Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks

Gideon Creech
B.E.(Elec) Hons, M.Eng.Sc.

A thesis in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Engineering and Information Technology,
University College,
The University of New South Wales,
Australian Defence Force Academy.

2013

PLEASE TYPE

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Creech**	
First name: **Gideon**	Other name/s: **Humphrey**
Abbreviation for degree as given in the University calendar: **PhD**	
School: **SEIT**	Faculty: **UNSW Canberra**

Title: **Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks**

Abstract 350 words maximum: (PLEASE TYPE)

Current anomaly host-based intrusion detection systems are limited in accuracy with any increase in detection rate resulting in a corresponding increase in false alarm rate. Furthermore, present technology is largely limited in scope to the Linux operating system, with the popular Windows family of computers forced to rely on signature-based protection schemes.

This thesis investigates the development of a new approach to host-based intrusion detection system design with the specific aims of improving performance beyond that of existing technology and developing a cross platform approach to intrusion detection. This research has made three original and significant contributions to the field, and represents a marked advance in the body of knowledge.

The first major contribution is the development of a new semantic approach to system call data processing, allowing the creation of host-based intrusion detection systems for the Linux operating system which perform significantly better than existing approaches. Performance was evaluated against existing datasets and also against a new modern dataset designed as part of this research.

The second key contribution is the development of a new theory which allows the deployment of traditional system call centric Linux anomaly-based intrusion detection systems on the Windows operating system for the first time. This significant technological advance means that protection against zero-day attacks is now possible on this operating system for the first time. These results were tested using a second new dataset designed as part of this research.

The final key contribution of this thesis is the development of a new attack methodology which is able to bypass traditional Windows signature-based defences without any obfuscation. The revelation of this new attack technology is an important contribution in and of itself as it allows the community of researchers worldwide to address this important weakness in current approaches. Notwithstanding this threat, host-based intrusion detection systems which use the first two new theories outlined in this thesis are shown to be able to detect this new attack class with a high degree of accuracy, allowing effective protection and significantly mitigating this threat.

G. Creech	[illegible]	**30 Jan 2014**
Signature	Witness	Date

ORIGINALITY STATEMENT

Signed G. Creech ....

Date **30 Jan 2014** ....

Abstract

Current anomaly host-based intrusion detection systems are limited in accuracy with any increase in detection rate resulting in a corresponding increase in false alarm rate. Furthermore, present technology is largely limited in scope to the Linux operating system, with the popular Windows family of computers forced to rely on signature-based protection schemes.

This thesis investigates the development of a new approach to host-based intrusion detection system design with the specific aims of improving performance beyond that of existing technology and developing a cross platform approach to intrusion detection. This research has made three original and significant contributions to the field, and represents a marked advance in the body of knowledge.

The first major contribution is the development of a new semantic approach to system call data processing, allowing the creation of host-based intrusion detection systems for the Linux operating system which perform significantly better than existing approaches. Performance was evaluated against existing datasets and also against a new modern dataset designed as part of this research.

The second key contribution is the development of a new theory which allows the deployment of traditional system call centric Linux anomaly-based intrusion detection systems on the Windows operating system for the first time. This significant technological advance means that protection against zero-day attacks is now possible on this operating system for the first time. These results were tested using a second new dataset designed as part of this research.

The final key contribution of this thesis is the development of a new attack methodology which is able to bypass traditional Windows signature-based defences without any obfuscation. The revelation of this new attack technology is an important contribution in and of itself as it allows the community of researchers worldwide to address this important weakness in current approaches. Notwithstanding this threat, host-based intrusion detection systems which use the first two new theories outlined in this thesis are shown to be able to detect this new attack class with a high degree of accuracy, allowing effective protection and significantly mitigating this threat.

For my Mother.

Acknowledgements

Research is a thrilling pursuit, but it does not occur in a vacuum and it is seldom the named author alone who has invested large amounts of time and energy in a published work. The people acknowledged here have played a huge role in the conduct of this research project, and they have my heartfelt thanks and appreciation.

I would like to take this opportunity to thank the following people for their support and assistance:

Professor Jiankun Hu, my research supervisor, for agreeing to take me on as a student and assisting throughout my research. I could not have asked for a better introduction to the academic process or a better guide for those first critical steps on the research path.

Dr Lawrie Brown, my co-supervisor, for his support and mentoring over many years. I first encountered Dr Brown when a Masters student, and have consequently been lucky enough to receive the benefit of his wisdom and experience for quite some time. I could not have asked for a better scholarly role model than Dr Brown, and am truly grateful for his involvement.

Lieutenant-Commander Stefaan De Brauwer, RAN, for taking a chance on me and posting me to full time study. Stefaan is in many ways the military counterpart to Dr Brown's scholarly mentoring, and I am extremely lucky to have had such an excellent career manager.

