Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses

Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, Sal Stolfo
{fapap, arh, shlomo, eeskin, sal}@cs.columbia.edu
Department of Computer Science
Columbia University
New York, NY

Contact Author: Eleazar Eskin <eeskin@cs.columbia.edu>
Keywords: anomaly detection, Windows registry, intrusion detection

Abstract

We present a host-based intrusion detection system for Microsoft Windows. The core of the system is an algorithm that detects attacks on a host machine by looking for anomalous accesses to the Windows Registry. The key idea is to first train a model of normal registry behavior for a host and to use this model to detect abnormal registry accesses at run-time. The system trains a normal model using data that contains no attacks and then at run-time checks each access to the registry in real time to determine whether or not the behavior is abnormal and corresponds to an attack. We evaluate the system by training the system on a set of normal registry accesses and then use the system to detect the actions of malicious software. The system is effective in detecting the actions of malicious software while maintaining a low rate of false alarms.

1 Introduction

Microsoft Windows is one of the most popular operating systems today, and also one of the most often attacked. Malicious software is often used to perpetrate these attacks. There are two widely deployed lines of defense against malicious software: virus scanners which attempt to detect the malicious software, and security patches that fix the security hole in the operating system that the malicious software exploits. Both of these methods for protecting hosts against malicious software suffer from the same drawback. They are effective against known attacks, but are unable to detect and prevent new types of malicious software.

Most virus scanners are signature based meaning they use byte sequences or embedded strings in software to identify certain programs as malicious [2, 16]. If a virus scanner’s signature database does not contain a signature for a malicious program, the virus scanner can not detect or protect against that malicious program. In general, virus scanners require frequent updating of signature databases otherwise the scanners become useless [21]. Similarly, security patches protect systems only when they have been written, distributed and applied to host systems. Until then, systems remain vulnerable and attacks can and do spread widely.

In many environments frequent updates of virus scanner signature databases and security patches are unlikely to occur on a timely basis causing many systems to remain vulnerable. This leads to the potential of very widespread destructive attacks caused by malicious software. Even in environments
where updates are frequent, the systems are vulnerable between the time new malicious software is created and the time that it takes for the software to be discovered, new signatures and patches created, and ultimately distributed to the vulnerable systems. Since malicious software may be propagated through email, often the malicious software can reach the vulnerable systems long before the updates are available.

Another line of defense is through intrusion detection systems (IDS). Host-based IDS systems monitor a host system and attempt to detect an intrusion. In the ideal case, an IDS can detect the effects or behavior of malicious software rather then distinct signatures of that software. Unfortunately, the commercial IDS systems that are widely used are signature based algorithms as well. These algorithms match host activity to a database of signatures which correspond to known attacks. This approach, like virus detection algorithms, requires previous knowledge of an attack and is rarely effective on new attacks. However, recently there has been growing interest in the use of data mining techniques, such as anomaly detection, in IDS systems [15, 17]. Anomaly detection algorithms build models of normal behavior in order to detect behavior that deviates from normal behavior and which may correspond to an attack [1, 4]. The main advantage of anomaly detection is that it can detect new attacks and can be an effective defense against new malicious software. Anomaly detection algorithms have been applied to network intrusion detection [4, 12, 14] and also to the analysis of system calls for host based intrusion detection [5, 7, 9, 13, 20]. There are two problems to the system call approach to host based intrusion detection which inhibits their use in actual deployments. The first is that the computational overhead of monitoring all system calls is very high which degrades the performance of a system. The second is that system calls themselves are irregular by nature which makes it difficult to differentiate between normal and malicious behavior which may cause a high false positive rate.

In this paper, we examine a new approach to intrusion detection that monitors a program’s use of the Windows Registry. We examine the use of the Microsoft Windows Registry as a sensor and demonstrate how to use data collected by the sensor to detect malicious activity. We present a system called RAD (Registry Anomaly Detection), which monitors the accesses to the registry in real time and detects the actions of malicious software.

The Windows Registry is an important part of the Windows operating system and is very heavily used, making it a good source of data. By building a sensor on the registry and applying the information gathered to an anomaly detector, we can detect registry activity that corresponds to malicious software. The main advantages of monitoring the Windows Registry is that registry activity is regular by nature, can be monitored with low computational overhead, and almost all system activities query the registry.

