PCPP: On Remote Host Assessment via Naïve Bayesian Classification

Thomas H. Morris and V.S.S. Nair
High Assurance Computing and Networking Lab (HACNet)
Southern Methodist University
Dallas, TX, USA
{tmorris, nair}@engr.smu.edu

Abstract

Private Computing on a Public Platform (PCPP) is a new paradigm in public computing in which an application executes on a previously unknown remote system securely and privately. The first step in the PCPP process is remote assessment of a prospective remote host to determine whether it is capable of executing the PCPP application and to classify the host as a potential threat or non-threat. This paper explores the use of a Naive Bayesian classifier to classify prospective remote hosts. We show that the Naïve Bayesian classifier learns to recognize subtle patterns in historical host measurements and performs the classification task accurately and with minimal negative performance implications.

1. Introduction

Recently, we introduced Private Computing on a Public Platform (PCPP) [8] [11] as a method for allowing clients to securely and privately execute applications on foreign hosts previously unknown to the client. PCPP is a two step process. First, PCPP performs a remote assessment of the prospective remote host to determine if the remote host is a threat or non-threat. After an acceptable remote host is found PCPP launches the application on the remote host and enforces a set of active security properties which assure that the PCPP application's control and data flow remain unaltered, unmonitored, and unrecorded.

In [11], we analogize PCPP to the security provided to a traveling dignitary. When a dignitary travels abroad his or her security is a two step process. First, an advance team is sent to scout the remote location. This advance team will work with local authorities to understand the security risks and safeguards that are already in place. If the advance team finds the risk at the remote location to be acceptable the trip will be approved. However, security does not stop there; rather, the visiting dignitary will also bring his or her own security detail when traveling. This security team will provide an additional security layer, in excess of local security safeguards, around the foreign dignitary while he or she is in the country.

This paper focuses on the PCPP remote assessment, which is akin to the work done by traveling dignitary's advance team. The PCPP remote assessment measures characteristics of the remote host via external scans and an on-host PCPP server daemon. These measurements are used to assess whether the PCPP client may proceed with execution on a given host. First, a checklist of minimum requirements is used to perform a relatively quick go-no-go check of the remote host. After the minimum requirements are met the PCPP client applies a Bayesian classifier on a larger set of measurements from the remote host. The Bayesian classifier, which has been trained from historical PCPP data, classifies the remote host as a threat or non-threat. If the Bayesian classifier classifies the host as a non-threat then execution on the remote host may commence.

In this paper we first provide a brief overview of the entire PCPP process. Next, we discuss the PCPP minimum requirements checklist. We then introduce the Naïve Bayesian classifier including a discussion of the data used to assess the classifier and the results we obtained from our classification experiments. Finally, we offer a section on future works and a conclusion.

2. Related Work

Grid computing allows pooling of compute resources across geographic areas and across organizations. Globus [1] offers a popular toolkit which can be used to create a working grid. Grids generally use some form of access control list to limit use of the grid to approved users. Once on a grid the user has access to an account with a set of privileges defined for his user type. User identity is commonly attained from PKI certificates for the users. Grid offers the ability to encrypt the communication channel between the user's computer and the grid host. Grid does not make guarantees about user privacy while executing on the remote host. Grid is intended to formally link networks of computers across university campuses or businesses.

Mobile agents are small code snippets which traverse networks performing tasks for their owner. These agents often run in sandboxes which allow them to be platform independent and contain the agent to reserved portions of the host machine to offer the host security against the agent. Many approaches have been proposed to ensure that the mobile agents control flow and data flow have not been altered [5] [6] [7] [8], or reversed engineered to invade the agents privacy [4]. We find no proposals for an assessment of the prospective host prior to agent execution outside of requiring a voucher or certificate from a third party [14].