Beverley Humphrey and **Dr George Humphrey**, my grandparents and career scientists of the highest calibre, for introducing me to the world of research before I could talk, and encouraging me to always ask *"Why?"*.

Gabrielle Le Grand, my wife and partner, for her encouragement, love and support, both for my research and in everything I do. I would not be the person I am today without her, and my mind would still be asleep. Quite simply, I could not have done this research without her by my side.

Finally, **Sonia MacDougall**, my late mother - never forgotten and always in my heart.

Publications

The following papers were produced in the course of this Ph.D. research:

Accepted for Publication

Refereed Journal Articles

G. Creech and J. Hu. A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns. *Computers, IEEE Transactions on*, PP(99):11, 2013.

Conferences

G. Creech and J. Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE,* pages 44874492, 2013.

G. Creech and F. Jiang. The application of extreme learning machines to the network intrusion detection problem. *AIP Conference Proceedings*, 1479(1):15061511, 2012.

G. Creech and F. Jiang. Semantics based multi-layered networks for spam email detection. *AIP Conference Proceedings*, 1479(1):15181523, 2012.

Under Review or Pending Submission

G. Creech and J. Hu. A New Virtual Kernel Theory Allowing the Application of System Call Intrusion Detection Analysis to the Windows OS. Manuscript submitted to *IEEE Transactions on Computers* for peer review.

G. Creech and J. Hu. A New Theory of Hacking Attacks - Stealth Attack Techniques and their Detection in the Windows OS. To be submitted to *IEEE Transactions on Computers* for peer review.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Cyber Security in the Modern Age

Recent years have seen a huge increase in the prevalence of computing assets and digital resources across the business, private and government sectors. Smart phones, tablet computing, wireless connectivity and similar innovations coupled with an increase in the affordability of digital devices has led to a previously unseen level of integration between the cyber and physical worlds. Whilst this phenomenon has had extremely positive benefits on productivity and efficiency, it has also engendered a significantly increased threat of malicious cyber activity. As hacking technology becomes increasingly widespread and accessible to less technically advanced operators through the use of open source hacking tool such as the Metasploit Framework (MSF) [1], so too does the performance of defensive technology deployed to limit the corresponding threat become ever more important.

The ramifications when defensive technology fails are critical. Recent media reporting is filled with examples of high profile hacking on critical service providers [2] and large multinational enterprises [3], as well as government departments [4] and key infrastructure [5]. Much of this threat arises from zero-day attacks, which is a term used to describe exploitation vectors which have not been previously encountered in the white hat community. As there is no *a priori* knowledge of the attack structure or method, defences are consequently unlikely to be able to protect against the new threat. This, in turn, leads to high impact compromises of seemingly well protected entities with an accompanying negative effect on society. Security researchers attempt to limit the number of undiscovered zero-day attacks by actively working to locate and patch the vulnerabilities before black hat hackers can take advantage of them, but the success of these efforts is debatable. A key protection against zero-day attacks is the use of anomaly-based Intrusion Detection Systems (IDSs), which is the research focus of this thesis. In many respects, the only persistent defence against zero-day attacks is this class of algorithm, and their importance in the modern cyber environment is hence extremely significant.

1.1.1 Current Situation

Digital devices are spreading throughout the world at phenomenal rates, with statistics for 2012 indicating the sale of 354 million personal computers, 66 million iPads and 1.7 billion mobile handsets during this year alone [6]. The increased level of connectivity possible with this many new endpoints in the world is significant, and it is clear that modern businesses are increasingly focussing on the productivity and efficiency dividends possible when the new wave of digital technology is exploited to its fullest capacity. Furthermore, the prevalence of these devices also strongly suggests that private use of computing resources is also increasing as individuals seek to leverage the prevalence of digital technology for socialising and recreation. As the numbers of endpoints themselves have increased, so too has the amount of data being transmitted around the world, and Internet service providers have had to expand their data networks to meet the growing demand accordingly. There are strong positive societal impacts on the increasing ease of access to technology, and many business sectors have arguably become dependent on the current level of connectivity as a key enabler for their business models. The negative aspect of the proliferation of digital devices, however, is the increased security risk which accompanies their spread; as the number of physical devices increases, so too does the attack surface offered to malicious entities.

The huge prevalence of endpoints in the modern cyber environment is deeply significant. Each device, whether it be a server, personal computer, tablet, or smart phone, is a potential target for hackers. The security of each individual endpoint is contingent on the security of the underlying operating system, the security of the installed programs, and the security of the data transmission protocols. Each time a digital transaction is enacted, such as sending email, transferring funds, or sharing locational information, hackers have an opportunity to penetrate the security perimeter and compromise the device. With the number of devices so large, as indicated above, there is significant risk that many endpoints are not using the latest versions of the Operating System (OS), programs have not been patched, and insecure transmission protocols are in use. Ultimately, cyber security is an expert discipline and many, if not most, end users are not expert in the application of cyber security principles. The huge number of endpoints in the modern world means that the cyber threat exists everywhere, with a significant portion of the world's population at risk.

