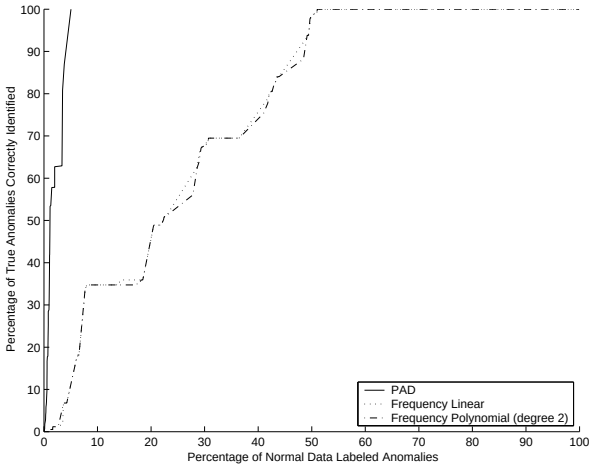

Figure 2. ROC curve for the kernels using frequency-based feature vectors (false positives versus true positives).

vectors. We have used a sigma value of 0.84 for our gaussian function. The binary linear kernel most accurately classifies the records. Figure 2 is the ROC curve for the linear and polynomial kernels using frequency-based feature vectors. The frequency-based linear and frequency-based polynomial kernels demonstrate similar classification abilities. Overall, in our experiments, the linear kernel using binary feature vectors results in the most accurate classification.

In Tables 2 and 3, information on the records and their discriminants are listed for the linear and polynomial kernels using binary feature vectors. From Table 2, it is seen that if the threshold is set at -1.423272, then the bo2kcfg.exe would be labeled as attack, as would msinit.exe and ononce.exe. False labels would be given to WINLOGON.exe, systray.exe and other normal records.

The results of the OCSVM system produce less accurate results than the PAD system demonstrated in [9, 14]. The PAD system is able to more accurately discriminate between normal and anomalous records. The OCSVM system labels records with fair accuracy, but could be improved with a stronger kernel, where more significant information is captured in the data representation.

The ability of the OCSVM to detect anomalies is highly dependent on the information captured in the kernel (the data representation). Our results show that kernels computed from binary feature vectors or frequency-based feature vectors alone do not capture enough information to detect anomalies as well as the PAD algorithm. With other choices of kernels, similar results will occur unless a novel technique which incorporates more discriminative information is used to compute the kernel. A simple example of this is if we have a dataset in which good discrimination depends upon pairs of features, then we will not be able to discriminate well with a linear decision boundary regardless of how we tweak its parameters. However, if we use a polynomial kernel we can account for pairs of features and will discriminate well. In this manner, having a well defined kernel which accounts for highly discriminative information is extremely important. For the purpose of this research, we believe our kernel choices are sufficient to reliably compare the OCSVM system with PAD.

The advantage of the PAD algorithm over the OCSVM system lies in the use of a hierarchical prior to estimate probabilities. A scaling factor (see equation (4)) is computed and applied to a Dirichlet prediction which assumes that all possible elements have been seen, giving varying probability mass to outcomes unseen in the training set. In general, knowing the likelihood of encountering a previously unencountered feature value is extremely important for anomaly detection, and it would be valuable to be able to incorporate this information into a kernel for use with our OCSVM system, perhaps by adding weighted "pseudo-counts" to the features in our frequency-based feature vectors.

6. Conclusions

By monitoring the Windows registry activity on a host system, we were able to use our OCSVM algorithm to label all records in the given experiments as either normal or attack with moderate accuracy and a low false positive rate. We have shown that since registry activity is regular, it can be used as a reliable anomaly detection platform. Note that it would also be informative to study detection rates for specific attack processes as a function of the discriminant threshold.

In the comparitive evaluation of our OCSVM system and the PAD system, we have shown that PAD is more reliable. However, understanding the reasons for this will lead to an improvement of the OCSVM system and will expedite the future development of anomaly detectors. Since there is currently no effective way to learn a "most optimal" kernel for a given dataset, we must rely on our domain knowledge in order to develop a kernel that leads to a highly accurate anomaly detection system. By analyzing algorithms (such as PAD) which currently discriminate well, we can identify information which is important to capture in our data representation and is crucial for the development of a more optimal kernel.

In the future, we plan on testing the system on file system accesses and on the Unix platform. We also plan to create a system to update the model as new data is labeled. This will help counter the effects of concept drift over time. Finding an efficient means of remodeling the data over time within

the OCSVM framework could improve the accuracy of the system.

Finally, since most users accept the default installation location when installing a program, the location of programs tends to be the same on all computers. Thus an attack does not need to query the registry for program location information. By forcing a location declaration other than the default location, a given program will not have the same location on all Windows machines. Attacks will have to query the registry to discover program locations, thus forcing all attacks to be monitored by the anomaly detector. A system such as this would improve the anomaly detection capabilities of the RAD system since no malicious attacks can bypass querying the registry. This would enhance the protection of the system against malicious users.