There are many commercial and freeware tools available to scan computers for security policy adherence and vulnerabilities. Virus scanners and spy ware scanners are found on large percentage computers and scan for two common types of vulnerabilities. I.T. departments use more sophisticated tools to scan for policy adherence and vulnerabilities. Such tools include the CIS Next Generation Scoring Tool [3] and NMAP [9]. With the exception of the virus and spy ware scanners these scanners are generally not used by the average computer user.

3. PCPP Overview

We have previously introduced PCPP [8] [11] as a two step process which allows applications to run securely and privately on foreign remote hosts. The two steps to the PCPP process are: first the initial remote assessment of a prospective host; and second the enforcement of a set of active security properties on the remote host while the PCPP application runs.

The PCPP remote assessment is also broken into a two step process. Measurements gathered from a remote scan of the prospective host and from a PCPP daemon running on the remote host are used to grade the host against a set of minimum requirements. If the minimum requirements are met, a Naïve Bayesian classifier is used to classify the remote host as a threat or non-threat.

First, the PCPP client scans the remote host externally. Second, a daemon running on the remote host gathers information directly from the host operating system. Depending upon the quantity and type of data gathered about the remote host, the length of these surveys can be extensive. However, it is expected that prospective hosts wishing to participate in the PCPP process will have previously installed and executed the PCPP remote host daemon, as well as all the PCPP host software, needed to provide the necessary measurements to the PCPP client. We anticipate that a PCPP host will re-execute the internal scan periodically, and when queried by a PCPP client, will provide the results of the latest scan.

Once the remote host assessment is completed a PCPP application may be launched on the remote host. In addition to the PCPP application, several other PCPP processes run on the remote host which work together to enforce a set of active security properties. The overall goals of these active security properties are to ensure that the PCPP application's control and data flow remain unaltered, unmonitored, and unrecorded.

Table 1: PCPP Control Flow Properties

	Property
1	Order of instruction execution must not be altered
2	The PCPP application execution must begin at a valid pre-determined start point
3	The PCPP application execution must end at a valid stopping point
4	The application code execution order must follow a legal path through all branches
5	Instruction execution shall not be monitored or recorded
6	Application code shall not be accessed by any process other than the PCPP application process itself
7	Detect control flow violations
8	Emergency shutdown/ clean-up upon control flow violation

The PCPP control flow properties listed in Table 1 protect the application control flow, while the PCPP data protection properties from Table 2 protect the PCPP application data.

Table 2: PCPP Data Protection Properties

	Property
1	Application data (stack, heap, files) shall remain unaltered
2	Application data (stack, heap, files) shall not be accessed by any process other than the PCPP application process itself
3	Application data (stack, heap, files) encrypted when not in use
4	Access to application data (stack, heap, files) restricted when not in use
5	Encryption of all input and output channels to and from a PCPP application
6	Encryption key protection
7	Detect data protection property violations
8	Emergency shutdown/ clean-up upon data protection property violation

PCPP processes running on both the PCPP client and remote host.

The PCPP client searches for prospective remote hosts. When a remote host is found, an assessment engine on the PCPP client scans the remote host and queries the remote host for the previously gathered on host remote assessment measurements. The assessment engine decides whether the host is an acceptable risk by first checking a set of minimum requirements and then by applying a Naïve Bayesian classifier to classify the host as either a threat or non-threat. The data used to train the Bayesian classification process is historical data from prior PCPP jobs which is collected and stored on the PCPP client.

When a remote host decides to become a prospective PCPP host, a set of PCPP applications are started on the hosts. These applications run as background tasks and include a virtual machine which provides a consistent environment for PCPP applications to run. An active security monitor application is also run which verifies the properties listed in tables 1 and 2 are met.

The active security monitor detects control flow violations using ECCA (Enhanced Control-flow Checking Using Assertions) [15] which requires modification of the application code at compile time and generates exceptions when the control-flow diverts from the legal path. The active security monitor prevents and detects data protection violations through a set of modifications of the operating system context switch and file access routines which apply an encryption process to all PCPP application data.

More details of the active security monitor and the implementation specifics required to enforce the properties in Tables 1 and 2 will appear in a subsequent paper which is under preparation.