Our anomaly detection algorithm is based on a registry-specific version of PHAD (Packet Header Anomaly Detection), an anomaly detection algorithm originally presented to detect anomalies in packet headers [17]. We show that the data generated by a registry sensor is useful in detecting malicious behavior. We shall describe how various malicious programs use the registry, and what data can be gathered from the registry to detect these malicious activities. We then apply an anomaly detection algorithm to this data to detect abnormal registry behavior which corresponds to the actions of malicious software. By showing the results of an experiment and detailing how various malicious activities use the registry, we show that the registry is a good source of data for intrusion detection. The paper will also discuss the modifications of the PHAD algorithm as it is applied in the RAD system.

The RAD system has three basic components: a registry sensor, a model generator, and an anomaly detector. The sensor serves to output data for each registry activity to a database where it is stored for training. The model generator reads data from the database and creates a model of normal behavior. The model is then used by the anomaly detector to decide whether each new
registry access should be considered anomalous. We present results of experiments evaluating the RAD system and demonstrate that it is effective in detecting attacks while maintaining a low rate of false alarms.

2 Modeling Registry Accesses

2.1 The Windows Registry

In Microsoft Windows the registry is a database of information about a computer’s configuration. The registry contains information that is continually referenced by many different programs. Information stored in the registry includes the hardware installed on the system, which ports are being used, profiles for each user, configuration settings for programs, and many other parameters of the system. It is the main storage location for all configuration information for almost all programs. The Windows Registry is the source for all security information: policies, user names, and passwords. The registry also stores much of the important configuration information that programs need to run.

The registry is organized hierarchically as a tree. Each entry in the registry is called a key and has an associated value. One example of a registry key is

HKCU\Software\America Online\AOL Instant Messenger (TM)\CurrentVersion\Users\aimuser\Login\Password

This is a key used by the AOL instant messenger program. This key stores an encrypted version of the password for the user name aimuser. Upon start up the AOL instant messenger program queries this key in the registry in order to retrieve the stored password for the local user. Information is accessed from the registry by individual registry accesses or queries. The information associated with a registry query is the key, the type of query, the result, the process that generated the query and whether the query was successful. One example of a query is a read for the key shown above. For example, the record of the query is:

Process: aim.exe
Query: QueryValue
Key: HKCU\Software\America Online\AOL Instant Messenger (TM)\CurrentVersion\Users\aimuser\Login\Password
Response: SUCCESS
ResultValue: "BCOFH1HBBBAHF"

The Windows Registry is an effective data source to monitor attacks because many attacks show up as anomalous registry behavior. Many attacks take advantage of Windows’ reliance on the registry. Indeed, many attacks themselves rely on the Windows Registry in order to function properly.

Many programs store important information in the Registry, notwithstanding the fact that other programs can arbitrarily access the information. Although some versions of Windows include security permissions and Registry logging, both features are seldom used (because of the computational overhead and the complexity of configuration options).

2.2 Analysis of Malicious Registry Access

Most Windows programs access a certain set of Registry keys during execution. Furthermore, most users use a certain set of programs routinely while running their machines. This may be a set of all programs installed on the machine or a small subset of these programs. Another important
characteristic of Registry activity is that it is regular over time. Most programs either only access the registry on start-up and shutdown, or access the registry at specific intervals. This regularity makes the registry an excellent place to look for anomalous activity since a malicious program may substantially deviate from this normal activity and can be detected as anomalous.

Many attacks involve launching programs that have never been launched before and changing keys that have not been changed since the operating system had first been installed by the manufacturer. If a model of the normal registry behavior is computed over clean data, then these kinds of registry operations will not appear in the model. Furthermore malicious programs may need to query parts of the registry to get information about vulnerabilities. A malicious program can also introduce new keys that will help create vulnerabilities in the machine.

Some examples of malicious programs and how they produce anomalous registry activity are described below.

- **Setup Trojan:** This program when launched adds full read/write sharing access on the file system of the host machine. It makes use of the registry by creating a registry structure in the networking section of the Windows keys. The structure stems from HKLM\Software\Microsoft\Windows \CurrentVersion\Network\LanMan. It then makes about eight new keys for its use. It also accesses HKLM\Security\Provider in order to find information about the security of the machine to help determine vulnerabilities. This key is not accessed by any normal programs during training or testing in our experiments and its use is clearly suspicious in nature.