Further compounding the security problem presented by the modern environment is the increasing complexity of both third-party applications and the underlying OS which drives the various endpoints. Many exploitation vectors rely on leveraging small bugs in the presented software interfaces to create an environment which leads to full system compromise, such as the famous stack-based overflows which first gained notoriety in [7]. Whilst modern software engineering practices take steps to try and eliminate all bugs with particular emphasis on those which may allow arbitrary code execution, it is functionally impossible to completely eliminate all programming errors from complex programs. This in turn means that the increasingly complex software environment presented to the contemporary user also provides the contemporary hacker with a rich and varied attack surface with numerous unknown bugs. Some of these bugs will inevitably eventually be exploitable by a sufficiently talented hacker, leading to a significant yet difficult to quantify risk of zero-day attacks.

The spread of digital devices throughout the world has also led to a reasonable spread of market offerings in both hardware and software options. Increasingly, interoperability between OSs and third-party programs is considered highly desirable, and this leads to previously proprietary file formats being reverse engineered to allow display by a wide range of third-party software products [8]. Even if a vendor makes the standard for their products public, the rigour of implementation will naturally differ between software development organisations. This means that a standard may well be secure, but a particular implementation may not be. This is a uniquely significant risk in cryptographic applications, which underpin much of the security for data in motion. The increased need for interoperability carries with it an increased risk of software bugs as a result of the complexity of the task at hand, and support for legacy systems may at times negate the cutting edge of security defences. This is not to imply that interoperability should be abandoned, however; standardisation of protocols is an important part of the modern digital world, but it does carry with it the risk of a security compromise if implementation of the standard is inadequate. Rather, it is important that the increased risk presented by this aspect of the cyber environment is acknowledged and security controls increased accordingly in order to compensate.

The last great factor of the modern environment which warrants specific attention in this introduction is the huge increase in bandwidth which has accompanied the renaissance of computer technology in which we currently find ourselves. The era of dial-up modems is well and truly gone, and data exfiltration in the contemporary environment can be performed much more rapidly after an initial network compromise has been successfully conducted. This means that the ramifications for an entity if they are hacked are potentially much more significant; rather than waiting days and weeks to download large databases as was the case in previous decades, current data thieves are able to rapidly remove large quantities of confidential information with a correspondingly stark impact on the business activities of the compromised entity. The increase in network traffic also makes detecting low bandwidth stealthy hacking activity much more difficult, requiring detailed log and packet analysis to detect these actions amidst the background noise.

The discussion above makes it clear that the current environment presents a uniquely exploitable attack surface to contemporary hackers, and security mechanisms must be optimised as much as possible in order to mitigate any significant portion of the systemic risk present in the on-line world of today.

1.1.2 The Threat

The threat of hacking is steadily increasing against this backdrop of an evermore permissive cyber environment. A simple comparison between *Symantec's* 2011 [9] and 2013 [10] Internet security threat reports clearly indicates that not only are the general occurrences of malicious on-line activity increasing but so too is the threat to individual endpoints within any given network. This is particularly concerning as many organisations often concentrate protection in firewalls and bastion systems. This approach is effective to some extent at defeating traditional hacking attacks against a network's web facing presence, but it does not account for the trends reported in [10] which make it clear that targeted attacks at end users are not only increasing, but are inherently capable of bypassing these choke point protection technologies. One of the key recommendations in [10] is for an increase in endpoint protection, and it is in this realm that anomaly-based Host Based Intrusion Detection System (HIDS) can make a uniquely powerful contribution.

The nature of the threat faced by modern Internet users can be broadly categorised into three main groups, namely cyber criminals, so-called activists, and state-sponsored organisations, also known as Advanced Persistent Threats (APTs). Each group has a distinct set of goals, which naturally affects their corresponding target selection. Fortunately, strong endpoint defences are capable of defeating attacks from all three classes of hacker, although other aspects of the defensive perimeter can be optimised for the specific nature of the most likely threat class.

The term *cyber criminals* refers to those hackers who work either individually or as part of a larger collective in pursuit of financial reward. Their activities take many forms, and include credit card fraud, identity theft, theft of financial credentials, corporate espionage, data theft and data ransoming activities. Irrespective of the particular nature of their criminal focus, all of their activities are geared towards a financial pay-off and it is this aspect of their motivations which differentiates them from the other two classes of hacker.