7. Acknowledgements

We would like to thank Eleazar Eskin, Shlomo Hershkop, Andrew Howard, and Ke Wang for their helpful comments. Katherine Heller was supported by an NSF graduate research fellowship. Krysta Svore was supported by an NPSC graduate fellowship.

References

[1] Aim recovery. URL: `http://www.dark-e.com/des/software/aim/index.shtml`.

[2] Back orifice. URL: `http://www.cultdeadcow.com/tools/bo.html`.

[3] Backdoor.xtcp. URL: `http://www.ntsecurity.new/Panda/Index.cfm?FuseAction=Virus\&VirusID=659`.

[4] Browselist. URL: `http://e4gle.org/files/nttools/`, `http://binaries.faq.net.pl/security\_tools`.

[5] Happy99. URL: `http://www.symantex.com/qvcenter/venc/data/happy99.worm.html`.

[6] Ipcrack. URL: `http://www.geocities.com/SiliconValley/Garage/3755/toolicq.htmlhttp://home.swipenet.se/~w-65048/hacks.htm`.

[7] L0pht crack. URL: `http://www.astack.com/research/lc`.

[8] Setup trojan. URL: `http://www.nwinternet.com/~pchelp/bo/setuptrojan.txt`.

[9] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo. Detecting malicious software by monitoring anomalous windows registry accesses. *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, 2002.

[10] A. Arnold. Svm anomaly detection c code. *IDS Lab, Columbia University*, 2002.

[11] V. Bartnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.

[12] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, NY, 1970.

[13] D. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222–232, February 1987.

[14] E. Eskin. Anomaly detection over noisy data using learned probability distributions. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, 2000.

[15] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 120–128, 1996.

[16] N. Friedman and Y. Singer. Efficient bayesian parameter estimation in large discrete domains. *Advances in Neural Information Processing Systems*, 11, 1999.

[17] S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[18] H. Javitz and A. Valdes. The nides statistical component: Description and justification. *Technical Report, SRI International, Computer Science Laboratory*, 1993.

[19] W. Lee, S. Stolfo, and P. Chan. Learning patterns from unix processes execution traces for intrusion detection. *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56, 1997.

[20] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[21] W. Lee, S. Stolfo, and K. Mok. Data mining in work flow environments: Experiences in intrusion detection. *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99)*, 1999.

[22] M. Mahoney and P. Chan. Detecting novel attacks by identifying anomalous network packet headers. *Technical Report CS-2001-2*, 2001.

[23] B. Scholkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1472, 2001.

[24] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

Program Name	Label	Number of Records	Min. Record Value	Max. Record Value
REGMON.EXE	NORMAL	259	-0.794953	-0.280406
SPOOLSS.EXE	NORMAL	72	-1.152717	-0.021361
CloseKey	NORMAL	429	-1.082720	-0.374784
OpenKey	NORMAL	502	-0.959895	-0.365539
QueryValue	NORMAL	594	-1.082909	-0.374972
EnumerateValue	NORMAL	28	-0.570206	-0.284935
DeleteValueKey	NORMAL	3	-1.078758	-0.370822
AimRecover.exe	NORMAL	61	-1.082720	-0.374784
aim.exe	NORMAL	1702	-1.064796	-0.356860
ttssh.exe	NORMAL	12	-0.969706	-0.375161
ttermpro.exe	NORMAL	1639	-1.083098	-0.285123
NTVDM.EXE	NORMAL	271	-0.798204	-0.410065
notepad.exe	NORMAL	2673	-1.083098	-0.285123
CMD.EXE	NORMAL	116	-1.139322	-0.375161
TASKMGR.EXE	NORMAL	99	-0.570017	-0.284935
_INS0432._MP	NORMAL	443	-1.423272	-1.423272
WINLOGON.EXE	NORMAL	399	-1.423272	-1.423272
systray.exe	NORMAL	17	-1.423272	-1.423272
em_exec.exe	NORMAL	29	-1.423272	-1.423272
OSA9.EXE	NORMAL	705	-1.083098	-0.375161
fi ndfast.exe	NORMAL	176	-1.083098	-0.375161
WINWORD.EXE	NORMAL	1541	-1.083098	-0.375161
winmine.exe	NORMAL	21	-0.429351	-0.429351
POWERPNT.EXE	NORMAL	617	-1.083098	-0.285123
PING.EXE	NORMAL	50	-1.083098	-0.375161
QueryKey	NORMAL	11	-0.712317	-0.375161
wscript.exe	NORMAL	527	-1.083098	-0.375161
AcroRd32.exe	NORMAL	1598	-1.083098	-0.375161
0”	NORMAL	404	-1.083098	-0.375161
WINZIP32.EXE	NORMAL	3043	-1.083098	-0.375161
explore.exe	NORMAL	108	-1.083098	-0.375161
EXCEL.EXE	NORMAL	1782	-1.083098	-0.375161
bo2kss.exe[2]	ATTACK	12	-0.712317	-0.375161
bo2k_1_0_intl.e[2]	ATTACK	78	-1.083098	-0.375161
browselist.exe[4]	ATTACK	32	-0.798770	-0.411763
bo2kcfg.exe[2]	ATTACK	289	-1.423272	-1.423272
bo2k.exe[2]	ATTACK	883	-1.423272	-1.091776
mstinit.exe[2]	ATTACK	11	-1.423272	-1.423272
runonce.exe[2]	ATTACK	8	-1.423272	-1.423272
Patch.exe[2]	ATTACK	174	-1.083098	-0.375161
install.exe[3]	ATTACK	18	-1.083098	-0.375161
xtcp.exe[3]	ATTACK	240	-1.083098	-0.285123
l0phtcrack.exe[7]	ATTACK	100	-0.798581	-0.285123
LOADWC.EXE[2]	ATTACK	1	-1.423272	-1.423272
happy99.exe[5]	ATTACK	29	-0.570017	-0.411575