- **Back Orifice 2000:** This program opens a vulnerability on a host machine, which grants anyone with the back orifice client program complete control over the host machine. This program does make extensive use of the registry, however, it uses a key that is very rarely accessed on the Windows system. HKLM\Software\Microsoft\VBA\Monitors was not accessed by any normal programs in either the training or test data, which allowed our algorithm to determine it as anomalous. This program also launches many other programs (LoadWC.exe, Patch.exe, runonce.exe, bo2k1p_int1.e) as part of the attack all of which made anomalous accesses to the Windows Registry.

- **Aimrecovery:** This is a program that steals passwords from AOL users. It’s actually a very simple program that simply reads the keys from the registry where the AOL Instant Messenger program stores the user names and passwords. The reason that these accesses are anomalous is because Aimrecovery is accessing a key that usually is accessed and was created by a different program.

- **Disable Norton:** This is a very simple exploitation of the registry that disables Norton Antivirus. This attack toggles one record in the registry, the key HKLM\SOFTWARE\INTEL \LANDesk\VirusProtect6\CurrentVersion\Storages\Files\System\RealTimeScan \OnOff. If this value is set to 0 then Norton Antivirus real time system monitoring is turned off. Again this is anomalous because of its access to a key that was created by a different program.

- **L0phtCrack:** This program is probably the most popular password cracking program for Windows machines. It retrieves the hashed SAM file containing the passwords for all users and then uses either a dictionary or brute force approach to find the passwords. This program also uses flaws in the Windows encryption scheme which allows the program to discover some of the characters in the password. This program uses the registry by creating its own section in the registry. This will consist of many create key and set value queries, all of which will
be on keys that did not exist previously on the host machine and therefore have not been seen before.

Another piece of information that can be used in detecting attacks is that all programs observed in our data set, and presumably all programs in general, cause Explorer to access a key specifically for that application. The key

```
HKLM\Software\Microsoft\Windows NT \CurrentVersion\Image File Execution Options\processName
```

where processName is the name of the process being run is a key that is accessed by Explorer each time an application is run. Therefore this will make it easy to tell when new applications are run, which will be a starting point to determine malicious activity. In addition many programs add themselves in the auto-run section of the Windows Registry under

```
HKLM\Software\Microsoft\Windows \CurrentVersion\Run.
```

While this is not malicious in nature, this is a rare event that can definitely be a hint that a system is being attacked. Trojans such as Back Orifice utilize this part of the registry to auto load themselves on each boot.

Anomaly detectors do not look for malicious activity directly. They look for deviations from normal activity. It is for this reason that any deviation from normal activity will be declared an attack by the system. The installation of a new program on a system will be viewed as anomalous activity. Programs often create new sections of the registry and many new keys on installation. This will cause a false alarm, much like adding a new machine to a network may cause an alarm on an anomaly detector that analyzes network traffic. There are a few possible solutions to get around this problem. Malicious programs often install quietly so that the user does not know the program is being installed. This is not the case with most installations. The algorithm could be modified to ignore alarms while the install shield program was running because that would mean that the user is aware that a new program is being installed. Another option would be to simply prompt the user when a detection occurs, this way the user can let the algorithm know that this program is not malicious and it could therefore be added to the training set of data to update the anomaly detector for permissible software.

### 3 Registry Anomaly Detection

In order to detect anomalous registry accesses, RAD generates a model of normal registry activity. A set of five features are extracted from each registry access. Using these feature values over normal data, a model of normal registry behavior is generated. This model of normal consists of a set of consistency checks applied to the features. When detecting anomalies, the model of normal determines whether the values of the features of new registry accesses are consistent with the normal data or not. If not, the algorithm labels the access as anomalous.

#### 3.1 RAD Data Model

The RAD data model consists of 5 features directly gathered from the registry sensor. The five raw features used by the RAD system are as follows.

- **Process**: This is the name of process accessing the registry. This is useful because it allows the tracking of new processes that did not appear in the training data.
• **Query:** This is the type of query being sent to the registry, for example, `QueryValue`, `CreateKey`, and `SetValue` are valid query types. This allows the identification of query types that have not been seen before. There are many query types but only a few are used under normal circumstances.