The cost of cybercrime is increasing, and is already at phenomenal levels. In 2011, the global cost of cybercrime was estimated to be $388 billion [11], and *Symantec* calculated that the average cost per data breach in the United States in 2010 was $7.2 million [9]. These figures indicate the huge threat posed by this group of hackers, and underscores the severity of the risk associated with malicious cyber activity. Given that [10] strongly indicates that endpoint detection is a critical risk mitigator in the current environment, development of robust algorithms to fulfil this need is clearly driven by a strong financial imperative. The societal productivity gains which would be possible by any significant reduction in the cost of cybercrime alone clearly warrants research effort and is a strong motivator for the development of effective zero-day attack endpoint protection.

The second major group of malicious hackers which regularly conduct high-profile attacks are *"hacktivists"*, a somewhat colloquial term used to describe hackers with motivations arising from social or political causes. The skill level in this group is usually quite low, and the major focus of their hacking activities is on achieving publicity for their cause through website defacement or denial of service attacks [10]. Hackers of this group typically do not engage in high-end cybercrime as this does not fulfil their desire for publicity and promotion of their particular cause. On one hand, this focus means that it is unlikely that the activities of *"hacktivists"* will result in the same magnitude of effects seen in the cybercrime domain, but public loss of confidence in the company or organisation after its security is penetrated so noticeably has a steep second-order effect on the victim which may well linger for some time after the actual event.

The final major class of hacking threat is that posed by APTs. This threat has only really become an issue in recent years, with reports such as [12] clearly indicating that APTs are active in the modern cyber environment. The overwhelming evidence presented in [12] underscores the huge scope of APT activity, and news reports such as [4] make it clear that significant harm is done by the espionage-based activities of these groups. This class of hacker is potentially the most significant threat faced in the modern cyber environment, based largely on the scale of resources available to these groups due to their government sponsorship. Unlike cyber criminals who must fund their own activities, APTs have access to significant funds and infrastructure [12] and consequently are able to produce a much more sophisticated and dangerous threat. Endpoint protection is absolutely critical in defeating this threat due to the sophistication of the attack vectors [10,12], and the amount of money and resources invested by governments in these APT groups makes it unlikely that this threat is transitory. As such, the relevance of developing highly accurate zero-day attack endpoint protection to combat the rise of APTs worldwide is clearly critical, and a strong motivator for this research.

One of the side effects of increased Internet connectivity throughout the world has been an increase in the accessibility of hacking tools to a curious citizen. Tools such as MSF [1] provide ready-made hacking platforms with an extensive collection of inbuilt exploits and a simple user interface. The ease-of-use of these platforms, coupled with exploit aggregators such as `www.exploit-db.com`, has led to a growing number of security enthusiasts or "hobby hackers" overstepping the bounds of online legality. Whilst it is unlikely that open source hacking tools are directly and strongly linked the rise in global malicious cyber activity, they do afford interested parties who do not fit into one of the three main classifications detailed above with a means of exploring the security sphere, and perhaps may eventually provide recruits to these organisations.

Whilst hacking occurs as a result of many different motivations and in pursuit of many different aims, the effect is always negative and hence must be countered as much as practicable. As has been noted throughout this section, the importance of endpoint protection is growing markedly and is now widely acknowledged [10, 13]. This importance has led to the formulation of the research questions outlined in Section 1.3, and underpins all the research into HIDS presented in this thesis.

1.1.3 Countering the Threat

The diverse threat outlined above is almost impossible to defeat using any single protection method or security control. No matter how effective any individual security device may be, history has shown us that there exists a technique for bypassing it, even if this method is not yet in the public domain. This characteristic of the modern threat has led to the development of the overarching principle of *defence-in-depth* [14–18], which is a key design requirement for contemporary security systems. The *defence-in-depth* principle states that assets should be protected by overlapping perimeters of security controls so that any weaknesses in an individual layer are mitigated by the presence of the additional security layers [19]. Clearly, it is insufficient to provide numerous instances of the same type of security control, and each layer must represent a discrete security methodology in order to provide the fullest scope of protection to the asset.

Defence-in-depth is often implemented in modern systems through the use of perimeter firewalls, network-based intrusion detection systems, rigorous application of the principle of least privilege [19], anti-virus (AV) or host-based intrusion detection systems, and an effective application of sub-netting to segregate key assets from general intranets. Expert opinion in recent times has attached increasing importance to the role of endpoint protection in the overall *defence-in-depth* strategy, and remarks such as *"Endpoints must be secured by more than signature-based AV technology"* found on page 22 of [10] make it clear that reliance on traditional protection such as AV techniques is no longer appropriate; in light of comments such as this, anomaly-based HIDS clearly have a critical role to play in defending against the current scope of cyber threats.

