An Agent-based Bayesian Method for Network Intrusion Detection

John Pikoulas

A thesis submitted in partial fulfilment of the requirements of Napier University for the degree of Doctor of Philosophy

December 2003

“Every man has been made by God in order to acquire knowledge and contemplate”
Pythagoras

“If I were to wish for anything, I should not wish for wealth and power, but for the passionate sense of potential, for the eye, which, ever young and ardent, sees the possible. What wine is so sparkling, so fragrant, so intoxicating, as possibility!”

Søren Aabye Kierkegaard

Acknowledgements

I would like to dedicate my work to my parents Leonidas and Maria and my sister Vasiliki, that gave me the means and the motivation to finish my research, and to my girlfriend Karen that kept me motivated so I could be able to finish.

I would also like to say a special thanks to my supervisors Dr. W. Buchanan and Dr. M. Manion, who were so patient with me, and gave my all that I needed to finish my research, and become a better man. Also I would like to thank Dr. Kostantinos Tryantafyllipoulos for his vast contribution to my research, and the way that I think.

Declaration

I hereby declare that this thesis is my own work, except otherwise stated, and that it is not submitted or being under consideration for submission in any other university.

Table of contents

Abstract

Security is one of the major issues in any network and on the Internet. It encapsulates many different areas, such as protecting individual users against intruders, protecting corporate systems against damage, and protecting data from intrusion. It is obviously impossible to make a network totally secure, as there are so many areas that must be protected. This thesis includes an evaluation of current techniques for internal misuse of computer systems, and tries to propose a new way of dealing with this problem.

This thesis proposes that it is impossible to fully protect a computer network from intrusion, and shows how different methods are applied at differing levels of the OSI model. Most systems are now protected at the network and transport layer, with systems such as firewalls and secure sockets. A weakness, though, exists in the session layer that is responsible for user logon and their associated password. It is thus important for any highly secure system to be able to continually monitor a user, even after they have successfully logged into the system. This is because once an intruder has successfully logged into a system, they can use it as a stepping-stone to gain full access (often right up to the system administrator level). This type of login identifies another weakness of current intrusion detection systems, in that they are mainly focused on detecting external intrusion, whereas a great deal of research identifies that one of the main problems is from internal intruders, and from staff within an organisation. Fraudulent activities can often he identified by changes in user behaviour. While this type of behaviour monitoring might not be suited to most networks, it could be applied to high secure installations, such as in government, and military organisations.

Computer networks are now one of the most rapidly changing and vulnerable systems, where security is now a major issue. A dynamic approach, with the capacity to deal with and adapt to abrupt changes, and be simple, will provide an effective modelling toolkit. Analysts must be able to understand how it works and be able to apply it without the aid of an expert. Such models do exist in the statistical world, and it is the purpose of this thesis to introduce them and to explain their basic notions and structure.

One weakness identified is the centralisation and complex implementation of intrusion detection. The thesis proposes an agent-based approach to monitor the user behaviour of each user. It also proposes that many intrusion detection systems cannot cope with new types of intrusion. It thus applies Bayesian statistics to evaluate user behaviour, and predict the future behaviour of the user. The model developed is a unique application of Bayesian statistics, and the results show that it can improve future behaviour prediction than existing ARIMA models. The thesis argues that the accuracy of long-term forecasting questionable, especially in systems that have a rapid and often unexpected evolution and behaviour. Many of the existing models for prediction use long-term forecasting, which may not be the optimal type for intrusion detection systems.

The experiments conducted have varied the number of users and the time interval used for monitoring user behaviour. These results have been compared with ARIMA, and an increased accuracy has been observed. The thesis also shows that the new model can better predict changes in user behaviour, which is a key factor in identifying intrusion detection.

The thesis concludes with recommendations for future work, including how the statistical model could be improved. This includes research into changing the specification of the design vector for Bayesian. Another interesting area is the integration of standard agent communication agents, which will make the security agents more social in their approach and be able to gather information from other agents.

1 Introduction

1.1 Introduction

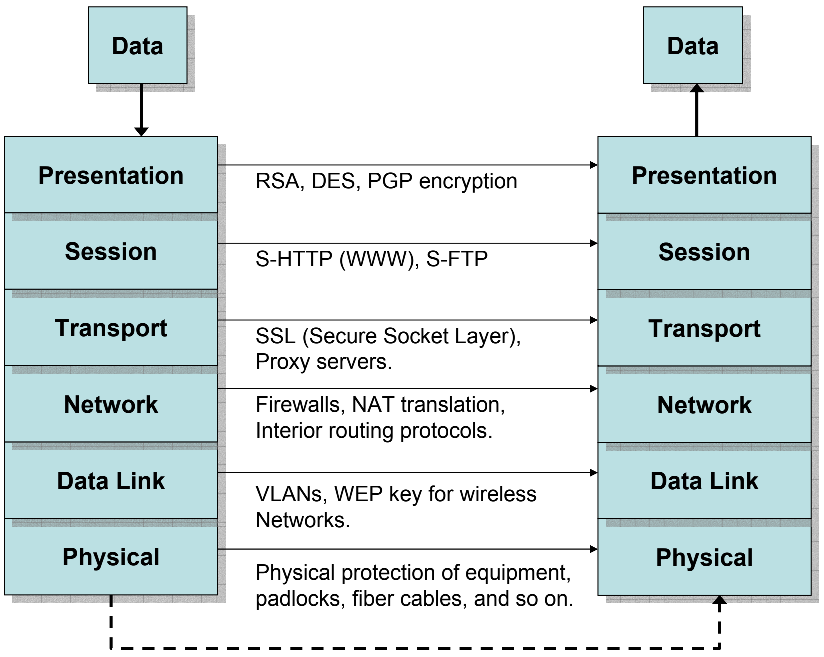
This chapter introduces the thesis. It defines the learning outcomes and measure of achievements. Along with this, the motivation of the work is defined, and a summary of the contribution of the research work. The chapter also includes background sections on Internet security and operating systems, and typical security breaches. Finally, the chapter outlines the structure of the thesis.

1.2 Motivation

Security is one of the major issues in any network and on the Internet. It encapsulates many different areas, such as protecting individual users against intruders, protecting corporate systems against damage, and protecting data from intrusion. It is obviously impossible to make a network totally secure, as there are so many areas, which must be protected. Chapter 2 expands on this and presents the diagram shown in Figure 1.1, which illustrates some of the methods that can be used to protect a network. This just encapsulates data as it travels over the network, whereas security protects users, data and systems against both internal and external intruders. It is well known that many security systems, such as firewalls, are used to exclude external intruders, but these can offer few boundaries if an intruder attacks a system from within the network.

Security is also complicated by new areas of development which must be protected against, such as:

- **Enhanced transaction processing**. With distributed applications, single transactions may span multiple hosts and multiple processes. However, applications must still guarantee the atomic integrity of transactions (that is, a unit of work).
- **Portable clients and servers**. In distributed environments, both users and their applications can be portable. On the Internet, users can access applications from virtually any physical location. Similarly, system administrators may move applications and software components among various machines based on such factors as process loading, hardware failure, performance, and so on.
- **Enhanced data sets**. Applications no longer deal with only simple data types. Current technologies allow system designers to incorporate enhanced objects such as video, audio and multimedia into even the most basic applications.



Data
Data
Presentation
Session
Transport
Network
Data Link
Physical
RSA, DES, PGP encryption
S-HTTP (WWW), S-FTP
SSL (Secure Socket Layer), Proxy servers.
Firewalls, NAT translation, Interior routing protocols.
VLANs, WEP key for wireless Networks.
Physical protection of equipment, padlocks, fiber cables, and so on.
Presentation
Session
Transport
Network
Data Link
Physical

Figure 1.1 Protecting data at differing levels

With the move toward distributed processing, applications must continue to appear seamless to users. This can be difficult to implement, as there are changes to the underlying architecture. Furthermore, distributed architectures increase this burden because they provide users easy access to a myriad of applications.

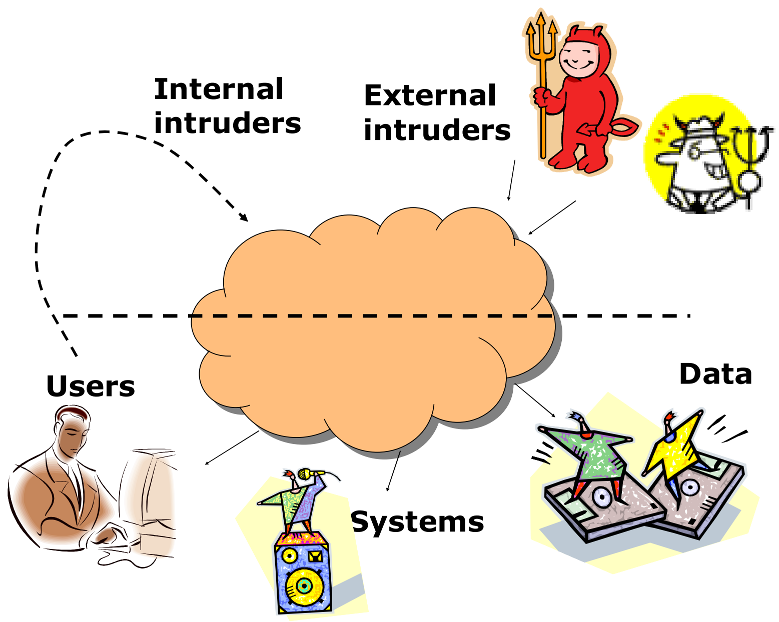


Figure 1.2 Protecting users, systems and data against internal and external intruders

Distributed computing often contains many compromises for developers, such as giving ease-of-access that will typically increase security concerns. As a result, system designers that are developing for distributed environments benefit greatly when security is integral, as effective security cannot be added on as a *patch* [1.1].

Unfortunately, most security systems in distributed systems are based on a passive and static model. A passive system is one which uses a defined set of rules, which are checked against, and a static system is one in which the system does not change its rules. In this model, once someone passes the security barrier, there are few methods of baring further intrusion, until it is too late. Passive authentication mechanisms are easy to break.

1.3 The Subject of the Research

Our research consists of evaluating current techniques for internal misuse of computer systems, and tries to propose a new way of dealing with this problem. The authentication problem described in Section 2.1, has direct implications to the internal security of computer networks, and most of the time it results in internal misuse. The research uses intelligent agents to monitor the behaviour of each user, and uses Bayesian statistics to evaluate the user behaviour, and predict the future behaviour of the user.

Some important parameters for the research were:

- **Distributed systems**. Software agents have been used as most computer systems are now interconnected, which makes the system as distributed as possible. As much as possible the security system uses a small footprint size, in terms of both processing overhead and memory size, so it can be scalable and work in any size of computer system. The only common thing that all the computers have is that they are connected to some kind of network, either on an internal (intranet) and/or external (Internet).
- **Agent-based systems**. To optimise the distribution of software components, the system is based around software agents. The *user-end* agent is deployed across all the nodes in the network, and the *core* agent, is deployed on a server. The user-end agent monitors each user who logs into a nodes, and logs their activities. The first action of the user-end agent is to contact the core agent, and get the required user profile. This profile contains the specific user information on the previous history actions of the user, and prediction values for this user session. The user-end agent monitors all the actions of the user, and the resources that are accepted, in terms of both local resources and network resources. Each user-end

agent collects information from the user behaviour, and sends them back to the core agent. The core agent is responsible for storing user profiles for all users that have access to the protected network. It is also responsible for giving the user profile to the user-end agent, each time that a user profile is requested. It is also responsible for processing new profile data from each user and create new prediction data using the new Bayesian model, that are stored in the user profile.

- **Results processing and forecasting**. The research uses a mathematical model which allows for monitoring of results in a fast and highly adoptable system. The research has concluded that the problem of processing our monitoring data and making a prediction on these data is a linear problem and thus uses a statistical model rather than a neural networks version. The decision was based on the analysis of the initial data from the first range of experiments. For this, it was seen that a linear fashion is observed from user behaviour. If the data were not linear, we would have exponential curve-like graphs, or some kind of fractal shaped graphs. Thus, the research uses a short-term prediction using linear methods, and statistical methods are used to solve the prediction problem. These factors focus on the use of the multivariate Bayesian statistical method, which allows for the processing of all the different attributes of each user, concurrently, and can be highly modifiable, with minimum effort. By changing only the timing variable (the difference in time between two consecutive observations) of our statistical model, it is possible to process different statistical monitoring data from different users, something that is virtually impossible with other more *traditional* statistical systems.
- **Intrusion data**. To verify the models used, the research has collected internal intrusion data from computer systems. It is obviously a difficult task to generate reliable intrusion data, as not all of the misuses are reported further than the information systems of the companies where the misuse has occurred [1.2].

1.4 Prediction Techniques

In this thesis, the terms, *prediction* and *forecasting* are used interchangeably, mainly for practical and reference reasons. Often authors use one of the terms discussing the same statistical concept. Forecasting is really the *realisation* of the future, based on some present and past knowledge. In general, there will be some sort of information on hand that can be used to anticipate the future. According to the kind of information that is received, and the kind of predictions that are being produced, there are different types of systems. For example, there can be qualitative forecasting, which is based on some non-numerical basis of information, or quantitative forecasting. This is based on an arithmetic translation of information, which can also be short-term or long-term forecasting.

For every case, this thesis will apply different methods and will always be aware of the particular restrictions and assumptions of the methods used. It will also use the relevant methodology that has been proven very effective for short-term predictions. It is argued that the accuracy of long-term forecasting questionable, especially in systems that have a rapid and often unexpected evolution and behaviour. For example, how would it have been possible to predict the Love Bug virus, the day before it occurred? And, even in its first moments, how could it be possible to predict its spread the next day?

Computer networks are now one of the most rapidly changing and vulnerable systems, where security is now a major issue. A dynamic approach, with the capacity to deal with and adapt to abrupt changes, and be simple, will provide an effective modelling toolkit. Analysts must be able to understand how it works and be able to apply it without the aid of an expert. Such models do exist in the statistical world, and it is the purpose of this thesis to introduce them and to explain their basic notions and structure.

1.5 Learning outcomes of the PhD

The SQA (Scottish Qualifications Authority) have defined four main learning outcomes for a PhD. These are defined below, and along with the measure of achievement.

LO1. The creation and interpretation of new knowledge, through original research, or other advanced scholarship, of a quality to satisfy peer review, extend the forefront of the discipline, and merit publication.

The research work has defined a new agent-based system, based on an application of a novel forecasting method. It has merited publication at several key points:

1. Triantafyllopoulos K and Pikoulas J, 'Multivariate Bayesian regression applied to the problem of network security', Journal of Forecasting.
2. Pikoulas J, Buchanan W, Mannion M and Triantafyllopoulos K, ‘An Intelligent Agent Security Intrusion System’, 9th IEEE Conference in ECBS, Lund, Sweden, April 2002.
3. Pikoulas J, Buchanan W, Mannion M and Triantafyllopoulos K. ‘An

agent-based Bayesian forecasting model for enhanced network security'. Proceedings of the Eighth Annual IEEE International Conference and Workshop On the Engineering of Computer Based Systems-ECBS 2001. IEEE Comput. Soc. 2001, pp. 247–54.

4. Pikoulas J, Buchanan WJ and Triantafyllopoulos K, 'An Intelligent Intrusion Detection Environment using Software Agents', Thirteenth International Conference Software & Systems Engineering and their Applications, Paris, December 2000.
5. Pikoulas J, Mannion M, Buchanan WJ, 'Software agents and computer network security', Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000). IEEE Comput. Soc. 2000, pp. 211–217.

LO2. A systematic acquisition and understanding of a substantial body of knowledge, which is at the forefront of an academic discipline or area of professional practice.

The work has involved investigating network intrusions systems, and methods of attack. From this, an agent-based model was created for the new method. These experiments initially used a small network with one server and two workstations. It also used different users and different timings to measure the results of the system. This experiment ensured that the system was functioning. It used just one user to test that the agents were working, and were monitoring each user correctly. It also tested that the agents were communicating properly and exchanging the required information.

The next phase of experiments ensured that the system could handle an increased information load, and ensured that the prediction system was working correctly. To measure the proposed system against another model, the results were compared with a commercially statistical software package. It was seen from this that the proposed system could handle more of an information loading, and that the results were correct.

In the final phase, the experiments used a large amount of data from our users and processed them to prove that the system can handle all kinds or information loads. These loadings verified that the system is only limited by the computational power of the server, and the speed of the network connections. It also showed at which numerical points the statistical model was producing errors, and that have to be addressed for further research.

LO3. **The general ability to conceptualize, design and implement a project for the generation of new knowledge, applications or understanding at the forefront of the discipline, and to adjust the project design in the light of unforeseen problems.**

The project idea started from the observation of how powerful and damaging, a common user can be when they decide to harm the computer system. This idea was the starting point of the research as to what computer security is, and how it can be broken using the most conventional and common operating systems that exist on the market today, and why? The research also involved modelling typical user behaviour and actions, and how these can be quantified. After an extensive research on security, the conclusion was that many commercial operating systems have very little internal security, and implement very few methods to protect against potential threats. The methods that were employed were typically passive ones. This was the weakness that focused on the research seeking a method which could make security both active and adaptive.

The research also uses software agent technology, as this is the technology that is most suitable for distributed system applications. Software agents are robust and highly adoptable software entities and, if used correctly, have minimum impact on the use of computational power, and memory usage of the system.

The research started on prediction systems, especially short-term prediction, and their accuracy and response times. At first, it was decided that the prediction of the user behaviour, if broken down to simple parts, could be linear. Thus sophisticated and very computational consuming methods, like Neural Networks, could be replaced by simpler methods. From our research on prediction techniques, it was found that Bayesian statistics are more suitable for the statistical prediction model. Bayesian statistics produces a compact short-term prediction model, that is highly configurable, and almost maintenance free.

Designing and determining the new technology was an important part of the project, but putting all the various parts together was another large task. The software environment used was Java-based, as it is highly versatile, and a platform independent programming language. It also provides ease-of-access to network programming, using the TCP and IP protocols.

The model started from the inner core, using key parts of the project: the communication mechanism; the implementation of the prediction model; and the format of the messages that the agents use to exchange information. After finishing this implementation and after unit testing all the various parts, the system components were put together, and used to test the model.

LO4. A detailed understanding of applicable techniques for research and advanced academic inquiry.

In this research project, many new ideas and technologies were used from different disciplines. For example, software agent technology was found to be the best technology for a distributed and secure environment. This involved research into the different aspects of software agents, their applications, and the way that other researchers have implemented them. Software agents, and agent in general, are very new and research on them is growing. At an early stage of the research, we had to draw a logical conclusion on various questions for which we could not find any answers, as there was no current research on the subject yet.

The research has studied different methods of network intrusion, which are typically not fully documented (as many organisation do not want intrusions into their network known), especially for internal intrusions.

The research also investigated user behaviour, and how to quantify user behaviour, so it can be compared and measured. As we are monitoring user behaviour there is obviously an ethical point of view. Thus, the benefits of increased security must have a benefit which over-rules the disadvantages of user monitoring. It is proposed that user behaviour would not be used in a normal network but would be applied, along with other methods, to a highly secure network, especially in a government or military network.

Another element of the research was in the usage of statistics. For this, various statistical methods were investigated, and were evaluated for their application to short-term prediction. For this, the research has to understand the differences on those systems, and how a statistical model could be used to overcome the problems these systems had.

We also researched the tools that we used to develop and test the project. This involved the comparison of the most common programming languages to determine which one was more suitable, and especially in terms of development tools, books and papers, and feedback from the programming community.

1.6 A New Computer Era

The availability of cheap and easy-to-find computing and networking technology has driven a large amount of people to get on-line and connect to Internet. In January 1993 there were 1,313,000 connected to the Internet, while in January 2002 were more than 100,000,000 [1.1]. With this growth comes the problem of computer security, which has been a problem since the first BBS (Bulleting Board System). With this, there was a problem in safeguarding corporate information against users and organisations who wish to abuse data, systems or users on networks.

Current security methods tend to be based on passwords, which are inherently weak. The system presented in this research uses a software agent and statistical prediction techniques to detect and prevent unauthorised security breaches by internal users. The experimental results show that predicting the user moves can be the best solution of detecting and preventing a security breach, in a highly secure environment.

At present, computer security is a major worry for organisations, and this will continue as long as there is information, which can be stolen or damaged. The laws, which relates to this type of crime are still being developed, and will take some time to implement.

Many users think that the main problem relating to security is caused by external users (often known as *hackers*), but Carter and Catz [1.5] have shown that the primary threat comes from individuals inside an organisation. This observation results in the fact that much more emphasis has to be placed on internal control mechanisms of systems, such audit log analysis. A recent example is the Hotmail [1.6] security breach:

> Microsoft was informed on August 29, 1999, of a security loophole, which allowed anyone to read the Inbox of any Hotmail user provided the username, was known (which is often relatively easy as many users advertise there Hotmail address). If its owner was not accessing the targeted account (also dependent on account settings), all other email functions were available (delete, forward and send). Reports varied on the duration of the vulnerability, but a Swedish newspaper informed Microsoft on Sunday morning, and Microsoft took its Hotmail gateway servers offline for a couple of hours on the Monday morning. Access to Hotmail was restored was soon, but several sources also reported in finding another vulnerable Hotmail server later in the day. By Monday evening, the vulnerability was removed. It is not clear how long the vulnerability was known before Sunday, but one source said it was known eight weeks before.

It is obviously difficult to ever have a totally secure network which is totally secure against intrusion as differing types of intruders have differing financial budgets. Network administrators must know how secure there network is and the level of intrusion that would be required to compromise the network (Figure 1.3). At the lower layers, such as the home user and data mining professional, the budget is likely to be relatively low, and easy to protect against. At the upper levels, such as large-scale military and government activities, which have large budgets it is extremely difficult to guard against.

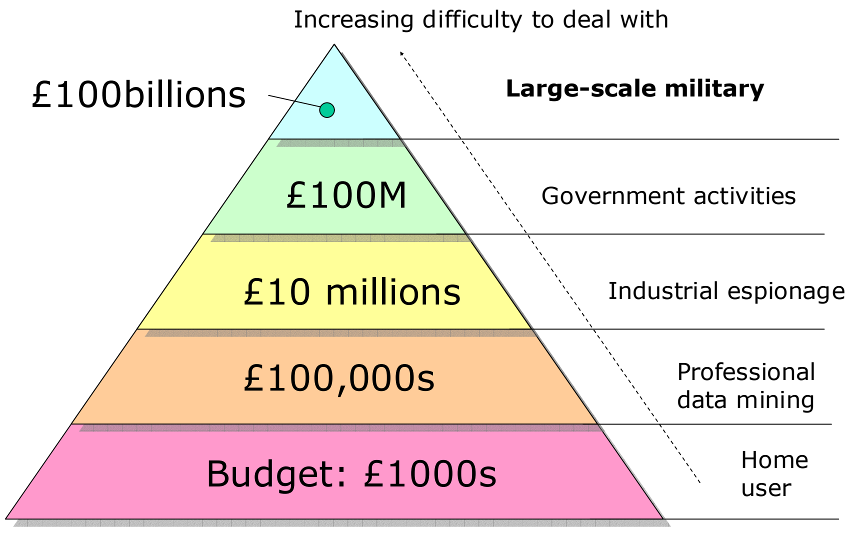


Figure 1.3 Varying budgets for intrusion

In many cases, people that called themselves *hackers* [1.7] create security breaches. In the early days of intrusion, these intruders did it for self-projection and not to improve their finances. Nowadays, there is a great deal to be gained from breaching system security, these individuals tend to be:

- **Professional programmers and IT specialists**. They typically have extensive knowledge of the protocols and hardware that are used by organisations. It is also typical that they work in teams, and have some *inside* knowledge of the organisational systems.
- **Government agents**. Typically, ex-military using advanced *information warfare* methods (such as CIA software agents [1.7]). For example, according to the Sunday Times on 4 August 1996, American intelligent agents have intruded into the computer of the European parliament and European commission as part of an international espionage campaign aimed at stealing economic and political secrets. Countries, including China and Korea, have also been targeted by US secret agents, and it is likely that spies of the future will use electronic surveillance techniques rather than traditional spying methods.

Coldwell [1.9] identifies that lone intruders often gain the headlines in newspapers (over 5000 intrusion attacks alone in the last decade in the USA, UK, Germany and Australia), but he argues that these are being used as 'fall guys' for a cover-up on computer fraud within organisations and through organised crime.

No security measure can ever guarantee a risk free environment, but increased security can make the system less easy-to-use. Many businesses require to grant access to some parts of their system and must make it easy to use, thus increasing potential exposure. In addition, the majority of people that want to be protected from intrusions rely on a specific security application that is, in most cases, a passive one, with no ability to adapt to the changes of the system that it protects. Proper security controls require planning and careful implementation and in most cases are not simply installed as a *security tool*. Forestalling potential security breaches requires careful monitoring and management, and it is critical that these controls deter real problems. Threats change as fast as both technology and business. Adaptation and improvisation are the key features of a security system. No hardware or software element can ever be immune from security weaknesses.

Many organisations will typically rely on an operating system for system security. To provide a measure of how secure a system is, the US Government defines certain security levels: D, C1, C2, B1, B2, B3 and A1, which are published in the *Trusted Computer Security Evaluation Criteria* books (each of which has a different coloured cover to define their function). These include:

- **Orange book**. Describes system security.
- **Red book**. Interpretation of the Orange book in a network context.
- **Blue book**. Application of Orange book to systems not covered in the original book.

For example, Windows NT uses the C2 security level. It has the following features:

- **Object control**. Users own certain objects and they have control over how they are accessed.
- **User names and passwords**. These are unique user's names, which are defined within groups, and have restricted access to objects.
- **No object reuse**. Once a user or a group has been deleted, the user and group numerical IDs are not used again. New users or groups are granted a new ID number.

- **Security auditing system**. This allows the system administrator to trace security aspects, such as user login, bad logins, program access, file access, and so on.
- **Defined keystroke for system access**. In Windows NT/2000/XP, the Ctrl-Alt-Del keystroke is used by a user to log into the system.

1.7 Structure of this thesis

The format of the thesis is as follows:

- **Chapter 1: Introduction**. This provides an introduction of the thesis. It defines the motivation for the work, and how well the research has matched the learning outcomes.
- **Chapter 2: Computer Security Issues**. This outlines the different techniques for intrusion and misuse of computer systems. It will highlight some of the most important and dangerous aspects of the computer security risks.
- **Chapter 3: Computer Network Security System**. This outlines the usage of software agents, and should give a basic understanding of what software agents are, and how they are used in a security model.
- **Chapter 4: Prediction Techniques**. This deals with prediction methods, and gives a brief overview of statistical methods, and areas they most used. A particular focus will be on explaining Bayesian methods, as these are used in the proposed prediction model.
- **Chapter 5: Simple Tutorial on the Proposed Bayesian Method.** This demonstrates a short and simplified tutorial of the statistical method used in this research. The aim of this is to introduce to the reader to an easy example so they can understand how it works, and can understand the following chapters.
- **Chapter 6: Proposed Statistical Prediction Method.** This covers the new statistical model that will be used in the proposed model.
- **Chapter 7: Software Agent Security System.** This explains, in detail, the software agent system, the different parts it consists off, and how it works.
- **Chapter 8: Experiments**. This outlines the different sets of experiments that were untaken.
- **Chapter 9: Results**. This presents results of the experimentation, as compared with data from other statistical models.
- **Chapter 10: Conclusions**. This is the final chapter and it deals with the main conclusions from the research, how successful it was, and proposes future research plans.

1.8 References

[1.1] *Secure Computing with Java.* Now and the future. Sun Microsystems. http:// java. sun. com/ marketing/collateral/security.html, June 12, 2002.

[1.2] *News Coverage on Security Breaches by Authorized Users*, http://www.waveset. com/ Features/ inside_threat.html

[1.3] Internet Software Consortium, Domain Survey, http://www.isc.org/ds/,

[1.4] Carter and Catz, *Computer Crime: an emerging challenge for law enforcement*, FBI Law Enforcement Bulleting, pp 1-8, December 1996.

[1.5] Scotty Strunk, *Intrusion Detection FAQ*, SANS Institute Resources, December 2, 1999.

[1.6] Sunday Times, *CIA Spies On Europe*, August 4, 1996, London, UK.

[1.7] Roger Blake, *Hackers in The Mist,* North-western University, December 2, 1994.

[1.8] National Institutes of Health. *Center for Information Technology,* http://www.alw.nih.gov/Security/securityprog.html#commercial, October 1998.

[1.9] Coldwell RA, *Should we be giving graduates in information technology hacking skills?,* Computer Education, No. 88, Feb. 1988, pp. 24-27.

2 Computer Security Issues

2.1 Introduction

This chapter provides an understanding of the methods used by intruders in order to gain entry to a system, and outlines various methods that intruders use to break computer security, alongside some operating system weaknesses that intruders exploit to gain access to computer systems. Unfortunately, no system can ever be totally secure, even with the strongest of hardware and/or software protection.

Our contribution is focused on internal misuse of computer networks, and typically operates after the initial user authentication. For this, many different techniques can be employed to tackle the same point of failure: the authentication phase. Our proposal deals with this condition when the user is authenticated correctly and allowed access to the network, but start to misbehave, either to damage to the network, or become involved in actions against the rules of the organisation.

2.2 Protecting networks

Most users typically think that security only relates to the protection of data on user systems. From an organisational point-of-view, it encapsulates other area such as:

- **Data protection** [2.1][2.2][2.3][2.4]. This is typically where sensitive or commercially important information is kept. It might include information databases, design files or source code files. One method of reducing this risk is to encrypt important files with a password, and another is to encrypt data with a secret electronic key (files are encrypted with a commonly known public-key, and decrypted with a secret key, which is known only by user who has the right to access the files).
- **Software protection** [2.5][2.6]. This involves protecting all the software packages from damage or from being misconfigured. A misconfigured software package can cause as much damage as a physical attack on a system, because it can take a long time to find the problem.
- **Physical system protection** [2.7][2.8][2.9]. This involves protecting systems from intruders who might physically attack the systems. Normally, important systems are locked in rooms and then within locked rack-mounted cabinets.
- **Transmission protection** [2.10][2.11]. This involves an intruder tampering with a

transmission connection. It might involve tapping into a network connection or total disconnection. Tapping can be avoided by many methods, including using optical fibres, which are almost impossible to tap into (as it would typically involve sawing through a cable with hundreds of fibre cables, which would each have to be reconnected as they were initially). Underground cables can avoid total disconnection, or its damage can be reduced by having redundant paths (such as different connections to the Internet).

Security is one of the major issues in any network and on the Internet. It encapsulates many different areas, from protecting individual users against intruders, to protecting corporate systems against damage. It is impossible to ever make a network totally secure, as there are many areas which must be protected. The most important, obviously, is to keep networking equipment in a secure environment. Figure 1.1 previously outlined some of the methods which could be used to protect a network at each of the layers of the OSI model. The physical protection of a network can involve using fibre cables instead of copper ones, as copper cables leak electromagnetic radiation, which can be used to tap into the communications. At the next layer (the **Data Link Layer**) a VLAN can be used to limit the transmission of data with a network. With a VLAN is it difficult for an intruder to tap into a network, unless than can configure the ports of the switch in which the VLAN connects to. In addition, network switches allow for point-to-point communications methods, where only the two devices communicating will be able to listen to the communications. This differs from a traditional Ethernet type network, which uses a bus topology where any device on the network can listen to the communications on the bus.

At the next layer (the **Network Layer**), firewalls are implemented on a router to define a security policy, for the network addresses and port which are allowed to flow into and out of a network. In a highly secure network, there is typically only one gateway into the network. This gateway operates the global security policy for the complete network. Along with this, there are other firewalls which implement localised security policy. Cisco Systems present a three-tiered architecture for network, where the lowest layer provides the connectivity, and is typically implemented by switches. The network layer above this implements the distribution layer, which interconnects networks using routers. These routers implement the security policies between workgroups. At the very centre of the network, there is the core element, which provides interconnectivity across a wide area.

Above this (at the **Transport Layer**), secure sockets can be used, which use encrypted information to pass data segments. At the **Session Layer**, messages can be encrypted using protocols, such as S-HTTP [2.12]. Finally, the data itself can be encrypted using a public- or private-key encryption scheme, such as RSA [2.13] or PGP [2.14]. The earlier that the data is encrypted the better, as even if an intruder can read the information at the other levels, they will find it difficult to recover the original data (normally only by brute force, or by time-based attacks). Details, though, given at the various layers could give some information on the data interchange. For example, an intruder might be able to determine the source and destination network addresses from the data packets, which could be used to the intruders benefit. A new technique called Network Address Translation (NAT) can be used at the network layer to hide IP addresses from intruders. With this, the NAT translates internal addresses into a generic address. It does this by remembering the connection that was made, especially the destination network address, and the source and destination ports.

A firewall is the routing computer, which isolates the intranet from the outside world. Another method is to use an intermediate system, which isolates the intranet from the external Internet. These intermediate systems include proxies and tunnels. A proxy acts on behalf of clients and sends requests from clients to a server. It thus acts as a client when it communicates with a server, but as a server when communicating with a client. A proxy is typically used for security purposes where the client and server are separated by a firewall. The proxy connects to the client side of the firewall and the server to the other side of the firewall. Thus, the server must authenticate itself to the firewall before a connection can be made with the proxy. Only after this has been authenticated will the proxy pass requests through the firewall. A proxy can also be used to convert between different versions of the protocols that use TCP/IP, such as HTTP. Each intermediate system is connected by TCP and acts as a relay for the request to be sent out and returned to the client.

The data link and physical layers are typically protected with physical methods, such as basing security on the physical address of networked devices, and on limiting physical access to the devices. At the network and transport layers, advanced firewalls and authentication servers can be setup to limit intruders.

Major elements of security are **authentication**, **authorization**, and **accounting** (AAA). These allow for enhanced security for who is allowed to log into a network, what they are allowed to do, and logs the things that they have done. Typically this security is applied at the edge of a network, using a network access server (NAS). These servers must use a secure protocol in order for a host to communicate with a specialised security server. This server contains a database of the users and their associated passwords, and any other configurations. On routers, there are three main security protocols:

- **TACACS+** [2.15]. This system uses a centralised validation of users. It maintains a database, which is a TACACS+ daemon running on a host computer, and provides for individual configuration of authentication, authorization, and accounting.
- **RADIUS** . This is a distributed client/server system. It is an open-standard and is specified in RFCs 2865, 2866, and 2868. With RADIUS, a client typically runs on a router, which sends authentication requests to a centralized RADIUS server.
- **Kerberos** [2.16]. This uses a secret-key encryption authentication method, but does not allow for authorization or accounting. It was designed to provide authentication, where a third party authenticates each of the parties to the other. For this, a trusted Kerberos server provides authentication tickets to users, which have a limited life.

The Authentication Problem

Often security systems base themselves on *authentication*, which is the process of determining whether an object (such as a user, a hardware device or a software component) is, in fact, the object that they are declared to be. In private and public computer networks, including the Internet, authentication is commonly achieved through the use of logon passwords [2.17].

Passwords are typically implemented at the session layer of the OSI model, and are one of the weakest areas for a point of network intrusion. The best protection at the layers of the OSI model, such as firewall protection, data encryption, and physical security will count for very little if a user's password has been breached. It is well known that reusable passwords have traditionally been the weakest link in host security, and are often poorly chosen and therefore easy to guess. It is also the case that password databases, such as ones used in Microsoft Windows and in Unix, are poorly protected, and are typically based on hash methods, which can be easily broken with exhaustive search techniques. In recent years the reusable password has become a favourite target for network eavesdroppers [2.18], and passive monitoring of shared networks is undetectable. It has thus become easier as computing trends have favored small, single-user workstations that give many users privileged network access. Since passwords are often be sent *in the clear*, they can be easily captured by eavesdroppers and used at a later time for network intrusions. Intruders often leave behind automated *sniffer* programs so that they can harvest passwords for later use.

Some systems can use Kerberos encryption for all sensitive network traffic, and foil both active and passive attacks. But the user is required to use only *kerberized* hosts to participate, thus remote logins over an unprotected link or from terminals are still vulnerable. This problem also exists with other application-level encryption protocols like SSH. They are well protected tools and should be used when available, but in many situations they are either unavailable or inconvenient.

One-time passwords (OTP) are another convenient way to solve this problem, as they allow users to be authenticated over insecure links without revealing any sensitive information. As the name implies, a one-time password is only good for one authentication session, after which it automatically expires, and a new (previously unused) password becomes valid. OTP schemes are less secure than Kerberos and SSH in that they only provide protection against passive attacks. On the other hand OTPs are more flexible than Kerberos and provide most of the protection required, as passive attacks are the most common and the hardest to detect.

The disadvantage with OTP schemes, is that they require the use of hand-held authenticators or *smart cards* to participate in cryptographic challenge/response dialogues with a remote host. With small numbers of users such a system may be feasible, but at large sites, the prospect of arming everyone with expensive smart-cards is daunting and undesirable.

Inside attacks

Inside attacks are the most common way of attacking a computer. The advantage of this way of attack is that the intruder is authorized for the system that they are attacking, so the authentication is not an object. One common solution to this attack is to monitor every action of the user [2.19]. This monitoring typically uses automatated software that is static, and is too complicated to install and maintain. It may not always make the right decisions, and the results often raise the wrong alert flags. Another disadvantage is that constant monitoring of the users, creates a corresponding decline in creativity and productivity.

2.3 Security loopholes

Software always has inefficiencies known as software bugs. System administrators and programmers cannot track down and eliminate all possible holes, and intruders can often use these to find a security hole to break in to [2.1]. Software inefficiencies are exploited in the: server daemons; the client applications; the operating system: and the network stack. Software bugs, which could allow intruders into a system, can be classified in the following manner:

- **Buffer overflows**. Most of the security holes that occur are due to buffer overflows For example a programmer may reserve up to 256 characters to hold a login username, as they assume that no-one will ever have a login name which is larger than this limit. Unfortunately, this is an opportunity for an intruder to send more than

256 characters, which may cause the system to act unreliably and allow the intruder into the system. Intruders can find these loopholes by either examining the source code of the system and then search it for any security loopholes. They may also look at the machine code of the program, or may even examine every place the program has input and try to overflow it with random data. If the program crashes, there is a good chance that carefully constructed input will allow the intruder to break in. Note that this problem is common in programs written in C/C++, but rare in programs written in Java (as Java has a dynamic allocation of arrays, and has automated garbage collection, which frees the memory of arrays which are not used any more).

- **Unexpected combinations**. Programs are usually constructed using many layers of code, including the underlying operating system as the bottom most layer. Intruders can often send input that is meaningless to one layer, but meaningful to another layer. The most common language for processing user input on the WWW is PERL, which normally sends this input to other programs for further evaluation. A common intrusion technique would be to enter something like "| *mail < /etc/passwd*". This is executed because PERL asks the operating system to launch an additional program with that input. However, the operating system intercepts the pipe '|' character and launches the 'mail' program as well, which causes the password file to be emailed to the intruder.
- **Unhandled input**. Most programs are written to handle valid input, but many programmers do not consider what happens when somebody enters input that does not match the specification (such as entering invalid characters).
- **Race conditions**. Most systems today are multitasking and multithreaded. This means that they can execute more than one program at a time. With this, there is always a danger that if two programs need to access the same data they will access it at the same time. Imagine two programs, *A* and *B*, who need to modify the same file. In order to modify a file, each program must first read the file into memory, change the contents in memory, and then copy the memory back out into the file. A race condition occurs when program *A* reads the file into memory, then makes the change. However, before *A* gets to write the file, program *B* steps in and does the full read/modify/write on the file. Now program *A* write its copy back out to the file. Since program *A* started with a copy before *B* made its changes, all of *B*'s changes will be lost. Since you need to get the sequence of events in just the right order, race conditions are very rare. Intruders normally have to try many times before they get it right, and are able to intrude into the system.

While many of the lower layers of the OSI model now are fairly well protected against intruders, the layers above these provide many of the major problems. This typically occurs with operating systems problems, and email and Internet browser problems, especially related to electronic worms. For example, Microsoft identified that there were security vulnerabilities in two ActiveX controls (scriptlet.typelib and Eyedog), which allowed a web page to take unauthorized action against a person who visited it. Specifically, the web page would be able to do anything on the computer that the user could do. This security breach was used to spread a worm, as it allowed the Active X controls to be marked as safe for scripting, and can therefore be called by Internet Explorer. Both these controls have rather benign uses, but someone determined could create a worm, which could easily spread through the Internet.

Wilson [2.27] outlines that many intruders make use of vulnerabilities available to them, and he argues that managers in the government and private sector organisations should be more accountable for updating their systems with the latest operating systems fixes.

2.4 System configuration

No software system is ever completely free of bugs, or properly configured. Unfortunately, software developers have made software more complex, and have generally focused on adding applications, rather than focussing on software reliability. After a series of embarrassing bugs in Microsoft Windows, Chris Mason at Microsoft puts it into context with:

> "Since human beings themselves are not fully debugged yet, there will be bugs in the code no matter what you do",
> "We could conceivably put a company out of business with a bug in a spreadsheet, database, or word processor",

Thus like any man-made system, there are likely to be weaknesses in it, which the human did not envisage.