• **Key:** This is the actual key being accessed. This allows our algorithm to locate keys that are never accessed in the training data. Many keys are used only once for special situations like system installation. Some of these keys can be used to create vulnerabilities.

• **Response:** This describes the outcome of the query, for example `success`, `not found`, `no more`, `buffer overflow`, and `access denied`.

• **Result Value:** This is the value of the key being accessed. This will allow the algorithm to detect abnormal values being used to create abnormal behavior in the system.

Some records are anomalous because they have a value for a feature that is inconsistent with the normal data. However, some records are anomalous because they have an inconsistent combination of features although each feature itself may be normal. Because of this, we examine pairs of features. For example, let us consider the registry access displayed in Table 1. The basic features for the normal program `aim.exe` versus the malicious program `aimrecover.exe` do not appear anomalous. However, the fact that the program `aimrecover.exe` is accessing a key that is usually associated with `aim.exe` is in fact an anomaly. Only by examining the combination of the two raw features can we detect this anomaly.

### 3.2 RAD Classification Algorithm

Using the 5 features that we monitor from each registry access, we train a model over normal data which allows us to classify each observed access as either normal or malicious.

Any anomaly detection algorithm can be used to perform this modeling. Since we aim to monitor a significant amount of data in real time, the algorithm must be very efficient. We apply a probabilistic algorithm described in Eskin, 2002 [6] and here we provide a short summary of the algorithm. The algorithm is similar to the heuristic algorithm that was proposed by Chan and Mahoney in the PHAD system [17], but is more robust.

In general, a principled probabilistic approach to anomaly detection can be reduced to density estimation. If we can estimate a density function $p(x)$ over the normal data, we can define anomalies as data elements that occur with low probability. In practice, estimating densities is a very hard problem (see the discussion in Schölkopf et al., 1999 [18] and the references therein.) In our setting, part of the problem is that each of the features have many possible values. For example, the `Key` feature has over 30,000 values in our training set. Since there are so many possible feature values, relatively rarely does the same exact record occur in the data. Data sets with this characterization are referred to as sparse.

Since probability density estimation is a very hard problem over sparse data, we propose a different method for determining which records from a sparse data set are anomalous. We define a set of consistency checks over the normal data. Each consistency check is applied to an observed record. If the record fails any consistency check, we label the record as anomalous.

We apply two kinds of consistency checks. The first consistency check evaluates whether or not a feature value is consistent with observed values of that feature in the normal data set. We refer to this type of consistency check as a first order consistency check. More formally, each registry record can be viewed as the outcome of 5 random variables, one for each feature, $X_1, X_2, X_3, X_4, X_5$. Our
Table 1: Registry Access Records. Two registry accesses are shown. The first is an normal access by AOL Instant Messenger to the key where passwords are stored. The second is a malicious access by AIMrecover.exe to the same key. The final column shows which fields register as anomalous. Note that the composite keys are useful for detecting the anomalous behavior of AIMrecover.exe. This is because under normal circumstances only AIM.exe accesses the key that stores the AIM password. Another process accessing this key is very suspicious.
consistency checks compute the likelihood of an observation of a given feature which we denote $P(X_i)$.

The second kind of consistency check handles pairs of features as motivated by the example in Table 1. For each pair of features, we consider the conditional probability of a feature value given another feature value. These consistency checks are referred to as second order consistency checks. We denote these likelihoods $P(X_i|X_j)$. Note that for each value of $X_j$, there is a different probability distribution over $X_i$.

In our case, since we have 5 feature values, for each record, we have 5 first order consistency checks and 20 second order consistency checks. If the likelihood of any of the consistency checks is below a threshold, we label the record as anomalous.