The key contribution that anomaly-based HIDS can offer to the *defence-in-depth* principle is that these devices have an ability to detect zero-day attacks and are not dependent on signature files. Furthermore, they analyse real-time system behaviour rather than examining files in storage as is usually the case with an AV system. This means that effective deployment of HIDS as part of the perimeter provides the overall security apparatus with the ability to detect zero-day attacks in real time; this is a powerful contribution and a key enabler for effective security in the modern environment. Unfortunately, the nature of anomaly-based HIDS methodologies means that high Detection Rate (DR) is often accompanied by unacceptably large False Alarm Rate (FAR). This issue is the main limiting factor of anomaly-based HIDS effectiveness, and the production of a system with high DR and low FAR would be an incredibly valuable contribution to global cyber security.

Creation of such an HIDS is contingent on developing efficient algorithms, rich data sources, a means of implementing the core algorithm on all major OSs, and the ability to provide robustness against hacking attacks deliberately crafted to bypass detection. This is a challenging task, and forms the basis for the research questions which underpin this PhD research.

1.2 Characteristics of Modern HIDS Technology

HIDS have three key components, namely a data source, sensor and Decision Engine (DE). Whilst each component provides a central aspect of intrusion detection functionality, the data source element is significantly unexplored compared to the other two key aspects. Sensing is usually conducted by some form of OS hook, such as the BSM audit module used in compiling the KDD datasets [20, 21] and the *Procmon* [22] program which was used extensively by this research in Chapters 4 and 5 to extract data from the Windows OS. Most modern OS have high reliability options for performing this sampling function, and as such there is little scope for improvement in this area. DE functionality is usually provided by implementing one of the many abstract machine learning algorithms and applying it to the specific intrusion detection problem. As such, developments in this part of the central IDS functionality are usually contingent on advances in the theoretical machine learning field rather than an IDS-specific innovation. The data source itself, however, provides the raw information upon which activity classifications are made. Logically, any improvement in the accuracy of the data source must consequently translate into improved systemic accuracy, assuming that all other aspects of the IDS remain static. Hence, developing a better data source for an HIDS would be a significant contribution to the field and likely result in markedly better performance.

Current approaches to the data source problem use one of three main types of information. The first and most common data source used in anomaly-based HIDS is system calls, as pioneered by Forrest [23]. Forrest's seminal theory suggested that contiguous sequences of system call patterns with a fixed length could be used as an accurate discriminator between normal and anomalous behaviour. She introduced the Sequence Time-Delay Embedding (STIDE) algorithm as a means of evaluating these fixed length patterns, with significant success. A wide range of DEs have subsequently been used to analyse system call patterns, and this data source is widely accepted as the superior option when available.

Some OSs such as Windows, however, do not provide ready access to system call data and the two other main sources of HIDS information have evolved to fill this gap. The second main data source is the use of log file entries, with research such as [24, 25] demonstrating reasonable success using this approach. Finally, the third data source consists of registry entry manipulations, which allows for some measure of successful detection as demonstrated in [26–28]. Both these approaches are discussed more fully in Chapter 2, but are widely acknowledged as a sub optimal solution when compared to system call analysis and are generally only used when this data source is unavailable.

Acceptable performance for modern HIDS solutions is generally considered to be approximately 95% DR for approximately 5% FAR. Table 2.1 in Chapter 2 provides a selection of performance evaluations for several different DE implementations and data sources, and this generic assessment is supported by the performances listed in this table. This thesis has made several important contributions to the field which are outlined in Section 1.4, but the central and underpinning innovation of this research is the provision of a new semantic feature. This feature differs from the syntactic, or linear, method introduced by Forrest and subsequently widely used by other researchers in that it considers non-linear patterns in system call traces for the first time. As is soundly demonstrated in this thesis, the consideration of these non-linear patterns allows for significantly increased decision accuracy allowing DEs which use this new data source to classify behaviour on both Linux and Windows systems with close to 100% DR for an FAR under 2%. This is clearly a significant contribution, and strongly underscores the merit of this new data source.

1.3 Research Questions

Given the discussion above and the technical background outline in Chapter 2, the following research questions were formulated and investigated by this research in order to attempt to mitigate the current threat, at least to some extent:

- Can the performance of contemporary HIDSs be improved by harvesting greater information from existing data sources?
- How can decision features with greater information content be extracted from existing data sources?

- How can anomaly-based HIDS methodology be applied to the Windows OS?
- Are stealth attacks capable of bypassing a modern HIDS possible without trivialising payload impact?
- Is it possible to detect these proposed stealth attacks by using a data feature with greater information content?

1.4 Contribution to the Field

The motivation of this research was to develop a platform independent security solution capable of detecting attacks which the local system had no prior knowledge of. By creating a high accuracy HIDS capable of reliably detecting local zero-day attacks, the threat of a successful global zero-day attack is also effectively mitigated. The number of different OSs now in use throughout the world makes platform independence a key motivator of this research, as successful achievement of this requirement would greatly increase the utility of the resulting solution.