Table 2. Information about test records for the linear kernel in the binary setting. The maximum and minimum discriminants are given for each process, as well as the assigned classification label. Listed next to the attack processes is the attack source. [1] AIMCrack. [2] BackOrifice. [3] Backdoor.xtcp. [4] Browse List. [5] Happy 99. [6] IPCrack. [7] L0pht Crack. [8] Setup Trojan.

Program Name	Label	Number of Records	Min. Record Value	Max. Record Value
REGMON.EXE	NORMAL	259	-4.062785	-1.524777
SPOOLSS.EXE	NORMAL	72	-5.422540	-0.272565
CloseKey	NORMAL	429	-5.210662	-1.788163
OpenKey	NORMAL	502	-4.828603	-1.758730
QueryValue	NORMAL	594	-5.211228	-1.789106
EnumerateValue	NORMAL	28	-3.311164	-1.542890
DeleteValueKey	NORMAL	3	-5.1955757	-1.766465
AimRecover.exe	NORMAL	61	-5.210285	-1.792879
aim.exe	NORMAL	1702	-5.148589	-1.703827
ttssh.exe	NORMAL	12	-4.860299	-1.794766
ttermpro.exe	NORMAL	1639	-5.211794	-1.543456
NTVDM.EXE	NORMAL	271	-4.234352	-1.794766
notepad.exe	NORMAL	2673	-5.211794	-1.543456
CMD.EXE	NORMAL	116	-5.388013	-1.794766
TASKMGR.EXE	NORMAL	99	-3.309843	-1.543456
_INS0432._MP	NORMAL	443	-6.239865	-6.239865
WINLOGON.EXE	NORMAL	399	-6.239865	-6.239865
systray.exe	NORMAL	17	-6.239865	-6.239865
em_exec.exe	NORMAL	29	-6.239865	-6.239865
OSA9.EXE	NORMAL	705	-5.211794	-1.789672
fi ndfast.exe	NORMAL	176	-5.211794	-1.794766
WINWORD.EXE	NORMAL	1541	-5.211794	-1.789672
winmine.exe	NORMAL	21	-1.794766	-1.794766
POWERPNT.EXE	NORMAL	617	-5.211794	-1.543456
PING.EXE	NORMAL	50	-5.211794	-1.789672
QueryKey	NORMAL	11	-4.022096	-1.789672
wscript.exe	NORMAL	527	-5.211794	-1.789672
AcroRd32.exe	NORMAL	1598	-5.211794	-1.794766
0”	NORMAL	404	-5.211794	-1.789672
WINZIP32.EXE	NORMAL	3043	-5.211794	-1.789672
explore.exe	NORMAL	108	-5.211794	-1.789672
EXCEL.EXE	NORMAL	1782	-5.211794	-1.789672
bo2kss.exe[2]	ATTACK	12	-4.022096	-1.789672
bo2k_1_0_intl.e[2]	ATTACK	78	-5.211794	-1.789672
browselist.exe[4]	ATTACK	32	-4.087124	-1.789672
bo2kcfg.exe[2]	ATTACK	289	-6.239865	-6.239865
bo2k.exe[2]	ATTACK	883	-6.239865	-5.245378
mstinit.exe[2]	ATTACK	11	-6.239865	-6.239865
runonce.exe[2]	ATTACK	8	-6.239865	-6.239865
Patch.exe[2]	ATTACK	174	-5.211794	-1.789672
install.exe[3]	ATTACK	18	-5.211794	-1.794766
xtcp.exe[3]	ATTACK	240	-5.211794	-1.543456
l0phtcrack.exe[7]	ATTACK	100	-4.194165	-1.543456
LOADWC.EXE[2]	ATTACK	1	-6.239865	-6.239865
happy99.exe[5]	ATTACK	29	-3.309843	-1.794766

Table 3. Information about test records for the second order polynomial kernel in the binary setting. The maximum and minimum discriminants are given, as well as the assigned classification label. Listed next to the attack processes is the attack source. [1] AIMCrack. [2] BackOrifice. [3] Backdoor.xtcp. [4] Browse List. [5] Happy 99. [6] IPCrack. [7] L0pht Crack. [8] Setup Trojan.