What remains to be shown is how we compute the likelihoods for the first order ($P(X_i)$) and second order ($P(X_i|X_j)$) consistency checks. Note that from the normal data, we have a set of observed counts from a discrete alphabet for each of the consistency checks. Computing these likelihoods reduces to simply estimating a multinomial. In principle we can use the maximum likelihood estimate which just computes the ratio of the counts of a particular element to the total counts. However, the maximum likelihood estimate is biased when there is relatively small amounts of data. When estimating sparse data, this is the case. We can smooth this distribution by adding a virtual count to each possible element. This is equivalent to using a Dirichlet estimator [3]. For anomaly detection, as pointed out in Mahoney and Chan, 2001 [17], it is critical to take into account how likely we are to observe an unobserved element. Intuitively, if we have seen many different elements, we are more likely to see unobserved elements as opposed to the case where we have seen very few elements.

To estimate our likelihoods we use the estimator presented in Friedman and Singer, 1999 [8] which explicitly estimates likelihood of observing a previously unobserved element. The estimator gives the following prediction for element $i$

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

(1)

if element $i$ was observed and

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

(2)

if element $i$ was not previously observed. $\alpha$ is a prior count for each element, $N_i$ is the number of times $i$ was observed, $N$ is the total number of observations, $k^{\alpha}$ is the number of different elements observed, and $L$ is the total number of elements. The scaling factor $C$ takes into account how likely it is to observe a previously observed element versus an unobserved element. $C$ is computed by

$$C = \left( \sum_{k=k_0}^L \frac{k^{\alpha} + N}{k^{\alpha} + N} \right)^{-1} \left( \sum_{k \geq k_0} m_k \right)^{-1}$$

(3)

where $m_k = P(S = k)\frac{\Gamma(k+1)}{(k+1)!} \frac{1}{\Gamma(k^{\alpha}+1)}$ 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. Although the computation of $C$ is expensive, it only needs to be done once for each consistency at the end of training.

The prediction of the probability estimator is derived using a mixture of Dirichlet estimators each which represent a different subset of elements that have non-zero probability. Details of the probability estimator and its derivation are given in [8] and complete details of the anomaly detection algorithm are given in [6].
Note that this algorithm labels every registry access as either normal or anomalous. Programs can have anywhere from just a few registry accesses to several thousand. This means that many attacks will be represented by large numbers of records where many of those records will be considered anomalous.

4 Architecture

![RAD System Architecture Diagram]

Figure 1: The RAD System Architecture. RegBAM outputs to the data warehouse during training model and to the anomaly detector during detection mode.

The basic architecture of the RAD system consists of three components, the registry auditing module, the model generator, and the real-time anomaly detector. An overview of the RAD architecture is shown in Figure 1.

4.1 Registry Basic Auditing Module

The RAD sensor is composed of the Basic Auditing Module (BAM) for the RAD system which monitors accesses to the registry. BAMS implement an architecture and interface for sensors across the system. They include a hook into the audit stream (in this case the registry) and various communication and data-buffering components. BAMS use an XML data representation similar to the IETF standard for IDS systems [11]. BAMS are described in more detail in [10].

The Registry BAM (RegBam) runs in the background on a Windows machine where it gathers information on registry reads and writes. RegBam uses Win32 hooks to tap into the registry and log all reads and writes to the registry. The software uses a similar architecture to SysInternal’s Regmon [19]. After gathering the registry data, RegBam can be configured for two distinct uses. One use is as the data source for model generation. When RegBam is used as the data source for model generation the output data is sent to a database were it is stored and later used by the model
generator described in Section 4.2 [10]. The second use of RegBam is as the data source for the real-time anomaly detector described in Section 4.3. While in this mode the output of RegBam is sent directly to the anomaly detector where it is processed in real time. An alternative method to collect the registry accesses is to use the Windows auditing mechanism. All registry accesses can be logged in the Windows Event Log. Each read or write can generate multiple records in the Event Log. However, this is problematic because the event logs are not designed to handle such a large amount of data. Simple tests demonstrated that by turning on all registry auditing the Windows Event Logger caused a major resource drain on the host machine and in many cases caused the host machine to crash. The RegBam application provides an efficient method for monitoring all registry activity with far less overhead than the native tools provided by the Windows operating system.

4.2 Model Generation Infrastructure

Similar to the Adaptive Model Generation (AMG) architecture [10], the system uses RegBam to collect registry access records. Using this database of collected records from a training run, the model generator then creates a model of normal usage.