System configuration bugs can be classified in the following manner:

- **Default configurations**. Most systems are shipped to customers with default, easy-to-use configurations. Unfortunately, *easy-to-use* typically means *easy-to-break-in*. Almost all Unix or Microsoft Windows computers which are initially installed can be easily intruded into.
- **Lazy administrators**. A surprising number of machines are configured with an empty root/administrator password. This is because the administrator is too lazy to configure one and wants to get the machine up and running as quickly as possible, with the minimum of fuss. Unfortunately, they never get around to fixing the password later, allowing intruders easy access. One of the first things an intruder will do on a network is to scan all machines for empty passwords.

- **Hole creation**. Virtually all programs can be configured to run in a non-secure mode. Sometimes administrators will inadvertently open a hole on a machine, such as a TELNET or FTP server connection. Most administration guides will suggest that administrators turn off everything that does not positively need to run on a machine in order to avoid accidental holes. Note that security-auditing packages can usually find these holes and notify the administrator.
- **Trust relationships.** Intruders often *island hop* through the network exploiting trust relationships. A network of machines trusting each other is only as secure as its weakest link.

2.5 Password cracking

Password cracking is the most common method of intruders trying to break in to a computer network and is a special category all to itself. It is an important area as passwords are typically used at the authentication layer for the session layer. Reasons intruders achieve their target are the following:

- **Weak passwords**. Most people use the names of themselves, their children, the spouse, the pet, or their car model as their password. There are also users who choose *password* or simple stay with a default password name (or even do not have a password at all). This gives a reduced list of possibilities that an intruder can try. If these possibilities are entered over a relatively long time period, the system would not report any errors for incorrect entry of a users' password.
- **Dictionary attacks**. Failing the above attack, the intruder can next try a *dictionary attack.* In this attack, the intruder uses a program, which tries every possible word in the dictionary. Dictionary attacks can be done either by repeatedly logging into systems or by collecting encrypted passwords and attempting to find a match by similarly encrypting all the passwords in the dictionary. Intruders normally have a copy of the English dictionary as well as foreign language dictionaries for this purpose. They all use additional dictionary-like databases, such as common names (as in weak passwords) and lists of common passwords. Passwords are also built using a concatenation of common names (such as frogcar). Some systems overcome this by informing the user that they are using a password which is formed with words from a dictionary (and may even bar the user from selecting that password).
- **Brute force attacks.** This is similar to the Dictionary attack, but the intruder tries all possible combinations of characters. This is relatively easy with short passwords, but difficult with long ones. For example a short 4-letter password consisting of lower-case letters can be cracked in just a few minutes (roughly, half a million possible combinations), whereas a 7-character password consisting of upper and lower case, as well as numbers and punctuation (which gives 10 trillion

combinations) can take months to crack assuming that the system can try a million combinations a second (in practice, a thousand combinations per second is more likely for a single machine).

In most networked systems, administrators force users to change password every six months a year. Most of the time the user will actually select the same password, which makes it easier for the intruder to get into the system as they intrude into the computer system once and copy the encrypted password file that the operating system manufacturer predefines its location. In most cases, intruders use more than one computer to decrypt the password file. Once they have the password, they can login like normal computer user and examine the computer system or copy information, depending of the security clearance level that the stolen user data have.

2.6 Internet security problems

The Internet can cause a great deal of problems as it can allow open access for external users. Typical attacks include:

- **E-mail bombing**. This is where an external user(s) continually send an identical e-mail message to a particular address. E-mail spamming is a variant of this, where the same message is sent to many users, at a single time. This is made worse if the recipient actually responds back to the e-mail spamming message using all the recipients on the address list, as this will also flood the network with unwanted e-mail messages. E-mail bombing without the permission of the user is illegal in many countries and there should always be a message on the e-mail message which identifies the method that can be used to delete a user's name from an e-mail bombing database.
- **E-mail spoofing**. This is where external users setup incorrect e-mail addresses, and then send e-mails to other users. This could be either to cloak the identity of the person, or to try to pretend to be another known person. On a personal level, when pretending to be another person, this can be particularly disturbing, as many users think that the e-mail address in the `From:` field is always from that person. The cloaking is typically used with e-mail bombing, where a false user name is used to send e-mail bombs.
- **Denial of service** (DOS) attacks. These are severe attaches where external users continually try and access a server, typically a WWW server, an email server or network routers, and try to slow it down to the point that no-one else can access it. Unlike many other attacks there is very little that can be done about it, apart

from tracing the source, and trying to block the transmissions.

2.7 Network attacks

Typical network attacks can be categorised as [2.21] for external attacks are:

- **IP spoofing attacks** [2.22][2.23]. This is where the intruder steals an authorized IP address. Typically, it is done by determining the IP address of a computer and waiting until there is no one using that computer, and then using the unused IP address. Several users have been accused of accessing unauthorized material because others have used their IP address. A login system which monitors IP addresses and the files that they are accessing over the Internet cannot be used as evidence against the user, as it is easy to steal IP addresses.
- **Packet-sniffing** [2.24]. This is where the intruder listens to TCP/IP packets which come out of the network and steals the information in them. Typical information includes user logins, e-mail messages, credit card number, and so on. This method is typically used to steal an IP address, before an IP spoofing attack. Most TELNET and FTP programs actually transmit the user name and password as text values; these can be easily viewed by an intruder.
- **Sequence number prediction attacks**. Initially, in a TCP/IP connection, the two computers exchange a start-up packet which contains sequence numbers. These sequence numbers are based on the computer's system clock and then run in a predictable manner, which can be determined by the intruder.
- **Trust-access attacks**. This allows an intruder to add their system to the list of systems, these are then allowed to log into the system without a user password. In UNIX this file is the *.rhosts* (trusted hosts) which is contained in the user's home directory. A major problem is when the trusted hosts file is contained in the root directory, as this allows a user to log in as the system administrator.

For internal abuse attacks:

- **Passwords attack**. This is a common weak-point in any system, and intruders will generally either find a user with an easy password (especially users which have the same password as their login name) or will use a special program which cycles through a range of passwords. This type of attack is normally easy to detect. The worst case of this type of attack is when an intruder determines the system administrator password (or a user who has system privileges). This allows the intruder to change system set-ups, delete files, and even change user passwords.
- **Session hi-jacking attacks**. In this method, the intruder taps into a connection between two computers, typically between a client and a server. The intruder then simulates the connection by using its IP address.

- **Shared library attacks**. Many systems have an area of shared library files. These are called by applications when they are required (for input/output, networking, graphics, and so on). An intruder may replace standard libraries for ones that have been tampered with, which allows the intruder to access system files and to change file privileges. The intruder might tamper with dynamic libraries (which are called as a program runs), or with static libraries (which are used when compiling a program). This would allow the intruder to possibly do damage to the local computer, send all communications to a remote computer, or even view everything that is viewed on the user screen. The intruder could also introduce viruses and cause unpredictable damage to the computer (such as remotely rebooting it, or crashing it at given times).
- **Social engineering attacks**. This type of attack is aimed at users who have little understanding of their computer system. A typical attack is where the intruder sends an email message to a user, asking for their password. Many unknowing users are tricked by this attack. From the initial user login, the intruder can then access the system and further invade the system. In one research study it was found that when telephoned by an unknown person and asked what their password was, 90% of users immediately gave it, without asking any questions.
- **Technological vulnerability attack**. This normally involves attacking some part of the system (typically the operating system), which allows an intruder to access the system. A typical one is for the user to gain access to a system and then run a program, which reboots the system or slows it down by running a processor intensive program. This can be overcome in operating systems such as Microsoft Windows and UNIX by granting re-boot rights only to the system administrator.

2.8 Conclusions

This chapter has discussed some of the implications of computer security. We saw that there are many ways in which intruders can harm a computer system. Some times, it is weaknesses of the operating system, or of the security software packages configuration, or human error. Especially, when there are large numbers of computer users, and the logs created for monitoring these users are very large, most often the logs are examined after a breach of system security.

Another conclusion that we can draw from this chapter is that pre-emptive action against computer misuse is a method that can deliver maximum-security effects, if substantiated with correct evidence.

2.9 References

[2.1] Principles of Data Protection, http://www.dataprotection.gov.uk/principl.htm.

[2.2] Information Commissioner's Office, *Corporate Plan 2001-2004*, http://www.dataprotection.gov.uk/dpr/dpdoc.nsf/ed1e7ff5aa6def3080256636 0045bf4d/d3b92b72a098804e80256b120045d37c/$FILE/2001cp.pdf,7th December 2001.

[2.3] SUBMISSION TO THE HOME OFFICE BY THE DATA PROTECTION COMMISSIONER, *Data Protection Act 1998: Post – Implementation Appraisal*, 7th December 2001.

[2.4] Data Protection Act 1998, The Stationery Office Limited, ISBN 0 10 542998 8, Crown Copyright 1998.

[2.5] 1394 Trade Association, http://grouper.ieee.org/ groups/ 802/ 17/ documents/ presentations/ sep2001/ draft_lbr_oam_01.pdf, LARA-16-1-T-Nov00/0.40:2, July 11, 2000.

[2.6] Lynn Jones and Tamitha Carpenter, *Decentralized Network Monitoring & Response: A Motivating Example*, Stottler Henke Associates, Inc., 1107 NE 45th St., Suite 427 Seattle, WA 98105, http://64.81.14.30/ReliabilityWeb/SAPmisconfig.html.

[2.7] W. Ware *et al.*, *Security controls for computer systems*, Rand Corp. Tech. Rep. R-609, 1970. (Classified confidontial.) (SFR)

[2.8] R. Turn and W. Ware, *Privacy and security in computer systems*, I-A1 *Amer. Scientist*, vol 63, pp. 196-203, Mar.-Apr. 1975. (I-A1)

[2.9] Jerome H. Saltzer and Michael D Schroeder, *The Protection of Information in Computer Systems*, Project MAC and the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology Cambridge, Mass. 02139, Copyright © 1975 by J. H. Saltzer.

[2.10] Adam H. Li, Jay Fahlen, Tao Tian, Luciano Bononi, So-Young Kim, Jeong-Hoon Park and John Villasenor, *Generic Uneven Level Protection Algorithm for Multimedia Data Transmission over Packet-Switched Networks*, IEEE International Conference on Computer Communications and Networks Proceedings, pp. 340-346, October 2001.

[2.11] Bill Pearson, *Digital Transmission Digital Transmission Content Content Protection*, Intel Corporation, June 16, 1999, http://www.dtcp.com/data/dtcp_public.pdf

[2.12] Adam Shostack, *An Overview of SHTTP*, May 1995, http://www.homeport.org/~adam/shttp.html

[2.13] Francis Litterio, The Mathematical Guts of RSA Encryption, Copyright © 1999-2001, http://world.std.com/~franl/crypto/rsa-guts.html

[2.14] Adam Back, *PGP Timeline*, http://www.cypherspace.org/~adam/timeline/.

[2.15] Philip Smith, *ISP Case Study – PIPEX Version 1.0*, Consulting Engineering, Office of the CTO, Cisco Systems Inc, Friday, June 19, 1998 ISP/IXP Networking Workshop.

[2.16] Johan Danielsson and Assar Westerlund,Kerberos 4 from KTH, 18/8/1999, http://www.pdc.kth.se/kth-krb/doc/kth-krb_toc.html.

[2.17] *SearchSecurity.com Definitions*, http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci211621,00.html.

[2.18] Computer Emergency Response Team, Advisory CA-94:01: Ongoing Network Monitoring Attacks. February, 1994.

[2.19] *Dr. Crime's Terminal of Doom and Other Tales of Betrayal, Sabotage and Skullduggery*. Jun. 1, 2002 Issue of CIO Magazine.

[2.20] Modori Asaka, Shunji Okasawa and Atsushi Taguchi, *A Method of Tracing Intruders by Using Mobile Agents*, Bunkyo Green Court Center Office, 1999.

[2.21] Robert Graham, *FAQ: Network Intrusion Detection Systems*, URL: http://www. robertgraham.com/ pubs/network-intrusion-detection.html, Version 0.7.1, Oct. 1999.

[2.22] National Institutes of Health. Center for Information Technology, http://www.alw.nih.gov/Security/securityprog.html#commercial, October 1998.

[2.23] W.J. Buchanan. Handbook of Data Communications and Networks, Kluwer, 1998.

[2.24] SamsNet, *A Hacker's Guide to Protecting Your Internet Site and Network*, URL:http://mx.nsu.ru/Max_Security/ch28/ch28.htm

[2.25] NetworkICE Corporation, *Packet Sniffing*, http: //www.networkice.com/ advice/ Underground/ Hacking/ Methods/ Technical/ Packet_sniffing/default.htm

[2.26] Alan Ramsbottom, FAQ: NT Cryptographic Password Attacks & Defences, 1997.

[2.27] Wilson C, *Holding management accountable: a new policy for protection against computer crime*, Proceedings of the IEEE 2000 National Aerospace and Electronics Conference, NAECON 2000, IEEE, Oct 2000.

3 Computer Network Security Systems

3.1 Introduction

This chapter describes software agent technology related to security. It provides an outline of software agents, and how they are used in current research. As previously mentioned, network security has become a major issue, especially when connecting through interconnected networks. Computer network security programs can be categorised as follows [3.9]:

- **Security enhancement software**. This enhances or replaces an operating system's built-in security software (such as with Mangle It, Passwd+ [3.45] and Shadow [3.46]).
- **Authentication and encryption software**. This encrypts and decrypts computer files (such as with Kerberos [3.47, 3.48], MD5 [3.49], RIPEM, and TIS Firewall Toolkit [3.50]).
- **Security monitoring software monitor**. This monitors different operations of a computer network and outputs the results to system administrators (such as with Abacus Sentry [3.50, 3.51], COPS [3.52], Tripwire [3.53] and Tiger).
- **Network monitoring software**. This monitors user's behaviour or monitors incoming or outgoing traffic (such as with Argus, Arpwatch [3.54] and ISS).
- **Firewall software and hardware**. This runs on the Internet/intranet gateway of a network, and checks all incoming network traffic for its contents at the network and transport layers of the OSI model. At the network layer, typically the Internet Protocol (IP) addresses are filtered for their source and/or destination, and at the transport layer, the TCP ports and monitored (thus FTP and TELNET traffic could be blocked for incoming data traffic, but SMTP (electronic mail) could be allowed).

These methods are generally centralised and have very little real-time response. Thus, they have no prediction to foresee future user events. The methods outlined also tend to have a central focal point for the security (typically a main server), which could itself become the focus of an attack (such as a denial-of-service attack). The security methods involved in this research are focused on distribution, where the system does not depend on a central point of failure. It also gathers user behavioural information and based on this, it makes a prediction on what the user might do in the future.

A firewall, for example, can implement filtering on IP addresses, and ports, but can also define TCP and UDP idle timeout values and DNS timeouts. This is mainly because multiple connections can be used in Denial-of-Service (DOS) attacks. Along with this, the router can define limits for DoS attack thresholds, such as defining the limits for triggering the deletion of existing half-open sessions. This can be defined as a one-minute session threshold, or the maximum number of incomplete sessions. The firewall, for all its power in filtering data, is only using fixed rules, which are implemented in an efficient manner. It cannot foresee events that will occur, and are typically only responding to previously known attacks.

In choosing a combination of network security programs, the dominant issues are: cost; the desired level of security; and the characteristics of the existing operating system environment. Three mechanisms for illegal behaviour detection are commonly used in computer network security programs [3.9]:

- **Statistical anomaly detection** [3.64]. Many secure systems, such as standard network operating systems and routers, support an audit log, which can store events on applications, security options and on general system properties, and whether they are successful or not. In statistical anomaly detection, the data in these logs can be used to detect network intrusion patterns.
- **Rule-based detection**. These systems use rules to define illegal behaviour. A firewall is basically a rules-based system, which routes based on these filtering rules. If a data packet fails one of the rules, it is deleted from the network. A rules-based system is easy to implement as it can typically relate directly to the security policy of the network. Humans understand rules, and are comfortable with the idea of computer-based systems operating on a fixed set of rules, of which they cannot detour from. Jha and Hussan [3.65] use agents to detect an intrusion based on pre-defined rules. The agent then informs the systems manager, and blocks any further intrusion. They also use a reactionary approach where a separate host-based agent checks log files in order to detect system anomalies caused by successful attacks. Their system is thus defined as pre-emptory and reactionary.
- **Hybrid detection**. This is an amalgamation of statistical anomaly detection and rule-based detection.

3.1.1 Statistical anomaly detection

Statistical anomaly detection systems analyse audit-log data to detect abnormal behaviour. A profile of expected online behaviour for a normal user is predefined and derived from how an organisation expects a user to behave, and from a system administrator's experience, of the way a user is expected to use system resources. The audit logs are analysed and processed for statistical patterns of events for typical operations from which to determine usage patterns. These patterns are compared to the user's profile.

Anderson [3.10] led a project called Safeguard, which adapted the NIDES statistical anomaly-detection subsystem, which profiled the behaviour of individual applications. Statistical measures were then used to determine the proper usage of an application, and what differentiates this from inappropriate usage. With Safeguard, a statistical score is assigned to the operation of applications and represents the degree to which current behaviour of the application corresponds to its established operational pattern. This demonstrated the ability of statistical profiling tools and differentiates the scope of execution from general-purpose applications. It also highlighted that statistical analysis could be effective in analysing activities other than individual users, such as the monitoring applications, rather than the users. The Safeguard analysis greatly reduces the required number of profiles and computational requirements, and the number of typical false-positives to false-negatives security breaches. These results suggest the possible usage of performing statistical analyses on activities at higher layers of abstraction. The system warns administrators that there has been a possible intrusion when a profile is different to a usage pattern. The major drawback with this technique is that it cannot predict extreme changes in user behaviour, which is unfortunate, as changes in a user's behaviour could identify a security breach. It is well known that organisations can often identify users who are committing a fraudulent act within the organisation when they change the behaviour [3.42], such as working different hours, or using differing types of applications. Adair [3.42] highlights that organisations must implement seven basic defence mechanisms to overcome fraud:

- **Physical security**. This includes locking equipment, especially servers and networking equipment, and setting up alarm systems.
- **Personnel security**. Staff should be employed whose main responsibility is to keep data and systems secure. This should also involve security being built into the design, implementation and maintenance of systems. There is typically weaknesses in systems which have not has security built into it at the design stage, and

often security can be seen as an after-thought.

- **Encryption of sensitive data**.
- **Technical surveillance inspection**.
- **Line security**.
- **Suppression of compromising emanations**. This includes shielding of computer equipment.
- **System security**. This includes software access control, logging mechanisms and authentication controls.

If possible, all of these should be implemented in a highly secure environment. Willison [3.43] advances the argument, that organisations must implement a preventative approach to reduce the dangers of fraud. For this, he recommends that *crime-specific opportunity structures* can be created which conceptualise a fraud environment. A key element of this is changes in a user's normal behaviour.

Heatley et al. [3.44] recommend the technique of data mining computer audit logs to detect computer misuse, in terms of computer fraud, information theft, software privacy, and violations of privacy. In this, the authors argue that the access of files can be monitored to identify fraud, such as:

- **Access to sensitive files**. This relates to a class of sensitive files which can be defined as the subject of improper access. In a networked environment, these might relate to important password or user information, such as password databases.
- **Access to random files**. This relates to the selection of files using a random process.
- **Access relating to groups of users**. This defines the probability that a user-selected file is from the sensitive class, is the same for all the member of a group of users.

3.1.2 Rules based detection

Rules based detection systems use a set of rules that define typical illegal user behaviour. These rules are formed by analysing previous patterns of attack, and analyses the audit-log data of a particular user and compares it with the rules. The drawback of this system is that the basic rules are predefined by system administrators, and cannot detect any new attack techniques. If a user exhibits behaviour that is not prescribed by the existing rules, the user can harm the system without being detected.

IDES [3.11] is security enhancement software that stores knowledge about a system's known vulnerabilities, its security policies and information on previous intrusions. The information it uses to determine the network state is limited to the data packet header. As it does not examine the contents of the data packet, it may miss critical information about the nature of the data that goes through the network. It also scales poorly, especially where there are many machines on a high-speed network.

A Kumar and Spafford enhancement [3.13] uses pattern matching, as attacks can be classified as patterns, which are matched against occurrences (the status of the system at that moment) in the system. These patterns can encode dependencies between system conditions and temporal conditions. Crosbie and Spafford use autonomous agents [3.14], which are trained to detect anomalous activity in network system traffic. A drawback of this approach is that the system requires considerable training by a human operator before it becomes effective.

3.1.3 Hybrid detection

Hybrid detection systems are a combination of statistical anomaly detection and rule-based detection systems. Typically, these use rules to detect known methods of intrusion and statistical based methods to detect new methods of intrusion, as illustrated in Figure 3.1.

CMDS (Computer Misuse Detection System) [3.15] is a security-monitoring package that provides a method to watch for intrusions. It detects and thwarts attempted logins, file modifications, Trojan horse installation, changes in administrative configurations and many other signs of intrusion. In addition, it constantly monitors for difficult detection problems like socially engineered passwords, trusted user file browsing and data theft that might indicate industrial espionage. CMDS supports a wide variety of operating systems and application programs. The drawback of this system is that it uses statistical analysis to make additional rules for the system. This is a drawback, as it can only detect attack patterns that have been used in the past and that have been identified as attack patterns, or predefined by the system operators. It also generates long reports and graphs of the system performance that requires to be interpreted by a security expert.

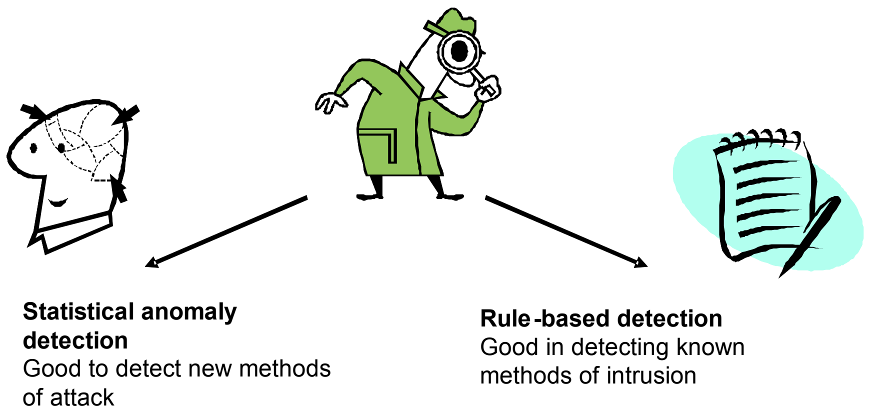

Statistical anomaly detection
Good to detect new methods of attack
Rule-based detection
Good in detecting known methods of intrusion

Figure 3.1: Statistical anomaly and rule-based detection

3.2 Software agents

Agents are programs which automate user tasks [3.1, 3.3], and have great potential in the development of mobile and distributed computing. Every agent satisfies the following four properties: reactive, autonomous, goal-oriented and temporally continuous, as illustrated in Figure 3.2. White [3.3] has defined software agents as automated tasks that otherwise we would have to do ourselves. He also distinguishes agents from other utility software programs by both the ability to perform in distributed computing environments and the ability to supply some domain knowledge to automating tasks for users.

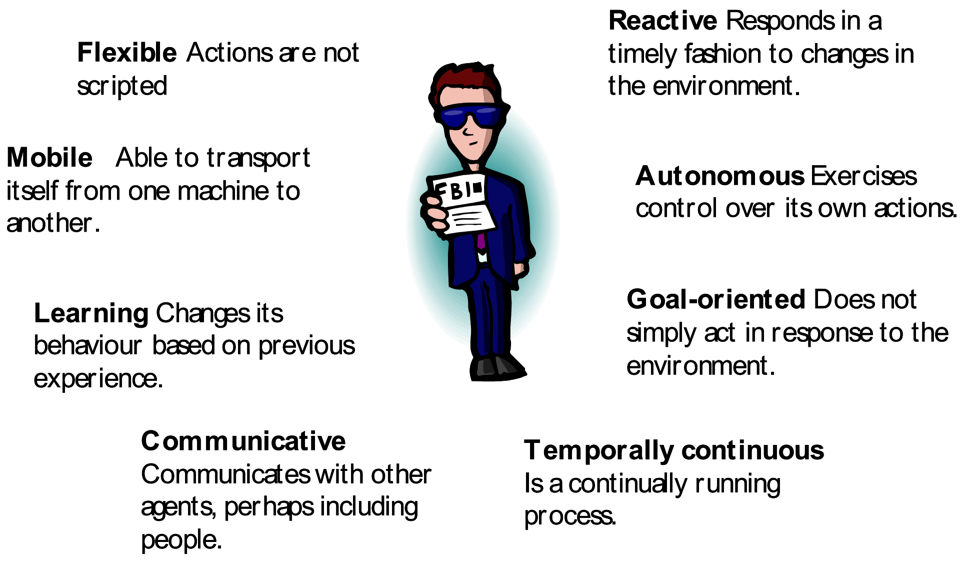

Flexible Actions are not scripted
Mobile Able to transport itself from one machine to another.
Learning Changes its behaviour based on previous experience.
Communicative Communicates with other agents, perhaps including people.
Reactive Responds in a timely fashion to changes in the environment.
Autonomous Exercises control over its own actions.
Goal-oriented Does not simply act in response to the environment.
Temporally continuous Is a continually running process.
FBI

Figure 3.2: Agent properties

Simple agent definitions include:

- **Co-operative agents**. These agents communicate with each other, and react to their environment. An agent's view of its environment would thus be very narrow, as it has limited sensors. Co-operation between agents thus exists when the actions of an agent achieves not only the agent's goals, but also the goals of other agents. An example of this in a computer system is in the creation of a word processor which uses a spell checking agent, a grammar checking agent and a text layout agent. Each of the agents only senses their own environment and has a limited viewpoint of the overall system, but together they intercommunicate to enhance the goals of the complete system.
- **Reactive agents**. These agents, like humans, are event-driven, and do not possess internal models of their environment. Instead, they react to a stimulus or an input that is governed by an event within its environment. This environmental event triggers a reaction or response from the agent. In the human body, an example of this is in the immune system which sends out lymphocytes when the body is attacks by a foreign body.
- **Weak agents**. Wooldridge suggests that weak agents are autonomous, and thus have control over their own actions and states. They also have some form of social interaction, either with humans or other agents using an agent-communication language. These agents also react and respond to their environment through sensing and perceiving the changes in the physical world, and display active knowledge of their goal through their actions. A good example of this, in a computer network, is where agents are built into networking devices and communicate using the SNMP protocol. These agents work autonomously and gather information, which they can then report to other network agents, or to human operators. An SNMP agent will typically monitor network traffic, network faults, and so on.
- **Strong agents**. Maes' defines interface agents as '*computer programs that employ Artificial Intelligence to provide active assistance to a user with computer-based tasks.*' Maes' research utilized existing applications and connected them to a learning interface agent, which gradually built a knowledge base of what the human operator may do in certain situations. It is desirable that an agent should exhibit a learning potential to improve its decision-making methods.
- **Mobile agents**. Mobile Agents are software processes capable of moving around networks such as the Internet, interacting with other hosts, gathering information

> on behalf of their owner, and returning with any information that was requested by the owner. Mobility is another way that agents can be classified, that is being either a static agent or mobile agent. A static agent stays on the system it was installed on, whereas a mobile agent can move its location. Mobile agents thus have to be able stop their execution in order to move around a network, and only continue executing when they arrive at a host capable of re-starting code. The suspension of execution is a serious disadvantage of mobile agents. A static agent on the other hand has the ability to execute other tasks while it is communicating with other static agents. Its execution never has to stop. Mobile agents, unfortunately, suffer from many security problems, especially in authenticating themselves to a receiving device, and vice-versa. Once these problems have been overcome, mobile agents may provide the ultimate in distributed systems, where programs can actually migrate themselves to the place that the communication takes place, rather than communicating over the network. The research in this thesis is based on static agents, who do not move around the network, and communicate using traditional client-server architectures.

Naylor et al. [3.4] have argued that agents are the next natural step in the development of distributed systems. Their great advantage is that they are designed to run over distributed computing systems, whereas traditional utility programs typically run on a peer-to-peer type connection. They are particularly useful when working remotely from a server (especially when there is no current network connection), and for processing data that can be presented in a convenient form. Agent-based systems also create more robust, and distributed, software environments. This is because agents can be written so that they only have a single goal, and the code can be designed around that code. Most software is now written with multiple goals, and the larger the software becomes the more bugs it tends to have. Agent-based systems can be likened to embedded systems development, where software entities are carefully crafted in terms of processing requirements, memory usage, and so on.

A major problem with agent technologies is that there has previous been few defined standards for their implementation and intercommunication. This is now being overcome with the Foundation for Intelligent Physical Agents (FIPA) [3.7] which has approved certain standards for agent communication. Many agents that are now being created comply with the standards generated by the FIPA. This research has not implemented these standards, as the objective of the research is to investigate a novel distributed security mechanism. The agents developed could be easily modified so that they could communicate using the FIPA standards.

The requirement for agent standards and in applying them to real-life applications has been highlighted with the EU-sponsored FACTS project [3.55, 3.56]. FACTS involved commercial organisations such as BT, Alcatel, BELL and Nortel, and academic organisation such as Imperial College (UK) and the Insituto Tertino di Cultura (Italy).

3.3 Software agents and their use for security enhancement

There are several security systems which use agent technology as a tool for detecting abnormal behaviour in a network. They use different security issues and adapt to the dynamic networking environment, such as:

- **Intrusion Detection Inter-component Adaptive Negotiation** (IDIAN) [3.17]. This project developed a negotiation protocol to allow a distributed collection of heterogeneous IDs to inter-operate and reach agreement on each other's capabilities and needs. Moreover, the negotiation is dynamic so the information generated and processed can evolve as the Intrusion Detection System IDS evolves for environment changes.
- **Adaptive Intrusion Detection** (AID) system [3.18]. This is an ongoing at the Brandenburg University of Technology at Cottbus. The system is designed for network audit-based monitoring of local area networks and is used for to investigate networks and privacy-oriented auditing. AID has a client-server architecture consisting of a central monitoring station and several agents (servers) on the monitored hosts. The central station hosts a manager (client) and an expert system. The agents then take the audit data that is collected by the local audit functions and converts it into an *operating system independent* format. This then allows for the monitoring of a heterogeneous UNIX environment. Next, the audit data is transferred to the central monitoring station, and is buffered in a cache and then analysed by an Rtworks-based real-time expert system [3.57]. Finally, the system manager provides functions for the security administration of the monitored hosts, and controls audit functions. It also requests new audit data by controlled polling and returns the decisions of the agent's expert system. Secure RPC (Remote Procedure Call) allows for the communication between the manager and the agents. The expert system uses a knowledge base with state-oriented

attack signatures, which are modelled by *deterministic finite state machines* and implemented as rule sequences. The security officer, through a graphical user interface, can access relevant monitoring capabilities. In addition, the expert system archives data on completed and cancelled attacks involving users and creates security reports. The weakness of the system are that it is not totally distributed, as it still relies on a central point to make security decisions, and is passive as it only generates reports, and does not take immediate actions.

Other approaches include:

- **Theoretical approach**. Abdelaziz Mounji and Baudouin Le Charlier [3.19] have built a security model based on logic programming and formalism. Unfortunately, their model is too specific, and is difficult to reconfigure and upgrade. It is also not suitable for cross-platform implementations.
- **Hardware specific**. Some other approaches are hardware specific, such as the Cisco intrusion detection system [3.20]. The Cisco Secure IDS includes two components: sensor and director. Cisco Secure IDS sensors are high-speed network *appliances*, and analyse the content and context of individual data packets to determine if traffic is authorised. If an intrusion is detected, such as a SATAN (System Administrators Tool for Analysing Networks) attack, a ping sweep, or if an insider sends out a document containing a proprietary code word, IDS sensors detect the misuse in real-time and forward alarms to an IDS director management console for geographical display. The offender is then removed from the network. The main disadvantage of the system is that it requires all the routers and network devices to perform to a Cisco-derived specification.
- **Conservation of flow**. John R. Hughes et al. [3.21] have developed a monitor for routers to examine system behaviour. Specifically they use the law of Conservation of Flow which states that an input must either be absorbed, or sent on as an output (possibly with modification). This is an attractive tool to analyse network protocols for security properties, and it can detect disruptive network elements that launch Denial of Service (DOS) attacks, by absorbing or discarding packets. The system uses WATCHERS [3.58] which is a distributed network monitoring protocol designed to detect and isolate these malicious routers [3.59]. Its use requires several assumptions about the protocols being analysed. They examine the WATCHERS algorithm to detect misbehaving routers. The research has shown that without sufficient verification of its assumptions, the Conservation of Flow

principle can be defeated [3.60].

- **Database security**. Despite the necessity of protecting information stored in database systems (DBS), existing security models are insufficient to prevent misuse, especially insider abuse by legitimate users. Liu [3.61], for example, outlines how the database system could isolate attacks by rewriting SQL statements. Even though there are available means to guard the information stored in the database system against misuse, they are seldom used by security officers because security policies of the organisation are either imprecise or not known at all. Detection of Misuse in Database Systems (DEMIDS) [3.23] is a security misuse system tailored to relational database systems and uses audit logs to derive profiles that describe typical behaviour, in particular, insider abuse. The system is a specialised application to the general type of report creating systems, and generates reports according to a user's database behaviour.

Other IDS's generate reports that system administrators have to search through, and rely on the existing system software applications to gather the intrusion data, such as CyberTrace [3.22]. System administrator know that it is difficult, if not impossible, to tell what is going on the network, and can use tools such as tcpdump [3.62], and LANalyzer [3.63], to filter the raw data. Thus, an important objective of the research is to create a tool that cannot only collect network traffic, but also makes a judgement on possible intrusions. As CyberTrace identifies high priority connections with numbers (zero to 10), an administrator can review the highlights and even replay those sessions, and ignore connections with low priorities numbers (30 to 100).

3.4 Internet intrusion detection systems

The requirement for Internet Intrusion Detection systems grows as the networks, including the Internet, grows. Many vendors claim to have the ultimate solution, such as Computer Associates (CA) and McAfee Security. Unfortunately, as networks grow in size and complexity, protecting them from threats, such as from low-level protocol attacks, and server and desktop intrusion, it becomes more difficult. Viruses and malicious applets traversing the internal network pose other dangers. In addition, it is necessary to detect and block inappropriate network access to internal services and desktops, as well as outside URLs.

3.4.1 eTrust Intusion Detection

eTrust Intrusion Detection [3.24] detects patterns in network traffic that indicate potential intrusions. It can detect a denial-of-service attack, and take appropriate action based upon predefined policies before they have an impact on the network. This system significantly reduces training levels and the time required to manage and ensure a safe network. The overall security policy compliance is also easier as it gives detailed statistical reports on policy violations, and where they came from.

Distinctive Features of eTrust are:

- **Network access control**. This uses a rules based system to define the users that can access specific resources on the network, ensuring only authorised access to network resources.
- **Advanced anti-virus engine**. A virus-scanning engine detects and blocks network traffic containing computer viruses. It also protects users from innocuously downloading virus-infected files.
- **Comprehensive attack pattern library**. This automatically detects attack patterns from network traffic.
- **Packet sniffing**. This operates in stealth mode, and is undetectable to attackers.
- **URL blocking**. Administrators can designate URLs that users are not allowed to visit.
- **Word pattern scanning.** Administrators can define word patterns that may indicate policy violations, as this prevents sensitive data from being sent without authorisation through e-mail or the Web.

This intrusion detection system is build specifically for dealing with external intrusions, and specifically for intrusions from unknown hosts and intrusions to WWW and email servers.

3.4.2 GrIDS

GrIDS constructs graphs which represent hosts and activity in a network [3.25], such as the tracking of a worm. Worms are programs that propagate themselves across a network using resources of one machine to attack others [3.27]. For example, if a worm starts from host A and then initiates connections to hosts B and C and causes them to be infected. These two connections are reported to a GrIDS module, which creates a new graph representing this activity and records when it occurred. If enough time passes without further activity from hosts A, B, or C, the graph will be forgotten. However, if the worm spreads quickly to hosts D and E, then this new activity is added to the graph and the graph's time stamp is updated. By examining the pattern of generated graphs, the GrIDS system can determine if an attack has been made and generate a report.

3.4.3 Bro

Bro is a stand-alone system for detecting network intruders in real-time by passively monitoring a network link over which the intruder's traffic travels over [3.26]. It has high-speed monitoring, real-time notification, a clear separation between mechanism and policy, and can be extended. For this, Bro is divided into an *event engine* that reduces a network traffic stream into a series of higher-level events, and a *policy script interpreter* that interprets event handlers written in a specialised language used to express the site's security policy. These event handlers can update state information, synthesise new events, record information to disk, and generate real-time notifications through a system log.

3.4.4 AAFID - An Architecture for Intrusion Detection using Autonomous Agents

A recent approach to the intrusion detection problem is to use software agents, such as the AAFID environment [3.28]. As agents are independently running entities, they can be added and removed from a system without altering other components. There is thus no need to restart the IDS when there is a system change. The elements of the AAFID architecture are:

- **Agents**, **transceivers** and **monitors**. These can be distributed over any number of hosts in a network. Each host may contain any number of agents that monitor specific events occurring in the hosts. All the agents in the host report their findings to a single transceiver.
- **Transceivers**. These reside on the host and oversee the operation of all the agents running on their host. They exert control over the agents running on that host, and they have the ability to start, stop and send configuration commands to the agents. The transceivers report their results to one or more monitors, who oversee a number of transceivers.
- **Monitors**. These have access to network-wide data, and they thus are able to perform higher-level correlation and detect intrusions that involve several hosts. Monitors can be organised in a hierarchical manner so that a monitor may in turn report to another higher-level monitor. In addition, a transceiver may report to more than one monitor to provide redundancy and resistance to the failure of one of the monitors.

The agent does not have the authority to directly generate an alarm, as this is the function of either the transceiver or monitor. By combining reports from different agents, transceivers build the status of their host and monitors. This can then be used to determine the status of their network. The main disadvantages of this system are that the monitors are single points of failure, and, if multiple monitors are introduced, there is a problem with the consistency and duplication of information. Another drawback is that the architecture does not specify access control mechanisms to allow for different users to have different levels of access.

3.4.5 NetSTAT

A network-oriented system is NetSTAT [3.29]. Network-based intrusion detection is challenging as network auditing produces large amounts of data, and different events related to a single intrusion may be visible in different places on the network. NetSTAT is a new approach to network intrusion detection, as it uses a formal model of both the network and the attacks. From this, NetSTAT is able to determine which network events have to be monitored and where they can be monitored.

3.4.6 IDA

The Information-technology Promotion Agency (IPA) in Japan [3.41] developed a network intrusion detection system called Intrusion Detection Agent system (IDA). This employs mobile agents to avoid some of the problems experienced by conventional IDSs. IDA allows mobile agents to trace intruders, collecting information only related to the intrusion along the intrusion-route, and deciding whether, in fact, an intrusion has occurred. These functions enable efficient information retrieval, and make it possible to detect compromised intermediate hosts. The idea behind the system is that the agents do not **always** monitor the user behaviour, but they wait and see if the user generated some intrusion pattern. When this happens, the agent starts monitoring the user and tries to determine if an intrusion is in progress. The drawback of the system is that the system tried to have the minimum amount of overhead. This it will concentrate on general intrusions, and will thus must some of the less obvious ones.

3.4.7 Ludovic Me

Genetic algorithms (GAs) have been used to construct intrusion detection systems, such as Me, L [3.30]. This uses a security model that analyses audit log files using genetic algorithms. According to him, predefined intrusion detection can be viewed as a pattern-matching problem (that is, finding a regular expression with a back-referencing operator in a string). The NP-completeness of this problem makes classical algorithms quite impossible to apply to real audit logs. Thus, he decided to eliminate time from the attack scenarios. This means that attacks become sets of user actions rather than sequences of user actions. The resulting problem remains NP-complete, thus a heuristic had to be founded to solve the problem in a realistic execution time. His choice relies on genetic algorithms, which were initially proposed by John Holland in the 1970s. These are based on the mechanism of natural selection in a population of individuals. Each individual is evaluated through a mathematical function, which models that problem. The population is randomly generated and then evolves, and the best-evaluated individuals being favoured. Genetic algorithms allow for a quasi-optimal solution to a given problem by evaluating only a very small proportion of the completely possible solutions. The time that taken to extract the attack information of the audit log file is not affected by the size of the file, but only by the number of attacks. This approach to the intrusion detection problem just confirms the attack, but does nothing to prevent it. Genetic algorithms, though, could be useful in detecting attacks which do not follow a logical flow, and do not fit into the normal rules of attack.

3.4.8 Reactive Intrusion Detection (RID)

An active firewall is one method of securing a network against external intruders, where the firewall is able to collect attack data and reconfigure itself for future attacks. One example is RID from Gauntlet Firewalls [3.31]. The LURHQ Corporation have also developed a firewall-integrated anomaly, host and network based intrusion detection product. This intrusion detection tool designed to unobtrusively monitor, detect, and respond to suspicious activity in *real-tim*e. By integrating into the Gauntlet firewall, RID eliminates the need for a piecemeal approach to network security.

RID can be used in conjunction with one or more firewalls throughout a network and has comprehensive configuration capabilities. It uses an up-to-date attack signature database which identifies all **known** attacks, as well as the ability to identify *abusive behaviour*. RID can be customised with different reaction profiles for every type of attack or host. Unfortunately the system is mainly aimed at detecting external intruders.