A TLS session is required to encrypt and authenticate traffic between the job dispatcher and the virtual machine.

4. Go/No-Go Requirements

The first step in the PCPP remote host assessment is verification of a set of go/ no-go properties pertaining to the host. These properties are a minimum set of system capabilities and security requirements.

Table 3: Remote Host Go/No-Go Requirements

Requirement Number	Description
1	Virus detection present
2	Virus signatures up to date
3	Last virus scan <= 14 days
4	Intrusion detection present
5	Software firewall present
6	Spyware detection present
7	Spyware signatures up to date
8	Last spyware scan <= 14 days
9	Root kit detection present
10	Root kit signatures up to date
11	Last root kit scan <= 14 day
12	Open critical ports
13	OS Version
14	OS security patches up to date
15	External port scan data must match data gathered via internal host scan
16	Prior host classification as non-threat

Table 3 lists the minimum set of requirements for the no/go-scan. For the items related to virus detection, intrusion detection, software firewall, spyware detection, root kit detection, and OS versions, an approved list of products will exist. The remote assessment scanners will support only these approved versions. If a remote host is using software unapproved by the PCPP client for one or more of the go/no-go requirements the requirement will not be satisfied.

The remote host will be scanned externally to generate a list of open ports, software versions for services running on the open ports, and OS version information. The data gathered externally must match data provided to the on-host PCPP scanner. Additionally, PCPP clients keep a database of information collected about previous PCPP jobs. If the prospective host has been used for a previous job it must have been classified as a non-threat.

5. Remote Host Classification

The second phase of the PCPP remote host assessment employs a Naïve Bayesian classifier [12] to predict whether the remote host is a threat or non-threat based upon historical PCPP experience. We measure the value of a set of 32 attributes primarily related to the security configuration of the remote host for each PCPP job run. We postulate that new PCPP hosts will violate the PCPP active security properties from Tables 1 and 2 when the new host has many attribute measurements in common with historical hosts which were detected for violation of the active security properties. Therefore, we want new hosts, with a number of attributes in common with previously known malicious hosts, to be classified as threats. Furthermore, we postulate that new PCPP hosts will *not* violate the PCPP active security properties from Tables 1 and 2 when the new host has many attribute measurements in common with historical hosts which were *not* detected violating the active security properties. We want new hosts with much in common with previously known *non*-malicious hosts to be classified as non-threats.

We use a Naïve Bayesian classifier and a training database containing the historical attribute measurements and the posterior classification (result of active security monitors) from all prior PCPP jobs to predict if a prospective host is a threat or non-threat. If a host is predicted to be a non-threat the pending PCPP job may be launched on the host. If the host is classified as a threat the pending job will not be launched on the host.

5.1. Naïve Bayesian Classification

The Naïve Bayesian classifier classifies new hosts by applying equation (2) to compute the probability that tuple t_i is a member of class C_j.. Tuple t_i is the 32 attribute tuple collected from the prospective host. We compute probabilities for each class C_j in C (1). The new host is assigned to the class with the higher probability from equation (2).

The three terms of equation 2 each provide separate levers in predicting the class of the new tuple.

First, the probability of tuple t_i given class C_j, defined in (3), is a product of the rates of class membership for all of the attributes measurements found in the tuple. When many attribute measurements have high rates of membership in class C_j, the probability that tuple t_i is a member of class C_j increases. Conversely, when few attribute measurements have high rates of membership in class C_j, the probability that tuple t_i is a member of class C_j decreases.

Second, the probability of class C_j, defined in (5), is actually the portion of the historical data which belongs to each class. When the historical data has a high rate of a given class, the probability of the new tuple being a member of that class increases and when a given class is rare, the probability of the new tuple being a member of that class decreases.

Finally, the denominator of equation (2) is the probability of the tuple itself. We note that the denominator is independent of class. It is the same for both classes and does not impact the comparison of *P(threat* | t_i*)* and *P(non-threat* | t_i*)*. As such, it can be left out of the calculations and the numerators alone can be computed and compared to provide our classification.