This PhD has made three key contributions to the field, and has successfully addressed the initial research questions. First, a new HIDS algorithm has been developed which provides superior performance on both legacy and modern datasets, achieving a level of accuracy significantly above that available using existing technology. Second, a method of translating system call based HIDS methodology to the Windows OS has been created. This new method allows the application of any HIDS algorithm to Windows, with performance remaining comparable to that observed on the Linux OS. This is a significant contribution, as it allows high accuracy anomaly-based HIDS protection on Windows platforms for the first time.

Finally, a new theory of payload creation was developed which allows high impact stealth attacks, again for the first time, allowing accurate baselining of IDS performance against this niche threat. Furthermore, the original HIDS algorithm developed in the first part of this research was shown to have an inherent capability to detect these new stealth attacks, providing an effective low footprint defence without any increase in system resource use; the new system has higher accuracy than any existing algorithm, without the disadvantages discussed in Section 2.6. As a result of the three innovations detailed above, high accuracy HIDS protection is now available for both Linux and Windows operating systems, with the threat of stealth attacks effectively countered.

1.5 Thesis Structure

Chapter 2 provides technical background for the research detailed in this thesis, and situates the research within the existing literature. Numerous topics are explored, including HIDS theory, DE options, data sources and IDS design methodologies. Existing research into attacks designed to bypass system call based HIDS is also outline in this chapter, with an analysis of the limited potency and impact of this existing attack technology.

Chapter 3 introduces the first new theory derived from this research effort. This new theory uses semantic analysis of system call patterns to produce previously unseen levels of classification accuracy in the Linux OS with a correspondingly significant contribution to the level of protection afforded by these systems. The performance of this new theory was tested against existing datasets such as the KDD collection [20], but also against a new dataset custom made for this research. This new dataset, the Australian Defence Force Academy Linux Dataset (ADFA-LD), uses modern hacking attacks against a modern OS variant and consequently provides the HIDS research community with a contemporary benchmark for evaluation.

Chapter 4 details another new theory which allows the application of system call based analysis techniques to the Windows OS for the first time. The new Virtual Kernel (VK) theory is a significant contribution as it allows for the enhanced protection of system call analysis in this popular OS for the first time. A dataset was created to verify the performance of this theory, and the Australian Defence Force Academy Windows Dataset (ADFA-WD) has been made publicly available so that other researchers can evaluate their own VK-inspired HIDS algorithms against a common benchmark.

Chapter 5 contains a new theory of attack technology which creates extremely stealthy payloads capable of bypassing over 47 modern AV systems. This new attack process uses system call sequences located in the normal baseline of a target program to create shellcode with a high degree of similarity to normal system activity, hence making it extremely difficult to detect the resulting attack. Fortunately, the new semantic and VK theories outlined in Chapters 3 and 4 allow high accuracy detection of these new attacks, significantly mitigating the corresponding threat.

Concluding remarks are presented in Chapter 6, with examples of semantic dictionary entries provided in Appendix A and collected pseudocode for the original processes developed in this thesis as Appendix B.

Chapter 2

Background and Literature Review

Intrusion Detection Systems (IDSs) are an incredibly important part of modern cyber defence activities and have been a developing area of cyber security for decades. As the digital world meshes more and more with the physical world, driven by the explosion of technology through vectors such as smart phones, tablet computers and increasingly versatile personal computers, so too will the security threat increase proportionately. The design of IDS systems is a complex task, with many engineering trade-offs which must be balanced to provide the optimal system for each actual deployment. Technology in this field has developed significantly since the early days of modern computing, with contemporary IDS representing finely tuned and highly complex classification systems.

Modern IDS have three basic components, namely a data source, sensor and Decision Engine (DE). As is discussed more fully in the later sections of this chapter, most Linux IDS since the seminal work of Forest [23] use a system call based data source. This research revolutionised the IDS field, and has been responsible for significant increases in data quality, almost irrespective of Operating System (OS). The sensor component of an IDS monitors changes to the data source, ideally in real time, and converts the raw data into a form usable by the DE. Finally, the DE itself uses the provided data to evaluate the system or network activity and classify it as either normal or anomalous. The research presented in this thesis is focused on improving the first two elements of the IDS triad, providing significant innovation for both data source and sensor. Abstract DE design was not the focus of this research, noting that machine learning is a niche field in its own right, and hence existing DE algorithms were adapted and used to evaluate the new IDS algorithms generated by the new developments in the first two IDS components.