3.4.9 Snort

Snort is a libpcap-based [3.32] packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS). It has rules-based logging for content pattern matching, and detects a variety of attacks and probes, such as buffer overflows [3.33], stealth port scans, CGI attacks, SMB probes, and much more. Snort also has real-time alerting capability, with alerts being sent to syslog, Server Message Block (SMB) *WinPopup* messages, or a separate *alert* file.

Snort fills an important *ecological niche* in the realm of network security [3.38]. Its main features are:

- It is a cross-platform, lightweight network intrusion detection tool that can be used to monitor small TCP/IP networks and detect a wide variety of suspicious network traffic, as well as outright attacks. A lightweight intrusion detection system can be easily used on most nodes on a network, with a minimal disruption to operations. Lightweight IDS's should be cross-platform, have a small system footprint, and be easily configured by system administrators who need to implement a specific security solution in a short amount of time. They can be any set of software tools, which can be assembled and put into action in response to evolving security situations. Lightweight IDS' are small, powerful, and flexible enough to be used as permanent elements of the network security infrastructure.
- It provides administrators with enough data to make informed decisions on the proper course of action in the face of suspicious activity.
- It can be used to quickly fill potential holes in a network's security coverage, such as when a new attack emerges and commercial security vendors are slow to release new attack recognition signatures.

Snort is configured using command line, which switches optional Berkeley Packet Filter [3.34] commands. The detection engine is programmed using a simple language that describes per packet tests and actions. Ease-of-use simplifies and expedites the development of new exploit detection rules. For example, when the IIS Showcode WWW exploits [3.35] were revealed on the Bugtraq mailing list [3.36], Snort rules detected the probes and signatures were available within a few hours. Snort shares commonalties with both sniffers and NIDS, and its main objective is to stop attack from outside the network, but not from the inside.

Snort has also been highlighted as suffering from too many alarms [3.66] and not being able to detect new types of attack [3.67], but some of these problems are now being overcome with AI techniques. Snort is now becoming popular in monitoring network traffic, and can even be implemented in hardware for gigabit Ethernet bit streams [3.68].

3.5 Visualisation of IDS's

Mathematical models are used to visualise intrusion, as visually presented information can encode large amounts of complex, interrelated data, that can be easily view by human operators [3.39]. The limitations of traditional IDS techniques are as much a function of the ability of a human to process large amounts of information as they are limitations of the techniques themselves.

A spicule is a spheroid geometric primitive, which is used as the basis for the visualisation model. One spicule represents the system model for one host on a network. The spicule's volume gives a measure of the security fitness of the host, where security fitness is a weighted sum of factors that add or detract from the vulnerability of the host. Computer systems with high security fitness will have smaller spicules than those of higher risk. Since the security fitness can be represented in this spherical fashion, it enables the usage of volumes and radii of a spicule as a mathematical property. Spicules also model other features of system activity, viewing them in terms of vectors. The values of some of these features, called tracking vectors can be normalized between 0 and 100%. An example of a tracking vector is CPU utilisation, which cannot exceed 100%. As a featured value grows, its associated tracking vector travels along a path on the surface of the spicule towards the vertical axis. Vectors, which cannot be normalised in this way, are called fixed vectors. These vectors are located at the equator of the spicule and can grow in magnitude coplanar to the horizontal axis. An example of a fixed vector is the number of child processes forked by a particular user. A signature can be obtained from both tracking and fixed vectors by tracing the path from a starting state to an end state. This signature can, in turn, be used as a mathematical property. The model is a mathematical characterisation of the types of attacks that can occur to a specific system.

3.6 Using the system administrator

Some intrusion detection systems rely on the system administrator in order to work correctly. En Garde Systems, Inc. T-sight [3.40] is a good example, as it is based on the principle of manual intrusion detection. Their IDS is highly configurable and the system administrators must know the system well so they can transport their knowledge of the system to the IDS. En Garde Systems believes manual intrusion detection is essential to comprehensive security, and knowledge and techniques in this field create the *backbone* of T-sight. It is felt that automated intrusion detection, which relies on flags generated by a static list of *signature* attacks, can give administrators a false sense of security. Although automated features are useful, they cannot keep up with the fact that security threats are constantly changing and may differ from company to company. Even if an organisation has firewall, it may not be completely safe, as a firewall cannot help with internal attacks. Consequently, a manual intrusion detection system is preferred as it is totally customisable. T-sight has a user-friendly, *out-of-the-box* interface that allows the administrator to adapt many functions to their needs, particularly on how data is viewed and what types of data are viewed. Since an administrator understands their network better than anyone does, it makes sense to allow them to be able to adjust the functionality of the tool so that it will be used on that network. This intrusion detection system is thus enhancing the monitoring utilities of the operating system and does not provide any prevention from the attacks.

3.7 Conclusions

In this chapter, we have discussed different techniques of software agent environments, and have analysed the different categories of software agents. We also have analysed different implementation of software agents, to enhance computer security. One major conclusion is that there is no definitive method which can be used to completely protect a network against intrusion.

In most cases, the implementation of the techniques has been achieved by installing security enhancement software on a centralized server. When this software crashes, or is breached, the complete network is at risk. To overcome this, a highly distributed system, such as using agents, can be used as these disperse network security management around the network. To address this, this research has built a security enhancement environment in which security management is dispersed across the network using software agents. In addition, even if the intrusion detection system is real-time, it can detect the intrusion after the action, but never before. We use a hybrid method, and thus get the best qualities of rules based systems and the statistical anomaly methods. In addition, to address the problem of detecting intrusions after they take place, we use a new statistical model based on Bayesian multivariate regression proposed by Pikoulas and Triantafyllopoulos [3.12], which takes into account user behaviour and generates a predicted profile, so that the intrusion system has sufficient information to foresee the future user actions.

3.8 References

[3.1] Jennings N and Woolridge M, ‘Agent Technology: Foundations, Applications and Markets’, Springer. 1998.
[3.2] Bradshaw J, ‘Software Agents’, MIT Press, 1997.
[3.3] White J, ‘Telescript Techology: Mobile Agents’, Software Agents, MIT Press, 1997
[3.4] Buchanan WJ, Naylor M, Scott AV, ‘Enhancing network management using mobile agents’, Proceedings Seventh IEEE International Conference and Workshop

on the Engineering of Computer Based Systems (ECBS 2000). IEEE Comput. Soc. 2000, pp.218–226.

[3.5] Harrison C, Chess D and Kershenhaum, 'Mobile Agents: Are they a good idea?'. Technical Report RC 19887, IBM TJ Watson Research Center, 1995.

[3.6] Lange D, Oshima M., 'Seven Good Reasons for Mobile Agents', Communications of the ACM, 42(3): 88–89, Mar 1999.

[3.7] Foundation for Intelligent Agents Website. w://www.fipa.org.

[3.8] http: //www.omikron.de/~ecr/ nthack/ samfaq.htm.

[3.9] Chris Herringshaw, Detecting Attacks on Networks, IEEE Computer Magazine, pp 16–17, Dec. 1997.

[3.10] Debra Anderson, 'Detecting Unusual Program Behavior Using the NIDES Statistical Component', IDS Report SRI Project 2596, Contract Number 910097C (Trusted Information Systems) under F30602–91–C–0067 (Rome Labs), 1995.

[3.11] T. Lunt, H. Javitz, A. Valdes, et al., 'A Real–Time Intrusion Detection Expert System' (IDES), SRI Project 6784, Feb. 1992. SRI International Technical Report.

[3.12] J Pikoulas and K Triantafyllopoulos, 'Bayesian Multivariate Regression for Predicting User Behaviour in a Software Agent Computer Security System', 20th International Symposium on Forecasting, Lisbon, Portugal, June 21, 2000.

[3.13] Sandeep Kumar and Gene Spafford, 'A Pattern Matching model for Misuse Intrusion Detection', Proceedings of the 17th National Computer Security Conference, Oct. 1994.

[3.14] Mark Crosbie and Gene Spafford, 'Active Defence of a Computer System using Autonomous Agents', COAST Group, Dept. of Computer Science, Prudue University, Technical Report (95–008),2–3, Feb 1995.

[3.15] The Computer Misuse Detection System, http://www.cmds.net/, 1998.

[3.16] M. Wooldrige and N. Jennings, Intelligent Agents: Theory and Practice, 1995.

[3.17] Richard Feiertag, Lee Benzinger, Sue Rho, Stephen Wu, Karl Levitt, Dave Peticolas, Mark Heckman, Stuart–Staniford–Chen and Cui Zhang, 'Intrusion Detection Inter–component Adaptive Negotiation', Report F30602 – 97 – C – 0187, DARPA, Sep. 29 1999.

[3.18] Brandenburg University of Technology at Cottbus, Adaptive Intrusion Detection system, 1994 to Spring 1996.

[3.19] Abdelaziz Mounji and Baudouin Le Charlier, 'Detecting Breaches in Computer Security: A Pragmatic System with a Logic Programming Flavor', Institut d'Informarique, Aug., 1996.

[3.20] Cisco, Cisco Secure Intrusion Detection System Sensor, February 2000.

[3.21] John R. Hughes, Tuomas Aura and Matt Bishop, 'Using Conservation of Flow as a Security Mechanism in Network Protocols', Report NAG21251 for National Aeronautical and Space Administration, 1999.

[3.22] Cyber Trace, CyberTrace – Intrusion Detection, September 1999.

[3.23] Christina Yip Chung, Michael Gertz and Karl Levitt, ‘DEMIDS: A Misuse Detection System for Database Systems’, Department of Computer Science, University of California at Davis, 1997.

[3.24] Computer Associates, eTrust Intrusion Detection, http://www.cai.com, 2000.

[3.25] Steven Cheung, Rick Crawford, Mark Dilger, Jeremy Frank, Jim Hoagland, Karl Levitt, Jeff Rowe, Stuart Staniford–Chen, Raymond Yip and Dan Zerkle, ‘The Design of GrIDS: A Graph–Based Intrusion Detection System’, Jan. 1999.

[3.26] Vern Paxson, ‘Bro: A System for Detecting Network Intruders in Real Time’, Network Research Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, January 1998.

[3.27] D. Seely, ‘A tour of the worm’, IEEE trans on Software Engineering, Nov. 1991.

[3.28] Jai Sundar Balasubramaniyan, Jose Omar Garcia–Fernandez, David Isacoff, Eugene Spafford and Diego Zamboni, ‘An Architecture for Intrusion Detection using Autonomous Agents’, COAST Laboratory, Purdue University West Lafayette, IN 47907–1398, Jun. 1998.

[3.29] Giovanni Vigna and Richard A. Kemmerer, ‘NetSTAT: A Network–based Intrusion Detection Approach’, Reliable Software Group, Department of Computer Science, University of California.

[3.30] Ludovic Me, ‘Genetic Algorithms, a Biologically Inspired Approach for Security Audit Trails Analysis’, May 1996.

[3.31] LURHQ Corporation, Reactive Intrusion Detection 2.0 for Gauntlet Firewalls, Conway, SC 29526.

[3.32] Van Jacobson, Craig Leres and Steven McCanne, Lawrence Berkeley National Laboratory, 1994, http://www–nrg.ee.lbl.gov/

[3.33] Smashing the Stack for Fun and Profit, Aleph1, Phrack #49, 1996, http://www.phrack.co.

[3.34] Steven McCanne, Van Jacobson, The BSD Packet Filter: A New Architecture for User level Packet Capture, USENIX Technical Conference Proceedings, 1993.

[3.35] NT ISS Showcode ASP Vulnerability (Bugtraq ID #167), http://www.securityfocus. com,Parcens/L0pht, May 1999.

[3.36] Bugtraq Mailing List, archives and vulnerability database are available at Security Focus,

[3.37] http://www.securityfocus .com

[3.38] Martin Roesch, Snort – Lightweight Intrusion Detection for Networks, Stanford Telecommunications Inc., Nov. 1999.

[3.39] Greg Vert, Deborah A. Frincke and Jesse C. McConnell, ‘A Visual Mathematical Model for Intrusion Detection, Center for Secure and Dependable Software’, University of Idaho, 1998.

[3.40] En Garde Systems, Inc., T–sight, Nov. 1998.

[3.41] Modori Asaka, Shunji Okasawa and Atsushi Taguchi, A Method of Tracing Intruders by Using Mobile Agents, Bunkyo Green Court Center Office, 1999.

[3.42] Adair R, Jewell S, Computer Fraud: Is there cause for concern?, Public Finance and Accountant, Vol. 12, No. 2, April 1986, pp. 27–29.

[3.43] Willison R, Reducing computer fraud through situational crime prevention, World Conference on Information Security, Kluwer, 2000, pp. 99–109.

[3.44] Heatley SK, Otto JR, Data mining computer audit logs to detect computer misuse, International Journal of Intelligent Systems in Accounting, Finance and Management, No. 3, Sept. 1998, pp. 125–134.

[3.45] Matt Bishop, Computer Incident Advisory Capability, Authentication tools,, http://ciac.llnl.gov/ciac/ToolsUnixAuth.html

[3.46] John F. Haugh, Matt Bishop, Computer Incident Advisory Capability, Authentication tools, http://ciac.llnl.gov/ciac/ToolsUnixAuth.html

[3.47] B. Clifford Neuman and Theodore Ts'o. Kerberos: 'An Authentication Service for Computer Networks', IEEE Communications, 32(9):33–38. September 1994

[3.48] John T. Kohl, B. Clifford Neuman, and Theodore Y. T'so, 'The Evolution of the Kerberos Authentication System', In Distributed Open Systems, pages 78–94. IEEE Computer Society Press, 1994

[3.49] Ronald L. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc., 'The MD5 Message–Digest Algorithm',Internet RFC 1321 (April 1992).

[3.50] http://www.geocities.com/ResearchTriangle/7003/fwtk.htm

[3.51] PSIONIC Technologies, Psionic PortSentry, http://www.psionic.com/products/portsentry.html.

[3.52] Dan farmer, 'COPS Overview', http://www.fish.com/cops/overview.html

[3.53] Charles Kolodgy, Roseann Day,Christian A. Christiansen, and John Daly, 'Technology to Invoke Trust in IT — The Tripwire Solution', IDC,5 Speen Street, Framingham, MA 01701.

[3.54] SecurityFocus online, Arpwatch, http://online.securityfocus.com/tools/142.

[3.55] Nurez–Suarez J. O'Sullivan D. Brouchoud H. Cros P. Moore C. Byrne C. ‘Experiences in the use of FIPA agent technologies for the development of a personal travel application’, Proceedings of the Fourth International Conference on Autonomous Agents. ACM. 2000, pp.357–64. New York, NY, USA.

[3.56] Suarez J. O'Sullivan D. Brouchoud H. Cros P. Moore C. Byrne C., Personal Travel Market: I'll book your trip for you sir,Cybernetics and Systems 2000. Proceedings of the Fifteenth European Meeting on Cybernetics and Systems Research. Austrian Soc. Cybernetic Studies. Part vol.2, 2000, pp.663–8 vol.2. Vienna, Austria.

[3.57] Arzen K–E., ‘A survey of commercial real–time expert system environments’, Artificial Intelligence in Real–Time Control 1992. Selected Papers from the IFAC/IFIP/IMACS Symposium. Pergamon. 1993, pp.483–90. Oxford, UK.

[3.58] Shipley G., ‘Watching the watchers: intrusion detection’, Network Computing, vol.11, no.22, 13 Nov. 2000, pp.135–6, 138, 140, 143–4, 146.

[3.59] Bradley KA, Cheung S, Puketza N, Mukherjee B, Olsson RA, ‘Detecting disruptive routers: a distributed network monitoring approach’, Proceedings. 1998 IEEE Symposium on Security and Privacy. IEEE Comput. Soc. 1998, pp.115–24.

[3.60] Hughes JR, Aura T, Bishop M, ‘Using conservation of flow as a security mechanism in network protocols’, Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE Comput. Soc. 2000, pp.132–41.

[3.61] Peng Liu, ‘DAIS: a real–time data attack isolation system for commercial database applications’, Proceedings 17th Annual Computer Security Applications Conference. IEEE Comput. Soc. 2001, pp.219–29.

[3.62] TCPDUMP, http://www.tcpdump.org/.

[3.63] LANanalyzer, Novell Systems, http://support.novell.com/servlet/tidfinder/2908065.

[3.64] Manikopoulos C, Papavassiliou S, ‘Network intrusion and fault detection: a statistical anomaly approach’, IEEE Communications Magazine, vol.40, no.10, Oct. 2002, pp.76–82. Publisher: IEEE, USA.

[3.65] Jha S, Hassan M. Building agents for rule–based intrusion detection system, Computer Communications, vol.25, no.15, 15 Sept. 2002, pp.1366–73. Publisher: Elsevier, Netherlands.

[3.66] Roesch M., ‘Snort – lightweight intrusion detection for networks’, 13th Systems Adminstration Conference (LISA '99). USENIX Assoc. 1999, pp.229–38. Berkeley, CA, USA.

[3.67] Schwartz DG, Stoecklin S, Yilmaz E., ‘A case–based approach to network intrusion detection’, Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002. Int. Soc. Inf. Fusion. Part vol.2, 2002, pp.1084–9 vol.2. Sunnyvale, CA, USA.

[3.68] Gokhale M, Dubois D, Dubois A, Boorman M, Poole S, Hogsett V., ‘Granidt: towards gigabit rate network intrusion detection technology’, Field–Programmable Logic and Applications. 12th International Conference, FPL 2002. Proceedings (Lecture Notes in Computer Science Vol.2438). Springer-Verlag. 2002, pp.404–13. Berlin, Germany.

4 Prediction Techniques

4.1 Introduction

This chapter describes the basic principles involved in the prediction model used in the research. It also introduces different prediction techniques, especially on statistical forecasting, and the differing techniques used in statistical forecasting using Bayesian statistics. Security systems of the future must take into account prediction, as new security breaches can result from unknown methods of attack. A good example of this is with the Love Bug virus, which spread from country to country as fast as the sun rose across world. Few systems detected it, as these systems did not have strong prediction techniques.

Whenever a new type of attack occurs the security systems can be quickly updated with new rules, and updated software patches, but, in many cases, these can occur too late to avoid some loss of service, or data. It can also be embarrassing for organizations if they have been seen as the focus of a security breach. Thus, it is important that prediction techniques are used on security systems. These can never run without producing false-positives, but their sensitivity can be fine-tuned by the system administrator. If too many false-positives are produced, they can swamp true-positives, thus good forecasting can aid the process of fine-tuning the intrusion detection process.

In statistics, we have two major forecasting techniques that are used: ARIMA models, and their variants, and Bayesian models. ARIMA models were first proposed by Box and Jenkins [4.4], and are widely used for short- and long-term prediction, especially in financial markets. They are complex models in that they have many variants, which must be set correctly before they give some accurate results. Bayesian models are first proposed by Rev. Thomas Bayes in 1763 [4.2], and they are now gaining growing support in the statistical community, as they are easier to configure and they give better results for short-term prediction. Both models are used in linear forecasting problems.

Bayesian methods were popular until the 1930s when they were seen as **subjective** and thus unscientific (as science often defines that experiments are conducted in an objective way). It fell out of favour for a while, but it now back in favour, and new applications of it include in medicine, such as in the investigation of the claims of a clot-reducing drug, and in the legal profession, in analysing the probability of the occurrence of DNA evidence in jury trails.

4.2 Forecasting methods

This section gives an brief introduction to forecasting and forecasting methods. The most widely used forecasting methods, especially for short analysis forecasting [4.1], are:

- **Multiple regression analysis**. This is used when two or more independent factors are involved and is widely used for intermediate-term forecasting. It is used to assess which factors to include and which to exclude, and can be used to develop alternate models with different factors. For example, in a security system, important factors are likely to be the applications and resources that a user uses, and how they typically operate a computer, rather than the sizes of the files that a user creates. It has been applied in many areas of forecasting weather patterns, such as with Kung et al. [4.9] who used it to predict the Indian monsoon. It has also been applied to others areas of science and technology, such as Yamada [4.6] who used it to estimating the peak loading of water demand and electrical power, whereas Gardener [4.7] has applied it to the forecasting of blood supplies.
- **Non-linear regression**. This method does not assume a linear relationship between variables and is frequently used when time is the independent variable. Renxiang et al. [4.8] has applied it to forecasting passengers on transport systems. They select dependent variables using statistical factors, rather than using subject judgements, and conclude that non-linear regression methods, in this application, are more effective than linear regression models.
- **Trend Analysis**. This method uses linear and non-linear regression with time as the explanatory variable used where the pattern varies over time. Menge [4.10] used trend analysis to predict short-range demand for electricity. The need for accuracy in forecasting demand in electrical supply, as providers must be able to foresee demand before they occur.
- **Decomposition analysis**. This is used to identify several patterns that appear simultaneously in a time series. It is time-consuming each time it is used, and is typically used to de-seasonalise a series. Huth [4.10] compared decomposition analysis to changes of temperature in Central Europe to other methods, such as multiple-linear regression methods, and concluded that it was a reasonable method for predicting long-term patterns.
- **Moving average analysis**. This is based on moving averages, which forecasts are based on a weighted average of past values. It has the advantage of being simple to implement. Smunt [4.11] applied moving average analysis to project planning,

and concluded that it can give better estimates for short-term costs than learning-curve models, and standard analysis methods.

- **Weighted moving averages**. This is a powerful and economical method, and uses methods like sum-of-the-digits and trend adjustment methods. It is widely used where repeated forecasts required. Many researchers have applied it to there work, especially in time series applications [4.13-4.15], and in networking algorithms, such as with Edge [4.16] who applied it to a timeout algorithm in packet switched networks.
- **Adaptive filtering**. This is another type of moving average, and includes a method of learning from past errors. It is typically used to respond to changes in the relative importance of trend, seasonal, and random factors [4.17, 4.18]
- **Exponential smoothing**. This is a moving average form of time-series forecasting. It is efficient with seasonal patterns, and easy to adjust for past errors. It is also easy to prepare follow-on forecasts, and where several different forms are used depending on presence of trend or cyclical variations. Synder [4.19] and Gardner [4.20] applied it to the seasonal forecasting of inventories, and Fei et al. [4.21] applied it to traffic forecasting. In networking, Brutlag [4.22] used exponential smoothing to predict aberrant behaviour over time. They proposed a model which allowed simultaneous, real-time monitoring of users over the network. The model proved to be fast and effective, but suffered from not being optimal.
- **Hodrick-Prescott filter**. This is a smoothing mechanism which is used to obtain long-term trends in a time series. It decomposes a given series into stationary and non-stationary components so that their sum of squares of the series from the non-stationary component is a minimum, with a penalty on changes to the derivatives of this component. It has been applied to several economic applications [4.23].

4.2.1 Forecasting by a regression analysis

Regression is the study of relationships among variables. Its main purpose is to predict, or estimate, the value of one variable from known or assumed values of other variables which relate to it. It includes:

- **Variables of interest**. To make predictions or estimates we must identify the effective predictors of the variable of interest. These are variables which are important indicators and can be measured at the least cost, which only carry a small amount of information, and are thus redundant.

- **Predicting the future**. Predicting a change over time or extrapolating from present conditions to future conditions is not the function of regression analysis. For this, time series analysis typically used.
- **Experiment**. This begins with a hypothesis about how several variables might be related and the form of their relationship.

Types of analysis include:

- **Simple linear regression**. This is regression using only one predictor and is called a simple regression.
- **Multiple regression**. This is where there are two or more predictors are used and multiple regression analysis must be used. Complex systems such as weather forecasting, which depends on variables such as temperatures, pressures, air movements, sea changes, and so on, use multiple regression methods.

The following are general steps on how to design, construct, feed data and get the results of a regression model [4.1]:

- **Data**. Since it is normally unrealistic to obtain information on an entire population, a sample is taken, which is a subset of the population. The sample may be either randomly selected or the x-values (the independent variable) are based on the capability of the equipment to be utilised in the experiment. Where the x-values are preselected, usually only limited inferences can be drawn depending upon the particular values chosen. When both x and y are randomly drawn, inferences can generally be drawn over the range of values in the sample.
- **Scatter diagram**. This is a graphical representation of the pairs of data, and can be drawn to gain an overall view of the problem. This makes it easier to determine an apparent relationship. If the points lie within a band described by parallel lines, then there is a linear relationship between the pair of x and y values. If the rate of change is generally not constant, then the relationship is curvilinear.
- **The model**. If we have determined there is a linear relationship between t and y we require a linear equation stating that y as a function of x in the form $Y = a + bt + e$, where a is the intercept, b is the slope. The error term (e) is the error term accounting for variables that affect y, but are not included as predictors, and/or otherwise unpredictable and uncontrollable factors.
- **Least squares method**. To predict the mean y-value for a given t-value, we need a

line which passes through the mean value of both *t* and *y* and which minimises the sum of the distance between each of the points and the predictive line. Such an approach should result in a line, which is often called the *best fit* to the sample data. The least squares method achieves this result by calculating the minimum average squared deviations between the sample *y* points and the estimated line. A procedure is used for finding the values of *a* and *b* which reduces to the solution of simultaneous linear equations. Shortcut formulas have been developed as an alternative to the solution of simultaneous equations.

- **Solution methods**. Techniques of matrix algebra can be manually employed to solve simultaneous linear equations. When performing manual computations, this technique is especially useful when there are more than two equations in two unknowns. Several well-known computer packages are widely available and can be utilised to relieve the user of the computational problem, all of which can be used to solve both linear and polynomial equations: the BMD (Biomedical Computer Programs) [4.24] from UCLA; SPSS (Statistical Package for the Social Sciences) [4.25] developed by the University of Chicago; and SAS (Statistical Analysis System) [4.26].
- **Use and interpretation of the regression equation**. The equation developed can be used to predict an average value over the range of the sample data. The forecast is good for short to medium ranges.
- **Measuring error in estimations**. The scatter or variability around the mean value can be measured by calculating the variance, which is the average squared deviation of the values around the mean. The standard error of estimate is derived from this value by taking the square root. This value is interpreted as the average amount that actual values differ from the estimated mean.
- **Confidence intervals**. Interval estimates can be calculated to obtain a measure of the confidence that a relationship exists. These calculations are made using t-distribution tables. From these calculations, it is possible to derive confidence bands, a pair of non-parallel lines, of which the narrowest at the mean values express the confidence in varying degrees of the band of values surrounding the regression equation.
- **Assessment**. This determines how confidant that a relationship actually exist. The strength of this relationship can be assessed by statistical tests of that hypothesis such as the null hypothesis, which are established using t-distribution, R-squared, and F-distribution tables. These calculations generate a standard error of the re-

gression coefficient, an estimate of the amount that the regression coefficient *b* will vary from sample to sample of the same size for the same population. An analysis of variance (ANOVA) table can be generated which summarises the different components of variation. When comparing models of differing size (different numbers of independent variables and/or different sample sizes), it must use the Adjusted R-Squared, because the usual R-Squared tends to grow with the number of independent variables. The standard error of estimate (that is, square root of error mean square) is a good indicator of the *quality* of a prediction model as it *adjusts* the error sum of squares (EMS) for the number of predictors in the model as follow:

$$EMS = \frac{\text{Error sum of squares}}{N - \text{Number of linearly independent predictors}} \tag{4.1}$$

If one keeps adding useless predictors to a model, the EMS becomes less stable. R-squared is also influenced by the range of your dependent value so if two models have the same residual mean square, but one model has a much narrower range of values for the dependent variable, that model will have a higher R-squared. This explains the fact that both models will do as well for prediction purposes. A considerable portion of the output of the computer programs previously mentioned are devoted to a description of the tests of significance of the regression.

The following sections, discuss Bayesian statistics and Bayesian models.

4.3 Bayesian statistics

Rev. Thomas Bayes [4.2], an English clergyman, first proposed that data could be converted into probabilities. He thus established a mathematical basis for probability inference (a means of calculating, from the number of times an event has not occurred, the probability that it will occur in future trials). He set down his findings on probability in the *Essay Towards Solving a Problem in the Doctrine of Chances* (1763) [4.27], which was published in the *Philosophical Transactions of the Royal Society of London*.

Bayes competes against **frequentism** methods, which determine the probabilities of the frequency of an event over a long period. It would thus judge that a coin would have a 50% chance of landing heads, and a 50% chance of landing tails. For this we must this take a large enough population, and draw definite theories on the probability of an event. Short-term and medium-term results do not necessarily go as the probability defines. For example, in the short-term it is unlikely that we would get the same number of heads as tails for a coin flip. Thus, how do we determine that a coin is operating as it should? For example if we tip as coin 100 times, and determine that it lands on head for 60 times, and tails for 40 times. In Bayesian, we can actually determine the chances of getting heads based on the evidence of the past, and can thus take into account any possibilities that a coin has been tampered with.

Key terms which are important in analysing Bayes theorem are:

- **Prior probability**. This is the probability of a theory being correct, prior to taking into account new evidence. For example, if the chances of a student successfully completing a course of PhD studies is 30%, then before a students starts their studies the chance of them successfully completing their studies is 3 in 10.
- **Posterior probability**. This is the updated probabilities of a theory being true, after new data has been taken account of. For example, with PhD studies, a new survey could take place, which gave up-to-date statistics for the completions rates. This might change the probability of successful completions.
- **Likelihood ratio**. This is a measure of how strong the evidence is. The more probable it is, the higher the factor will be. This is obviously important in areas such as in DNA legal evidence, where an expert witness must define the chance that DNA evidence could be from another person. Bayesian methods have been applied to this area, and can produce likelihoods ratios of one in 100,000, or more.

Bayesian methods are now becoming popular over frequentism methods, as they can cope with large amounts of data, from which it is possible to draw strong conclusions, that define how well defined the conclusion is. Medical studies have applied this to disprove claims that a clot-reducing drug reduced heart attacks by 50%. It is also applied to areas such as defining the chances of weather predictions being true. It thus focuses on converting data into probabilities. In network security, it would determine the probability that users would operate in a certain way, based on the history, and not on some defined profile, that models the user. Many, at the time it was defined, and in future years, disapproved of the method. It general it basically defines the:

- **Likelihood**. It measures the strength of evidence and compares the relative likelihood of getting such evident if the theory was true, and if it is false. The larger the likelihood ratio, the better for making conclusions.
- **Prior probability**. This defines that the probability of the theorem for the data is

true, before collecting the data. This goes against our basic theory of life where things must be seen as false, before we get evidence to show that it is true. Imagine if a jury in a trail were to assume that a person was guilty before they were tried, and the defence would have to produce evidence to show that their client has not guilty. Also in the case of a person flipping a coin, the question we might initially ask is 'Has the coin be tampered with?'. Thus, we would assume, initially that the coin had been tampered with, and then the evidence of coin flips would show us if the coin had really been tampered with. Each new flip would vary our likelihood factor, so eventually we could say that there is a 1 in 100,000 chance that the coin had been tampered with. This gives humans an opportunity to understand the odds of an event, and make decisions.

In security, it could be defined that prior probability is that all users want to do damage against a system, and user behaviour data must be used to overcome this theory. This is actually a good viewpoint in a highly secure environment. Bayesian also has the advantage that it will base it judgements on posterior data, and not on a standard model of the user. In most cases, in a frequentism method, the user would be viewed as a non-intruder, and not of an intruder. Bayesian takes an opposite viewpoint, and will initially define that the theorem is initially true (a probability of 1), or false (a probability of 0). Bayes initially produced this theorem, but at the time could not produce the mathematics to solve it. Laplace (1771) eventually used Bayes' ideas, and used it to investigate the claim that there were more boys born in Paris, rather than girls.

A problem with Bayesian is obviously the initial viewpoint. With a new drug, the manufacturing company could define that the initial viewpoint is that it will be successful, whereas scientists investigating its success will define the opposite viewpoint. The same can be said for network security. From a users point-of-view the initial viewpoint would be that all users were non-intruders and that intruders would then be identified, while a network manager might have the opposite point-of-view, where all users are potential intruders, and they must show by their behaviour that they are not intruders. In the 1920s, researchers tried to overcome this problem by developing **objective** ways to test theories, which did not have a prior probability. These were not successful, and frequentism methods become popular, as Bayesian methods were seen as unscientific and **subjective**. The frequentism methods then become standard techniques, where p-values were used to check the statistical significance. Often a p-value of between than 1 and 20 identified that the result had a significant conclusion. Unfortunately, many have argued that this p-value does not actually improve the weighting of the conclusions over the Bayesian approach. Luckily, the 1980s saw an increase in computing power, which allowed Bayesian methods to be properly implemented in an efficient way. There was also a large increase in the amount of data generated, which, as Bayesian makes improved decisions the more data that it has, has made Bayesian an effective way to make decisions on. The applicability of Bayesian was initially highlighted by research, in the 1990s, into the benefits of a clot reducing drug. The drug company initially quoted that it had, using frequentism methods, a success rate of 50%. Several researchers, including Spiegelhalter at the London School of Hygiene, showed that, using Bayesian methods, the actually rate was less than this. No one could initially tell if these results were correct, but now after a decade after the research, it has been shown that the success rate was about a half the value quoted. Thus, Bayesian has better predicted this result. The Bayesian approach of viewing the drug as not being successful is a good one, as the drug must prove itself before it can be validated.

In DNA evidence, Bayesian has also changed the way that the legal professional looks at DNA evidence. Before the Bayesian method was applied to this evidence, expert witnesses regularly told that there were certain odds, such as 1 in 100 000, that DNA matches in the population. This is the probability of a person having the same DNA as another person. Bayesian argues that this is not true, and that all the evidence involved in the case should be taken into account. Frequencism methods would take the complete population, and determine the probability of these occurrences. Bayesian would argue that we should start with the assumption that it will match, and then gather data from around the evidence. This may show that there are localised changes which increases or decreases the probability of a conclusion. Thus, in network security, we can take into account localised factors. In fact, it can take into account individual users, rather than averaging users over a complete population.

Bayes and empirical Bayes (EB) [4.3] methods combine information from similar components of information and produce efficient inferences for both individual components and for shared model characteristics. Many complex problems are ideal settings for this type of synthesis. For example, county-specific disease incidence rates can be unstable due to small populations or low rates. *Borrowing information* from adjacent counties by partial pooling produces better estimates for each county, and Bayes/empirical Bayes methods structure the approach. Importantly, recent advances in computing and the consequential ability to evaluate complex models have increased the popularity and applicability of Bayesian methods. Bayes and EB methods can be implemented using modern Markov chain Monte Carlo (MCMC) computational methods. Properly structured Bayes and EB procedures typically have good frequentist and Bayesian performance, both in theory and in practice. This, in turn, motivates their use in advanced high-dimensional model settings (for example in longitudinal data or spatio-temporal mapping models), where a Bayesian model implemented with MCMC often provides the only feasible approach that incorporates all relevant model features.

4.3.1 Prediction in Detail

A time series is a collection of data throughout time, and are called observations. Normally they are collected at certain time intervals, such as every second, every minute, every day, every month, or every year. At each time *t*, we may have either one observation (univariate time series) or several observations (multivariate time series). This depends on what we want to model and predict. For example, if we want to produce predictions for an index, such as a financial index or the sales of a company, we can use univariate time series. However, if we want to predict intruder behaviour within a complete network we need to consider multivariate time series. The dependence structure is of particular interest, such as are the intruders act independently, and if they do not, can we derive information about one intruders's action from another? These questions will be answered in the following chapters.

Time series models have been used successively from the 1950s and on in a wide variety of real-life problems, especially with Box and Jenkins [4.4] and Kalman [4.5]. Both these researchers dominated the time series world, mainly because:

- The Box and Jenkins approach has an integrated implementation as well as the design part, in the 1980s. They have applied this to several areas, such as in the forecasting the demand for blood [4.7].
- The Kalman filtering approach has a unique representation power through the state-space models that makes it relatively easy to deal with, and thus reduces a complicated modelling to s relatively simple one.

Box and Jenkins models are usually referred to as ARIMA (Autoregressive Integrated Moving Average) models. In practice, these are based on the assumption that the series of interest is stationary, or it can be reduced to a stationary one using a transformation series. By stationary, it is defined that the mean and the variance of the series is constant over time and correlation over the series does not depend on time. This shows that the level of the series is constant throughout time, and that the variation of the series is always constant or approximately constant. These models require certain distributional assumptions that we avoid discussing here. The important thing to note is that an implicit assumption is that all the future forecasts depend on some kind of averages of the past observations. In certain cases, this is a valid assumption, however not always. Unfortunately, there is rarely justification for these assumptions. The ARIMA modeller is thus *hoping* that the forecasts will follow this simple rule and the only judgement on this is by comparing the predictions with the real data. However, care must be drawn here, especially for systems that change rapidly.

Let Y_t represent a single variable of interest that may be called the response at time *t*. The index *t* is a subset of the integer numbers values: 1, 2, 3,…, T. Then the simplest ARIMA model, abbreviated as AR(1), is defined by:

$$Y_t = aY_{t-1} + \varepsilon_t, \varepsilon_t \sim N[0,V], \tag{4.2}$$

where:

a	is a constant
ε_t	has a normal distribution with mean 0 and variance *V*. The normal distribution means that ε_t will be approximately zero and the uncertainty about this will be measured with the variance *V*.
V	is an indicator of how far from 0, ε_t is located. This, in practice, means that Y_t will be close to aY_{t-1}.

Now suppose that at time *t* we know the values of a, Y_{t-1} and that we want to predict the value of Y_t. ARIMA forecasting suggests that Y_t is predicted by aY_{t-1}. Since a, Y_{t-1} are known we can perform this prediction and go on to the next time *t*. When Y_t becomes available we can predict the value of Y_{t+1}. Using the ARIMA methodology, a (the parameter of the model) will be estimated by the past data at time t, namely $Y_1, Y_2, \dots, Y_{t-1}$.

This simple model reflects the underlined ARIMA methodology. The response Y_t will be a sum of previous values multiplied by constants and a sum of some other variables (called the moving average noises) multiplied by constants. If the initial series is not stationary, it must be reduced to a stationary one and then the methodology can by applied. This method is called as differentiation and it simply suggests instead of working with Y_t, considering the differentiated series $Z_t = Y_t - Y_{t-1}$. So, in this case, for all the past data we calculate Z_t's, hoping that they are stationary. If they are not stationary we have to take higher-order differences, such as $Z_t = Y_t - Y_{t-2}$. In addition, we go on like this until we get a stationary differenced series.

Box and Jenkins claim that any non-stationary time series can be reduced to a stationary one using the above methodology, but this is driven only from certain examples and there is no formal justification. When abrupt changes occur, the methodology gives poor results. Merely, a *bad* observation known as outlier, can dominate the forecasting method yielding very poor results. One possible action for improvement would be to consider time-varying coefficients a_t replacing the a's. However, ARIMA modelling is not able to handle such an approach. There is also another major difficulty that discourages the use of the Box and Jenkins approach. This is when multivariate modelling requires differencing that is restricted.

For example, consider a vector series Y_t that comprises of several scalar variables. Then by using $Z_t = Y_t - Y_{t-1}$ some of the scalar series (elements of Y_t) may be stationary, although some other non-stationary. If higher differences are applied the elements of Y_t that became stationary with $Z_t = Y_t - Y_{t-1}$, it is very possible to become non-stationary. It is thus if all the different elements of Y_t then the independent ARIMA modelling can be applied, but in doing so there is no gain using multivariate modelling. The modeller can thus consider several univariate models. In practice, such independent assumptions will never be justified and usually will be unrealistic. For these reasons, ARIMA modelling is very restricting and problem specific.

The Kalman filtering approach of time series modelling is based on the state space representation of a time series. This states that the formulation of the model is not explicitly given for all *t*, but only relevant to the previous time *t*–1. Y_t, either univariate or multivariate, is related through a number of known values (not necessarily constant over time) with several states. These states are random variables that are responsible for prediction and that are allowed to have an evolution throughout time. The relationship between the states and the response is fundamental and it is assumed linear. The reason for this assumption is that our system is not likely to have a chaotic behaviour and also that linear forms allow for full analytical results giving fast algorithms and saving significant memory. The state space form of a time series representation is independent of the Kalman filtering methodology and was discovered before the method. In fact, we use the state space form, and avoid the Kalman approach. The reasons for this will become evident when considering the model. The important thing to note is that the Kalman approach is again restricted on assuming some of the parameters of the model known. This is a common problem with ARIMA modelling and there is not justification on how the modeller chooses the variance between the variables. This is a well-known problem for our models and it will be considered in some detail in later chapters. It is important because it is closely related with independence among the variables, such as independence among the users and/or among the applications they use.

4.4 Conclusions

This chapter gives a general introduction on forecasting methods, and has analysed different forecasting methods, and the areas that each methods that each of the methods are used in. We also have made a general introduction to the prediction in general, and how it works. The focus of the chapter has been on an introduction on Bayesian methods, and ARIMA models, which are the most wide use forecasting models from forecasting practitioners. The general assumption in Bayesian applied to network security is that we can define that users can be viewed, initially, either as non-intruders, or as intruders. It is then up to the sampled data to show otherwise. It is likely in a highly secure network, that all users would initially be viewed as an intruder.

4.5 References

[4.1] Professor Hossein Arsham, *Statistical Data Analysis: Prove it with Data*, University of Baltimore, http://ubmail.ubalt.edu/~harsham/stat-data/opre330.htm.

[4.2] The Bayesian Institute, Bayes Thomas, http://www.bayesian.org/bayes.html

[4.3] Carlin B. and T. Louis, Bayes and Empirical Bayes Methods for Data Analysis, Chapman and Hall, 1996.