The math used in our implementation of the Naïve Bayesian classifier is discussed below. Equations 1-5 are derived from discussion in [12].

$$C \in (threat, non-threat) \tag{1}$$

$$P(C_j|t_i) = \frac{P(t_i|C_j)P(C_j)}{P(t_i)} \tag{2}$$

When classifying a tuple we compute the probabilities (2) of each class C_j in C (1) given the tuple t_i. We classify tuple t_i into the class with the higher probability.

$$P(t_i|C_j) = \prod_{k=1}^{32} P(t_{ik}|C_j) \tag{3}$$

The probability of tuple t_i given class C_j (3) is the product of individual conditional probabilities (4) for each attribute in the tuple. The individual probabilities from equation (4) are computed by counting the occurrences of the attribute values for the given class C_j. In practice, during our training step we create a table of probabilities from equation 4 for each attribute value for each class. We then use this probability table in the classification step to look-up the probabilities needed for equation (2).

$$P(t_{ik}|C_j) = \frac{count(t_{ik}|C_j)}{count(C_j)} \tag{4}$$

The probability of threat or non-threat (5) is simply a count of all incidences of the given class in the training database divided by the total number of tuples in the training database. We compute these probability values during training as they do not change when classifying a new tuple.

$$P(C_j) = \frac{count(C_j)}{count(threat) + count(non-threat)} \tag{5}$$

One of the primary advantages of the Naïve Bayesian classifier is its simplicity. There is no complex difference equation required, there is no need for the measurements to take a numeric form and the math needed to compute the probabilities necessary for Bayesian classification are simple. For each attribute we simply define sub-classes and then for computing the intermediate probabilities the majority of the work is spent counting occurrences of each attribute for each class.

Naïve Bayesian uses historical data gathered from previous PCPP jobs to measure the similarity of the prospective host to the hosts in the training set. Each attribute in the classification is treated independently, yet, Naïve Bayesian measures the probability of class member ship as product of probabilities from each attribute and therefore capable of noticing patterns of similarity in the data which humans and other classifiers such as decision trees cannot observe.

For the exercises in this paper we chose to use just two classes, threat and non-threat. It is possible to use the same Naïve Bayesian classification with more than two classes. For instance, it may be desirable to break the threat class into multiple levels of risk which could allow certain PCPP jobs with lower value to be run on low risk hosts which must be avoided by higher valued PCPP jobs.

5.2. Data collected from Remote Hosts

When a new PCPP job is launched the PCPP client will gather a new remote assessment tuple from the prospective host. After a PCPP job completes the host's remote assessment tuple and the result of the job are added to the classification training data. The result of the job, whether the active security monitor detected violations or not, is recorded as threat or non-threat, respectively.

In order to minimize PCPP job startup latency, we expect that the host will have precompiled the data required for Bayesian classification. This data will come from periodic scans of the host which are stored as tuples for use in the PCPP host assessment. Because the scan is run periodically it may be scheduled as a low priority task.

It is possible for the host to provide false answers to PCPP assessment inquiries. In this event a prospective host may fool the PCPP assessment into providing a non-threat classification when a threat was the correct classification. In such cases presumably the host would attack the PCPP job and this would be detected by the PCPP active security monitors. This would trigger an immediate shutdown of the PCPP job in progress, a classification of this host as a threat in the PCPP client training database, and seed the Bayesian classifier to classify similar hosts as threats in the future.