The model generator uses the algorithm discussed in Section 3 to build a model that represents normal usage. It utilizes the data stored in the database which was generated by RegBam during training. The model itself is comprised and stored as serialized Java objects. This allows for a single model to be generated and to be easily distributed to additional machines. Having the model easily deployed to new machines is a desirable feature since in a typical network, many Windows machines have similar usage patterns which allow for the same model to be used for multiple machines.

4.3 Real-Time Anomaly Detector

For real time detection to take place, RegBam hooks up with an anomaly detector feeding it live data to analyze. The anomaly detector will load the normal usage model created by the model generator and begin reading each record from the output data stream of RegBam. The algorithm discussed in Section 3 is then applied against each record of registry activity. The score generated by the anomaly detection algorithm is compared by a user configurable threshold to determine if the record should be considered anomalous. A list of anomalous registry accesses are stored and displayed as part of the detector.

4.4 Efficiency Considerations

An important consideration of this system is efficiency. In order for a system to detect anomalies in a real time environment it cannot consume excessive system resources. This is especially important in the case of registry attack detection because of the heavy amount of traffic that takes place in the registry. While the amount of traffic can vary greatly from system to system, in our case the traffic load is about 50,000 records per hour. Our distributed architecture is designed in order to minimize the resources used by the host machine. It is possible to spread the work load on to several separate machines so that the only thing running on the host machine is the lightweight RegBam. However this will increase network load due to the increased communication between components. These two concerns can be used to configure the system in order to create the proper proportion between host system load and network load. The RegBam module is a far more efficient way of gathering registry activity then turning on full auditing in the Windows Event Log.
5 Evaluation and Results

5.1 Data Generation

In order to evaluate the RAD system, we gathered data by running a registry sensor on a host machine. Beyond the normal execution of standard programs, such as Microsoft Word, Internet Explorer, and Winzip, the training also included performing tasks such as emptying the Recycling Bin and using the Control Panel. All data used for this experiment is publicly available online in text format at http://www.cs.columbia.edu/ids/rad.

The training data collected for our experiment was collected on Windows NT 4.0 over two days. This comprised approximately 500,000 records. The system was tested on a full day of attack data. This data was comprised of approximately 300,000 records most of which were normal program executions with a total of 12 attacks interspersed among normal process executions. 9 of these attacks were unique. We trained the anomaly detection algorithm presented in Section 3 over the normal data and evaluated each record in the testing set. We evaluate our system by computing two statistics. We compute the detection rate which is the percentage of attacks detected and the false alarm rate which is the percentage of normal records which are labeled anomalous. By varying the threshold for the inconsistency checks, we were able to vary the detection rate and false positive rate. We plot the false positive rate versus the detection rate in an ROC (Receiver Operator Characteristic) curve shown in Figure 2. At a certain threshold the detection rate was 97%. The false positive rate was .012%.

![ROC Curves for Registry Anomaly Detection](image)

Figure 2: The ROC curve showing the performance of the RAD system over the test set.

Many of the false positives were from processes that were simply not run as a part of the training data. If more training data were gathered it is likely that the amount of false positives would be greatly reduced.

We want to emphasize that the rate of false positives is typically much lower than for other IDS systems. This is partly because the accesses to the Windows Registry are more regular which makes normal registry easier to model.
6 Conclusions

By using registry activity on a Windows system we were able to label all processes as either attacks or normal. We have shown that registry information is very useful when detecting attacks against a Windows system. We have shown that registry activity is regular and described ways in which attacks would generate anomalies in the registry. Most important, we have shown that a system that uses only registry data can be effective as an intrusion detection system and would improve protection of systems in cases of new attacks that would otherwise pass by scanners that have not been updated on a timely basis. Our system is able to detect the actions of malicious software while keeping a low false positive rate.

Future plans include combining the RAD system with a system that evaluates Windows Event Log data. This will allow for various data correlation algorithms to be used to make more accurate system behavior models, which will provide a more accurate anomaly detection system. Part of our future plans for the RAD system include adding data clustering capabilities. This clustering will allow for groups of activity records to be considered malicious as a group rather than individually. We also propose to store the system behavior model as part of the registry itself. The motivation behind this is to use the anomaly detection algorithm to protect the system behavior model from being maliciously altered, hence making the model itself secured against attack. These additions to the RAD system will make the system a more complete and effective tool for detecting malicious behavior on the Windows platform.

References