[4.4] Box, G.E.P., Jenkins, G.M., and Reinsel, G.C. (1994) Time Series Analysis: Forecasting and Control, (3rd ed.). Prentice Hall, Englewood Cilffs, NJ.

[4.5] Kalman, R.E. (1963) New methods in Wiener filtering methods. In proceedings of the 1st Symposium of Engineering Applications of Random Function Theory and Probability, J.L. Bogdanoff and F. Kozin (Eds.). Wiley, New York.

[4.6] Yamada K. A general hybrid regression forecasting model and its applications. [Conference Paper] 1968 IFAC symposium on optimal systems planning. Institute of Electrical and Electronics Engineers. 1968, pp.165-75. New York, NY, USA.

[4.7] Gardner ES Jr. Box-Jenkins vs. multiple regression: some adventures in forecasting the demand for blood tests. Interfaces, vol.9, no.4, Aug. 1979, pp.49-54. USA.

[4.8] Renxiang Gao, Bao Liu, Shiying Zhang. Nonlinear forecasting models for passenger traffic. Transportation Systems: Theory and Applications of Advanced Technology. A Postprint Volume from the IFAC Symposium. Pergamon. Part vol.2, 1995, pp.627-32 vol.2. Oxford, UK.

[4.9] Kung EC, Sharif TA. Regression forecasting of the onset of the Indian summer monsoon with antecedent upper air conditions. Journal of Applied Meteorology, vol.19, no.4, April 1980, pp.370-80. USA.

[4.10] Menge EE., 'Improvements in time series trend analysis increase accuracy of small area demand projections', 1977 Joint Automatic Control Conference. IEEE. Part II, 1977, pp.981-6. New York, NY, USA.

[4.11] Huth R., 'Statistical downscaling of daily temperature in central Europe', Journal of Climate, vol.15, no.13, 1 July 2002, pp.1731-42. Publisher: American Meteorol. Soc, USA.

[4.12] Smunt TL., 'A comparison of learning curve analysis and moving average ratio analysis for detailed operational planning', Decision Sciences, vol.17, no.4, Fall 1986, pp.475-95. USA.

[4.13] Johnston FR, Boyland JE, Meadows M, Shale E., 'Some properties of a simple moving average when applied to forecasting a time series', Journal of the Operational Research Society, vol.50, no.12, Dec. 1999, pp.1267-71. Publisher: Stockton Press for the Oper. Res. Soc, UK.

[4.14] Vander Wiel SA., 'Monitoring processes that wander using integrated moving average models', Technometrics, vol.38, no.2, May 1996, pp.139-51. Publisher: American Soc. Quality Control; American Stat. Assoc, USA.

[4.15] Martz HF, Kvam PH, 'Detecting trends and patterns in reliability data over time using exponentially weighted moving-averages', Reliability Engineering & System Safety, vol.51, no.2, Feb. 1996, pp.201-7. Publisher: Elsevier, UK.

[4.16] Edge SW., 'An adaptive timeout algorithm for retransmission across a packet switching network', Computer Communication Review, vol.14, no.2, 1984, pp.248-55. USA.

[4.17] Bretschneider SI, Gorr WL., 'On the relationship of adaptive filtering forecasting models to simple Brown smoothing', Management Science, vol.27, no.8, Aug. 1981, pp.965-9. USA.

[4.18] Wheelwright SC, Makridakis S., 'An examination of the use of adaptive filtering in forecasting', Operational Research Quarterly, vol.24, no.1, March 1973, pp.55-64. UK.

[4.19] Snyder R., 'Forecasting sales of slow and fast moving inventories', European Journal of Operational Research, vol.140, no.3, 1 Aug. 2002, pp.684-99. Publisher: Elsevier, Netherlands.

[4.20] Gardner ES Jr, Diaz-Saiz J., 'Seasonal adjustment of inventory demand series: a case study', International Journal of Forecasting, vol.18, no.1, Jan.-March 2002, pp.117-23. Publisher: Elsevier, Netherlands.

[4.21] Fei He, Jian Li, Yue You. 'Design and implement of network traffic forecast system', 2001 International Conferences on Info-Tech and Info-Net. Proceedings (Cat. No.01EX479). IEEE. Part vol.5, 2001, pp.164-9 vol.5. Piscataway, NJ, USA.

[4.22] Brutlag JD., 'Aberrant behavior detection in time series for network monitoring', Proceedings of the Fourteenth Systems Administration Conference (LISA XIV). USENIX Assoc. 2000, pp.139-46. Berkeley, CA, USA.

[4.23] Cogley T, Nason JM. 'Effects of the Hodrick-Prescott filter on trend and difference stationary time series. Implications for business cycle research', Journal of

Economic Dynamics & Control, vol.19, no.1-2, Jan.-Feb. 1995, pp.253-78. Netherlands.

[4.24] Frane JW., ‘Recent developments in the maintenance and distribution of BMDP’, Computer Science and Statistics. Tenth Annual Symposium on the Interface. Nat. Bur. Standards. 1978, pp.221-4. Washington, DC, USA.

[4.25] Leece J, Parker F., ‘Use and misuse of SPSS (Statistical Package for the Social Sciences)’, Software-Practice & Experience, vol.8, no.3, May-June 1978, pp.301-11. UK.

[4.26] Immonen-Raiha P, Hatonen S, Torppa J, Toivanen A., ‘A statistical analysis system macro for age-standardized incidence rates’, Computer Methods & Programs in Biomedicine, vol.44, no.2, Aug. 1994, pp.79-83. Netherlands.

[4.27] Bayes T, ‘Essay Towards Solving a Problem in the Doctrine of Chances (1763)’, which was published in the Philosophical Transactions of the Royal Society of London. http://www.stat.ucla.edu/history/essay.pdf.

5 Simple Tutorial on the Proposed Bayesian Method

5.1 Introduction

The previous chapter justified the requirements for the Bayesian method, that is, that it is useful in initially defining a hypothesis, which is either true or false. This goes against frequencism methods which takes a long-term view on prediction. This method may have advantages in predicting intrusions, as it can quickly adapt to individuals user activities. In Chapter 4, it was argued that users who change their behaviour can be a suspect for network intrusion, in a highly secure environment. Thus, we can use Bayesian methods to make a prediction on future events that a user may access a certain resource. In this research, the resource is taken to be application usage, but could equally relate to the usage of any network-based resource, such as a file, a folder, a printer, and so on.

5.2 Simple introduction

The system is continuously monitoring the usage of system resources. We first made make some decisions on how we are going to measure our system resources usage, how are we going to express the result, and over what period. In this research, the time period is taken as one hour. Next, we decided to express this with a real value number and not an integer. This is because we wanted to keep the values small, so the initial experimentation will be smoother. We also decided that we are going to grade the ‘no usage’ measurement of measuring our system resources, with ‘zero’ and one full hour of ‘usage’ with the ‘one’. For example if we have a measurement of ‘0.3’ then the particular monitored system resource is used for 0.3×60min, which is 18 minutes.

Our prediction model is doing one-step forecasting. This means that each time that we want to make a forecast we can only do one hour ahead. When we will get the results from the forecasting of this hour, we can then forecast for the next hour, or the chosen time interval.

So let us assume that we have 20 observations and we want to make a forecast for the last five of them, as in Table 5.1. We take the 15th observation on our forecasting model, and then estimated a value for the 16th. We then compare it with the real observation that we get at the end of the hour (time interval), and put the 16th observation as a feed in our forecasting model in order to get the next one, and so on. As our model uses vectors, we can put an array of different observations together in our forecasting model and get the results. This speeds up our forecasting procedure. This is the case in real situations, when you we have to monitor 20 or 30 different system resources, and get prediction results.

Therefore, when we apply our first observation to our model, we are getting something like what it is described in Figure 5.1. In this, we have our initial observations and some initial vectors that we need for our calculations. The initial values in these vectors are generated empirically, so they will be suited to the model. As our experimentation proceeds, we adopt these values. We have to note that there has to be more research on our initial values and they way that they are chosen. After we feed the initial values into the model, we calculate the results for time T+1 if we consider that we are at time T. Also we feed the results back to our model for further forecasting values. When the forecasting is complete, we have a vector with all the predicted values for each period.

5.3 Example

Table 5.1 shows 20 observations, which contains values between zero and one. These describe the usage of a specific computer application in an hour. Thus, if the observed value is unity, the application is being used for one hour, and if the observed value is zero, it has not been used at any time within the hour. If the application defined as 0.2, it has been used for 12 minutes (0.2×60).

In this case, for simplicity, there is only one application being monitored (represented with the letter *r*) and we also have only one user (represented with the letter *n*). Thus, there is one user and a number of *r* = 1 applications of interest. We also get the results of the real observations every hour.

As presented in Chapter 6 there are two basic equations that describe the statistical model:

$$Y_t = F_t^T \theta_t + v_t, \qquad v_t \sim N[0, \sigma], \tag{5.1}$$

$$\theta_t = \theta_{t-1} + \omega_t, \qquad \omega_t \sim N[0, \sigma W_t], \tag{5.2}$$

where:

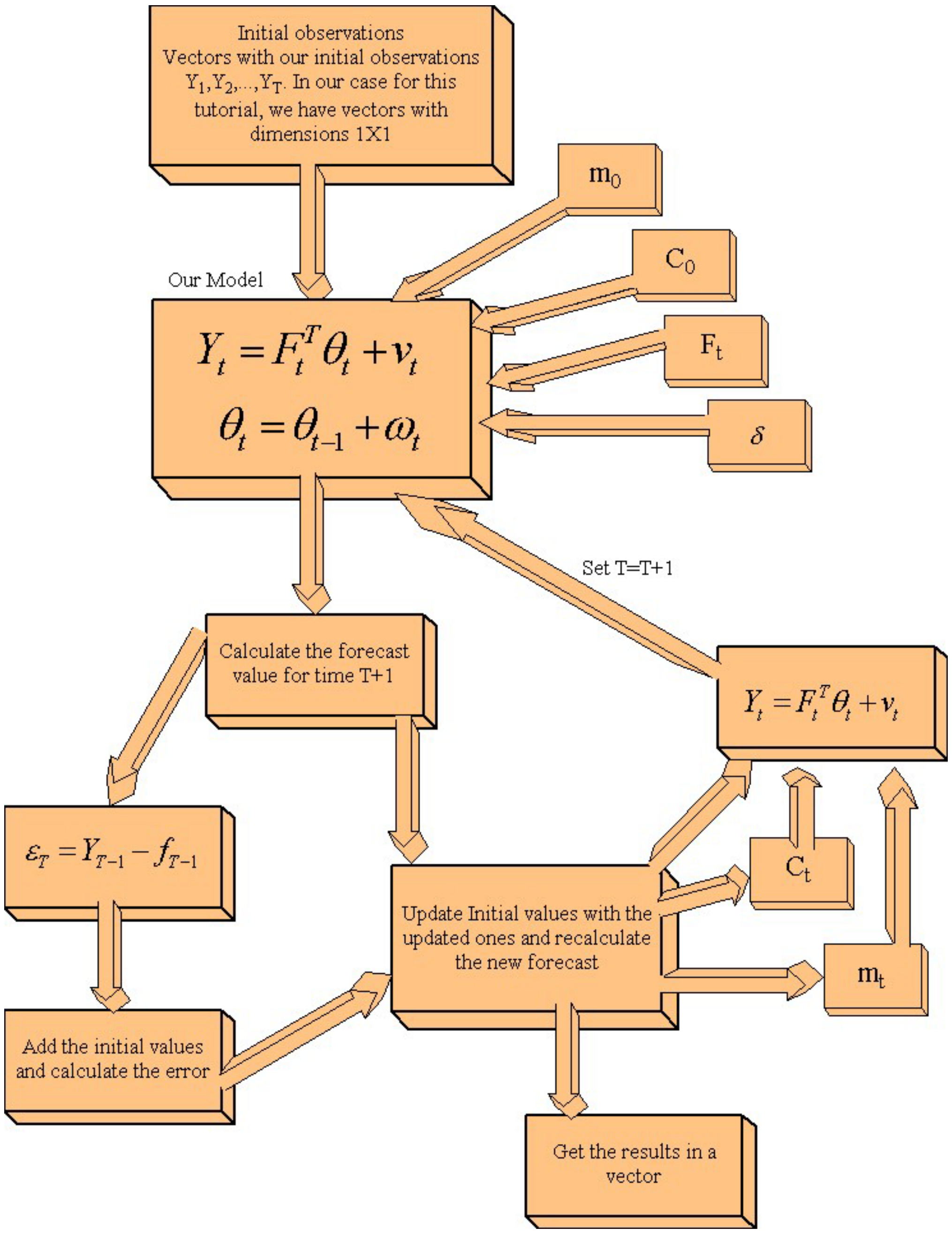
Y_t is a variable that we observe at time *t*.

F_t an n-dimensional vector (known as the design vector).

θ_t an n-dimensional parameter vector.

v_t a random variable that has a normal distribution with zero mean and variance σ.

ω_t is an n-dimensional random vector with a multivariate normal distribution with zero mean and variance matrix σW_t.



Initial observations Vectors with our initial observations $Y_1, Y_2, ..., Y_T$. In our case for this tutorial, we have vectors with dimensions 1X1
m_0
C_0
F_t
δ
Our Model
$Y_t = F_t^T \theta_t + v_t$
$\theta_t = \theta_{t-1} + \omega_t$
Set T=T+1
Calculate the forecast value for time T+1
$Y_t = F_t^T \theta_t + v_t$
$\varepsilon_T = Y_{T-1} - f_{T-1}$
C_t
Update Initial values with the updated ones and recalculate the new forecast
m_t
Add the initial values and calculate the error
Get the results in a vector

Figure 5.1: Bayes Model behaviour

Table 5.1: Real and predicted values

Observation Number	Real value	Predicted Value
1	0.0	
2	0.6	
3	0.4	
4	0.0	
5	0.9	
6	0.8	
7	0.5	
8	0.1	
9	0.0	
10	0.0	
11	0.3	
12	0.3	
13	0.1	
14	0.1	
15	0.0	
16	0.6	0.0
17	0.0	0.0
18	0.9	0.11
19	0.7	0.16
20	0.7	0.14

Equation (5.1) is called the observation equation as it shows how the observations (Y_t) are linked with the parameters (θ_t). Here the relationship is linear which is shown clearly if we write the vectors $F_t^T = (f_{1t}, f_{2t}, \ldots, f_{nt})$ and $\theta_t^T = (\theta_{1t}, \theta_{2t}, \ldots, \theta_{nt})$ (the symbol 'T' as a postscript refers to the transpose of a vector which is the arrangement as a row) and then apply the vector multiplication: $F_t^T \theta_t$. Doing this, equation (5.1) can be written as:

$$Y_t = f_{1t}\theta_{1t} + f_{2t}\theta_{2t} + \cdots + f_{nt}\theta_{nt} + v_t \tag{5.3}$$

where v_t has the same distribution as in (5.1).

Equation (5.3) is a linear equation which is why the model is called a *linear model.* Normally the values $f_{1t} f_{2t}, \ldots, f_{nt}$ will be constants not depending on *t,* such as $f_1, f_2, \ldots, f_n$. These values are assumed to be known and, in later chapters, we discuss their choice. Note that these design values (building the design vector) are the equivalent to the design matrix in regression [5.1]. To make the above more clear, suppose that we choose *n*=2. This means that we choose to have two parameters to model our series Y_t, namely θ_{1t}, θ_{2t}. The observation equation will be:

$$Y_t = f_1\theta_{1t} + f_2\theta_{2t} + v_t \tag{5.4}$$

The choice of *n* is left to the modeller. It does not have a practical interpretation, but it is proposed that we keep the number of parameters low, as the model uncertainty will increase with an increased number of parameters. This means that we will need more data to estimate the extra parameters, and so the system will not have fast response due to over-parameterisation. On the other hand a single parameter (*n*=1) will not suffice to properly forecast our time series. For these reasons, we use in all our applications a value of *n*=2.

Equation (5.2) is called the system or transition equation as it tells us how the parameters change over time. Here we allow for a slow change from time *t*–1 to time *t*. This equation makes the model *dynamic* as it does not assume that the parameters are appropriate for all times, giving a local suitability of the model. This will be discussed in more detail in Chapter 6.

At time *t* (hours) the random variable Y_t expresses the total time of application usage in the interval $[t-1,t]$. When we observe the value of Y_{t-1} we can use our model (Eqn. (5.1) and (5.2)) to give the forecast of the unknown Y_t. This is called one-step forecasting as it uses the current observation in order to predict the next observation. This is a simple idea behind the notion of our statistical prediction model. In practice, the model that we using is more complex in the fact that it is making one-step prediction for as many steps as we define.

Not only does our method provide the forecast as a single prediction number, it also gives forecast variance and the entire distribution of the forecasts (this will be covered in Chapter 6). The forecasts are produced by an algorithm using the observed values of $Y_1, Y_2, \ldots, Y_{t-1}$ as well as some initial values of the estimates of the vector θ_0 (θ_t at *t*=0). By these estimates we define its mean and variance at *t*=0, namely m_0, C_0 respectively. Then the algorithm forecasts the value of Y_1 as the quantity $F^T m_0$. Since we know Y_1, we can calculate the one-step prediction error, which is $e_1 = Y_1 - F^T m_0$. This tells us how close our forecast for *t*=1 ($F^T m_0$) is to the actual value Y_1. Ideally, it would be $e_1 = 0$, but this is only theoretically possible because our predictions cannot be perfect. However, we may like to monitor the errors so that our predictions are not unrealistic. The algorithm carries on by calculating the next forecast based on Y_1 and producing a forecast for Y_2. Every time we observe a value of our time series, we update the forecasts. The quantities m_0, C_0 must be initially set. Here, if we choose n=2, m_0, we will get a 2-dimensional vector (that means we will have to specify two single values for m_0) and C_0 will be a 2×2 matrix (that means we will have to specify three single values). Here we use $m_0 = (1,1)^T$ and:

$$C_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{5.5}$$

So we first have to calculate, for $t-1$, the value of m_{t-1} which is $m_{t-1} = E(\Theta_{t-1} \mid D_{t-1})$, where E is the Expectation of the Θ_{t-1} given D_{t-1}. We also need to calculate the C_{t-1} which is $C_{t-1} = Var(\Theta_{t-1} \mid D_{t-1})$, where Var is variance.

To calculate the predicted value, or the forecast for time t we need to calculate f_t first ,which is $F^t m_{t-1}$, and then we have to calculate Y_t.

So we have to calculate m again but this time for t=t:

$$m_t = m_{t-1} + (C_{t-1}F/(\delta Q_t))(Y_t - F^t m_{t-1}) \tag{5.6}$$

where $Q_t = F^T C_{t-1} F / \delta + \sigma$ and also $C_t = C_{t-1}/\delta + (C_{t-1}FF^T C_{t-1}/(\delta^2 Q_t))$. After substituting arithmetic values to our previous equations, we find the value of Y^t. And then we start again for our new value of $t-1$ and t.

A few more initial values are needed for the implementation of the full algorithm, but these will be discussed in more detail in Chapter 6. Table 5.1 shows an example of the actual values for 20 observations (time *t*=1 to 20) and the associated one-step forecasts for the last give observations (*t*=16 to 20). In Figure 5.2, the predicted values for this table (illustrated as a solid line) are shown against the real observations (illustrated as a broken line). It can be seen that the predictions do not follow the actual values. This is because the initial data sample size is too small for any prediction model to have very accurate results. In addition, it can be seen that the prediction line is following the real observations line, and that they have similar slopes. This is an important factor in the prediction, as it is used in the prediction results and depicts that the user will have an increase or decrease in the use of the monitored resource, by just observing the slope of the prediction graph.

Thus, from Figure 5.1 we can conclude that our model has the potential to give information about the future use of a specific system resource, even if the initial values that are read before the prediction is started is minimal, and that the slope of the resulting prediction is an important factor.

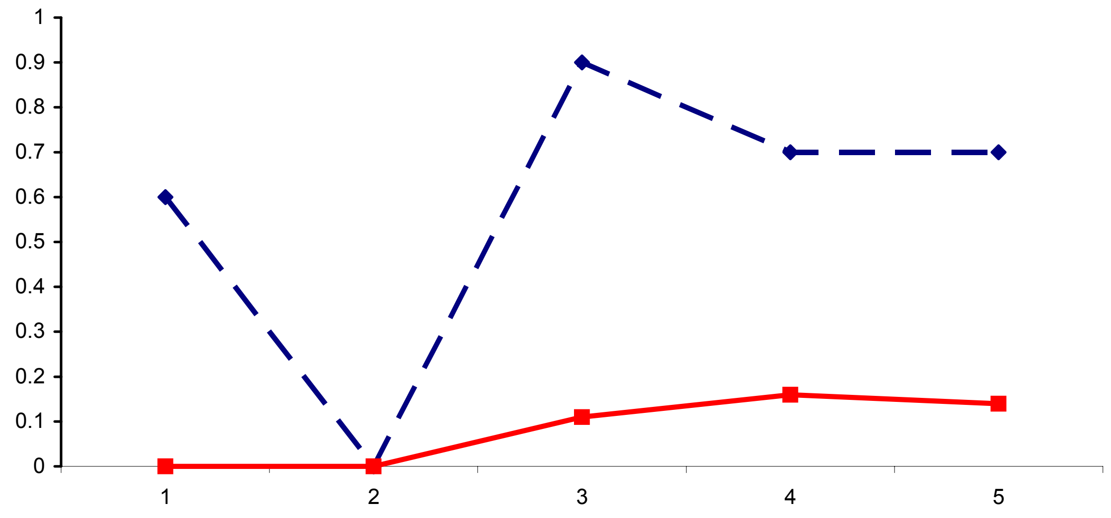


Figure 5.2: Prediction results of the tutorial

5.4 Conclusions

This chapter has outlined a simple situation for the proposed forecasting model. We have seen how the model calculates the forecasting data. In this case our prediction was only five values, so the model did not had the opportunity to learn much from the previous values, and thus cannot make an a very accurate prediction.

It can be seen that the model has potential in giving some indication about what the user will do next. In the following chapter we show exactly how our model works, and how we set up the initial values for the model.

5.5 References

[5.1] Sen A. and Srivastava M. (1990), *Regression Analysis*, Springer-Verlag, New York.

6 Proposed Statistical Prediction System

6.1 Introduction

This chapter defines the model published in the Journal of Forecasting [6.1]. It identifies the theoretical part of the model, and how it works. Bayesian is now applied to many areas, and is increasing in its relevance to many areas. In Chapter 4 has shown that Bayesian provides a better model than frequencism methods in areas such as the analysis of drug treatments, or in the analysis of DNA evidence. The model assumes a hypothesis that in order to detect an intruder that we must forecast the future events, and then monitor the actual match between the forecasted events, and the actual events. An intruder, or someone who is acting fraudulently within an organisation, is likely to show changes in their behaviour, especially if they have used the user ID and password of another user. A simple example is if the system monitors the keystrokes of the user, and determines their typing speed. If the user normally has a typing speed of 20 words per minute, and the system forecasts this, then an intruder can be detected if this rises over to 60 words per minute, or even drops to 10 words per minute. There are thus many differing monitors that can be applied to a user, such as mouse movements, keyboard strokes, processor usage, file access usage, disk size, usage of the graphical user interface, use of certain phases, email addresses used, and so on. In this research we focus only on application usage, as this can be easily measured, and can also be predicted, but it is important to understand that the methods can be easily applied to differing monitors, which may give improved intrusion detection.

6.2 Model

Our model has three phases of operation:

- **Observation stage**. An important phase of the Bayesian method is the observation phase, in which the system will try to understand the monitored trend.
- **Evaluation phase**. This is where the model makes a prediction and also monitors the user actual moves and determines a forecast. This stage is critical, as the model modifies itself according the need of the environment that it operates in.
- **One-step prediction**. The system then goes into a phase where it is confidence about its predictions, and makes one-step predictions.

As an example, let us assume that the user is logged in for 15 times and that the model is configured and it is ready to start predicting user moves. Instead of making five or 10-step prediction, like other mathematical models, our model makes a prediction for the next step. When the user logs-in and -out the model takes the actual behaviour of the user, and compares it with the one-step prediction that it has performed before and calculating the error. Thus, the next time a prediction is made for this user it will include the data of the last user behaviour. With this procedure, we maximise the accuracy of the prediction system.

The general multivariate model (DLM) is given by the next equations [6.1]:

$$Y_t = F_t^{'}\theta_t + v_t, \qquad v_t \sim N[0,\Sigma] \tag{6.1}$$

$$\theta_t = G_t\theta_{t-1} + \omega_t , \quad \omega_t \sim N[0,W_t] \tag{6.2}$$

We use multivariate models, as we want to incorporate and forecast several variables simultaneously. Again note that the parameters θ_t change both deterministically (through *t*) and stochastically (through the variance W_t), and thus make the model dynamic. Also standard ARIMA models are a special and restrictive case of the above model, when you set $F_t = F$, $G_t = G$ and $W_t = W$ (all these three components are constant over time). This is restrictive as all these components are likely to change over time because data changes over time and that there are other external sources of variation (such as extra subjective information about a variable). Moreover, the second equation is not observable. This means that we are never going to see any evolution or trend in a diagram or a graph. This is a hidden model that cannot assume W_t to be constant over time.

There is another problem that we cannot ignore in multivariate models. The variance matrix Σ will is not known. Often, in standard time series, it is assumed known and the researcher can easily jump to another problem. However, in practice, this is extremely difficult to set as a known matrix. It is very difficult to propose what variance to use to a system where 20 applications are considered and only 20 or 30 vectors are collected as data.

Therefore, for all these reasons we need to consider the dynamic models. In addition, the system could provide forecasting as much ahead as we like, proving accurate according to the results. For this purpose, we used a Bayesian framework, which virtually means that at time *t* we will have some kind of knowledge, which is a **subjective** belief, expressed in terms of a distribution. As said before the true strength of Bayesian is that it is subjective in its approach, and that we can apply a prior probability distribution. Thus, we have a prior distribution of $(\theta_t \mid D_{t-1})$ at time *t*. In other words, it is what we know before Y_t becomes available. Once this happens, we revise this prior belief, using the likelihood function (see Page 55, Chapter 4), to find the posterior distribution $(\theta_t \mid D_t)$ or revised, which is better and more accurate. Then according to simple calculations, we find the prior of time *t–1* and we calculate the posterior at *t+1*, only when information of the data Y_{t+1} comes into the system (which is, in our case, is the real behaviour of the user). The model used becomes:

- **Autoregressive moving average model**. The general model introduced by Box and Jenkins (1976) includes autoregressive as well as moving average parameters, and explicitly includes differencing in the formulation of the model. Specifically, the three types of parameters in the model are: the autoregressive parameters (*p*); the number of differencing passes (*d*); and moving average parameters (*q*). In the notation introduced by Box and Jenkins, models are summarized as ARIMA (*p*, *d* and *q*); so, for example, a model described as (0, 1, 2) means that it contains 0 (zero) autoregressive (*p*) parameters and 2 moving average (*q*) parameters which were computed for the series after it was differenced once.
- **Identification**. As mentioned earlier, the input series for ARIMA needs to be stationary, that is, it should have a constant mean, variance, and autocorrelation through time. Therefore, usually the series first needs to be differenced until it is stationary (this also often requires to logarithmically transform the data to stabilise the variance). The number of times the series needs to be differenced to achieve stationary is reflected in the *d* parameter (see the previous paragraph). In order to determine the necessary level of differencing, one should examine the plot of the data and autocorrelogram. Significant changes in level, such as strong upward or downward changes in user behaviour, normally requires first-order non-seasonal (*lag*=1) differencing; and strong changes of slope normally require second order non-seasonal differencing. Seasonal patterns require respective seasonal differencing (see below). If the estimated autocorrelation coefficients decline slowly at longer lags, and thus first-order differencing is normally required. However, one should keep in mind that some time series may require little or no differencing, and that over differenced series produce less stable coefficient estimates.

  At this stage we also need to decide how many autoregressive (*p*) and moving average (*q*) parameters are necessary to yield an effective, but still efficient, model of the process (that is, with the fewest parameters and greatest number of degrees of freedom among all models that fit the data). In practice, the values of the *p* or *q* parameters are rarely greater than two (see the forthcoming material for more specific recommendations).

- **Estimation and Forecasting**. At the next step (estimation), the parameters are estimated (using function minimization procedures), so to minimise the sum of squared residuals. The estimates of the parameters are used in the last stage (forecasting) to calculate new values of the series (beyond those included in the input data set) and confidence intervals for those predicted values. The estimation process is performed on transformed (differenced) data; before the forecasts are generated, the series needs then to be integrated[1] so that the forecasts are expressed in values compatible with the input data. This automatic integration feature is represented by the letter I in the name of the methodology[2].

In addition to the standard autoregressive and moving average parameters, ARIMA models may also include a constant, as described above. The interpretation of a statistically significant constant depends on the model that is used. Specifically:

- If there are no autoregressive parameters in the model, then the expected value of the constant is μ, the mean of the series.
- If there are autoregressive parameters in the series, then the constant represents the intercept.

If the series is differenced, the constant represents the mean or intercept of the differenced series. Thus, if the series is differenced once, and there are no autoregressive parameters in the model, the constant represents the mean of the differenced series, and therefore the linear trend slope of the un-differenced series.

ARIMA models are similar to our model. They use the existing data to calculate the parameters of the model. However, if, for example, some external information is available, such that we may know that it is user *xyz* and although they do not have an illegal user profile, it is very probable that at a specific point of time they will perform a huge invasion to an important application. ARIMA will try to change the parameters to adjust the model, but even in this case, it is doubtful how well the model will do in all the applications. With our DLM it is not a problem. Simply we add to the prior information we have, which is the external information. This is described as *expert intervention*, and the revised posterior probability takes into account the new knowl-

[1] Integration is the inverse of differencing.

[2] ARIMA – Auto-Regressive Integrated Moving Average.

edge. Our system is not assumed perfect when the model is fitted, and we let information, no matter what its type, to allow the system to learn and improve itself.

Now our model is slightly different than the one we use for illustration purposes. We find recurrence relationships, which are more natural to overall long formulae that ARIMA works out. We note that because ARIMA is quite complicated, and many practitioners end up with a simple subclass of ARIMA model, that does not even start with assumptions. This produces results that sometimes do not correspond to the real application. The only difficulty with the DLMs is the specification of the initial values, such that the algorithm may be put into practice. In general, this requires to be solved by the experience of the individual practitioner.

In our case, we have to specify the following: $m_0, C_0, S_0, n_0, \beta, \delta, F_t * m_0$ is the mean of $(\Theta_0 \mid D_0)$ and C_0 its variance. The choices made are:

- $m_0 = 0$. This is set when we expect that the prior distribution $(\Theta_1 \mid D_0)$ (the distribution of the parameter Θ at time *t* given D_0 – any initial information which is explicitly known) will not give any drift to Y_1. The fact that we expect this to happen, but we are not sure, so there is here an uncertainty, which is expressed by the variance C_0. It is natural and common policy to assume $C_0 = I$, the identity matrix. But care must be taken when we are very uncertain about our choice we MUST increase the diagonal elements of C_t. Of course, this affects all the following results somehow, but the approach is more realistic. In general, we will have more data vectors than 15, or 20 (our case), hence, initial values will dominate the actual estimates in a decreasing rate.
- $C_0 = I$. This is motivated by our belief that m_0 is not important to the following values of Y_t, t=1, ... S_0 is typically, almost always set to I and it has not got any special meaning. The only one we can find is that it is chosen such that according to the formula that we have to calculate S_t, S_0 must lead to acceptable results (symmetric matrices). The n_0 can be set to 0 (a case which implies $n_1 = 1$, without great loss) or $n_0 = 1$ (a case which implies $n_1 = \beta + 1$). The choice of n_0 is not crucial since there is theorem that states that S_t converges to Σ as *t* goes to infinity and it does not depend on n_0. But it must take mall values.

- δ. The δ choice is discussed with details in Ameen and Harrison (1982a) [5.1] where it is shown that it must be $0.85 < \delta < 1$ and quite high. Thus we have set it 0.95.
- β. The β is a discount factor as well. In this document we state that it has to be smaller than δ, as, in general, S_t is not so much influenced by the data as it is m_t. Note that δ is in A_t and so it influences m_t.

The components are defined as:

m_0	The mean of the influence of Θ_1, Y_1 from D_0, our initial info.
C_0	Dispersion of the above influence.
S_0	No meaning, and is an auxiliary quantity for S_t.
n_0	Same as above.
β	Factor of the influence of the data to the estimate S_t.
δ	Factor of the influence of the data to the estimate m_t.
F_t	A basic quantity that expresses the linearity of the model and gives different trends to the several values of Y_t, both for time series analysis (what has happened in the past) and forecasting (what will happen in the future).

Finally, we make clear that when we say *factor* in the above explanation we do not mean any percentage or whatever. Factor means discount factor, which means that the estimates of m_t and S_t are discounted somehow and in different rate, since both are influenced by data.

6.3 Intervention

Intervention is a mechanism for increasing the prediction accuracy to maximum. It is used when we have any additional information about the future behaviour of the system, then we can thus add them to the model **prior** the prediction. For example if there is some users that are keen of using illegal software or there are new users that there is not enough information about their behaviour, by applying the intervention mechanism, we increase the accuracy of the model and can make more accurate predictions.

In our model, we can observe this by looking at Chapter 8. In these we can observe that our model prediction is very close to the actual users behaviour for the application number 1 at the specific time *t=19*. We achieved this accuracy by applying the intervention technique. We can also observe that the ARIMA model did not make any prediction for this particular user behaviour (see [5.2]).

6.4 Conclusions

In this chapter we explained how our forecasting model works, and analysed all the parameters of our model, with an explanation of each of the initial parameter are, and how they are set. It has also included a general comparison with the ARIMA models, which are the current dominants on the forecasting practice. It will be shown in future chapters that the new model work better that ARIMA for this application.

From this chapter we can conclude some future research for our forecasting model. As we saw, some of the initial values of our forecasting model are set empirically from the feedback of the results of our experiments, or random with a minimum and a maximum limit. This sometimes has a negative effect to our results. Some future research on the subject can find some configuration equations for more accurate and predetermined initial values.

We also have saw in this chapter the application of the intervention method, which enable our forecasting model to have accurate results, even in a small number of observations. In the next chapter we will explain, in detail, how our software agent security environment is structured, how did we build it, and why.

6.5 References

[6.1] Kostas Triantafyllopoulos and John Pikoulas, '*Multivariate Bayesian regression applied to the problem of network security*', Journal of Forecasting, 21, pp 579-594.

[6.2] J.R.M. Ameen and P.J. Harrison, *Normal discount Bayesian models,* Journal of Bayesian Statistics, 1985.

7 Software Agent Security System

7.1 Introduction

Previously we described the different computer security vulnerabilities, and the need for additional security. In addition, it has been proposed the reasons for prediction in a highly secure environment. This chapter describes the software agent security enhancement system implemented in this research, and outlines its main elements, and how they intercommunicate.

Figure 7.1 illustrates the difference between agent-based security and centralized security. Typically, security is implemented on a domain basis, where a central server holds the database on the complete domain. When a user logs into the network the host interrogates the domain controller, which grants the login or not. The domain controller will then grant permissions, or not, based on the rules on its database. Unfortunately, this type of system puts a great deal of emphasis on the central server. It can thus become the focus of an attack on the network, typically with a denial-of-service (DOS) attack. If the central server becomes overburdened, or even stops working, the security for the whole domain will be affected.

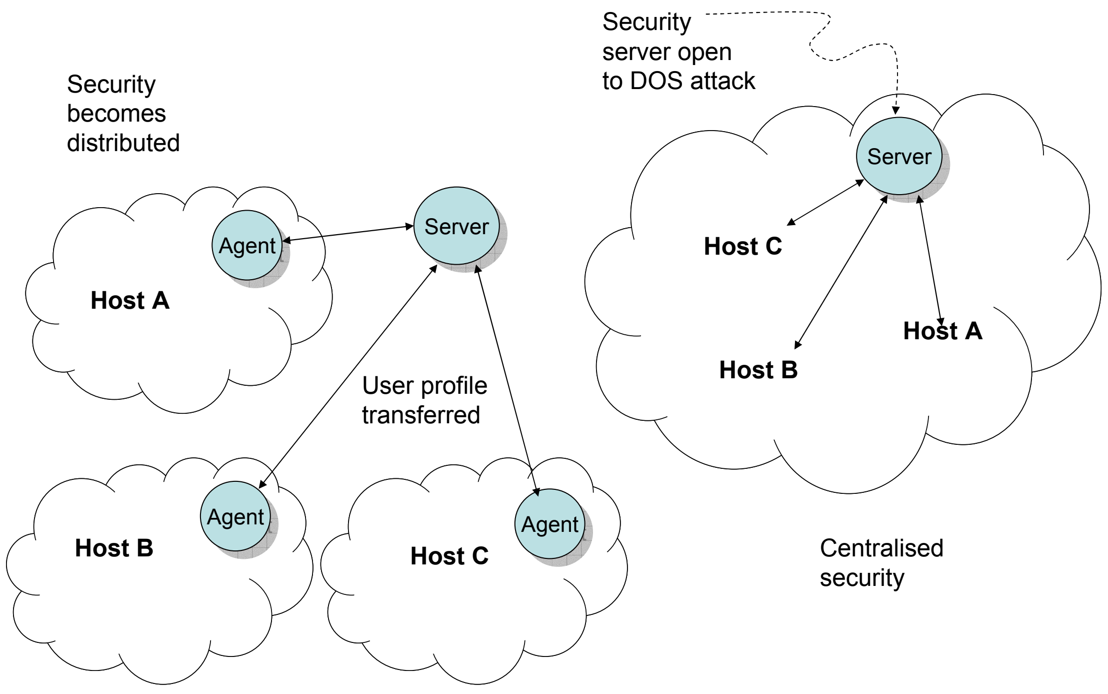


Figure 7.1 Distributed agent-based security and centralized security

An agent-based approach distributes the security, as a profile of the user is downloaded from the server to an agent running on a host, when the user initially logs in. The agent then becomes responsible for the security within the host. Its main responsibility is then to monitor the security conditions of the user until they log-off. The system implemented in this research uses agent which monitors the operation of a user based on previous events. The agent then forecasts into the future to determine their likely mode of operation. If this deviates from normal behaviour, the system administrator, via the server, is alerted [7.1]. The test results of the research have been applied to user behaviour for running applications, but could equally be applied to other user behaviour patterns, such as file access, folder access, keyboard usage, and so on.

Figure 7.2 illustrates the operation of the software agent security enhancement system. It consists of at least two software entities, the *user-end* agent and the *core* agent. The core agent is a collection of software objects that operate on the server and acts as a repository for user profiling information. The core agent focuses on prediction profiling work, in the background when the server is not very busy. If possible, the user-end agent must be a fast acting program for real-time monitoring and have a small memory footprint. Figure 7.3 illustrates the main component parts of a core agent, these are:

- **Transmitter.** This is responsible for getting secure information from the user-end agents, and sending back requested information and user profiles.
- **Actioner.** This informs the system administrator of any important feedback that it is getting from the various *user-end* agents.
- **Profile selector.** This selects the appropriate profile for the specific user, which then is then stored locally on the server, and is received back from transmitter.
- **Statistical prediction result engine.** This collects all the new profile information that the various user-end agents are sending. It compiles them through our novel prediction model, and creates some prediction results, that describe future user behaviour.
- **Main part.** This is the part that coordinates all the individual components of the core agent system.