Table 4: Attributes collected for Naïve Bayesian Classification

Item	Bayesian Ranges
Virus protection present	TRUE, FALSE
Virus protection brand	A,B,C,D,OTHER
Virus Signatures up to date	TRUE, FALSE
Virus days since last complete scan	GOOD, OKAY, BAD, TERRIBLE
Intrusion detection present	TRUE, FALSE
Intrusion detection brand	A,B,C,D,OTHER
Software firewall present	TRUE, FALSE
Software firewall brand	A,B,C,D,OTHER
Spy ware detection present	TRUE, FALSE
Spy ware signatures up to date	TRUE, FALSE
Spy ware days since last complete scan	GOOD, OKAY, BAD, TERRIBLE
Spy ware scanner brand	A,B,C,D,OTHER
Root kit scan present	TRUE, FALSE
Root kit scan brand	A,B,C,D,OTHER
Root kit signatures up to date	TRUE, FALSE
Root kit days since last scan	GOOD, OKAY, BAD, TERRIBLE
Open critical ports	TRUE, FALSE
Port server software up to date	TRUE, FALSE
OS patches up to date	TRUE, FALSE
OS automatic updates	TRUE, FALSE
OS version	XP_HOME, XP_PRO, WIN2K, OTHER
File system	NTFS,OTHER
IP region	USA, EUROPE, EASTEUROPE, ASIA, OTHER
Known bad MAC address	TRUE, FALSE
Password complexity	TRUE, FALSE
Expired passwords	TRUE, FALSE
Auto login disabled	TRUE, FALSE
Guest accounts disabled	TRUE, FALSE
Anonymous login disabled	TRUE, FALSE
Number of administrator accounts	TRUE, FALSE
Active X security level	TRUE, FALSE
Java Security level	TRUE, FALSE
MS Office macros	TRUE, FALSE

Table 4 shows all of the attributes and their legal values which we used in the Bayesian classification. The set of attributes collected was derived from a survey of information gathered by various vulnerability scanners available as freeware and commercially [3][9][10].

The cardinality of allowed values for most attributes was kept small intentionally. This limits the possibility of missing data in the training set and limits the number of counting steps required during training. Limiting allowed values also limits the granularity of the classifier. However, if it is found that more specificity is needed the possible value set can be expanded or single attributes can be replaced by multiple, more specific attributes. An example of limiting the legal values is found for the case of attributes which measure the number of days since the last (virus, root kit, spy ware) scan. We group these values in to the categories of GOOD, OKAY, BAD, and TERRIBLE. If we had left this attribute to be an integer from 0 to *N*, we would be required to compute *N* conditional probabilities for that attribute in the training step. This would be extra work for little extra classification value.

The actual attributes listed in Table 4 are less important than the concept of using a Naïve Bayesian classifier in this context. It is relatively easy to scan a host for many different types of attributes. If better indicator attributes are found, they can easily be used in place of or in addition to those listed in Table 4.

Data from prior PCPP jobs may be shared amongst PCPP clients. This would allow the training data to become sufficiently large for accurate classification sooner. Sharing remote assessment tuples from prior PCPP jobs will also disseminate information about new threats faster and potentially limit the impact from these threats. The remote assessment tuples from prior PCPP jobs may be shared via a central server or peer to peer exchange. Tuples should be shared only if their validity can be guaranteed. Without this guarantee shared data could be used to maliciously manipulate the results of the classifier.

5.3. Synthetic Data used to test the classifier

In order to judge the effectiveness of the Naïve Bayesian classifier we created synthetic constrained random data sets to both train our classifier and test our classifier. In all we created seven constrained random data sets. These data sets are listed in table 5.

Table 5: Naïve Bayesian Classification with Synthetic Data

Data Set Name	Description
1k	1K constrained random tuples with patterned threats and 1% random threats
10k	10K constrained random tuples with patterned threats and 1% random threats
100k	100K constrained random tuples with patterned threats and 1% random threats
1k_nr	1K constrained random tuples with patterned threats
10k_nr	10K constrained random tuples with patterned threats
100k_nr	100K constrained random tuples with patterned threats
Random	100K constrained random tuples with 50% random threats

In the preceding table, we use the term constrained random to mean, first, that the attribute values for each tuple were generally chosen at random while only being allowed to take legal values for each attribute as listed in Table 4. Second, some attributes where constrained to take a certain value for a certain percentage of the time and or situationally. For instance, the *virus protection present* attribute on the system was constrained to be TRUE 95% of the time. Further, if the *virus protection present* attribute was FALSE the *virus brand* attribute was forced to OTHER, the *virus signatures up to date* attribute was forced to FALSE, and the *virus days since last complete scan* attribute was forced to TERRIBLE.