Evaluating IDS performance is usually conducted by examining two fundamental metrics, Detection Rate (DR) and False Alarm Rate (FAR). A perfect system would have 100% DR whilst achieving 0% FAR, or in other words, the perfect IDS detects all attacks without ever classifying normal behaviour as anomalous. This is rarely possible, however, and represents a theoretical optimal performance rather than an achievable real-world system. A key principle in IDS research is the base-rate fallacy, which when applied to this field means that some normal behaviour will appear to be anomalous in sufficiently large populations, noting the difficulty of measuring *a priori* probabilities in complex systems [29]. This manifests itself as a residual FAR, which can at times significantly reduce the efficacy of an IDS. To some extent, FAR is a function of the DE's internal threshold; as DR is increased, so too is FAR. Given this phenomenon, IDS evaluation is usually conducted by plotting the Receiver Operating Characteristics (ROC) curve for a given implementation, with better performance equating to a greater area under the curve as discussed in [30]. This PhD research uses the ROC method as the prime evaluation tool as is common practice in the field[1], but also employs peak performance ratings for broad comparisons between algorithms.

An IDS can be deployed at two distinct levels. Network Based Intrusion Detection Systems (NIDSs) are designed to monitor traffic within an intranet in order to detect anomalous connections and malicious activity. Historically, NIDS have operated by measuring various characteristics of the packets on the network and using this data as the basis for classification decisions. With the increased use of encryption over the last 10 years, NIDS have lost significant potential data as the contents of each packet are now often obfuscated [37]. As such, modern NIDS are often reliant on data such as length of connection, connection characteristics, and protocol type to provide the majority of the decision feature [21]. This presents a very challenging task for NIDS designers and with the strongly encouraged use of `IPsec` in the new *IPv6* standard, it is likely that plaintext data sources for this type of IDS will soon become extinct.

Whilst NIDS will always have a part to play in modern layered security solutions, they are increasingly unable to provide a complete protection against malicious activity. This increases the importance of the other main type of IDS, the Host Based Intrusion Detection System (HIDS). An HIDS resides on a single host within a network and monitors the activity on that host. HIDS have access to all information, data, and system activity which the host itself is privy to; end-to-end encryption hence has little impact on the ability of the HIDS to provide an impermeable barrier against cyber intrusions. Traditionally, HIDS have focused on the host in isolation, but in recent times designers have started combining aspects of NIDS and HIDS to create a hybrid IDS which has access to decrypted network traffic and host behaviour [38]. This trend underscores the importance of the fundamental data component of

1. This evaluation technique is used widely in peer reviewed papers such as [31,32], and provides a better assessment of the IDS under review across the full range of possible decision thresholds. It is also extensively used in the evaluation of theoretical machine learning algorithms as outlined in [33–36], thus forming an appropriate performance metric for the applied machine learning algorithms used in the IDS field.

the IDS triad, and reinforces the significance of this PhD research in its quest to provide a better data source and sensing mechanism.

An important issue in computer security is defence against zero-day attacks. This class of attack refers to attacks which have not previously been observed and consequently have no logged signature in any database. As a result, signature-based systems are unable to detect or prevent zero-day attacks and provide no defence against this serious threat. Anomaly-based systems, however, operate by detecting divergences from an established normal system baseline. This class of IDS is thus able to detect any attack which creates a large enough divergence from the baseline; in effect, *all* attacks are zero-day attacks from an anomaly-based IDS's perspective. As such, detection of local zero-day attacks by an anomaly-based IDS provides directly comparable detection performance against global zero-day attacks. Indeed, the ability of anomaly-based systems to detect both local and global zero-day attacks is the major motivation for researching this class of IDS. When reading the remainder of this chapter, note that cited performance metrics for anomaly-based systems have been derived from local zero-day attacks but that these results are indicative for global zero-day attacks as well.

This research is focused on HIDS development, and NIDS will consequently not be addressed in any further detail. The focus of this research and the subsequent discussion in this chapter will hence be focused on HIDS technology, methods, and algorithms. The rest of this chapter is organised as follows: The assorted raw data sources for HIDS are outlined in 2.1, along with their strengths and weaknesses. Section 2.2 discusses the two main competing approaches to attack detection, with Section 2.2.1 introducing the main DE options which have been used in the HIDS field. Training time considerations for IDSs are outlined in Section 2.3, and the particular challenges facing modern IDS designers are presented in 2.4. The requirements and idiosyncrasies of Windows HIDS are presented in Section 2.5. Finally, the impact of stealth attacks[2] on system call based IDS is introduced in Section 2.6 with concluding remarks contained in Section 2.7.