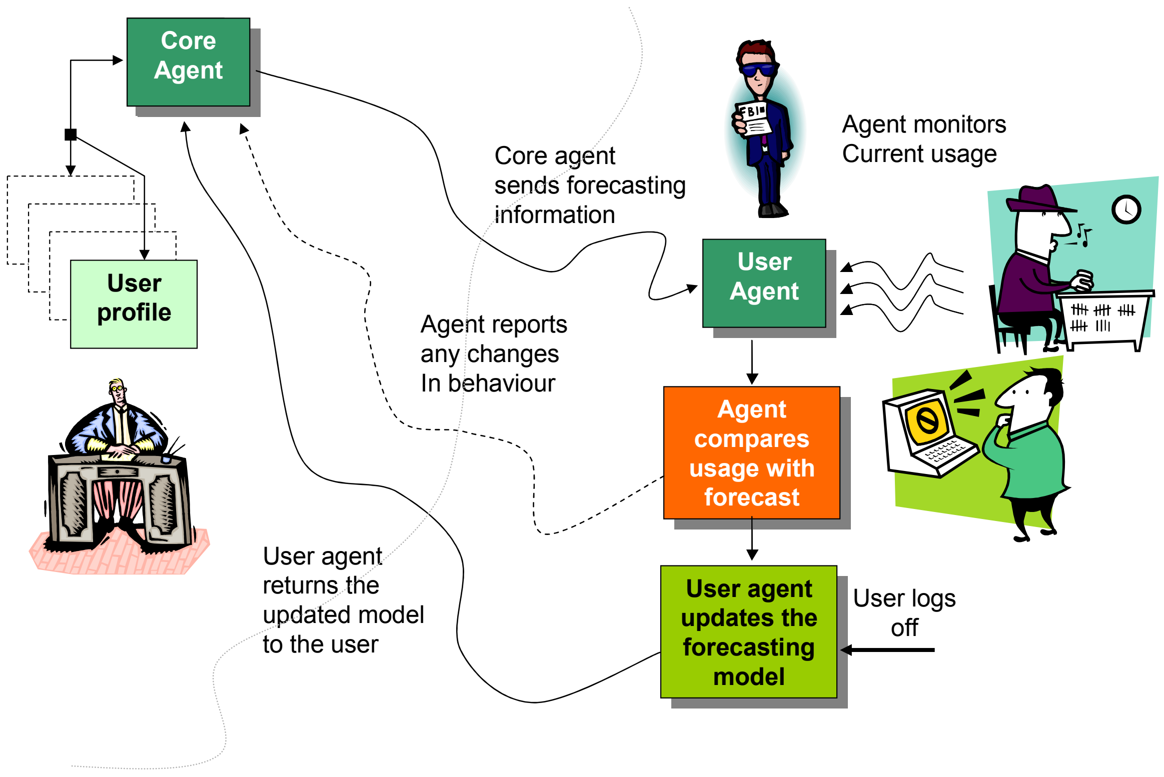

Core Agent
User profile
Core agent sends forecasting information
Agent monitors Current usage
User Agent
Agent reports any changes In behaviour
Agent compares usage with forecast
User agent returns the updated model to the user
User agent updates the forecasting model
User logs off

Figure 7.2 Software Agent Security Enhancement Topology

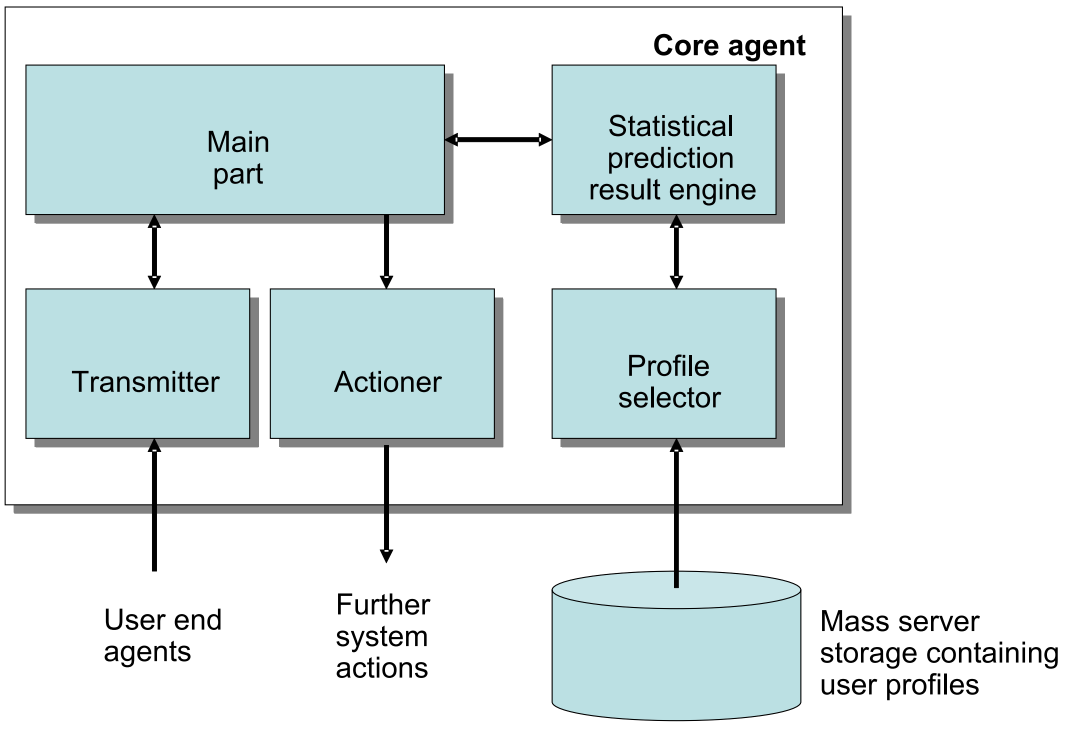


Figure 7.3 Core agent topology

A user-end agent resides on every host on the protected computer environment. This agent is responsible for monitoring each user that logs onto the host. It is started at the logon process, and has system administrator privileges. These privileges stop the agent from being shutdown. It comprises of:

- **A sensor**. This monitors the various software applications (for example, a word processor, a spreadsheet, or some system resources like the usage of a printer or scanner) that are currently being run by the user on that host. When a user logs-in, the sensor polls the user's activity every five seconds and records the user's identifier, each application's name, and process identifier.
- **A transmitter**. After the first polling by the sensor, the transmitter sends this information to the core agent. The core agent then responds by sending a user historical profile. With an audit-log file for a period of one month, we observed that the size of an average user profile was between 400KB and 600KB, with a download time of between three and five seconds.
- **A profile reader**. This reads the user's historical profile.
- **A comparator**. This compares the user's historical profile with the information read by the sensor. If the current behaviour profile does not fall within the accepted behaviour pattern defined by the user historical profile, the comparator provides the transmitter and sends the following to the core agent: user identifier; invalid behaviour type; and corresponding invalid behaviour type data.
- **An actioner.** This is an agent that executes the actions that have to be imposed on the system. According to the user behaviour, the actioner might have to send a visual message to the users screen for some warning or some additional help, or disable some actions of the user. The actioner part, along side with the main part, also uses that 'prediction data' for this user, to take preemptive actions on the user behaviour.
- **A main part.** This makes decisions according to the information that is fed back from the comparator and the sensor.

The operation of the user-end agent is illustrated in Figure 7.4. When the user-end agent is initiated, at the same time that a user logs-in, the transmitter part of the user-end agent is activated, and connects with the core agent. When the connection is successful, the user-end agent identifies itself by sending a simple message to the core agent. In this way, the core agent knows the user name, the IP address, and the name of the host that the user started the user-end agent on. When the core agent gets all this information it then sends the corresponding user profile to the user-end agent (Figure 7.5). This profile, as outlined in Figure 7.6, is a collection of rules and data, such as *real-time monitoring* rules and *prediction rules*. The real-time monitoring rules are set by the system administrator at the beginning of our enhance security environment. They include areas of the network computer system that the users can or cannot access, software applications that users cannot use on their hosts, and so on. These rules can even include FTP or WWW locations that they administrators feel that the users should not visit. When each individual user logs onto a host, these rules are updated automatically, according to user behaviour. The prediction rules are then made from the statistical model, in order to predict user behaviour actions, before an incorrect action takes place. They consist of sets of values that describe the next prediction time interval, and values that are generated by our 'statistical engine' and describe the level of use of the monitored resource. A sample of prediction rules is shown in Figure 7.6.

As seen Figure 7.6, the format of the user profile is simple. In the first section of the rules we have what the user is set to allow doing from the system administrators, when the system first installed. In the next section, we have what the system administrators are not allowing the user to do, in terms of resources that they are not allowed to access. Note here that they resources that are not specified are questionable and will be examined later by the system if the specific user accesses any of them. The next section contains the results of the evaluation of the user profile.

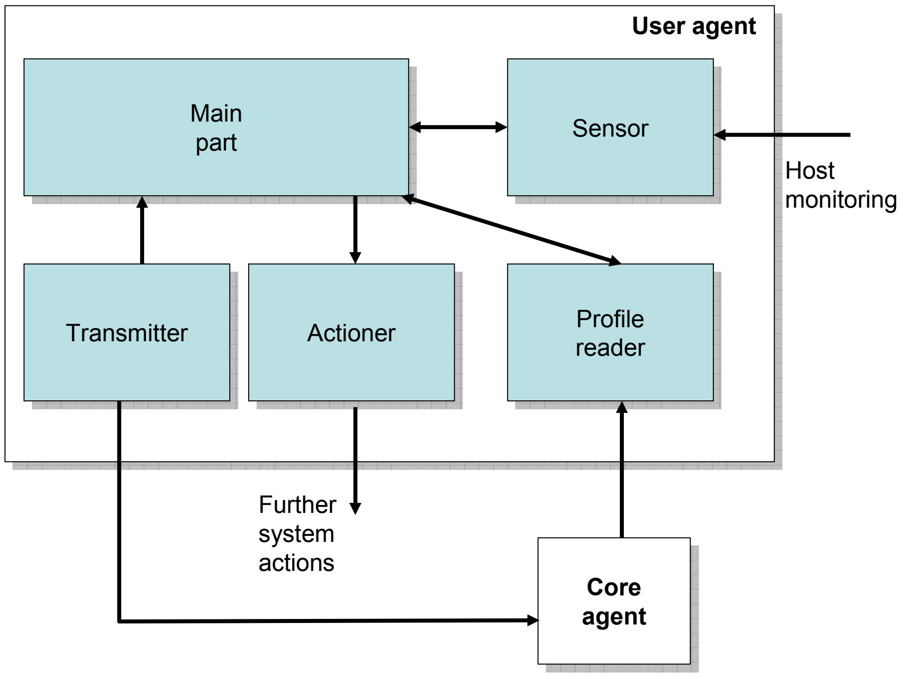


Figure 7.4 User-end agent topology

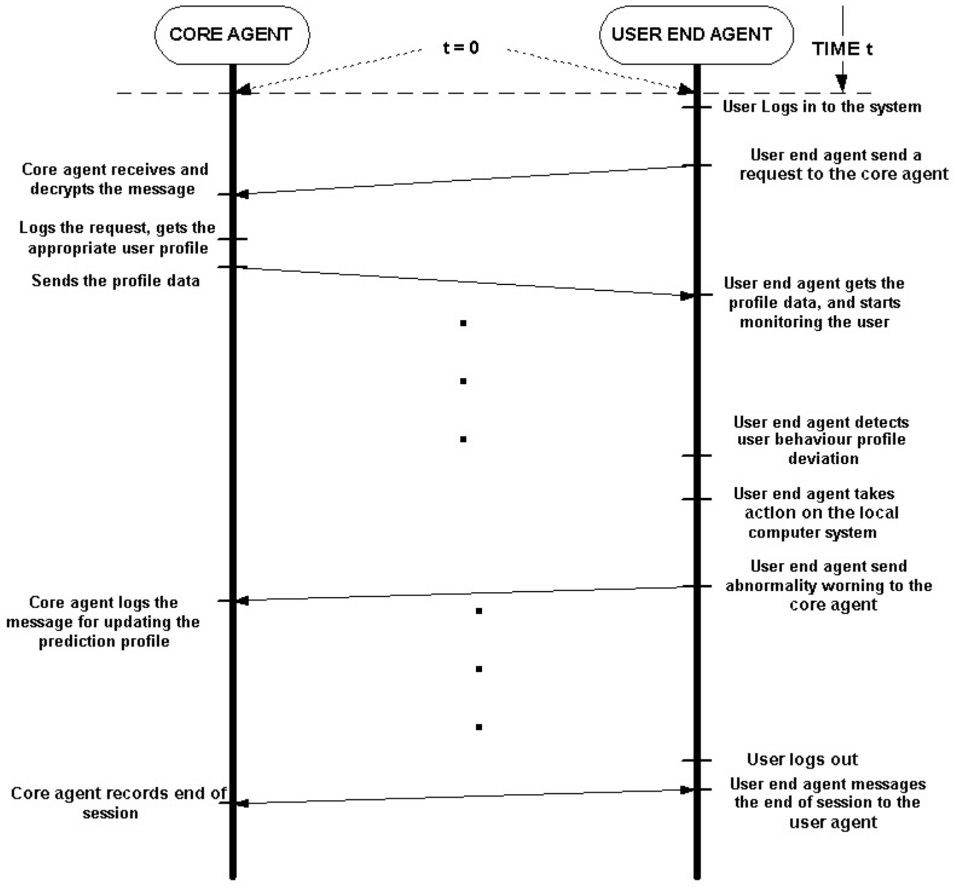
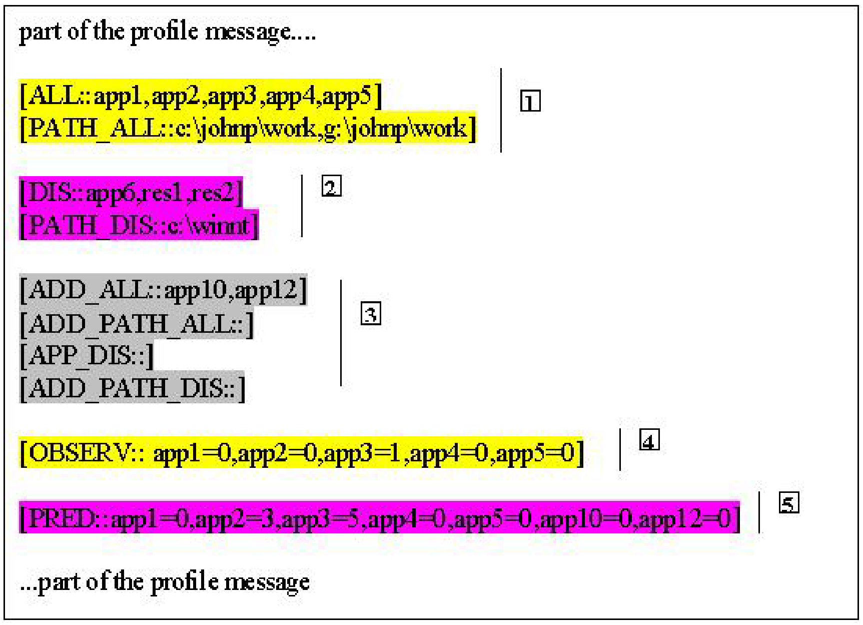

CORE AGENT
USER END AGENT
t = 0
TIME t
User Logs in to the system
User end agent send a request to the core agent
Core agent receives and decrypts the message
Logs the request, gets the appropriate user profile
Sends the profile data
User end agent gets the profile data, and starts monitoring the user
User end agent detects user behaviour profile deviation
User end agent takes action on the local computer system
User end agent send abnormality worning to the core agent
Core agent logs the message for updating the prediction profile
User logs out
User end agent messages the end of session to the user agent
Core agent records end of session

Figure 7.5 Interaction time chart between the two agents


part of the profile message....

[ALL::app1,app2,app3,app4,app5]
[PATH_ALL::c:\johnp\work,g:\johnp\work] 1

[DIS::app6,res1,res2]
[PATH_DIS::c:\winnt] 2

[ADD_ALL::app10,app12]
[ADD_PATH_ALL::]
[APP_DIS::]
[ADD_PATH_DIS::] 3

[OBSERV:: app1=0,app2=0,app3=1,app4=0,app5=0] 4

[PRED::app1=0,app2=3,app3=5,app4=0,app5=0,app10=0,app12=0] 5

...part of the profile message

Figure 7.6 Typical example of the rule part of the profile

7.2 Profile information

A profile is information that is compiled over time and describes the past user behaviour. It also contains basic rules that are set by the system administrators. All the profiles of the user are kept on the server that the core agent resides on. This information is passed to the user end agent when they request it.

The core agent and the user-end agents communicate with simple text messages. The format of the messages is a simple *send–receive* pairs. For example, when a user-end agent is loaded at the start of the user's session, it requests information from the core agent. It then sends this as a message that contains some information from the user-end agent machine such as the IP address, the name of the user, the date, and a message to request profile information from the server, in an encrypted format. The server sends the profile information back in an encrypted form.

The communication language that we used is a simple text language, as this allows the agents to interact, and exchange the information. After an initial interchange, the information exchange is small, so there is no need for additional formal language heuristics that can add complexity. In addition, most of the message is data, that includes the rules that are interpreted when their reach their destination, so there is no need for additional complexity.

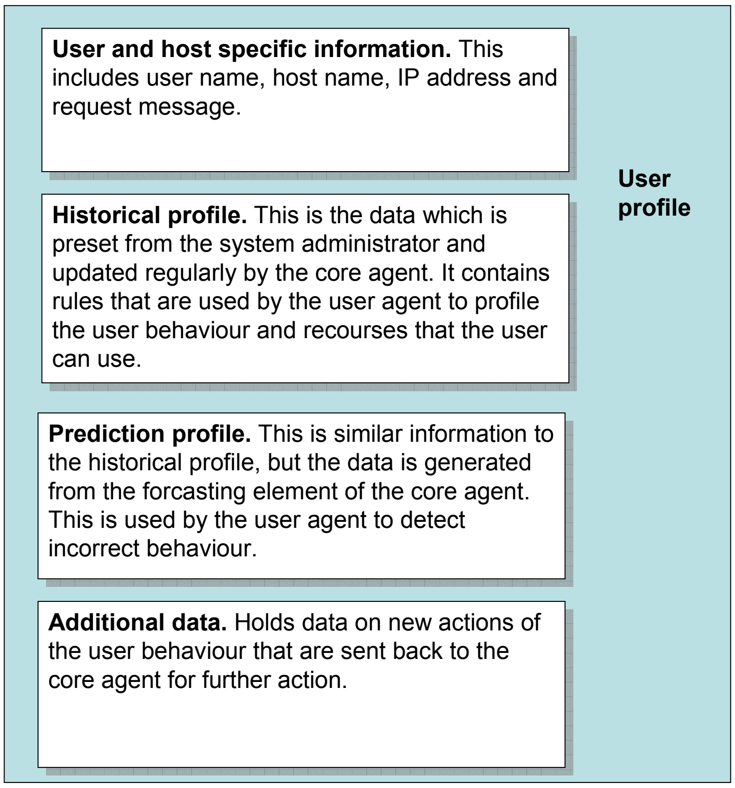

User and host specific information. This includes user name, host name, IP address and request message.
Historical profile. This is the data which is preset from the system administrator and updated regularly by the core agent. It contains rules that are used by the user agent to profile the user behaviour and recourses that the user can use.
Prediction profile. This is similar information to the historical profile, but the data is generated from the forcasting element of the core agent. This is used by the user agent to detect incorrect behaviour.
Additional data. Holds data on new actions of the user behaviour that are sent back to the core agent for further action.
User profile

Figure 7.7 Profile structure

7.3 Conclusions

This chapter has presented the distributed security system using agent technology. A 'core' agent on the server defines the security policy, while the 'user-end' agents actually implement the security on the hosts. Core agents act as data processing units and as data repositories for the user-end agents. User-end agents are individual software entities that act autonomously, by monitoring the user behaviour, and taking specific action, depending on user behaviour.

The forecasting part of the model resides on the core-agent side, and is responsible for processing the monitored data from the user-end agent, and generate forecasting data for the next session of the specific user. The 'user-end' agent gets this data when each user logs on to the computer network, and tries to predict future user behaviour. This data is not essential for the operation of our security enhancement system, but increases the protection of the system.

Both agents are created by different modules, and are highly configurable. Some future researchers might want to redistribute tasks for each agent, or create more specialized agents for different tasks. Future work might move the forecasting module from the 'core' agent and to the 'user-end' agent and repeat the experiments to discover if there is any difference in the results, and on system performance. In addition, there may be scope to create some different agents that are responsible for just processing data, on workstations that there idle, so the agents will be distributing not only the security, but also the workload of the processing requirements of the system.

7.4 References

[7.1] Pikoulas J, Buchanan W, Mannion M and Triantafyllopoulos K, 'An Intelligent Agent Security Intrusion System', 9th IEEE Conference in ECBS, Lund, Sweden, April 2002.

8 Experiments

8.1 Introduction

This chapter outlines the range of experiments for the security system, and details the first two experimental cycles. It covers the nature of our experiments, and describes the aims of these. Also discussed is the requirement for the logical flow of the experiments, along with the presentation of graphs with results that compare real observations and the results of the prediction statistical system.

The main objectives of the experiments are to examine:

- **The functional testing**. This proves the functionality of the model, and verifies it can be implemented in a typical domain.
- **The model against the functionality of the misuse purposes**. The misuse functionality was tested by monitoring user actions (such as the applications and resources that they used) in real-time, and blocking the use of the specific resource if this was flagged by the *comparator* mechanism on the user-end agent. The information for blocking the actions of the user is inside the profile that the user-end agent gets each time it is started at the logon of every user.
- **The model against the statistical purposes**. The statistical part was tested by processing the data in the core agent for each user, and creating or appending new information to the individual user profiles. After a user has logged-in and the user profile data was transferred from the core agent to the user-end agent, the *user-end* agent could use the comparator to not only check the user behaviour in real-time, but also check if the predicted profile is matching the one that is stored inside the user profile.

The experiments are divided in three different stages that targeted different results:

- **Stage 1**. This set of experiments verified that the model works, and to make tests on the entire agent environment.
- **Stage 2**. This set was an extension in the number of users and data that we processed with in the model. These are tests that are more extensive and verify some changes that were done in the statistical model and some regression testing to verify that the agent environment could work in a distributed system.
- **Stage 3**. This set of experiments verified that the model works, and gives the desired results. It also includes some regressions testing to verify that our agent environment could work with large volumes of data and users.

Figure 8.1 outlines the operation of the model, and the range of experiments. For this the user is monitored for their usage of an application. The window size (n) defines the number of previous observations used in the prediction, and the prediction number (z) defines the number of predictions into the future, based on a number of previous observations. The time between observations is defined by t. Our model has three stages of operation. The stages are:

- **Observation stage.** In this stage, the model is monitoring the user and records its behaviour.
- **Evaluation stage.** In this stage, the model makes a prediction and also monitors the user actual movements and calculates the result. This is a critical stage, as the model modifies itself according to the environment that it operates in.
- **One-step prediction.** In this stage, the model makes a single-step prediction. For example, assume that the user is logged in for 15 times and the model is configured, and it is ready to start predicting user moves. Instead of making a five or ten step prediction, like other mathematical models, our model makes a prediction for the next step. When the user logs in and out of our model, it takes the actual behaviour of the user, compares it with the one-step prediction that it has performed before and calculates the error. Therefore, the next time a prediction is made for this user it will also include the data of the last user behaviour. With this procedure we maximise the accuracy of the prediction system.

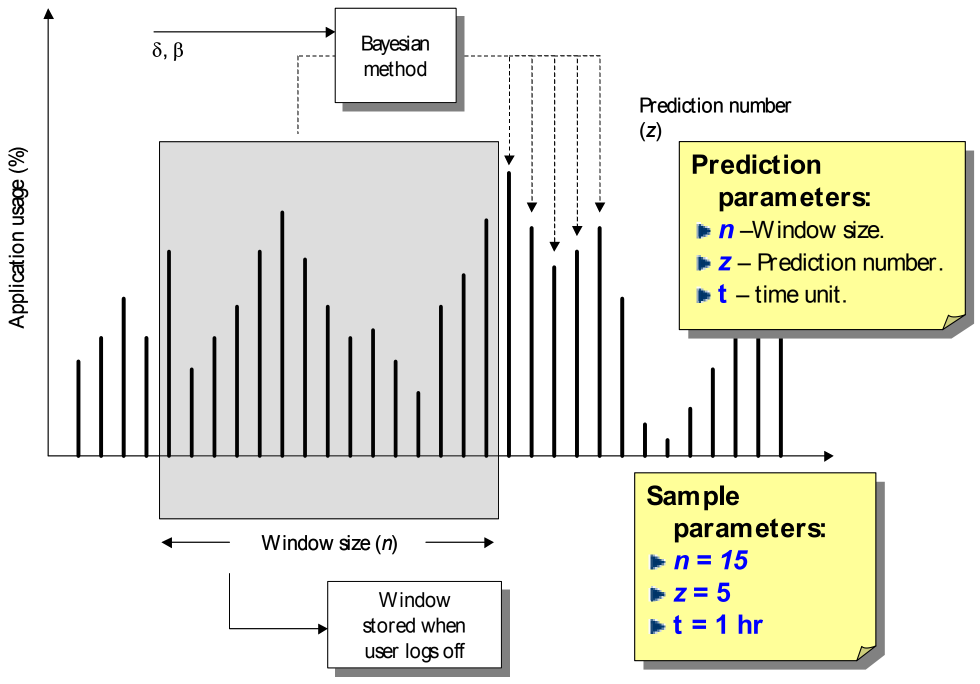


Figure 8.1 Experiment outline

8.2 System functionality

In the first two experiments, we test the functional aspects of the system. Functional testing involves tested that all the different software parts are operating correctly. The basic functional tests were, in sequence:

1. Test the core agent and measure the system resources. In this we found that when there were no user-end agents live, that core agent consumes less than 2% of the system resources. This is because the core agent is listening for any user-end agents every five seconds, and because it is multithreaded, the load in the system is minimal. After this, we started to start up user-end agents. The different software parts (core agent and user-end agent) where located on different computers on the same network. It was observed that each user-end agent was finding the core agent on the network, and was connecting successfully, and retrieving the required information.
2. Next a batch process was created to stop and start user-end agents. This allowed a regression test for the connection of our system. For this, three hosts were used; one of which contained a core agent and the other two with user-end agents. This was left to run for three days. For this user-end agents created connections with the core agent and destroyed them when finish. At each time there were 20 user-end agents running on each of the two computers. After the three-day regression experiment, we counted that the system made around 20,000 connections between the user-end agents and the core agent. At the end of the third day, the system was still up and running, so we concluded that our system could run robustly and the limit of the agent connections was only imposed by the performance of the computers and the memory that each host had to run our application.

8.3 Real-time detection

One of our key functionality tests in our security system is the real-time monitoring of each user. To test this we used users that were using the system and logging onto a university network to perform some predefined actions. They were given a degree of freedom by using anything additional to what the predefined steps were. The users were students of the School of Computer at Napier University, and they were familiar with computer use, and with the experiments that were run. They also had some experience of computer misuse. The time taken to gather the data took almost an academic year, mostly because of the difficulty to find spare time for the users to log in and test our system.

The experiment involved giving a predefined script to use applications such as Microsoft Excel, Microsoft Word, Internet Explorer and Borland's C++ Builder. For this, the user-end agents were configured to report any other software application that they used from the user, except these predefined applications. Every user-end agent scanned the host that they resided on for the status of the applications that are running, every five seconds. This time interval is user definable, but we chose this time interval because it does not add too much of an overhead on the system, and it is sufficient to categorize it as real-time detection. The processing overhead of the system is small, as threads are used to make every action, such as scanning for what applications are running, or contacting the core agent. The thread mechanism gives us the advantage of spreading-out processor-intensive tasks, over time, and share the processor with the rest of the applications on the same host. In addition, by using threads, we are letting the host give the same execution time to our process as to the rest of the processes that run on the host. Our system successfully detected all the applications that were running on each user host, and alerted the users and the system administrator of any applications that were not authorized, even when that applications were started from the command shell. The use of the command shell, as apposed to the use of icons, is often a pointer to an advanced user, and could highlight a possible breach of security if the user changes their behaviour from mostly using icon-based programs to command-line ones, or vice-versa.

In all cycles of our experiments, the real-time detection and the interconnectivity between agents was tested as the system had to be operational to get the statistical observations. For making the agents talk to each other, we used Java RMI (Remote Method Invocation). From this, it was possible for agents to exchange data, and use encryption to encrypt and decrypt the messages. As we can see in Figure 8.2, we have detailed the description of the creation of a message from user-end agent to sending it, and receiving it, from a core agent. This process gets more complex when there is more interaction between agents, and when the number of agents increases. All the communication between a core agent and a user-end agent takes at least the actions that described in the process in Figure 8.2.

Real-time detection is one part of the enhanced security system, and gets its results from the comparator, the part of the user-end agent that gets data from sensor and from the user profile and determines the behaviour of the user.

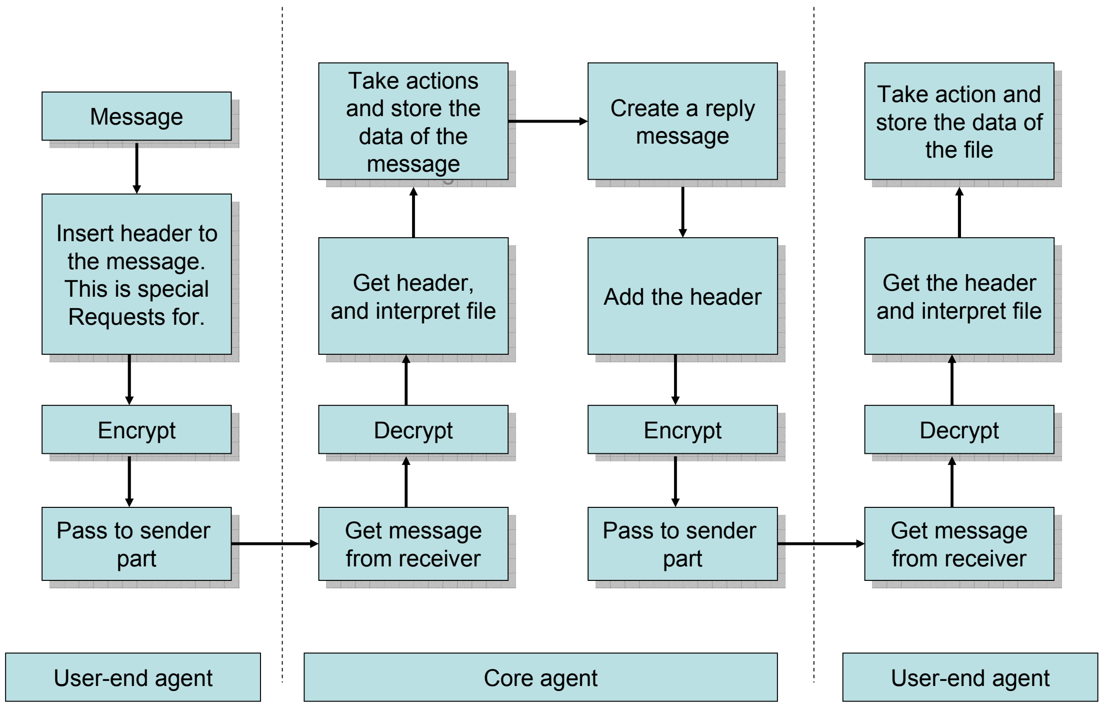

Message
Insert header to the message. This is special Requests for.
Encrypt
Pass to sender part
Take actions and store the data of the message
Get header, and interpret file
Decrypt
Get message from receiver
Create a reply message
Add the header
Encrypt
Pass to sender part
Take action and store the data of the file
Get the header and interpret file
Decrypt
Get message from receiver
User-end agent
Core agent
User-end agent

Figure 8.2 Creating and transporting messages process

8.4 Our Experimental Topology

In our first set of experiments, we used only two PCs running Windows NT Version 4. They both connected to the university network through a 10Mbps hub. We used one user, which worked on the workstation. There were only two applications that there were monitored. The user was logging in the computer for one week.

In our second set of experiments, we used the same hardware configuration, with one user, but this time we used five applications to be monitored as system resources. In our last series of experiments, we used four computers one that played the role of the server and three that played the role of the workstations. Again we used three users, but there are 10 applications monitored. The total amount of monitored time is 50 hours. These experiments took approximately eight weeks to complete, because of the workload of the users.

The users in all of our experiments did not have a specific path of actions to perform when they were logged in the system. They had a minimal instruction set in order to use some specific applications, but not specific scripts. An important factor is to allow user impulses. The results of this is that if we observe that some times the user did not use all of the monitored applications at all during the monitored session.

8.5 Our first set of statistical experiments

Intervention is a mechanism for increasing the prediction accuracy. This uses any additional information about the future behaviour of the system that can be added to the model prior the prediction. For example, if there are some users that are keen of using illegal software, or there are new users that there is not enough information on then we can apply the intervention mechanism, and thus increase the accuracy of the model and make more accurate predictions. Figure 8.3 illustrates intervention. We can observe from Figure 8.3 the sliding window is greyed. This is the initial minimal window that our prediction model uses to start producing acceptable results. In our case in these first experiments we used 20 initial observations, after which we generated some results. The prediction that we have is done for the next five observations. Our model is making a one-step prediction. This means that in order to find the next value or to make a one-step prediction, we need to know the previous observation. In other words, in order to make an observation for $t+1$ we need to know the value of the observation on time t. That does not stop the mathematical model from making predictions for more than one-step. In these experiments, we have done only five-step prediction as we needed to verify that we are getting the expected results. In following experiments, we are increasing the number of predictions. In our third set of experiments, we are making 100 step predictions of our observations.

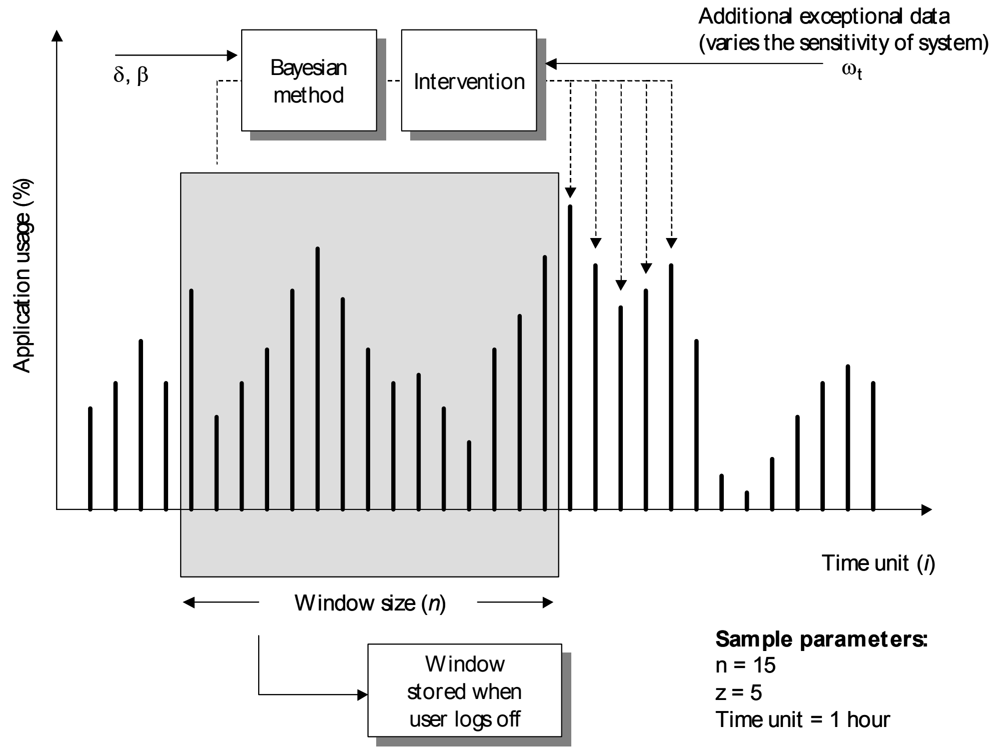


Figure 8.3 Intervention

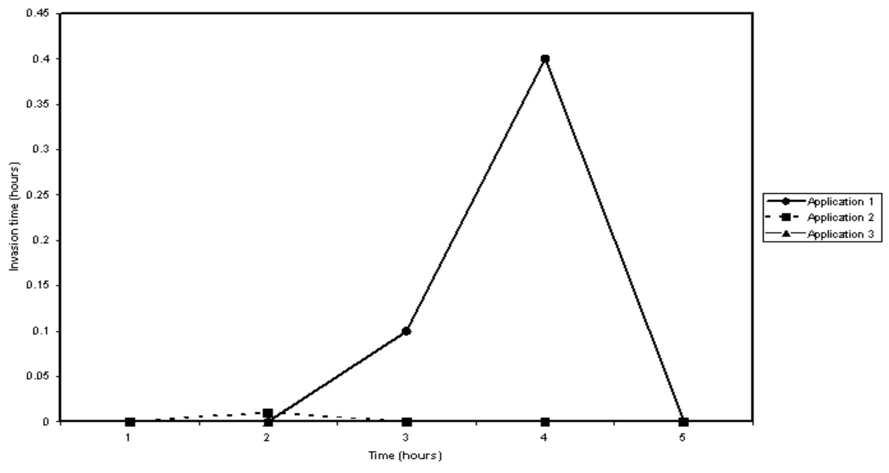


Figure 8.4 The Real Observations of the Model

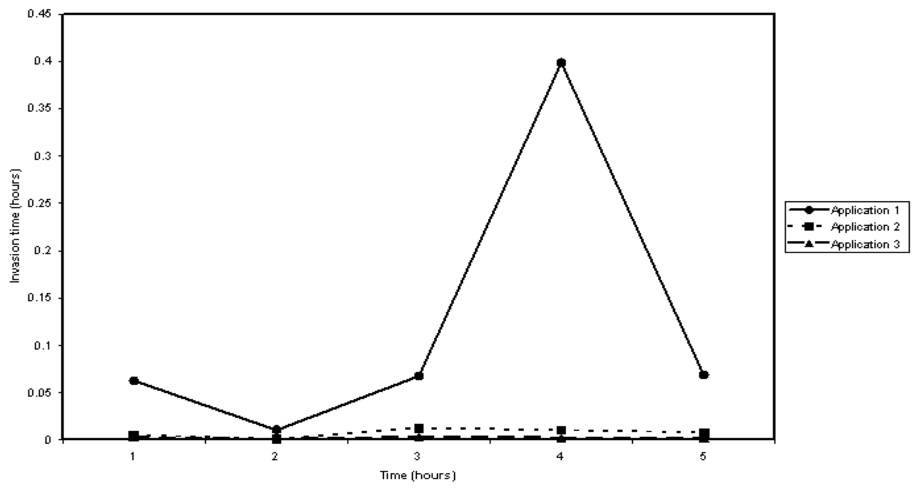


Figure 8.5 The proposed Model 5 step prediction

In our model we can observe that in Figure 8.4 and 8.5, that our model prediction is very close to the actual users behaviour for the application number one at the specific time *t=19*. We achieved this accuracy by applying the intervention technique.

The first experiments are made in order to test our security environment on what extend it works and to get some results from our proposed statistical model. These results can also be compared to other statistical models. In this case, there was one user that logged on to the system for 20 times and had a one-hour session each time. We monitored all their operations and all the applications that they used. In our prediction model, we had only three applications to predict. The results of which are shown in Table 8.1. The intervals are from zero to unit and they denote an hour. Therefore, for example, 0.3 means that the user used this program for 0.3 of the hour (18 minutes), in this specific hour of the system usage.

We used our prediction mechanism for the last five observations. As we can observe from the results, if we compare the graph in Figure 8.4 (for ARIMA), which is the real observations for the three applications, and the graph in Figure 8.5 (our model), we can see that the two figures are almost identical. We can see that they are less precise with the actual readings and they fail to predict the action of the user in application 1 at the time interval 19, in comparison with our model that predicted it with a very close figure.

Table 7.1: Invasion time (hours) in software agent environment

Hour	*Application 1*	*Application 2*	*Application 3*
1	0	0	0
2	0	0	0
3	0.01	0	0
4	0	0	0
5	0	0.1	0
6	0.2	0	0
7	0.25	0	0
8	0	0	0
9	0	0	0.05
10	0	0	0
11	0.01	0	0
12	0	0	0
13	0.3	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0.01	0
18	0.1	0	0
19	0.4	0	0
20	0	0	0

8.6 Our second set of experiments

In our previous experiments we used only one user and a limited observation of 10 times. This time our experiments are more generic. By showing with our previous experiments that our model works, we now use three users and with 100 observations. In this experiment the prediction model uses 50 observations in order to adapt to the user behaviour, and then apply the prediction model to the next 50 observations. For this experiment, time interval for the usage of the application is one hour.

The results graph the last 50 real observation values and our predictions, and in the *y*-axis defines the usage. The users were monitored for the following applications: Microsoft Word, Microsoft Excel, Microsoft Outlook, Microsoft Internet Explorer and

Borland C++. The users were free to use other applications and system recourses if they wanted to, but our system where monitoring only the above applications. There were no restrictions on when they use it, or how much time.

For these experiments we assumed that only the first application was legal to be used by all the users (Microsoft Word), and the rest were available to the system but restricted from the users. We again used the time period of one hour to make our prediction.

Figures 8.6 and 8.7 show the results without any sort of intervention. We can see that our model is capturing all the different anomalies, with sometimes, luck of capturing the length of the high peaks of the real-data graphs. The graphs show as prediction results that are taken from our agent environment. In the first two figures we can see that our model is able to keep with the changes of the user. Although our prediction is accurate, we have introduced the notion of the intervention that we apply in our last result (Figure 8.8). We can observe that the two lines are following each other. That means that our model is predicting very accurately the future behaviour of the user.

In all of our examples, we can see that our model is able to detect the deviations of the user behaviour, with maximum effects in the case of the use of intervention (see Figure 8.8).