There are three basic types of data listed in Table 5 which differ by how the classification as threat or non-threat was generated. Except for the data called *Random,* the data set names contain 1K, 10K, or 100K which indicate the number of tuples in the data set. The *Random* data set contains 100K tuples.

The six data sets, *1k, 10k, 100k, 1k_nr, 10k_nr,* and *100k_nr* all contain a set of hidden patterns. The hidden patterns are designed to show that the Naïve Bayesian classifier will generally detect these patterns and flag prospective hosts as threats. These patterns were generated in the data as additional constraints. First, anytime the *bad MAC address* attribute was TRUE, the tuple was classified as a THREAT. The *bad MAC address* attribute was allowed to be TRUE 0.5% of the time. Second, when the *Virus protection brand* attribute was B, the *virus days since last complete scan* attribute was BAD, and the *OS patches up to date* attribute was FALSE the class was forced to THREAT. Finally, when *root kit protection present* attribute was FALSE and the open critical ports attribute was TRUE the class was forced to THREAT.

In addition to the embedded patterns the three sets called *1k, 10k,* and *100k* contain an additional 1% of tuples which were classified as threat regardless of the value of the attributes in the tuple. The three sets called *1k_nr, 10k_nr,* and *100k_nr* do not contain patterns randomly classified as threat.

The *Random* data set contains no embedded patterns, rather 50% of the tuples are randomly classified as threats regardless of the value of the attributes and 50% are randomly classified as non-threats regardless of the value of the attributes.

All of the data mentioned above was created using Cadence Specman Elite [13], a commercial software used to automate test benches used for digital integrated circuit verification. Cadence Specman Elite executes code written in the 'e' language [14] which contains a random constraint solver which we used here to create our synthetic data. The constraint solver has three features we used for this project. First, we used the 'e' language to specify legal values for each of the attributes in our synthetic data tuple. Second, we specified the distribution of attribute values for certain attributes in our tuple. Third, we specified relationships between the attributes which sometimes override the random behavior (our embedded patterns). We then used the constraint solver to randomly choose tuples which met our specifications.

5.4. Our Implementation

We implemented our classifier in Perl. Our implementation is broken into two parts. First, we train our classifier with all of the tuples gathered from previous PCPP jobs, or in our case, randomly created synthetic tuples, and the resulting classification of those jobs. Second, our implementation classifies individual tuples from prospective hosts for new PCPP jobs.

In the training step we build a table of conditional probabilities $P(t_{ik}|C_j)$ (4) and compute the probabilities of each class $P(C_j)$ (5). These tables are built with a single pass through the data. The probability tables are stored in hashes to speed up search in the classification step. Computing the probability tables ahead of time allows significant speed up in computation of the final classification of a new prospective host.

For classification we simply look-up the $P(t_{ik}|C_j)$ (4) probabilities for each attribute measurement in our tuple to generate $P(t_i \mid C_j)$ (3). We then look up the class probability $P(C_j)$ (5). Finally, we compute the probability of class C_j given the tuple t_i, for each class in C (1), in our case threat and non-threat. We classify the tuple as a member of the class with the higher probability.

Occasionally in classification we encounter tuples with data values for attributes which require special treatment. We found two such cases.

First, we may find an attribute value in classification which was never sampled in the training data. In this case the conditional probability, $P(t_{ik} \mid C_j)$ (4) is zero for both the threat and non-threat classes. If these intermediate zero values are kept the resulting probability of class membership $P(C_j \mid t_i)$ (2) for both classes will be zero. In this case we ignore this attribute when computing the probability of class membership $P(C_j \mid t_i)$ (2).