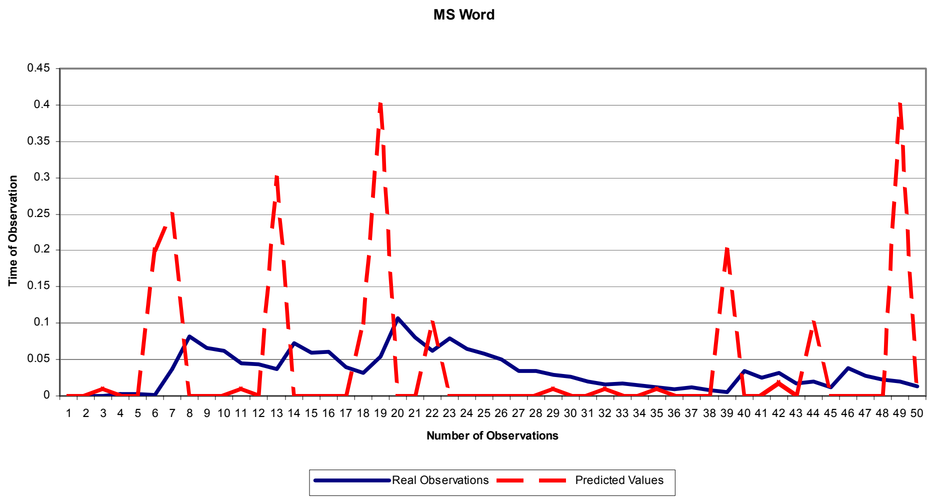


Figure 8.6 Application one Prediction and real values

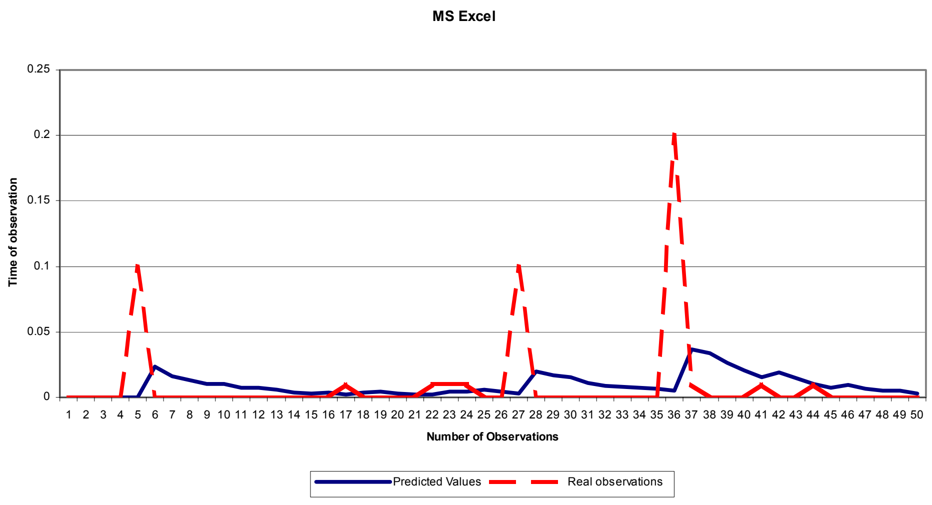


Figure 8.7 Application two Prediction and real values

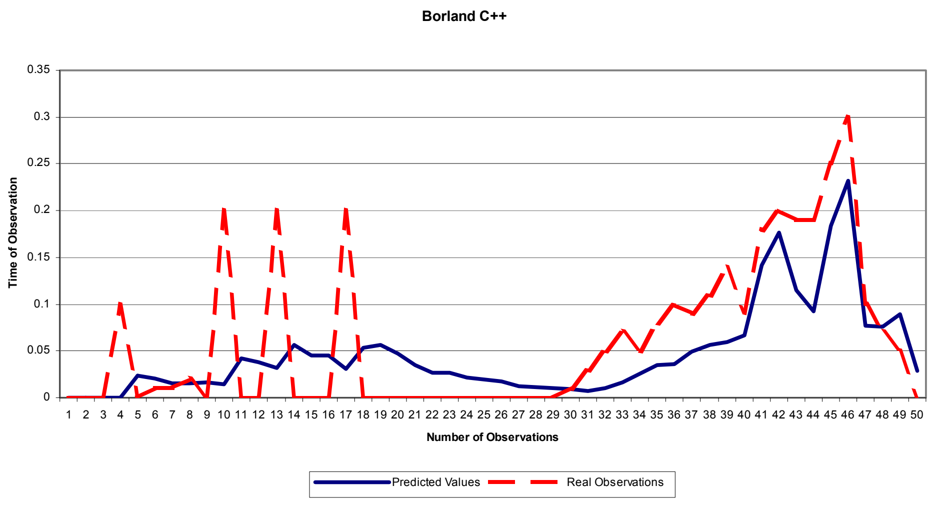


Figure 8.8 Application five Prediction and real values (with intervention)

Figures 8.9, 8.10 and 8.11 show the results that combine ARIMA model results, Bayesian model results, and the real results. We can observe that the ARIMA results, do not even try to follow the real observations, and in some cases we can observe negative values, which we cannot have since we cannot negative time. It does not make sense in our experimentation frame of mind. The results that we got for the ARIMA models, we obtained using 'S-PLUS 2000' statistical software package. We inserted all the range of the observations, and we made a prediction for the last ten observations, as we had done to our experimentation for our Bayesian model.

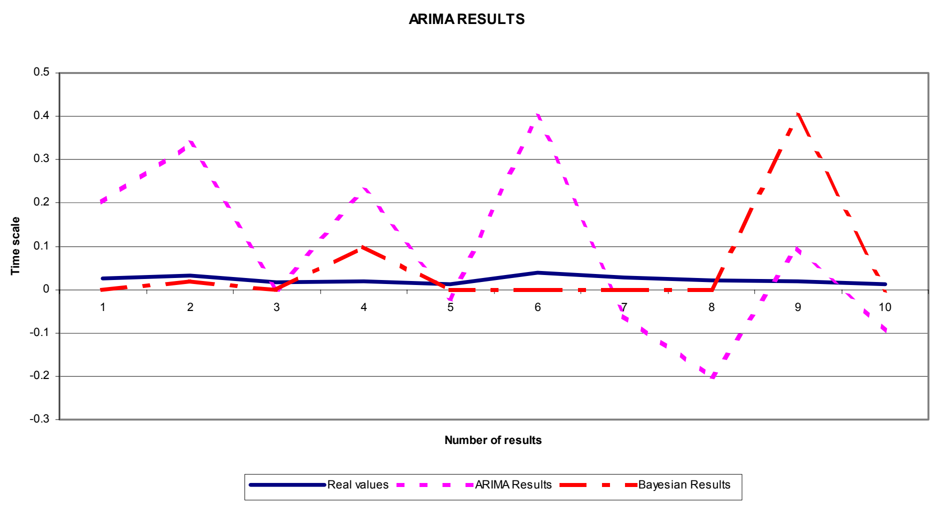


Figure 8.9 Arima and Bayesian results comparison chart for application 1

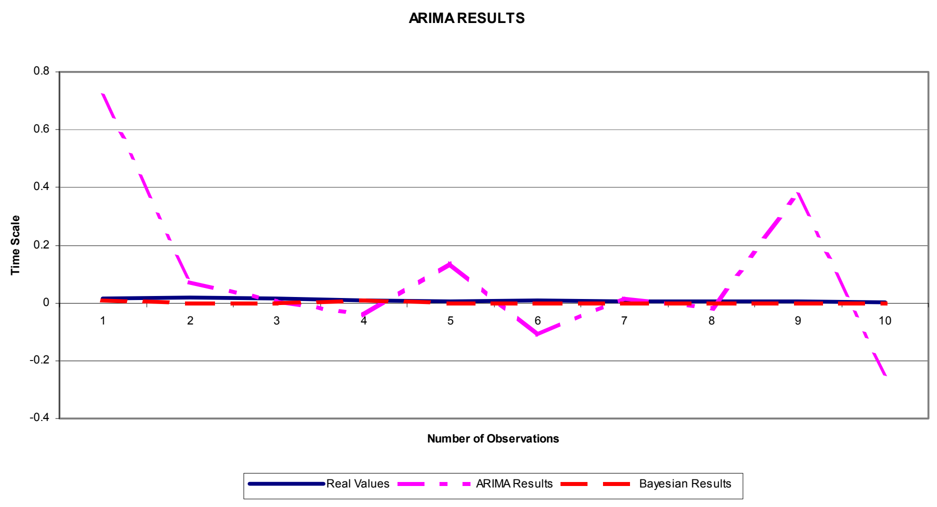


Figure 8.10 Arima and Bayesian results comparison chart for application 2

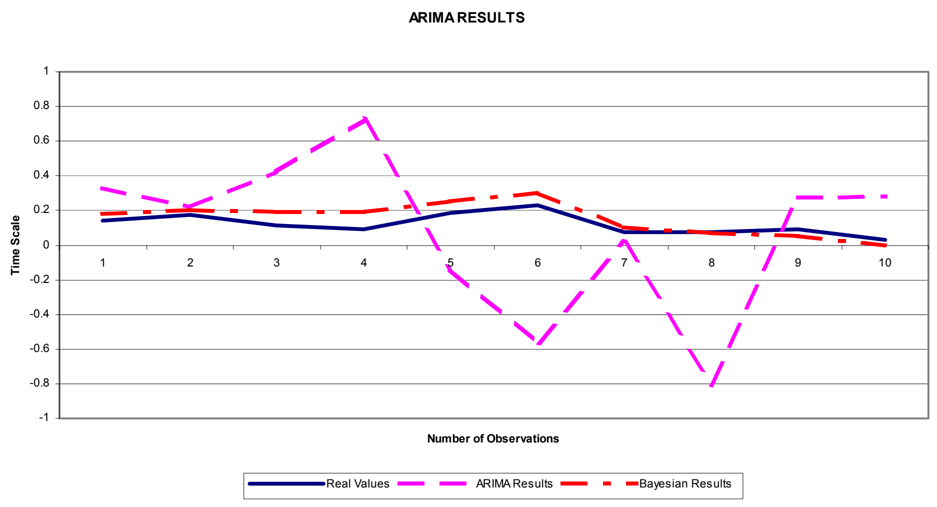


Figure 8.11 Arima and Bayesian results comparison chart for application 5

We can also find the deviation for the ARIMA model and the Bayesian model. For this we use the absolute values of the difference, so we can see more precise the accuracy of our method. This gives Figure 8.12, Figure 8.13 and Figure 8.14.

Since we represent the three last graphs as absolute values, we expect our graphs to be as closer to the x-axis of the graph, as possible. Again, we can see that the ARIMA models have a very large deviation from the x-axis, in comparison with the prediction results of our model.

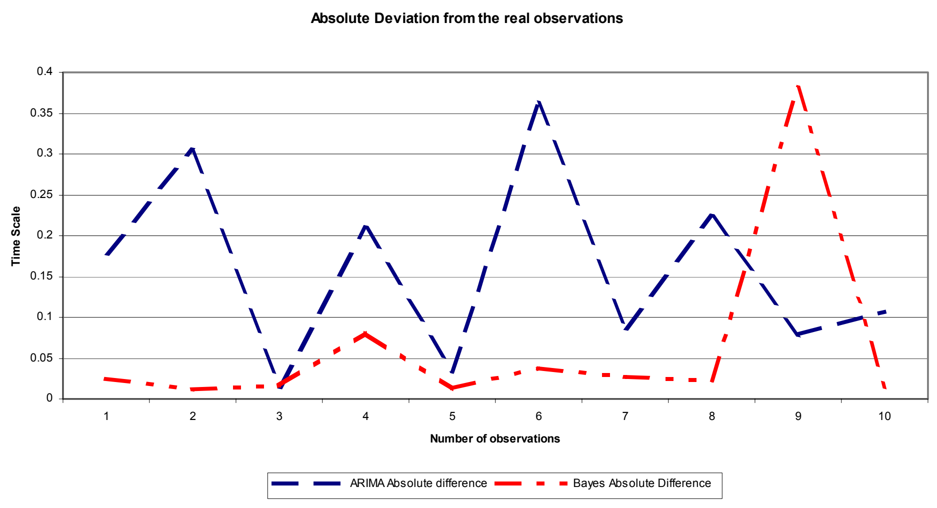


Figure 8.12 Absolute difference from the real observations for App1

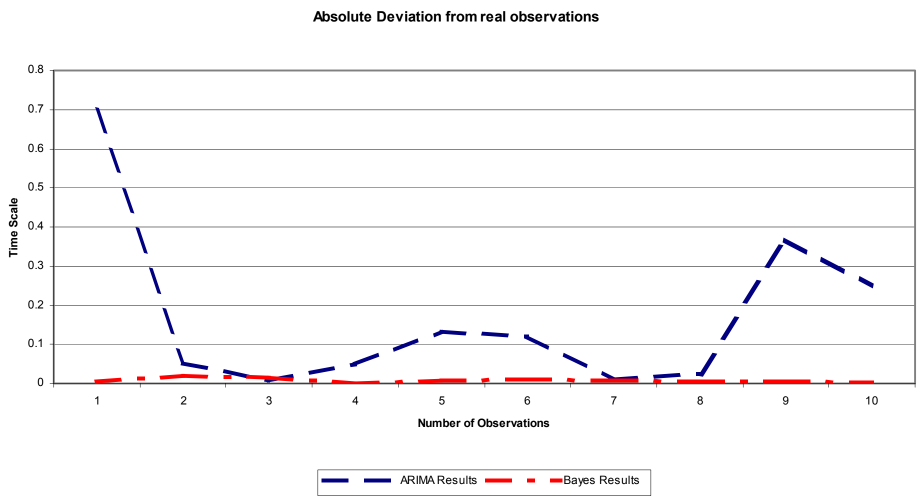


Figure 8.13 Absolute difference from the real observations for App2

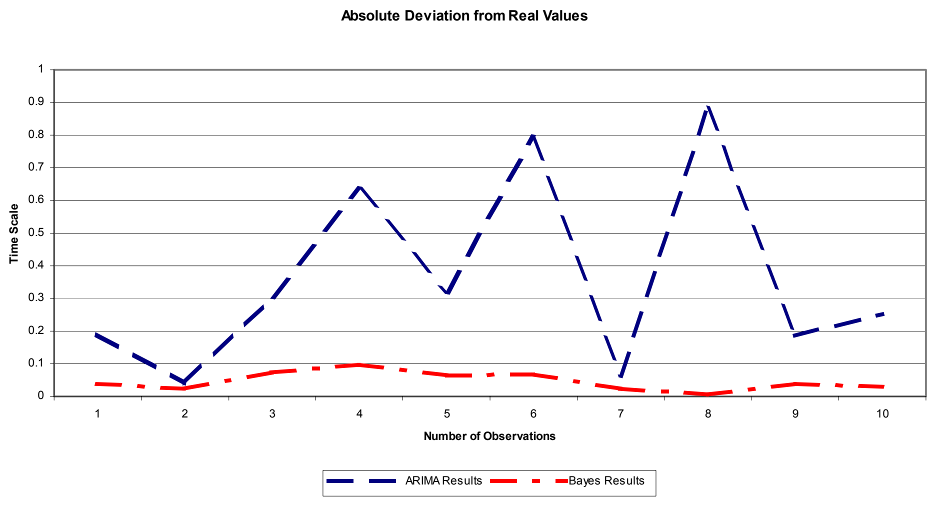


Figure 8.14 Absolute difference from the real observations for App5

8.7 Conclusions

These were the first two stages of our experimentation. We have found that our security enhancement is working properly, and producing correct results. We also made theses experiments in order to verify the other part of our security enhancement system, like to see if the different agent can communicate with each other, and transfer the correct information.

We have found that at a low number of observations, such as smaller than 30; we have some times, similar results with the ARIMA models. This is expected as the number of results is too small, and the values can sometimes take random values that are close to the correct ones.

We also found that the load on each workstation that each agent is residing is very small, and we typically observed an increase on workload of just 5%, and a 2% increase in the core agent. This activity occurs when the user has increased activity and when the agent is trying to monitor all their moves. In addition, the memory usage is regulated by Java's virtual machine, so we do not have any memory leaks, and our agent is using the exact memory that it needs.

On server side, we observed that our agent was creating the profiles for each user, was not larger than 800 KB. This considered a small amount of data, even if there are a large number of users. In addition, there was no problem trying to acquire large number of user profiles at the same time, as we use multithreading on the 'core' agent.

We found also found an error of between 0.2% and 2.5% of the real value of use behaviour.

9 Results

9.1 Introduction

This chapter outlines the results from a range of experiments which measure the performance, in terms of software overload, and prediction accuracy, of the prediction model. Early sections have described the elements of the statistical system variables that are tested, and how the different experiments we achieved we done. As mentioned in Chapter 8, the experiments verified that the model works, and gives the desired results. In addition, some regression testing has been done to verify that our agent environment could work with large volumes of data and users. The number of data that are processed in these experiments is large, so we cannot present all the data in this chapter.

The first step in the experiments was to try to quantify the different aspects of our system in order to test them. After this, the values which were expected were derived, and these were compared with the actual results from experiments. This will give a theoretical outcome of each experiment, and after the practical result is determined the two are compared to determine how success the model is.

The key factors for the statistic part of the experiments are:

- **Number of observations** (NoO)**.** Number of initial user behaviour observations, before the system start prediction results.
- **Time scale** (TS)**.** Time between two observations. For example, in our first two experimental cycles, we used a TS of one hour.
- **Prediction number size (in observations)** (PSz)**.** The number of predicted observations. For example, in our first test cycle, we used PSz of 10 and in the second test cycle the PSz was 25.

Table 9.1 to Table 9.6 outlines the range of experiments. We are also going to perform some extreme tests in order to find the limitations of out statistical engine, and try to find the reasons why these limitations exist. Figure 9.1 and Figure 9.13 outline the results of these tests.

Table 9.1 Keep TS and PSz constant, and change the NoO

NoO	*TS*	*PSz*
50	one hour	25
100	one hour	25
150	one hour	25
250	one hour	25

Table 9.2 Keeping TS and PSz constant, and change the NoO

NoO	*TS*	*PSz*
50	10 minutes	25
100	10 minutes	25
150	10 minutes	25
250	10 minutes	25

Table 9.3 Keeping NoO and PSz constant, and change TS

NoO	*TS*	*PSz*
100	one hour	25
100	2 hours	25
100	3 hours	25
100	5 hours	25

Table 9.4 Keeping NoO and PSz constant, and change TS

NoO	*TS*	*PSz*
100	10 minutes	25
100	15 minutes	25
100	20 minutes	25
100	30 minutes	25

Table 9.5 Keeping NoO and TS constant and change PSz

NoO	*TS*	*PSz*
100	10 minutes	25
100	10 minutes	50
100	10 minutes	100
100	10 minutes	250

Table 9.6 Keeping NoO and TS constant and change PSz

NoO	*TS*	*PSz*
100	one hour	25
100	one hour	50
100	one hour	100
100	one hour	250

One comparison of the two values that we getting in every time t is to compare the slope of the two graphs. In this way, even if the values do not match, if their slopes are close together, we have a very strong indication as to what behaviour the user will have. Therefore, we are including in our graphs some slope graphs. The purpose of putting the comparison of the two result slopes is to emphasize the fact that some times the graphs between the real values and the prediction statistical system values are not the same.

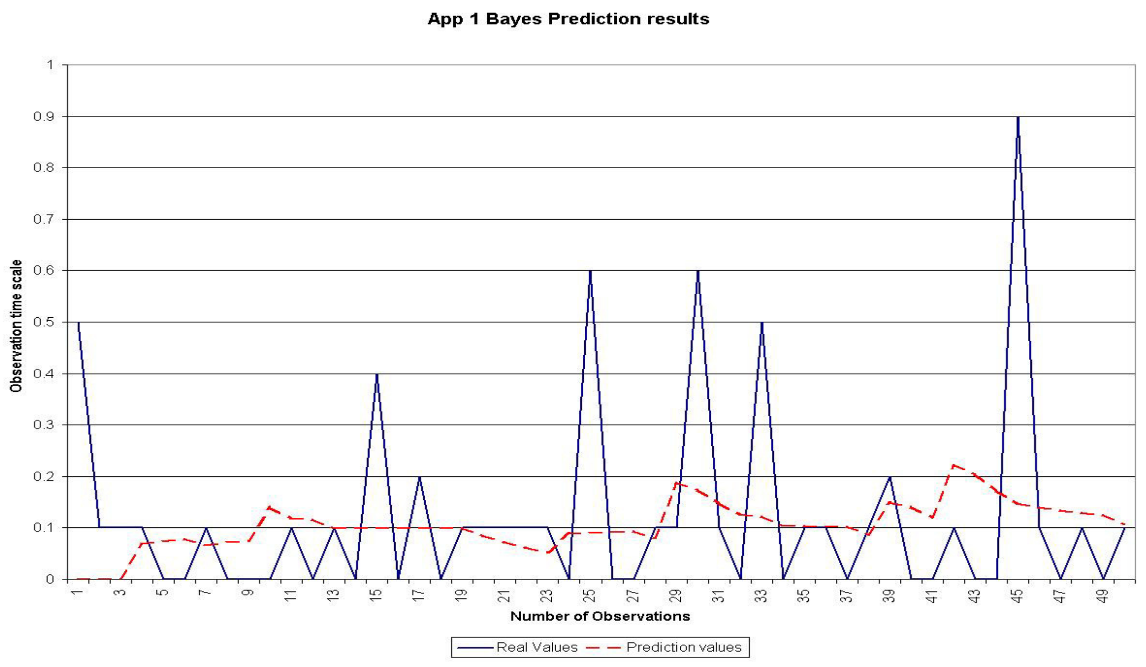


Figure 9.1 Prediction results with TS one hour and PSz of 50

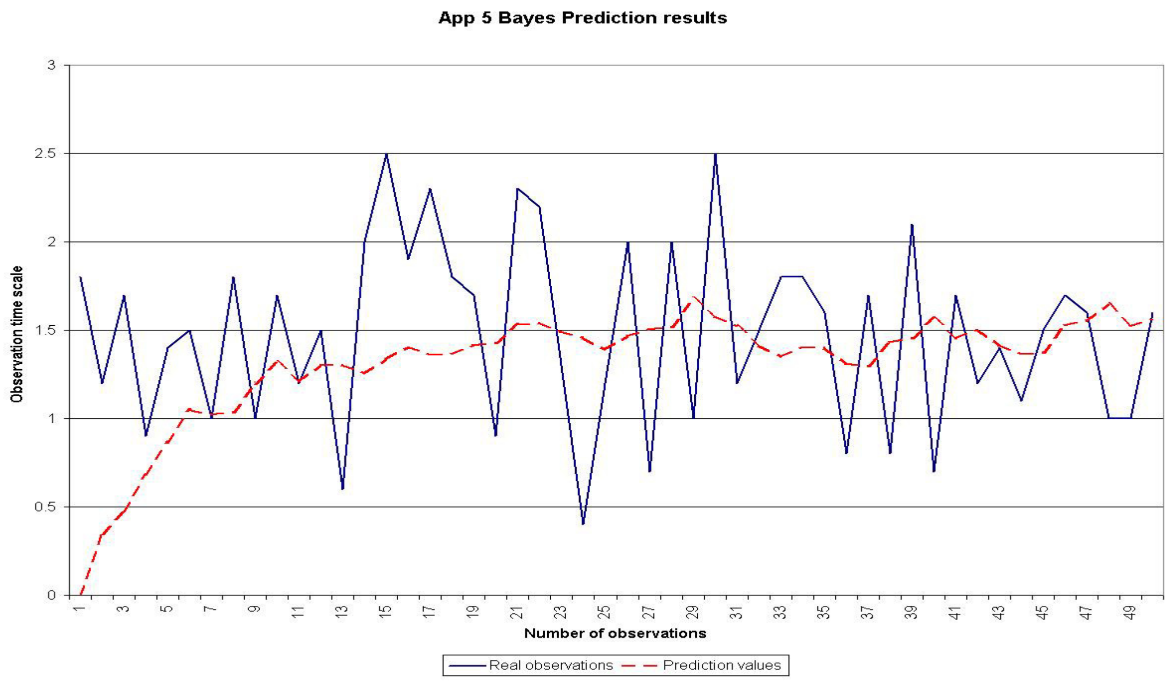


Figure 9.2 Prediction results with TS one hour and PSz of 50

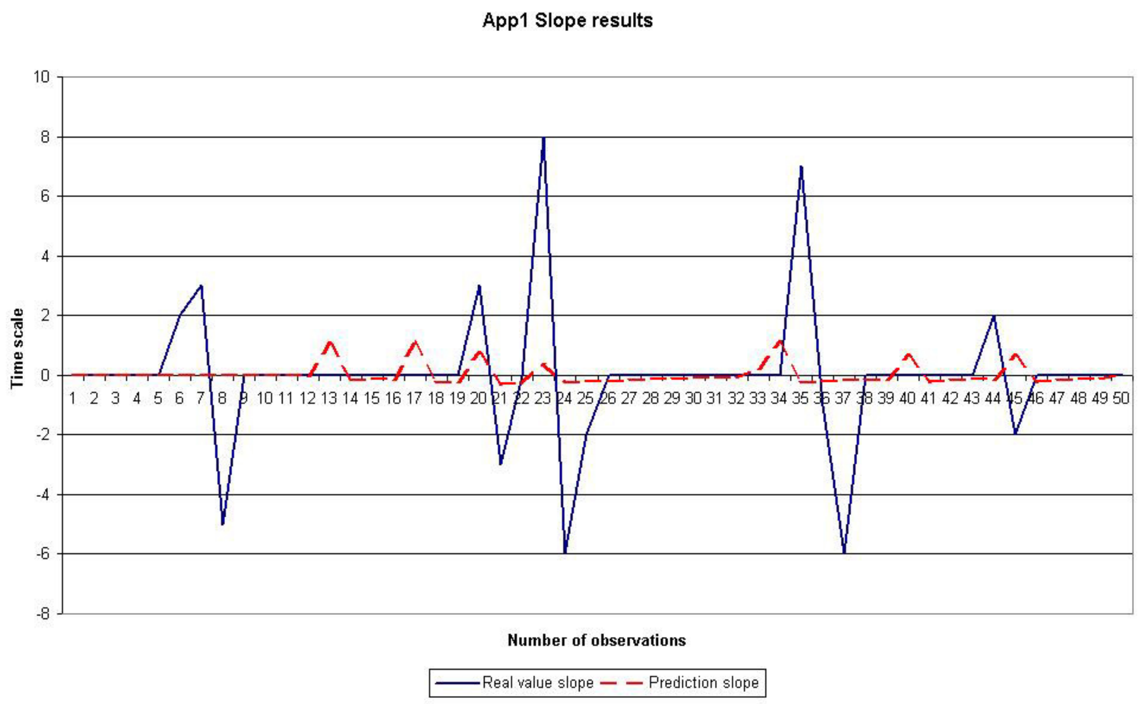


Figure 9.3 Slope graph comparison between the two results where TS is 10 minutes for app1

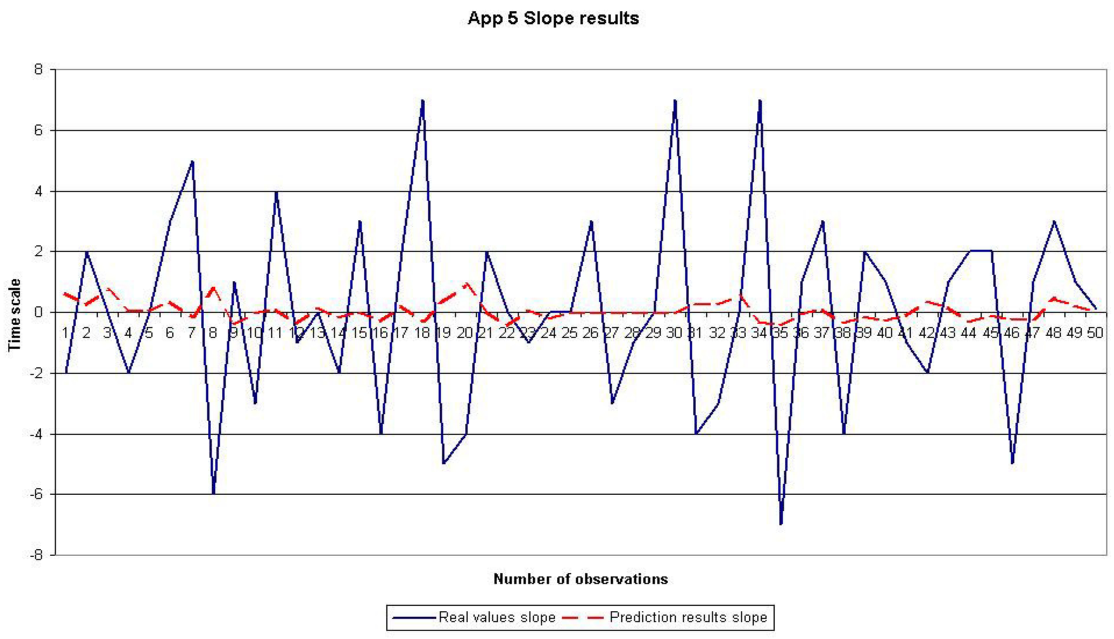


Figure 9.4 Slope graph comparison between the two results where TS is 10 minutes for app5