The second case which requires special treatment occurs when within the same tuple one or more attribute values always imply a threat classification while simultaneously one or more attribute values always imply a non-threat classification. Here, we mean that the attribute value was always associated with one class and never occurs with the other class. When we have separate attribute values that which always imply classification in opposite directions, the resulting posterior probabilities again become zero. We handle this case by dropping the attribute which implies non-threat and keeping the attribute which implies threat. This sways the final classification towards threat which for our application is the more conservative choice.

5.5. Classification Results

We ran our Naïve Bayesian implementation with the synthetic data a total of twelve times. For each run we trained the classifier with one data set and then classified all the tuples from a second data set.

The classification of each tuple in the test data was checked against the actual class of the tuple. We report the percentage of classifications which are correct, percentage of classifications which are incorrect, percentage of false negatives, and percentage of false positives in Table 6.

All of the first nine runs are trained with data which contains embedded patterns as we discussed earlier. We notice that when there is a pattern in the data, the Naïve Bayesian classification does a very good job of finding similar tuples in the test phase and classifying these correctly.

When the data set name contains the post fix *'_nr'* this means the data does not contain any random classification. In other words the class in these data sets always depends on the three patterns embedded in the data. When the data set name does not contain the *'_nr'* post fix the training data contains approximately one percent of the tuples randomly assigned to the threat class regardless of the presence of the aforementioned patterns. This can be seen in the results. For runs in which the test data contained this 1% random assignment to the threat class, the percentage of incorrectly classified tuples also approaches 1%.

In the final three runs the tuples in the training data were classified completely randomly. Since we have 2 classes, threat and non-threat, the split between classes was approximately 50% threat and 50% non-threat. The results from these runs show that the classifier found no trends in the data and consequently incorrectly classified the data approximately 50% of the time. This illustrates that when the data has no trends, the Naïve Bayesian algorithm can not accurately classify tuples.

We expect in real world application, there will generally be trends in the data similar to the ones we placed in the synthetic data. We know that certain attribute values lead to vulnerabilities which can be exploited.

Generally, a system which works to quickly remove these vulnerabilities will be a more secure system. Conversely, a system which does not fix vulnerabilities in a timely manner will be less secure.

TABLE 6: NAÏVE BAYESIAN CLASSIFICATION WITH SYNTHETIC DATA

Training Data	Test Data	% Correct	% Incorrect	%False Negative (Fatally Wrong)	% False Positive (Missed Opportunity)
10k	*1k*	99.2	0.8	0.8	0.0
100k	*10k*	98.9	1.1	0.9	0.2
10k_nr	*1k_nr*	99.3	0.7	0.2	0.5
100k_nr	*10k_nr*	99.6	0.4	0.1	0.3
10k_nr	*1k*	99.2	0.8	0.8	0.0
100k_nr	*10k*	98.9	1.1	0.9	0.2
1k	*10k*	98.9	1.1	0.9	0.2
1k_nr	*10k_nr*	99.9	0.1	0.1	0.0
10k	*1k_nr*	99.3	0.7	0.2	0.5
random 100k	*10k_nr*	49.9	50.1	1.1	49.0
random 100k	*1k_nr*	54.9	45.1	0.8	44.3
random 100k	*random 100k*	51.0	49.0	23.5	25.5

When these observations hold true, the Naïve Bayesian will do a good job classifying hosts. There will always be cases of new vulnerabilities which look like the random threats we introduced in our synthetic data. Generally, new vulnerabilities will tend to be classified as non-threats at first, then as more PCPP jobs are encountered where the vulnerability is exploited the trend will become apparent to the classifier and these tuples will be classified as threats.

A Dell branded Linux workstation was used to perform all of the computations for this paper. The workstation contains 2 Intel Pentium 4 2.8 GHz processors with 512 KB cache each. The workstation contains 4GB shared memory. The Linux kernel version is 2.4.21-37.ELsmp.