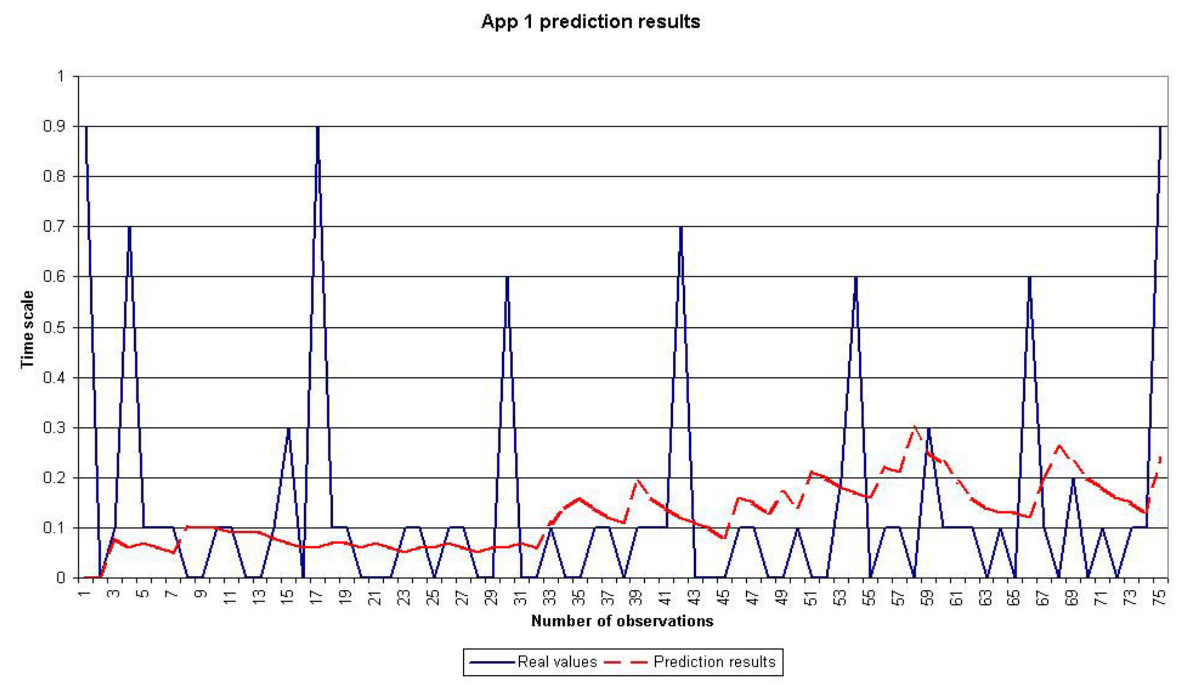


Figure 9.5 Prediction results with TS one hour and PSz of 75

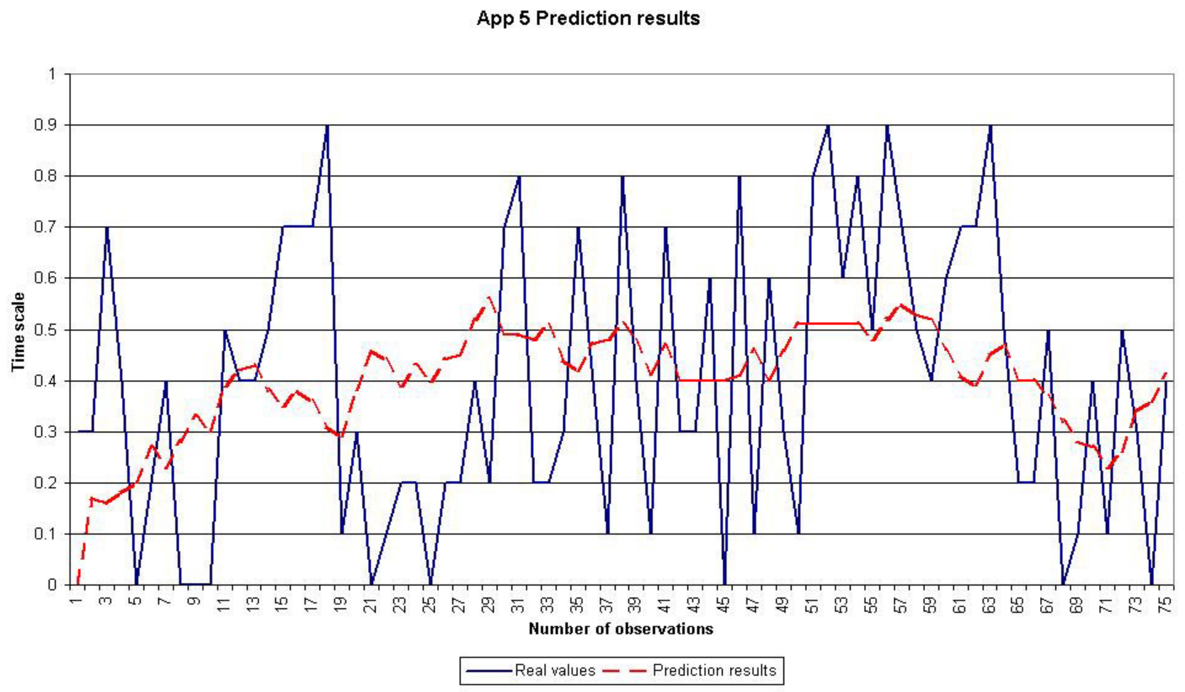


Figure 9.6 Prediction results with TS one hour and PSz of 75

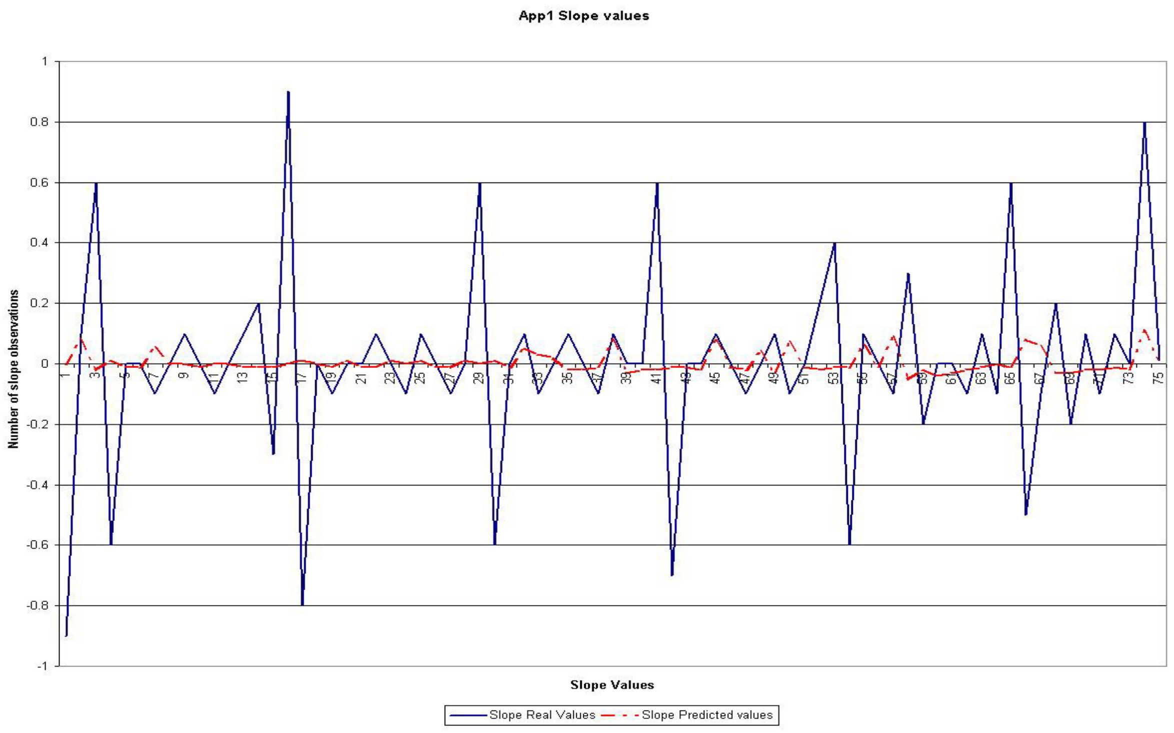


Figure 9.7 Slope graph comparison between the two results where TS is one hour

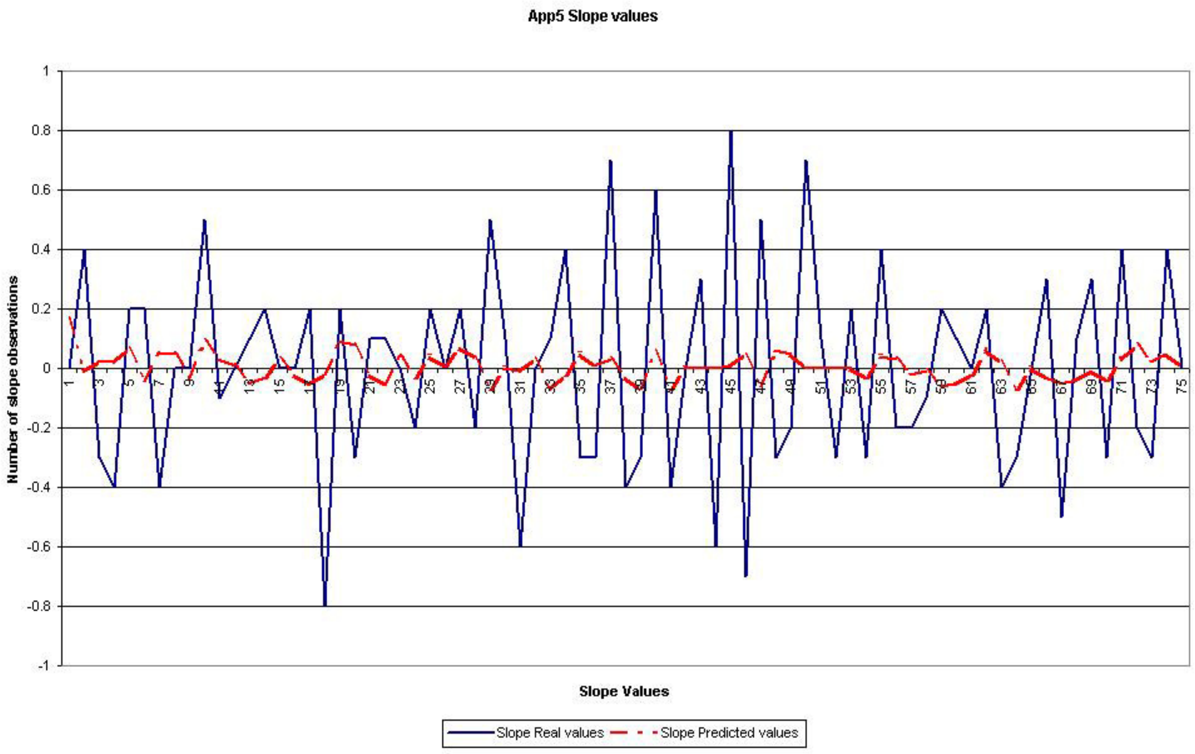


Figure 9.8 Slope graph comparison between the two results where TS is one hour

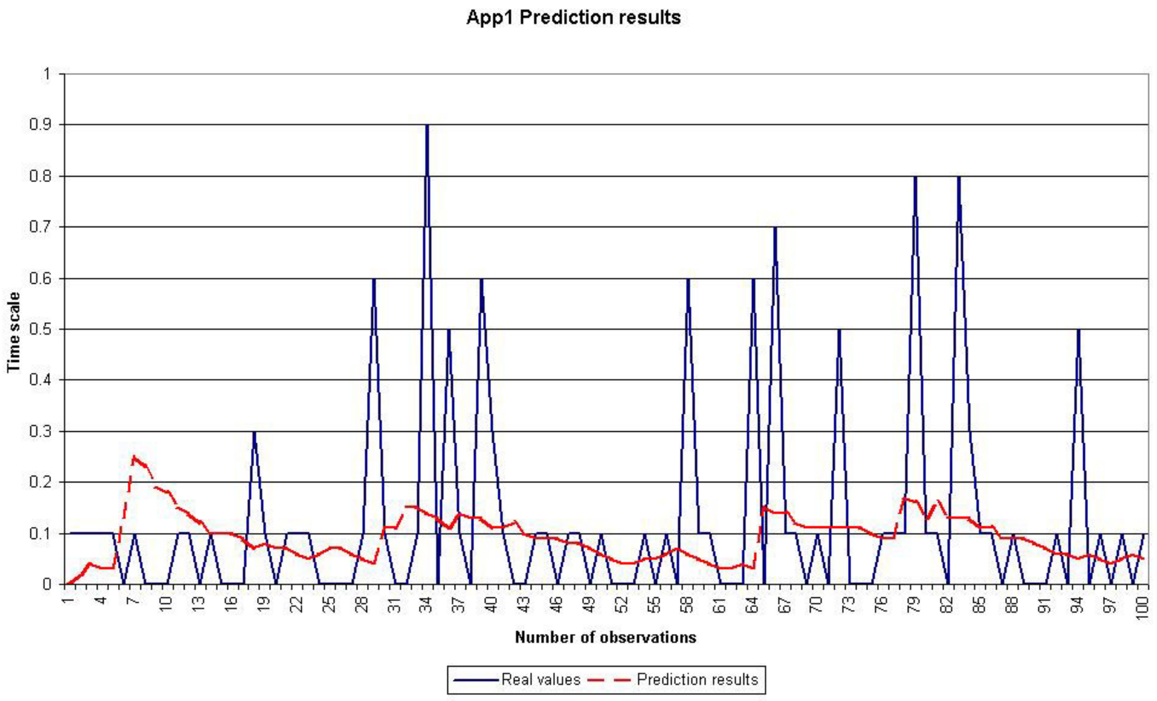


Figure 9.9 Prediction results with TS one hour and PSz of 100

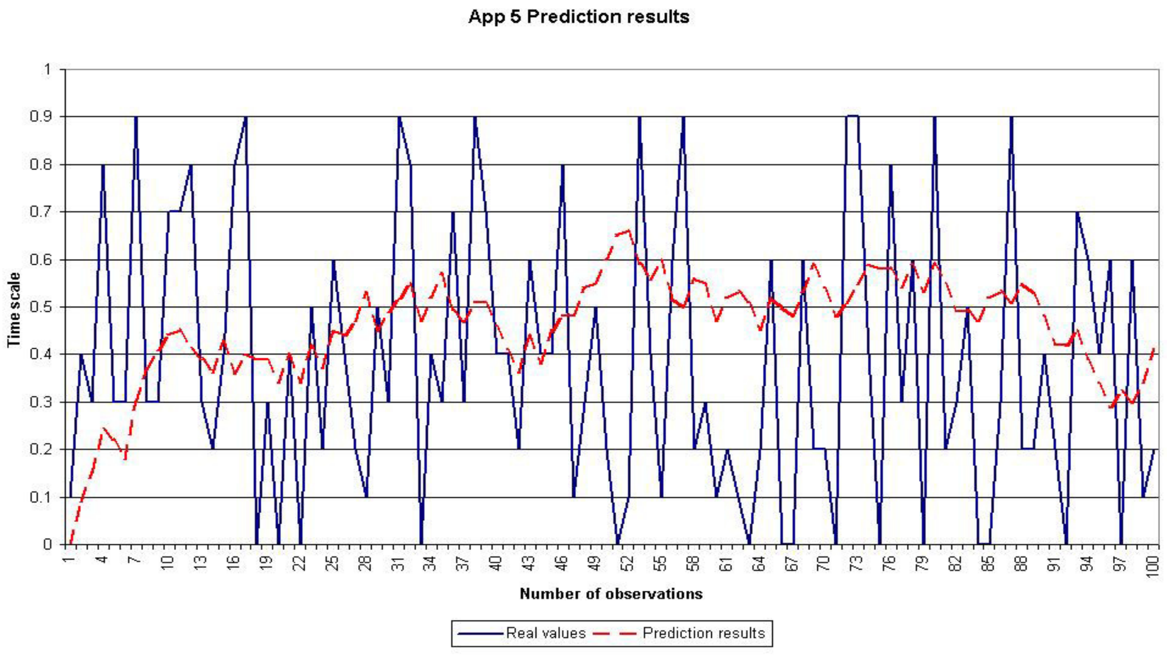


Figure 0-10

Figure 9.11 Prediction results with TS one hour and PSz of 100

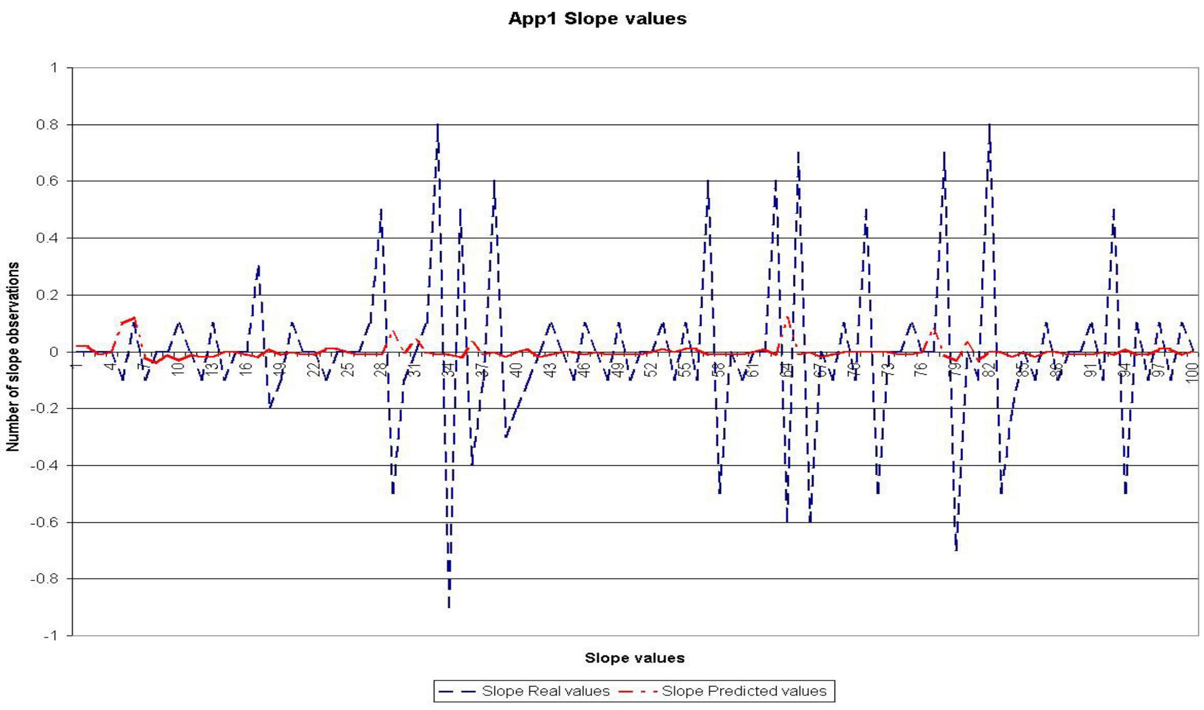


Figure 9.12 Slope graph comparison between the two results where TS is one hour

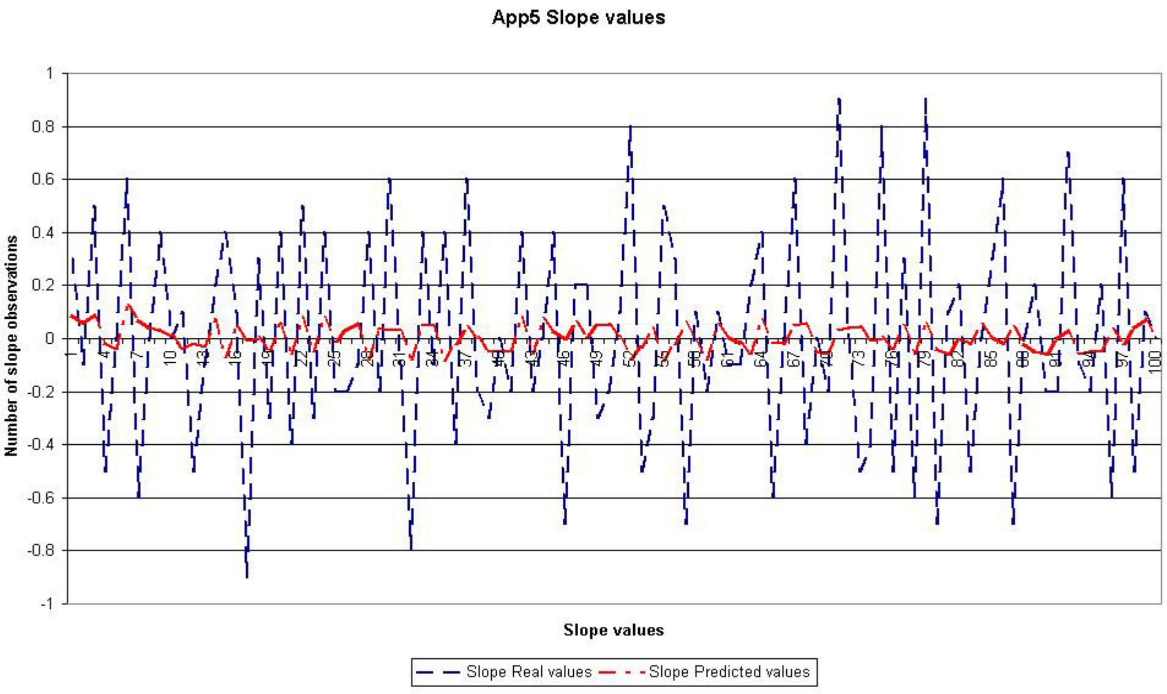


Figure 9.13 Slope graph comparison between the two results where TS is one hour

9.2 Experimental results of predictions with a time scale of 10 minutes

In this part of our experiments, we choose the time scale of our statistical model to be 10 minutes instead of one hour. This, thus, takes a snapshot of the user behaviour every ten minutes, and predicting the user behaviour for the next 10 minutes. The number of the experimental results is the same in both the previous experiments and these ones. As of the number of the results was large, we have chosen a part of our experimental results, and graphed it. Figure 9.14 to 9.24 gives these results. An analytical list of all the experimental results can be found on the additional material provided with the thesis.

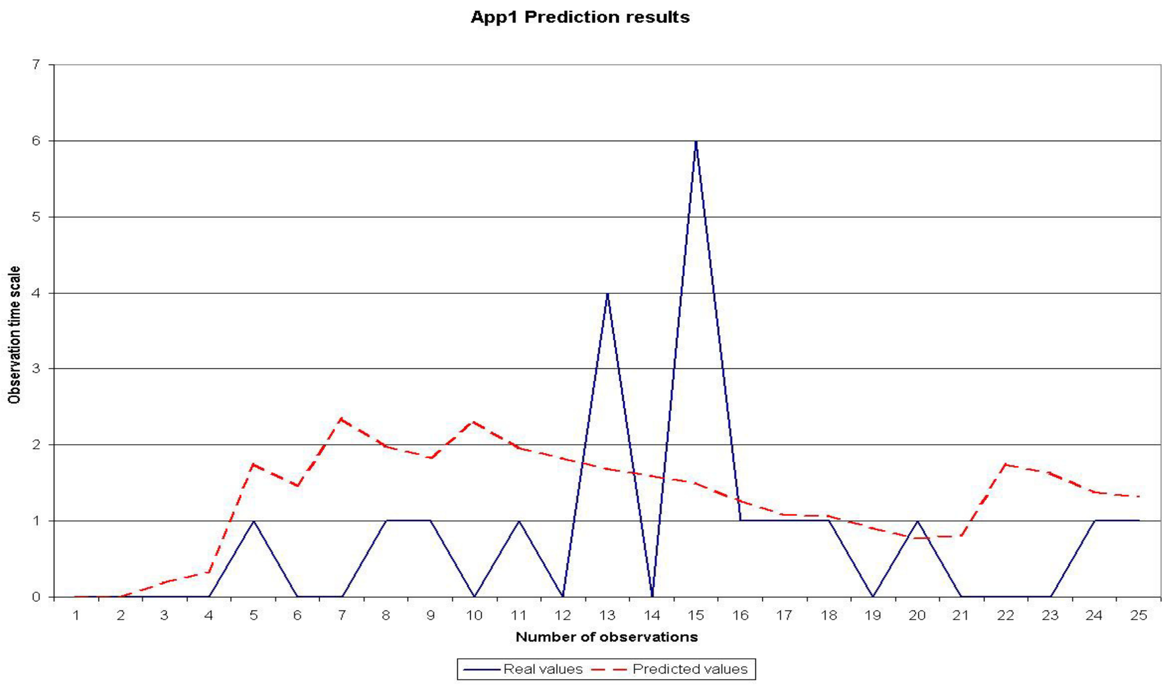


Figure 9.14 Prediction results with TS 10 minutes and PSz of 25

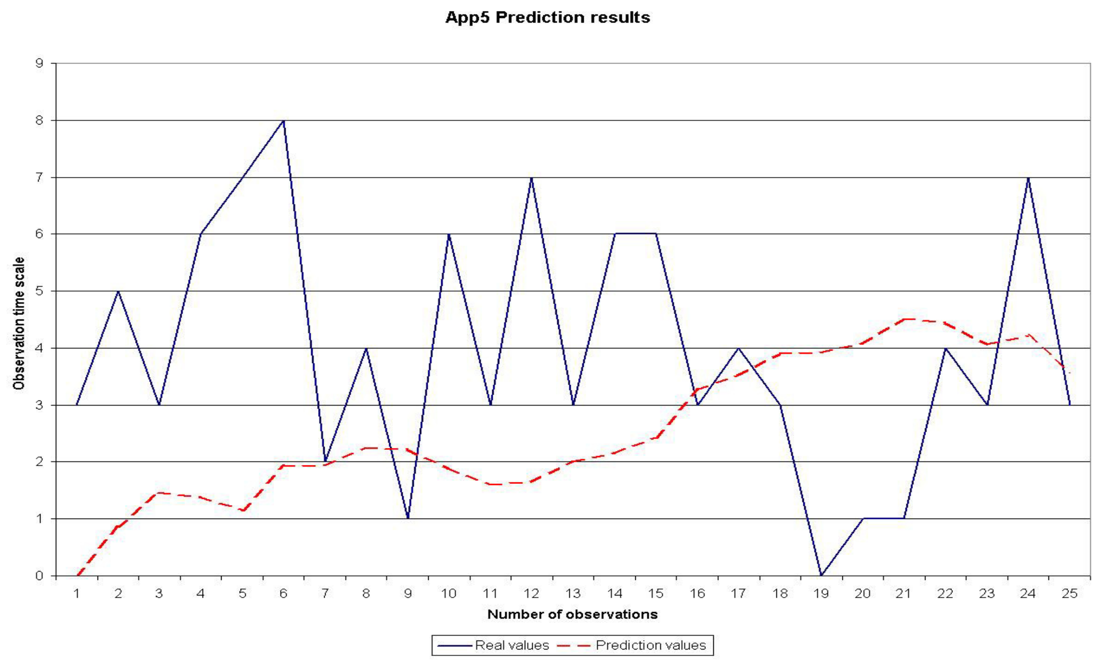


Figure 9.15 Prediction results with TS 10 minutes and PSz of 25

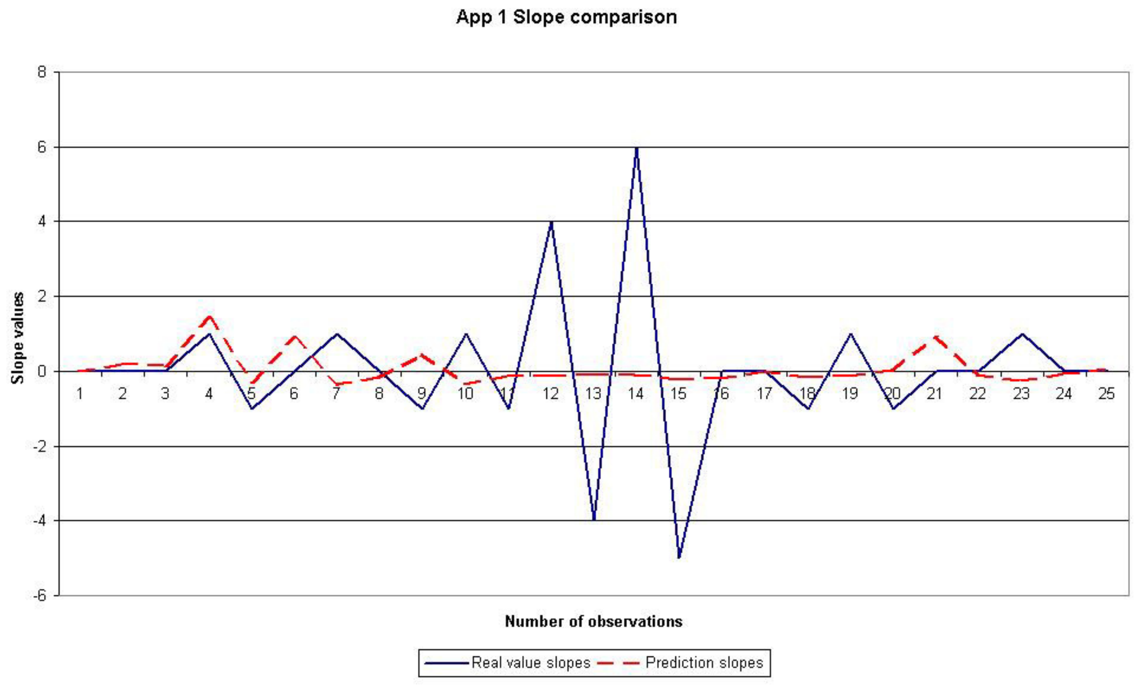


Figure 9.16 Slope Prediction results with TS 10 minutes and PSz of 25

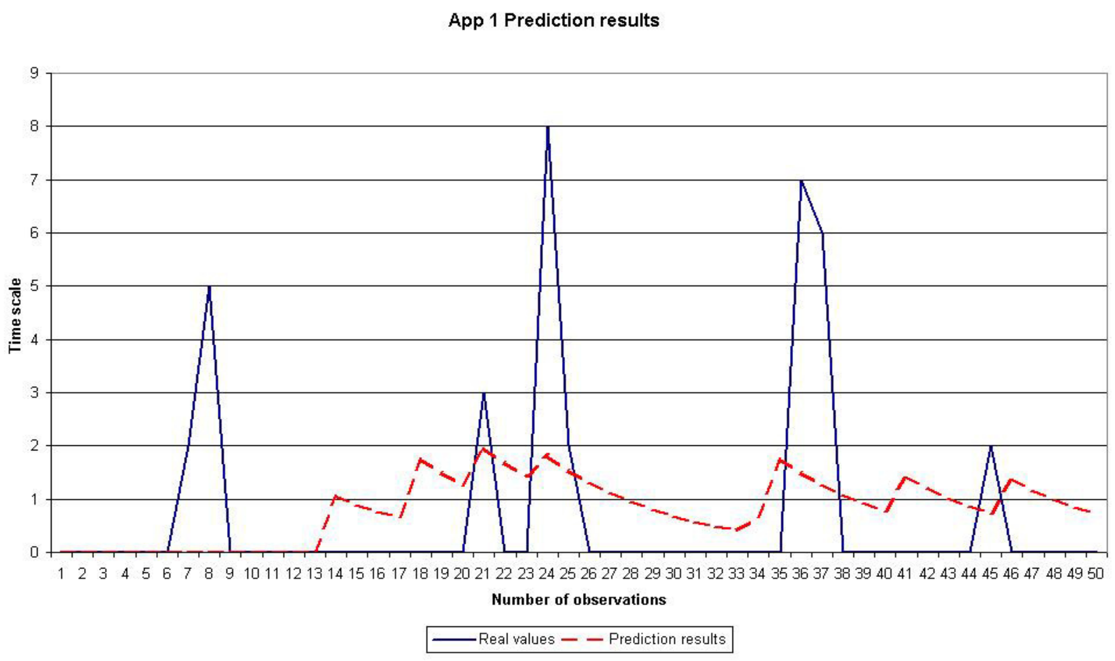


Figure 9.17 Prediction results with TS 10 minutes and PSz of 50

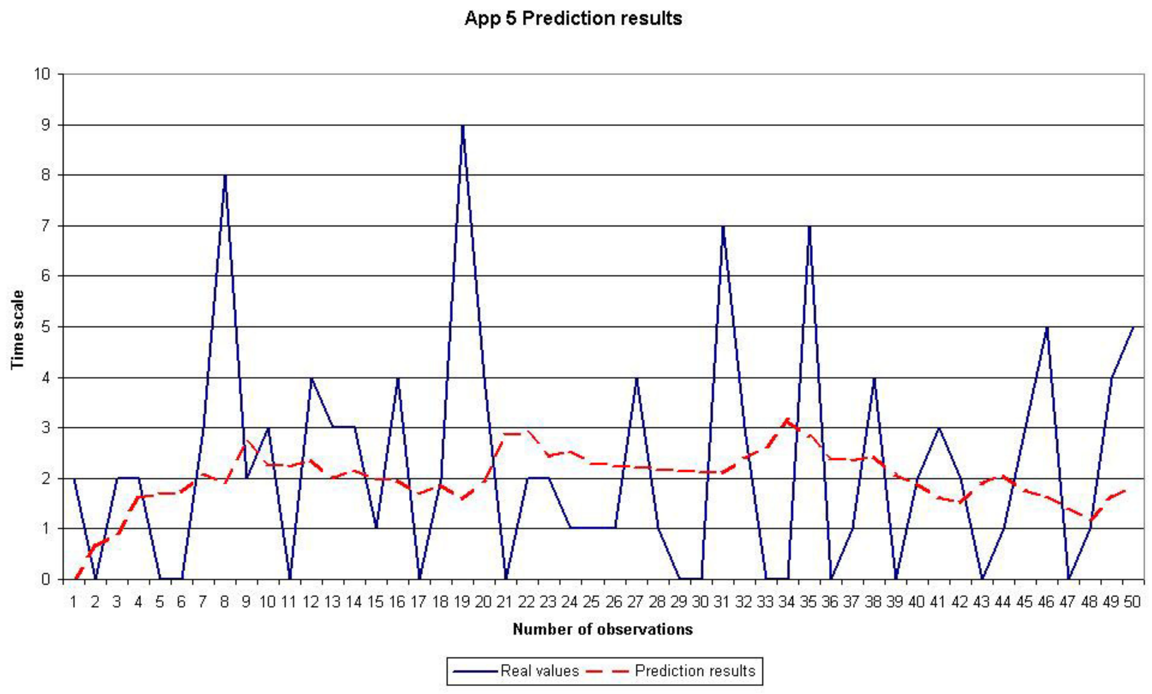


Figure 9.18 Prediction results with TS 10 minutes and PSz of 50

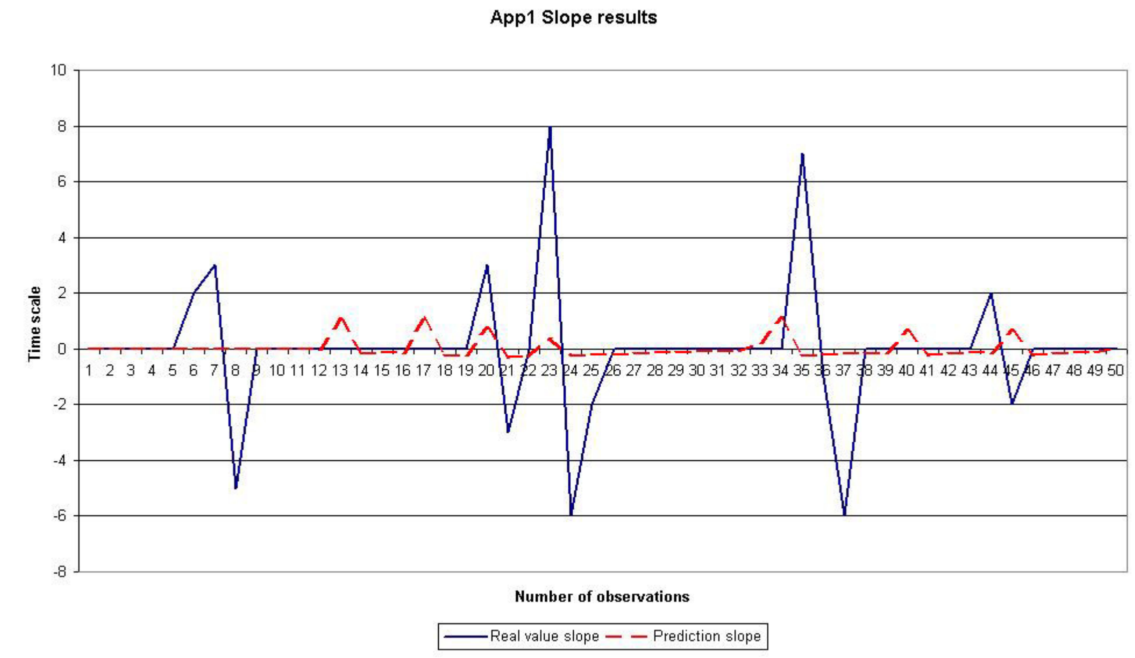


Figure 9.19 Prediction results with TS 10 minutes and PSz of 50

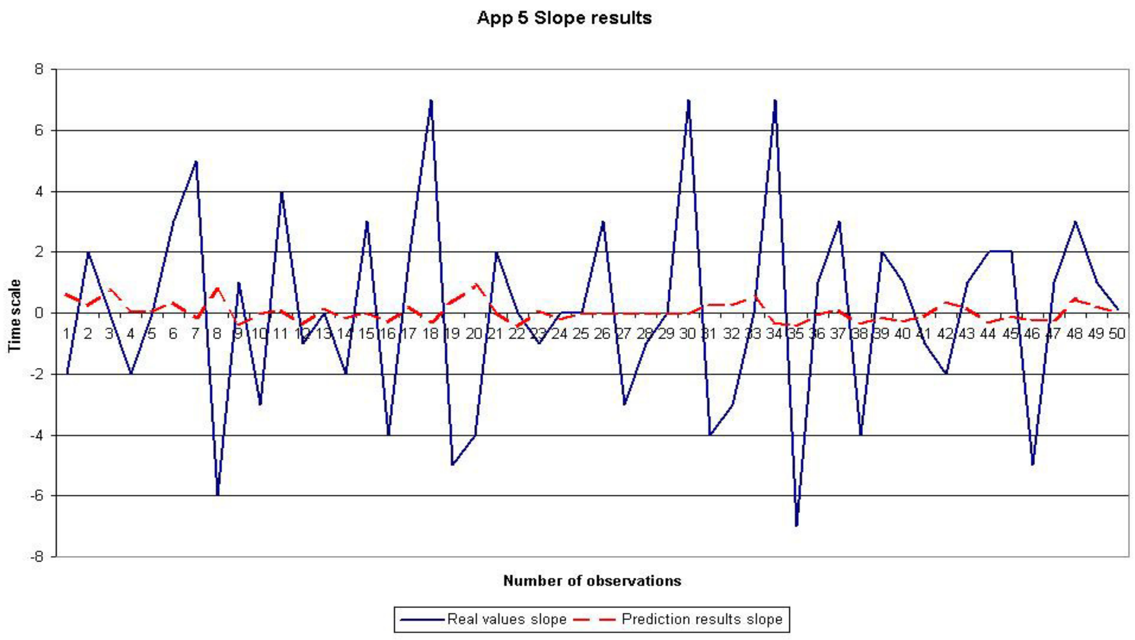


Figure 9.20 Prediction results with TS 10 minutes and PSz of 50

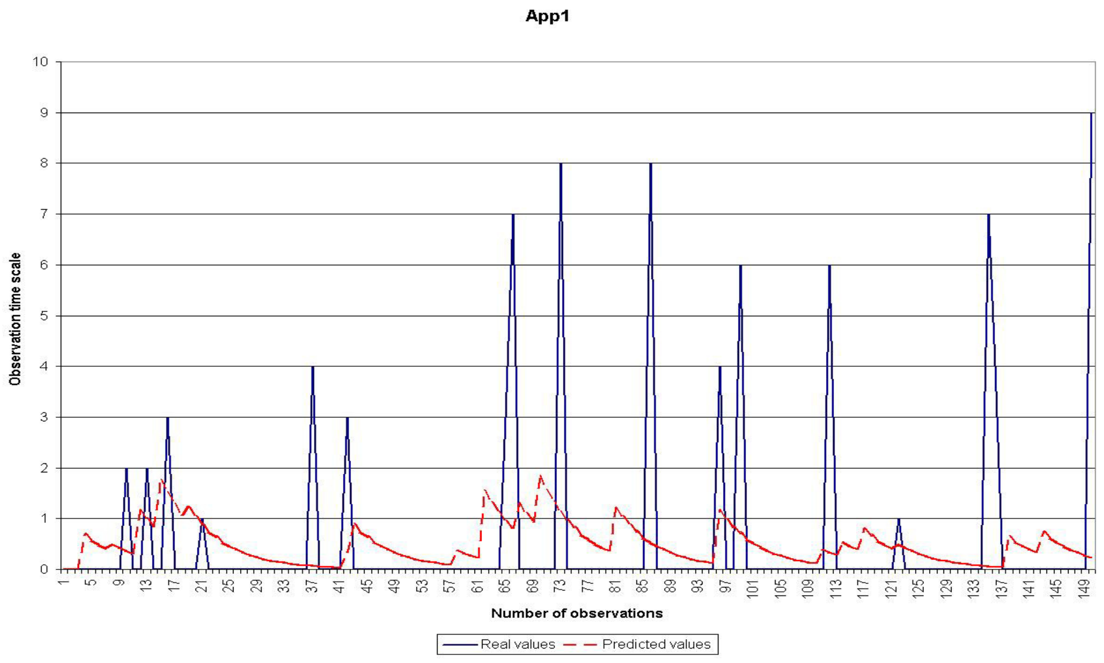


Figure 9.21 Prediction results with TS 10 minutes and PSz of 150

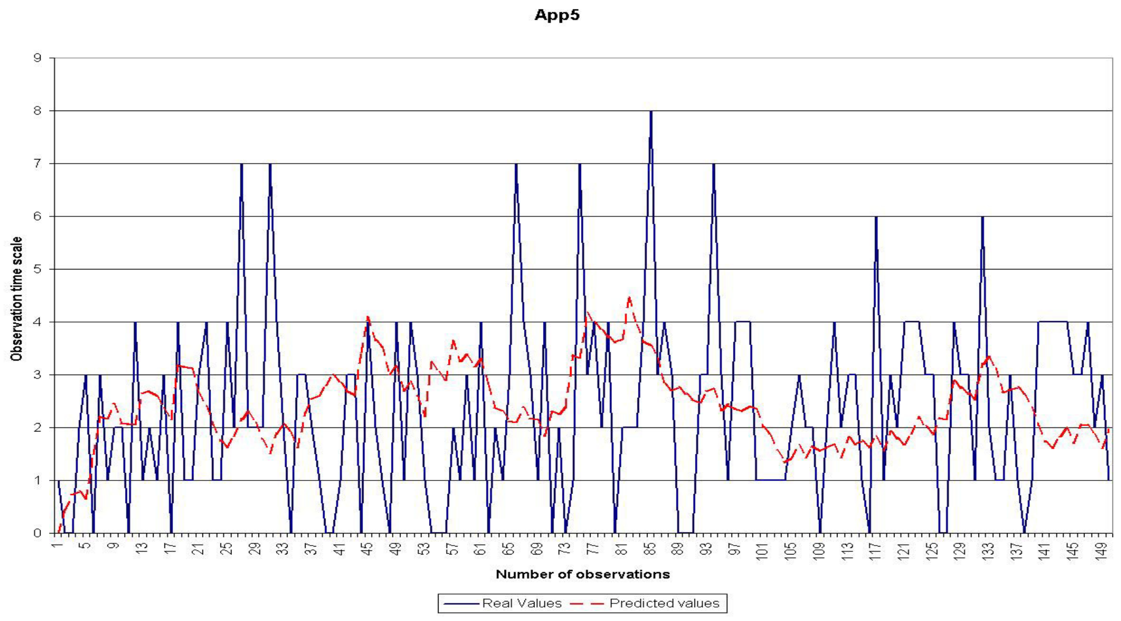


Figure 9.22 Prediction results with TS 10 minutes and PSz of 150

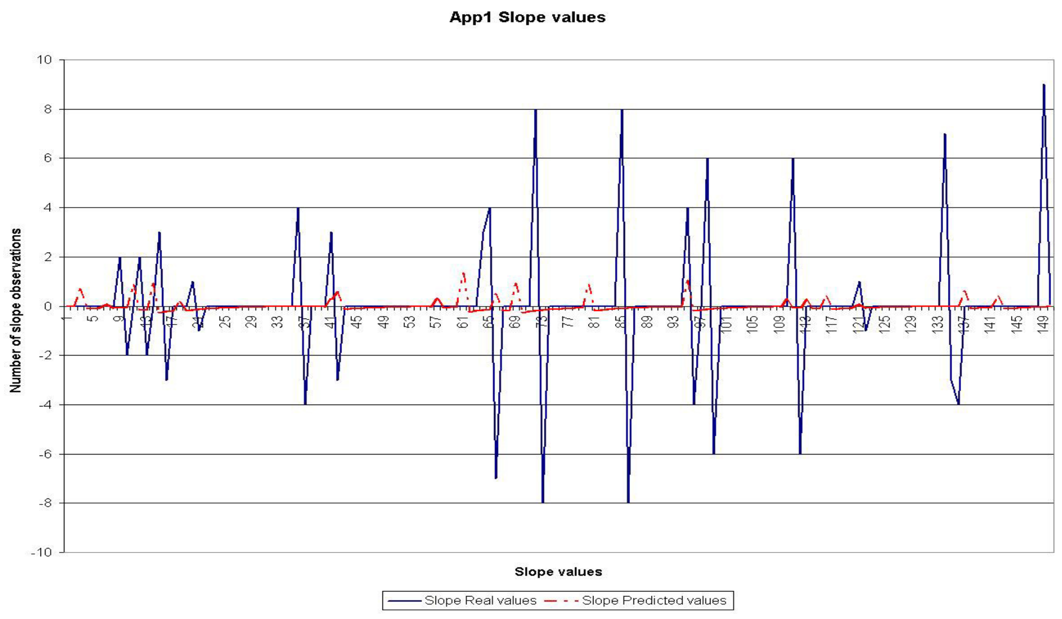


Figure 9.23 Slope graph comparison between the two results where TS is 10 minutes

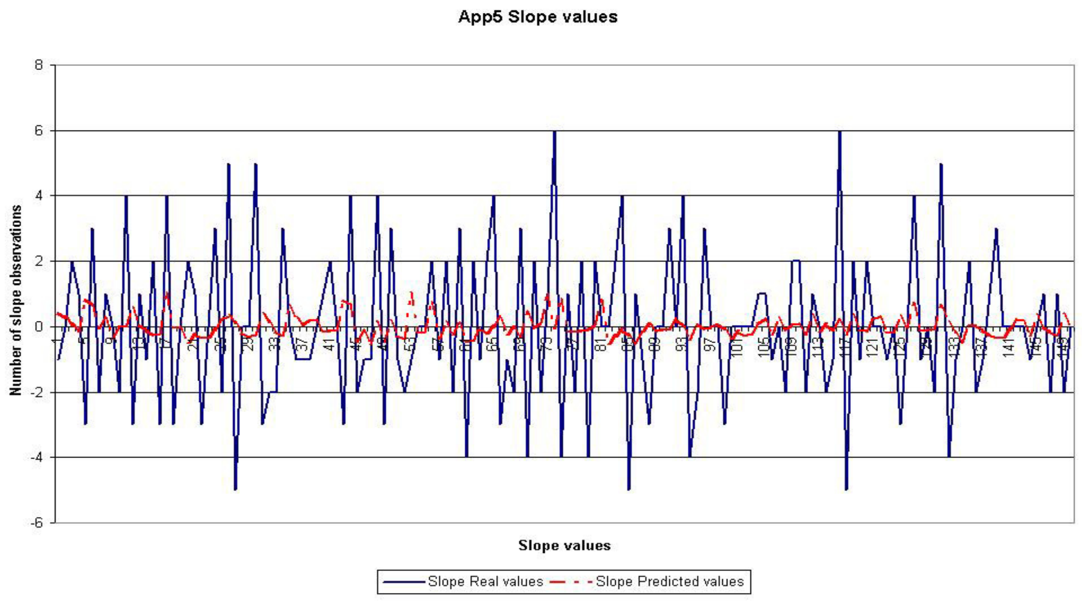


Figure 9.24 Slope graph comparison between the two results where TS is 10 minutes

Figure 9.25 to 9.28 show results using the same data, using ARIMA models and from our Bayes model. We can see the difference in the prediction; especially we can observe that our model can follow the real values, even in the smallest prediction periods.

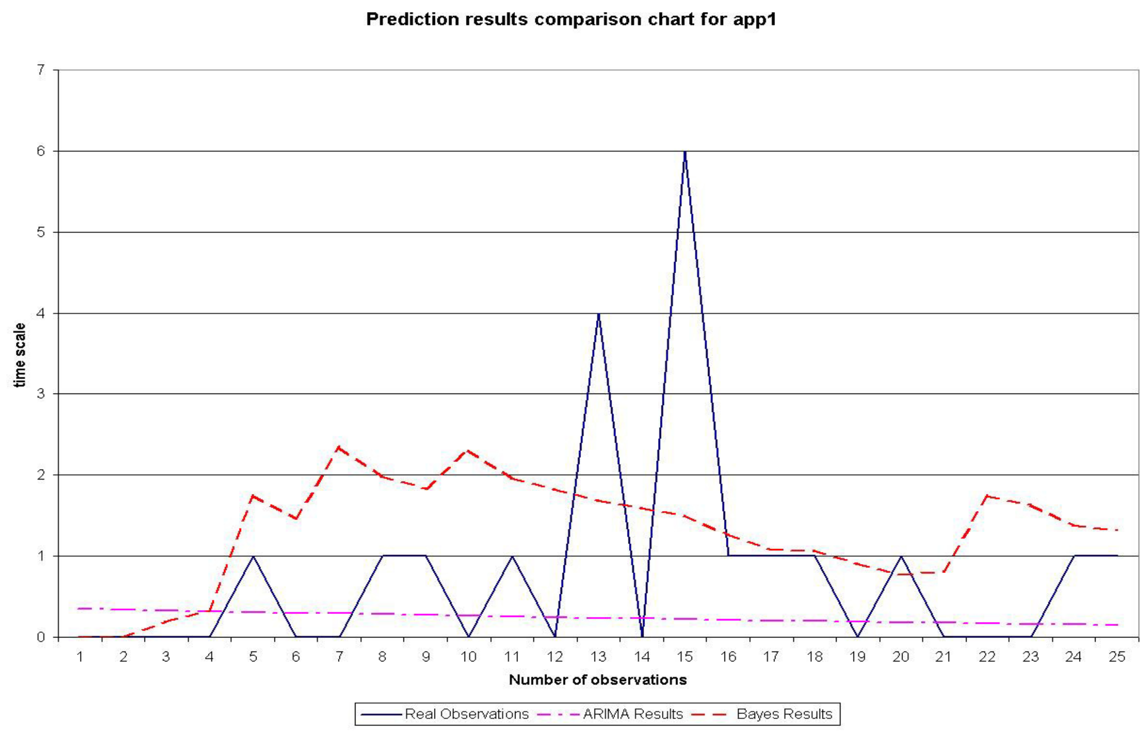


Figure 9.25 Prediction results for ARIMA and Bayesian models for app1

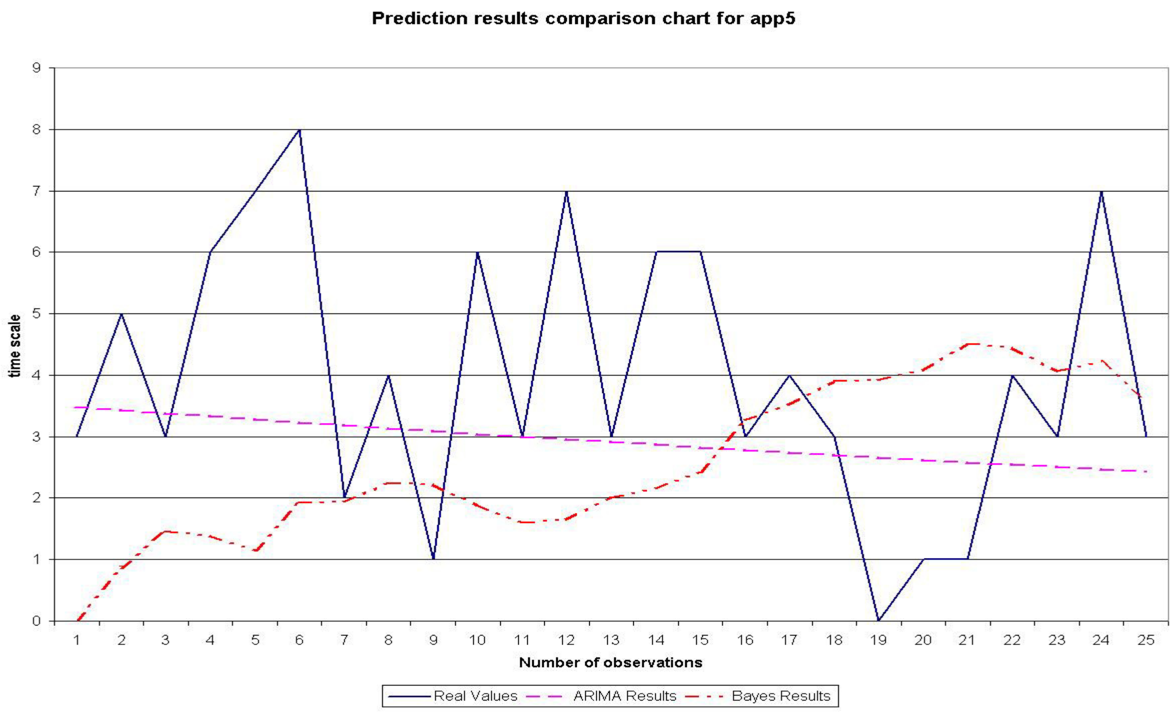


Figure 9.26 Prediction results for ARIMA and Bayesian models for app5

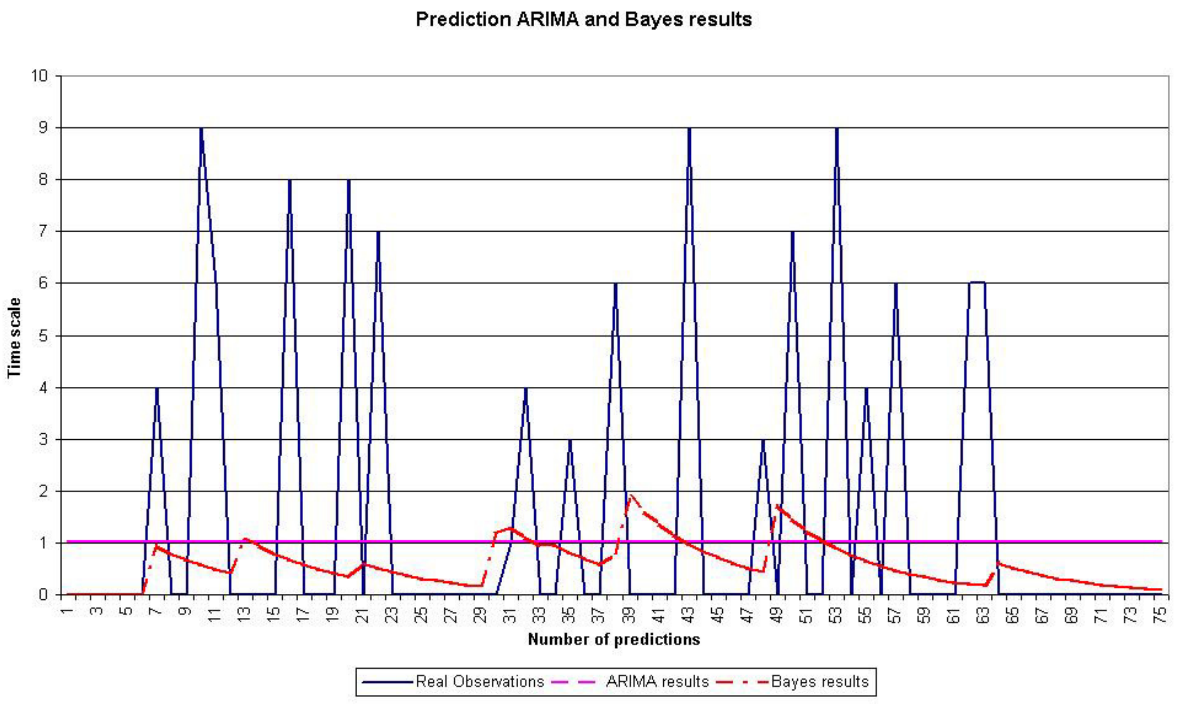


Figure 9.27 Prediction results for ARIMA and Bayesian models for app1

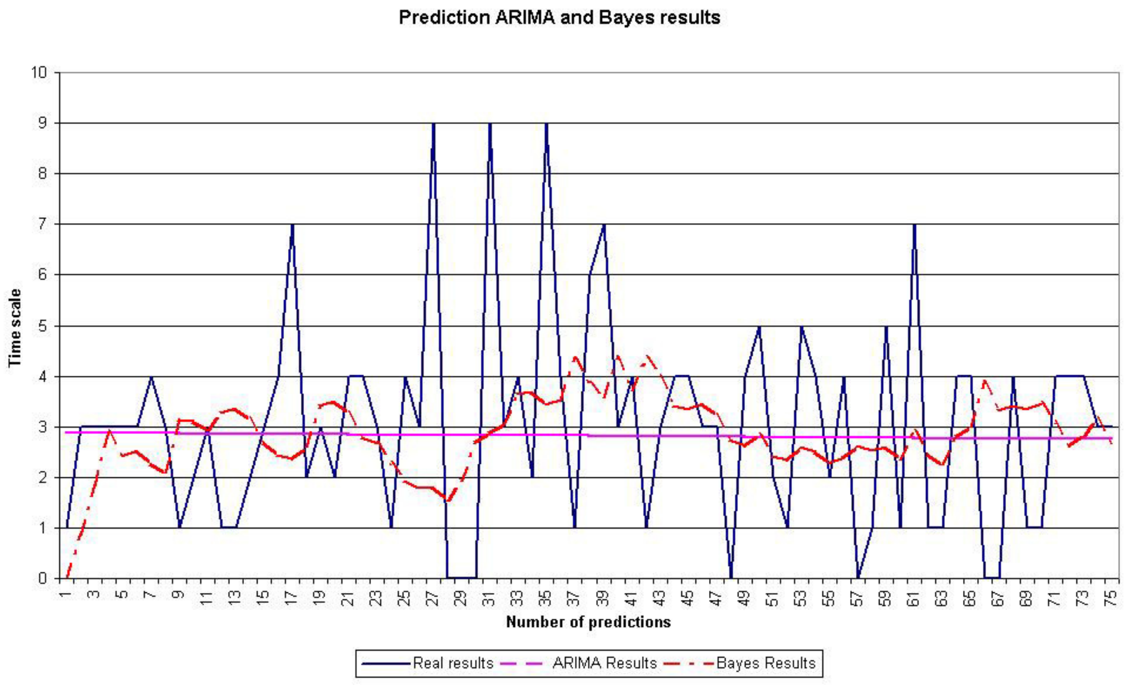


Figure 9.28 Prediction results for ARIMA and Bayesian models for app5

9.3 Result Evaluation

This section discusses the experiments, and tries to make some logical conclusions on the research. This will describe the extent or our experiments, and with the help of graphs, we explain why our results prove that our initial hypothesis is correct.

As described in Chapter 8, the first experiments were made to test the fundamental parts of our software agent system. For this, we had to verify that all the components of our system was working. The experiments were thus not very extensive. For this, we deployed agents in different computers and tested if they were communicating with each other. We have done some regression testing by forcing the user-end agents to make continuous connections and negotiation, with the core agent for a large number of times (about 3000) over a period of three days. This allows us to see if we could force the system to fail. After three day, the system was still functioning without stopping any software part of the system and rebooting any computer. This verified that our system could function properly.

The second aspect of our system that we tested was the real-time monitoring. This was done by gathering the first results from our users. We have observed that every application that the users were using, and was reported back to the core agent. The real-time monitoring runs running by the 'user-end agents' every five seconds. In order to keep the system usage to minimum levels, we are having made this monitoring multithreaded, so that the local computer load can determine how much priority it gives the process. With this observation, we have verified that our system were ready to be operational. We continue our experiments by collecting more data from users.

Every time that we were monitoring a group of user, we were testing not only the statistical prediction part of our system, but also the interconnectivity and the real-time monitoring. The focus of the tests is mainly to different aspects of the security agent system.

The second cycle of our experiments was targeted at the statistical prediction model. As this model is tailored particularly for our application, we had to modify many of its aspects, and some of its initial parameters are taken random. We had to make experimentation to verify that all the values that we had chosen were giving acceptable results. We can see from Chapter 8 that the graphs with the real values and the graphs from our prediction model were very close. These were some initial experiments that we made to see if the system was working. That is why the number of the observations was so small.

As we can see in Figure 9.29, that the two lines that represent the real observations and the predicted ones, are not exactly matching. However, we can observe that if we get the any two pair of points from both graphs and calculate their slope, we can see that it is very similar (Figure 9.31). This shows that even if our statistical prediction model can not some times predict how long the user will use a specific computer resource in the future, it can predict if the user is going to use this specific computer resource or not.

It can be seen from Figure 9.30 that by applying intervention on our results, we can maximise the prediction of our statistical model. Intervention is a simple method that can be applied on specific time-periods that are predefined, and by calculating the error of each prediction, we can make our system extremely accurate. The reason that we do not apply intervention always, is that we add a small calculation overhead to the system, and most of the time the prediction results that we are getting before apply intervention, are sufficient. As we mentioned earlier, by observing just the slope of our predictions, we can predict the future user behaviour.

App1 results

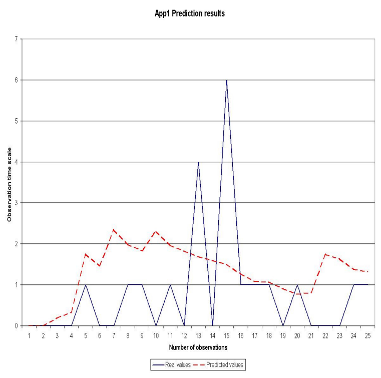


Figure 9.29 Application results

App5 results

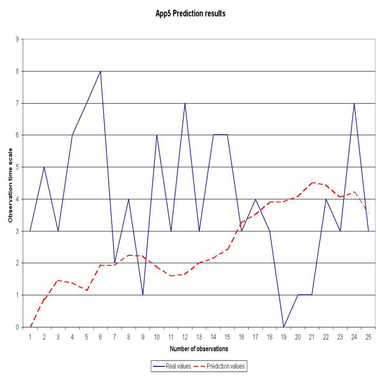


Figure 9.30 Application results

Slope results

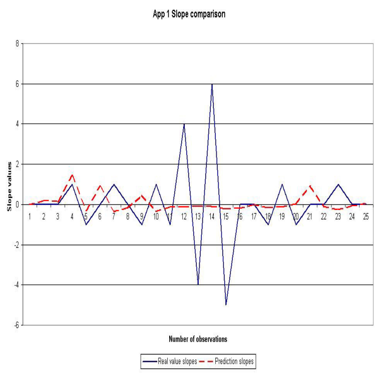


Figure 9.31 Slope results

Table 9.7 Observation results for two different applications

After experimenting with all of our software environment parts, and concluding that they work, we started our third cycle of experimentation. The aim of this cycle was to prove that our environment was working as predicted, and to find the limitations of our current model, and suggest future research based on our model.

As you can observe in this chapter, we first tried to isolate and quantify individual parameters of our system, so we can measure them and experiment with them. After that, we constructed tables that contained these parameters, in different combinations, so we can make sure that we have tested many of the different potential possibilities. These experiments also tested the proposed limits of our system.

For this we started by changing the time scale of the statistical system. As we said earlier, in our first experiments we used as time scale of once every one hour for the observations. In these experiments, we used again one hour and also 10 minutes, as time scale. We can observe some results in Table 9.8 and Table 9.9.

As we observe from these two tables, we can see that the two lines do not quite match each other. However if we observe the third graph (absolute values of the graph slope), we can see that the two lines are almost identical. This shows that the trend of our prediction is the approximately as the real observation. If we had the slope graphs of all of our results, we could observe that the slopes are almost identical. From this we can conclude that our model is working correctly, and the results are getting are similar to the expected ones.

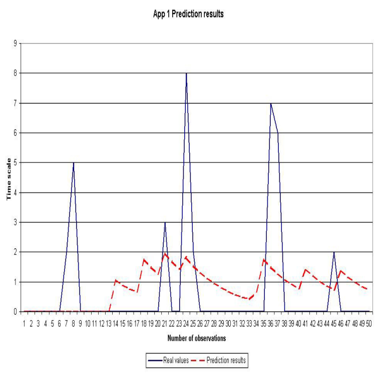
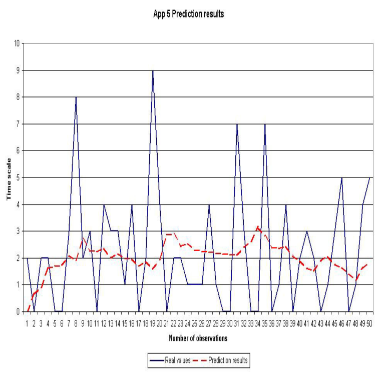
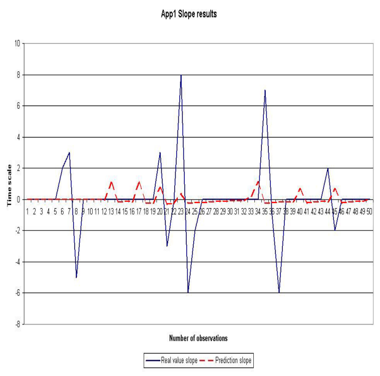
App1 results	App5 results	Slope results
		

Table 9.8 10 minute time scale results

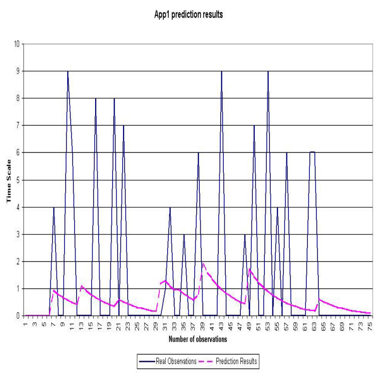
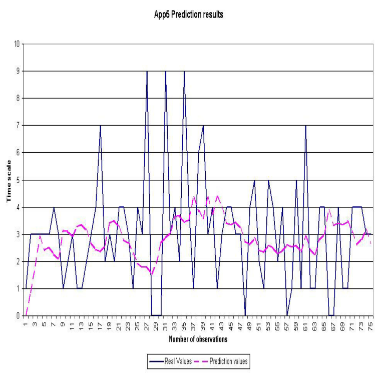
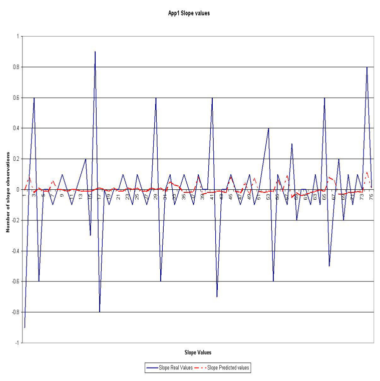
App1 results	App5 results	Slope results
		

Table 9.9 10 minute time scale results

Table 9.8 and 9.9 contain graphs with results from our model, and the real values, and the slope of each observation (which is the third graph). The slope is a very useful tool to compare our results with the actual values. The meaning behind the slope is that if we observe the slope of each prediction, we can predict if the user is likely to use the resource that we are monitoring, or not. The prediction does not have to match exactly the value of the real observation, but to have the same slope as the real observation. We can see in that in Figure 9.32, that although the real values and the prediction results are not exactly the same, we can still predict accurately the future value or behaviour of the user. By comparing the slope, for the graph for the same period, from the real values and the prediction values, we can see that both slopes are the same, or similar, which is enough to make a prediction.

We can now concentrate on the performance of our model. For this we use ARIMA to compare our results with. The reason for that is, is that this is the most wide used set of models. We are going to present some sets of graphs that we plot the results of our model, the real values, and results that we got from ARIMA model, when we processed our data with S-Plus statistical package. These results are shown in Tables 9.10 to 9.12.

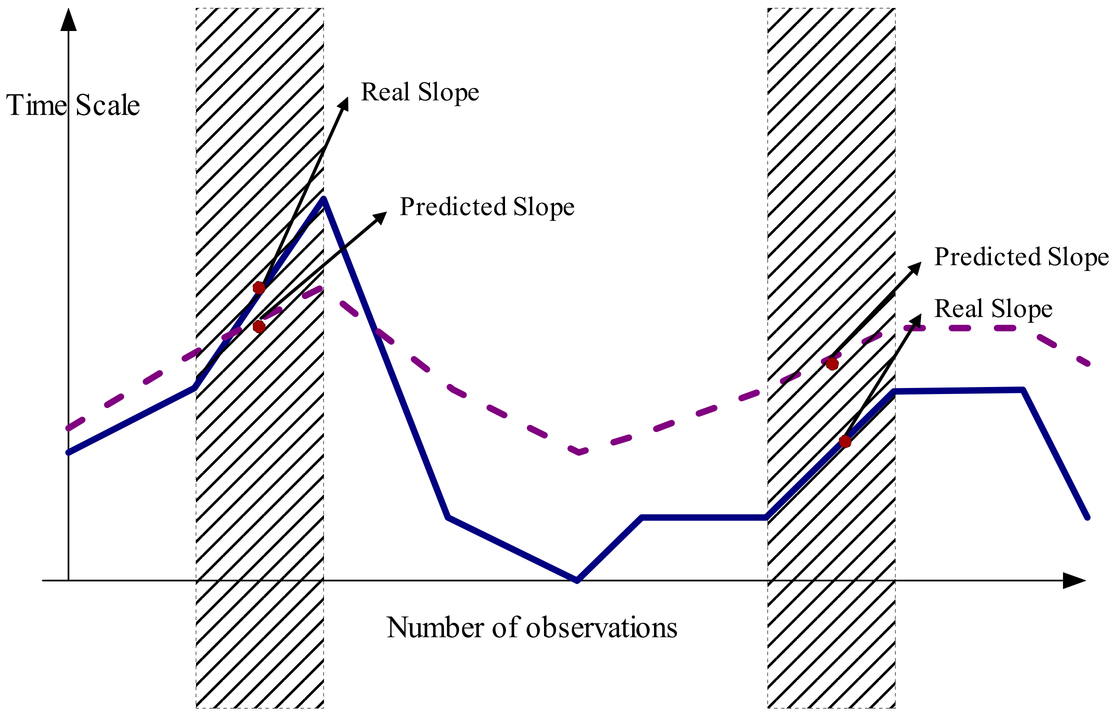


Figure 9.32 Slope comparison example

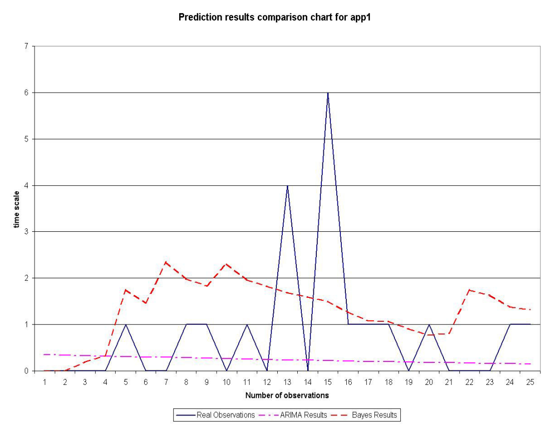
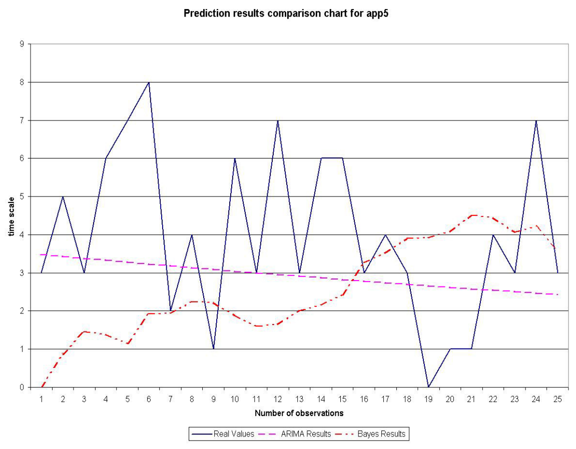
ARIMA and Bayes Results for app1	ARIMA and Bayes Results for app5
	

Table 9.10 ARIMA Bayes results for time scale 10 minutes

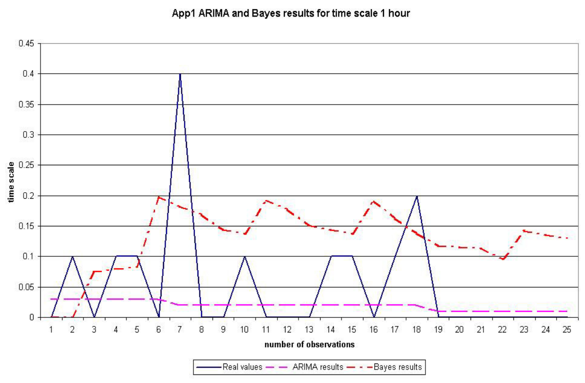
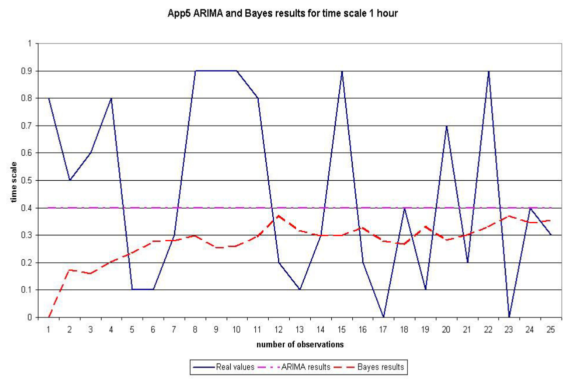
ARIMA and Bayes Results for app1	ARIMA and Bayes Results for app5
	

Table 9.11 ARIMA Bayes results for time scale 1 Hour and 25 numbers of predictions

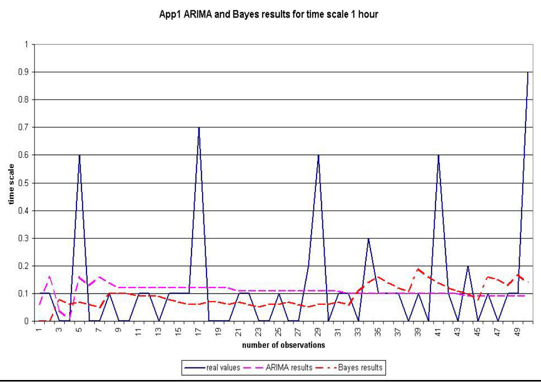
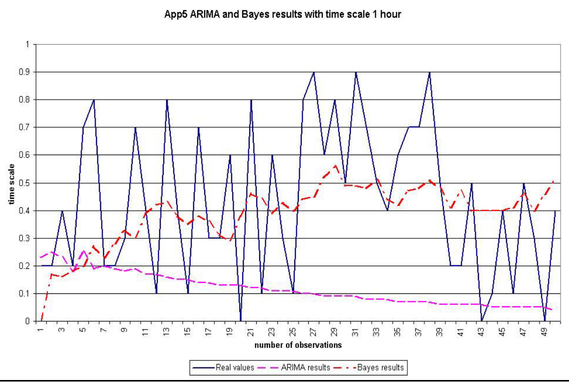
ARIMA and Bayes Results for app1	ARIMA and Bayes Results for app5
	

Table 9.12 ARIMA Bayes results for time scale 1 Hour and 50 numbers of predictions

We can see from the comparison graphs that the results we get from our model (Bayes) is more accurate than the ARIMA one. We can observe in every graph that the ARIMA result is almost a flat line. This is because ARIMA models cannot adapt to our fast changing system, and that ARIMA, as most of the forecasting models, are designed for long-term prediction.

One way to see the difference is if we graph the difference between the ARIMA and Bayes results, and the real values. Let us assume that we have the initial graph in Figure 9.33. We have results from both ARIMA and Bayes. Now we can find the difference between the Bayes results and the real values, which is X, and the difference between the ARIMA results and the real values, which is Y. If we make another graph, and plot the difference for the Bayes, and the ARIMA results, we will have the comparison. This method is just an indication and it is not always true, as we are taking more information from our model results graph. One factor that we cannot demonstrate with this set of test is if our results are precise enough or whether they can predict when a particular user will use a system resource. The only way to compare this is by looking the results graphs that contain information from both the ARIMA and Bayes models.

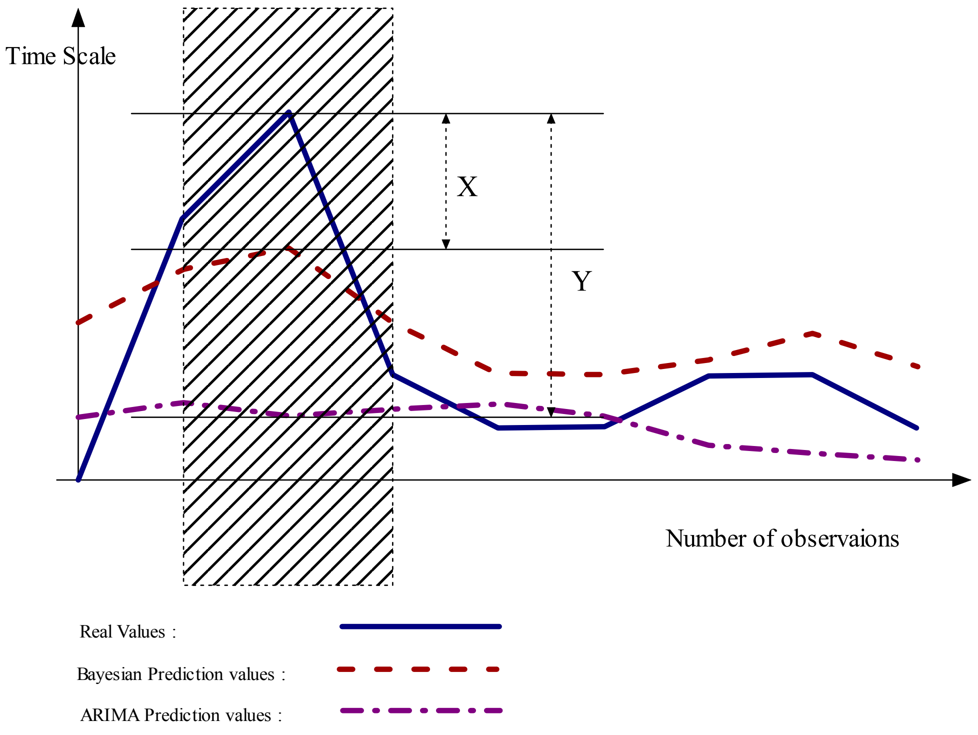


Figure 9.33 Prediction difference example

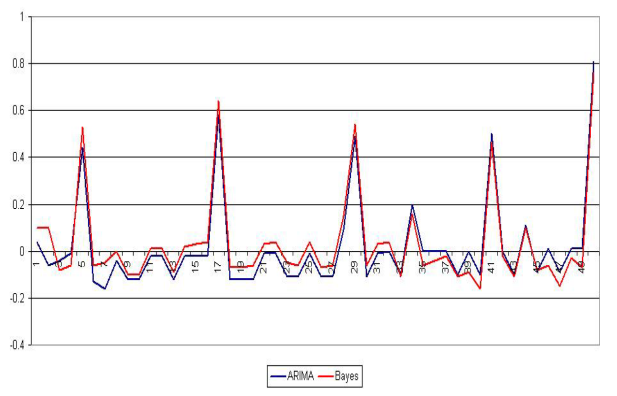
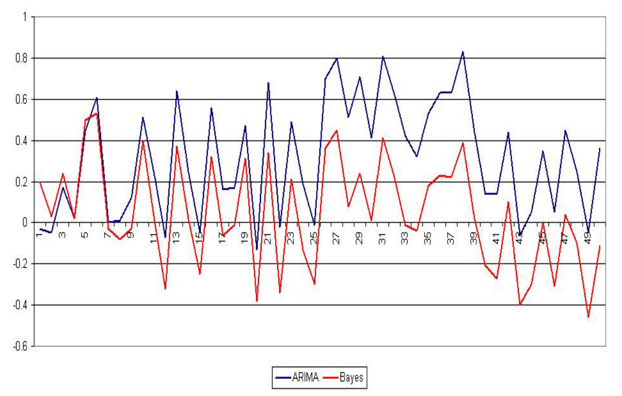
App1 ARIMA Bayes difference	App5 ARIMA Bayes difference
	

Table 9.13 Accuracy difference between ARIMA and Bayes models with time scale one hour and 50 observations

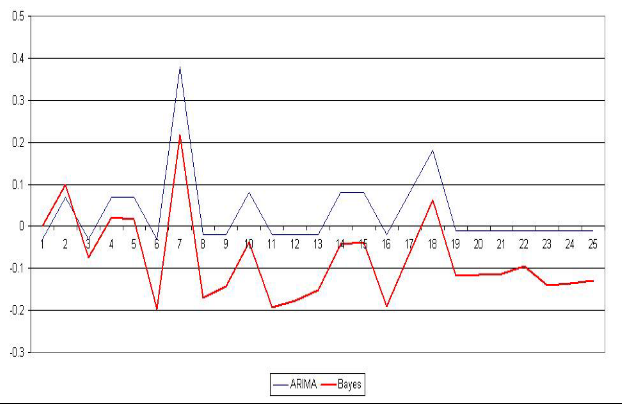
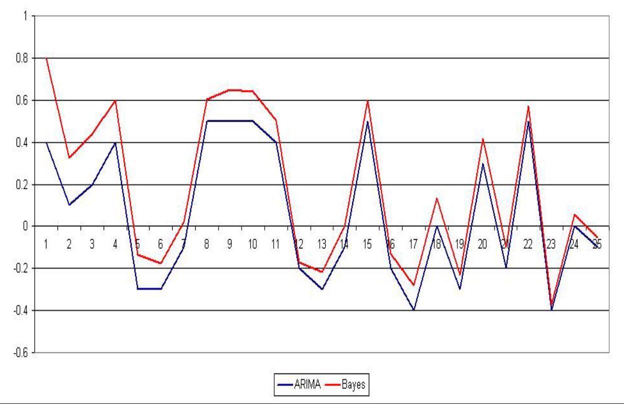
App1 ARIMA Bayes difference	App5 ARIMA Bayes difference
	

Table 9.14 Accuracy difference between ARIMA and Bayes models with time scale one hour and 25 observations

From the graphs, we can see that there is a difference between our model and the ARIMA model, as we can observe that our graph results are closer to the x-axis than the ARIMA results. We can also see from the comparison graphs that the results for a small number of predictions, typically smaller than 30, that both models has a small amount of randomness in their results. Thus, we can observe that sometimes on small number of observations, ARIMA models might give the same results as our model. This is to be expected, and we see from the graphs that for larger observation numbers, that our model is giving more accurate results than the ARIMA model.

9.4 Conclusions

We have made a comparison of our Bayesian prediction model with the real values, and with results from ARIMA model. We have found that our model, for the range of experiments conducts, that is much more precise than ARIMA models. We also saw that in some occasions, when the prediction is small, typically less than 30 observations, that sometimes our Bayesian model results has similar results to ARIMA. This is because both models cannot make proper predictions, as they do not have enough previous information. The results with a higher prediction observation number, gives improved results for the Bayesian model than the ARIMA model.

We have also seen that intervention can be used to improve the accuracy of the prediction. The drawback of this is that it requires more precise tuning of the data before we use them to our model.

10 Conclusions and Future Research

10.1 Conclusions

This chapter discusses the main conclusions of the work, and recommends future research. We started our research with a basic concept: *that computer security is not complete for today's computer networks*. Some of the reasons that we make this assumption, beyond our own experience, is that login and password access systems can be easily bypassed by intruders, and use this as an advantage to get into the rest of a system. Computer security is also largely based on human decision-making and most of the popular commercial operating systems are designed to be easy-to-use. Often computer security is left as a background task for skilled operators. Some of the reasons for security not being the first priority are that operating system vendors do not consider high security to be of concern of their customers; and that the complexity that the extra security might add to the operator may make the operating system less attractive as a purchase.

Our research does not, in any way, replace existing security methods, especially related to security at the network and transport layers. The proposed system tries to integrates with existing intruder detection systems, with the minimum of overhead. For this, we used a distributed systems architecture, in order to reduce single point-of-failures, manage fault tolerance, and redistribute the workload of the security enhancement to the entire system. Software agent technology is the most appropriate solution to this distributed system concept. For this, we have software entities which work autonomously and carry out decision making. These operate without the need for a central point. Software agents can also be easily upgradeabled and managed. They also are ad-hoc devices which give the opportunity of making changes to the system, without bringing the entire system down, thus not breaching the security protection. Agents thus overcome one of the major problems of software system, which is that software systems are becoming larger and more complex. This increase in size and complexity often increases the number of errors (*software bugs*) and the number of potential weak points of the system. Most users are now well aware that Microsoft Windows and associated software have many weaknesses which are often exploited by intruders. Many of these problems have been caused with the increasing complexity of the software. Agent technology overcomes this by engineering software entities which can be carefully crafted to meet a specific aim. They are also easier to test, and they have a limited number of inputs and outputs.

During our research on various intrusion detection and misuse detection systems, we discovered that most systems lack a vital component: that they only took action after an intrusion has been detected. We decide that this was a very serious weakness, and this led to the research on forecasting models. Our research revealed that most of the systems that used forecasting were either out-of-date, very difficult to configure, require specialized personnel, or the length of the prediction that they were making was long. These long-range predictions are good for making weather forecasts, long-range stock predictions and other kinds of long-range prediction, but they require a very large amount of data input, in order to be significant accurate. In intrusion detection, we wish to make short-term forecasts, as long-term forecasts may not accurately predict an intrusion in the short-term.

We first decided that our problem of intrusion detection was a linear one. This means that if we have a way to quantify the user behaviour over time, we could feed it into a forecasting model and get some results. We researched other prediction techniques, such as neural networks, but we decided that these were too complex for our problem. From the research on forecasting models, we found that Bayesian statistics fit exactly on our problem. Their general idea is that the models are small and can be easily configured. They can also produce accurate results in a short time. We have thus taken the general idea of Bayesian models, and fitted them into an intrusion detection system, and created a novel Bayesian statistical model. Chapters 4 and 5 deal with the mathematical aspects of our model, are based on papers [10.1, 10.6] and on a periodical publication in the *Journal of Statistics* [10.7].

Our new Bayesian model is fairly simple, as it takes some initial empirical values which we found to be optimum for each time scale. These values are generated after three cycles of experimentation, and are used to gain predictions on. One advantage of our model is that it can process, in parallel, all the different series of variables that we have and give results. By this, we mean that if we have for a user, N (where N is greater that unity) number of resources that we monitor, prediction is achieved by running our model only one time, in comparison with ARIMA models which requires to be run for each resource individually. The simplicity of the model can be observed in the third cycle of our experimentation, when we did experiments with differing time scales. This time scale is the time between the two observations, and the time for the next prediction. In our model, all that was required was to change the input data.

The results from the entire experimentation procedure were very positive. In the previous chapters, we made exact comparison of the results with the real values of the experiment, and with the ARIMA statistical models. It has been seen that the proposed models are accurate in predicting changes of user behaviour. From our experimentations we have observed that the model is between 0.2% and 2.5% more accurate than the ARIMA model. This is an arithmetic result and it is only specifies the arithmetic accuracy of our model. The results of our model are not interpreted arithmetically, but we look at the trends of our results. For this we look the slope our prediction to conclude on future user behaviour. The arithmetic accuracy defines as to how much the user will use or not, a specific resource. In the interpretation of whether the user is going to use the resource or not, the results interpretation of our model compares the slope of the results with the real values. Chapters 6, 7, are based are based on novel work and resulted in several papers [10.1–10.3].

Our security enhancement system can be applied to most computer networks. We designed our system to be an addition to the existing security of commercial operating systems. It is focused on a highly secure environment, where any intrusion can be taken as undesirable (such as in a secure government or military-related environment). Chapters 8 and 9 are the focus of the results on the research and is based on several papers [10.2–10.5].

10.2 Future work

In our security enhancement model, our agents are static, and reside on the resource that they monitor. This makes the deployment of the system relatively easy, but does not take into account network resources that may highlight intrusion. For instance, a tell-tail sign is maybe where workstation disk drives are reformatted, or crashed on-purpose, so that the intruder can exploit certain operating system security vulnerabilities. One way to solve this problem is to add a transport mechanism the user – end agent. In this way, the user-end agents can detect that a specific resource does not have an agent, and actually clone themselves, which would be sent to the network resource which was operating without one. In addition, a core agent could monitor network resources that are responsible for, and if it detects some lack of communication from a user-end agent, then it could create a new user-end agent and send it to the network resource. In this way, they can operate in a more *social* way, and will thus have more control over their environment.

Another aspect of future research, is to add a recognised agent language to their inter-agent communication process. In this research we have implemented a very basic language that has the purpose of exchanging basic information between the different types of agent. With an enhanced security system, the agents will have to be more social, meaning that they have to exchange larger amount of information, or more often. For this, an enhanced language could be an agent language such as

KQML [10.8]. In this way our agents could communicate with other agent societies, if necessary, without the need of upgrade, or another software translation layer.

Another future research can be on how the agents, in our environment, can interact with agents from other environments, or from the same environment. From a security aspect, the research has to find ways of identifying if the communicating agent has malice purposes, or not. A key to this is authentication. This is often relatively easy for agents in the same environment, by adding an encrypted signature that they have to exchange as the first part of their communication. But agents from different environments, made from different vendors, will be difficult to identify and trust. One way, is to build certificates, similar to ones that they use today form accessing secure information on the Internet. Each agent would thus have their own signature, and they could thus be identified and trusted, or not.

Future experimentation can support a larger number of users (such as with 50 users), and a larger number of monitored resources (such as up to 30 resources). From our current experimentation results, we did not have any problems with the number of the users or monitored resources. Also in our experiments we used a timescale interval of one hour, and also of 10 minutes. A fully range of experiments could reduce the interval to as low as one minute, and as high as one day, without making configuration changes to the statistical model. The aim of this would be to create a statistical model that can work with different ranges and different timescales, with the minimum configuration. In our experiments, we forced the system to the limit, such as making predictions with over a thousand values. For this very large numbers were generated (larger than 10^{60}), which do not make any physical sense. Future research would thus investigate these limits.

10.2 Future Research of our Statistical model

Our statistical model is unique in the Bayesian literature, as it does not make assumptions of independence among the various computer applications. One major extension to the model is to reform it to model a matrix of data, instead a vector of data. With this, we can have many users working at the same time and many applications considered. The benefit of such extension is mainly in computational effectiveness and portability, but also in mathematical clarity.

The matrix version of the model was briefly described in this thesis is an efficient approach. Future research would further develop the statistical properties of this model. A future project might mainly focus on the specification of the design vector F_t. In this thesis F_t is assumed known and in our experiments we chose it randomly. The specification of design components is an active research area of Bayesian statistics and the problem is significantly important. By working on the specification of F_t we may open a more general path of linear modelling in statistics. Our plan is to assume that F_t is slowly changing with t, such as considering a third equation in our model:

$$F_t = F_{t-1} + c_t$$

where c_t is an error factor measuring how much F_t differs from F_{t-1}.

c_t may be a random variable (in which case F_t will be random), or it may be a constant value (in which case F_t is deterministic). If F_t is assumed random, the analysis we propose in this thesis may extend to more general results. Linear modelling with random design vectors is a very important area in the statistical literature and in our problem it will solve the problem of the specification of F_t.

Security, in general, is a major part in many organizations and quality control is now a major element of strategic planning and management of future performance. Statistics plays an active role, especially in time-series analysis and forecasting. Employing modern statistical methodology and not relying on the past can maximum security. A future research would look at how our models can be fitted to other security environments.

This research has revealed some of the weaknesses in performance with relatively small data sets of ARIMA models. Some authors have been addressed these weaknesses and have pointed out that Bayesian models overcome such weaknesses (see, West and Harrison, 1997, page 166). However, no author has considered a comparison of Bayesian time series and ARIMA models from an application point of view. Future research may wish take further comparisons and would see the differences from a practitioner's perspective.

Another area, which is very likely to have a growth in the future, is to develop statistical software for Bayesian time series. Most of the existing software in this area lacks accessibility and user friendliness. Scientists and practitioners are starting to recognize the impact of Bayesian computation by developing computer-intensive software. Unfortunately, such software is not easy to be use by non-statisticians, and non-experts of the methods. Otherwise, we feel that Bayesian models will not be widely known and scientists may still use the older statistical methodology, like ARIMA and regression.

10.3 References

[10.1] Pikoulas J., Mannion M., Buchanan W., 7th IEEE ECBS Conference, NAPIER University, Edinburgh, 3-7 April 2000, with title "*Software Agents and Network Security*".

[10.2] Pikoulas, J., Buchanan, W.J., and Triantafyllopoulos, K. (2000) "*An intelligent intrusion detection environment using software agents*". In *Proceedings of the 13th International Conference of Software and Systems Engineering and their Applications (ICSSEA)*, Volume 4, Paris 6-8 December 2000.

[10.3] Pikoulas, J., Buchanan, W.J., Manion, M., and Triantafyllopoulos, K. (2001) "*An agent based Bayesian forecasting model for enhanced network security*". In *Proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer Based Systems - ECBS, IEEE Comput. Soc.*, 247-254, Los Alamitos, CA, 17-20 April 2001.

[10.4] Pikoulas, J., Buchanan, W.J., Manion, M., and Triantafyllopoulos, K. (2002) "*An intelligent agent intrusion system*". In *Proceedings of the 9th IEEE International Conference and Workshop on the Engineering of Computer Based Systems - ECBS, IEEE Comput. Soc.*, Luden, Sweden, 8-11 April 2002.

[10.5] John Pikoulas and Kostas Triantafyllopoulos, "*Bayesian Multivariate Regression for Predicting User Behaviour in a Software Agent Security System*",20th International Symposioum on Forecasting, June 21-24 2000, Lisbon, Portugal.

[10.6] Triantafyllopoulos, K. and Pikoulas, J. (2000) "*Bayesian multivariate regression for predicting user behaviour in a software agent computer security environment*". *Research Report* 375, Department of Statistics, University of Warwick.

[10.7] Triantafyllopoulos, K., and Pikoulas, J. (2002) *Multivariate Bayesian regression applied to the problem of network security*, Journal of Forecasting (to appear).

[10.8] Specification of the KQML Agent-Communication Language -- plus example agent policies and architectures, by The DARPA Knowledge Sharing Initiative External Interfaces Working Group.

Acronyms

ARIMA	AutoRegressive Integrated Moving-Average
BBS	Bulleting Board System
CGI	Common gateway interface
DLM	Dynamic Linear Model
DOS	Denial of Service
FTP	File Transfer Protocol
IDS	Intrusion Detection System
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
OSI	Open Systems Interconnection
RFC	Request for Comments
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman
SMTP	Simple Mail Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator, previously Universal Resource Locator