We measured the time spent in training and the time spent classifying the test data for each run. The average training time per tuple was 148 µS. The average time spent in classification per tuple was 422 µS. The time spent in training per tuple was about 1/3 of the time spent in classification per tuple. However, generally we will train with thousands of tuples, while we always classify only one tuple at a time. This means that the majority of time will actually be spent in training. This is acceptable because the training can be done on the PCPP client as a background task periodically as new tuples are collected. The classification step must always be done before a new PCPP job is launched on a remote host. As such, the classification step adds delay to the PCPP job completion time and therefore should be as fast as possible.

6. Future Work

We find that Naïve Bayesian classification fits our PCPP remote host classification problem well. However, we would like to follow up this research with more details in some areas. First, we would like to examine the use of clustering algorithms and visualization techniques to discover and show patterns in the collected PCPP assessment data. Second, we would like to explore the use of more classes perhaps having classes with varying degrees of risk for PCPP jobs with varying value. Third, we need to explore answering the question how much training data is required before we can trust the classifications our classifier.

7. Conclusion

In this paper we offer an approach for assessing PCPP remote hosts to determine if the host is acceptable candidate for running PCPP applications. Our solution breaks the problem into two parts. First, we scan the remote host to ensure that a set of go/no-go requirements are met. Second, we used historical data collected from previous PCPP jobs to train a Naïve Bayesian classifier. We then demonstrated the use of this classifier to classify prospective hosts as threats or non-threats. We found this approach to meet the needs of the PCPP remote assessment step.

References

[1] www.globus.org

[2] Wang, C. and Wulf, W. A, "A Framework for Security Measurement." Proc. National Information Systems Security Conference, Baltimore, MD, pp. 522-533, Oct. 1997. http://citeseer.ist.psu.edu/wang97framework.html

[3] The Center for Internet Security, http://www.cisecurity.org/

[4] Wang, C., Hill, J., Knight, J., and Davidson, J. 2000 Software Tamper Resistance: Obstructing Static Analysis of Programs. Technical Report. UMI Order Number: CS-2000-12., University of Virginia

[5] Farmer, W. M., Guttman, J. D., Swarup, V. 1996b. Security for mobile agents: Authentication and state appraisal. In Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS), E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds., Lecture Notes in Computer Science, vol. 1146, Springer-Verlag, New York, 118–130.

[6] Vigna, G. 1997. Protecting mobile agents through tracing. In Proceedings of the Third ECOOP Workshop on Mobile Object Systems: Operating System Support for Mobile Object Systems

[7] Yee, B. S. 1999. A sanctuary for mobile agents. In Secure Internet Programming: Security Issues for Mobile and Distributed Objects, J. Vitek and C. Jensen, Eds., Lecture Notes in Computer Science, vol. 1603, Springer-Verlag, New York, 261–274

[8] Morris, T., Nair, V.S.S., Private Computing on a Public Platform, Department of Computer Science and Engineering, Southern Methodist University, Technical Report 06-CSE-01, 2006

[9] Free Security Scanner For Network Explorations & Security Audits, www.insecure.org/nmap

[10] SANS Institute, www.sans.org

[11] Morris, T., Nair, V.S.S., PCPP: Private Computing on Public Platforms A New Paradigm in Public Computing, 2nd IEEE International Conference on Wireless Pervasive Computing, 2007, Feb. 2007 (to appear)

[12] M.H. Dunham. Data Mining: Introductory and Advanced Topics. Prentice Hall,Upper Saddle River, NJ, 2003

[13] Specman Elite Testbench Automation, http://www.verisity.com/products/specman.html

[14] Chess D., B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, Itinerant Agents for Mobile Computing. Technical Report, October 1995, IBM T.J. Watson Research Center, NY.

[15] Z. Alkhalifa, V. S. S. Nair, N. Krishnamurthy, and J. A. Abraham. Design and evaluation of system-level checks for on-line control flow error detection. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):627–641, 1999.