2.1 Data Sources

As introduced in the preceding sections, a fundamental component of an HIDS is the data source which provides the inherent information upon which the DE is able to classify activity. Since Forrest's first work [23], system calls have been widely acknowledged as the preferred data source for HIDS. System calls are the method by which programs are able to access core kernel functions, and provide a wide

2. For the purpose of the discussion in this section and the research contained in Chapter 5, the term *Stealth Attacks* is defined to be the superset of all payloads with a system call sequence which has been altered to avoid detection; given this definition, mimicry attacks are rightly viewed as a subset of stealth attacks.

range of allowed interactions with the low-level OS space. Given this low-level interface role, Forrest's hypothesis that behavioural anomalies would be represented by aberrations in the sequences of system calls is intuitively appealing. Extensive testing by Forrest [39–42] and the other research teams introduced in Section 2.2.1 has resoundingly proved this hypothesis. As a result, system call based IDS are now generally acknowledged to perform much better than a comparatively complex IDS DE evaluating a different data source.

The collection of system calls is facilitated by most modern OS with inbuilt tools, and often forms an integral part of system logging software. Collection is usually conducted by grouping system calls received by the kernel using the originating process and time of collection. The result of this process is a set of traces, where each trace contains the system calls of a single process in the order in which the kernel received them. There is rarely any information provided by the logging software about the time between consecutive system calls, but modern IDS algorithms typically do not require relative timing information to perform classification tasks.

Conventional system call analysis seeks to find information about the nature of the host's activity by analysing patterns in the sequences of system calls. Forrest's seminal work [23] introduced the Sequence Time-Delay Embedding (STIDE) algorithm which is based on a self/non-self analysis inspired by artificial immune systems and is outlined in detail in Section 2.1.1. This simple algorithm defines a mismatch ratio for each new trace of system calls, and performs a classification based on the magnitude of the mismatch. More complex DEs such as Hidden Markov Models (HMMs), Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) also use the patterns present in the system call traces to perform their assessment, but employ a more complicated decision process to classify the data in keeping with the methodology of the assessing DE. System calls are an appropriate data source for both per-process IDS as well as whole-of-system IDS, and generally scale well as the scope of the system is expanded.

Whilst system calls provide a much richer data source than the other options, there are several weaknesses associated with this approach. First, logging every system call made by every running process imposes a moderate drain on the host's resources. Hopefully, this usage can be minimised in real-time applications but it is still a processing overhead which can, at times, be quite steep. Second, the length of the collected system call trace can play a part in skewing DE assessment. The IDS engineer must make a decision as to how the IDS will process system calls, and either accept the delay in evaluation whilst a standard length of trace is collected or accept a potential reduction in accuracy if non-standard trace lengths are used. Fortunately, most modern IDS are robust enough to strike a balance between these two extremes, but the trade-off remains a fundamental part of system call based systems. These two weaknesses rarely impact academic research due to the controlled nature of the experimental environment, but have an important real-world effect on IDS implementations. As such, it behoves researchers to attempt to optimise their algorithms with these trade-offs in mind in order to produce a better overall algorithm, more suited to deployment in real-world applications.

System calls are not always the best approach, however, with accessibility often dependent on the particular OS that the host uses[3]. As such, two other main data types have been trialled by IDS researchers. The first alternate data source is log files, which has been used with moderate success as shown later in Table 2.1. On one hand, log files are an attractive data source for IDS use, as almost every modern operating system employs some form of system logging software. As this software is already running, increased overhead through data collection is low and much of the data cleansing is inherently performed by the logging process. Notwithstanding this convenience, log file-based IDS are almost always less accurate than a comparable system call based algorithm due to several issues with the log entry data format [43, 44].

Fundamentally and inescapably, log file entries represent interpreted data. The process of compilation by the logging software means that some data content is lost, and any subsequent decision based on these log entries is subject to any upstream parsing which may have been made by the logging program. As the logging functionality is generally provided as an inherent part of the OS, controlling the compilation of the entries themselves is beyond the purview of the IDS designer, consequently introducing uncertainty into the data stream which results in questionable decisions and reduced performance. Furthermore, log manipulation is a fundamental part of the post-exploitation phase of a hacking attack, with many tools such as the Metasploit Framework (MSF) [1] providing inbuilt functionality to manipulate the log entries. As such, the fidelity of the data cannot be guaranteed, casting doubt over the performance of the IDS as a result. Notwithstanding these real-world issues with log file based IDS, research such as [24, 25] has demonstrated the validity of the data source in a controlled environment; log file IDS are often used in the Windows OS, for example, where system call based algorithms do not mesh well with the intensive Dynamic Linked Library (DLL)-based architecture.

Finally, IDS based on registry access [26–28] in the Windows OS has had some limited success, although the absence of a common benchmark for evaluation makes generalisation of these results difficult. The basic principle of this family of IDS is that Windows programs frequently interacts with the OS registry [45]. As such, some measure of normalcy should be possible by examining these access requests and using them to generate a suitable feature for DE evaluation. Two main flaws

3. This problem as it relates to the Windows OS in particular is discussed in detail in Section 2.5.

have limited the effectiveness of this approach, however, and a perfect system would regardless be limited to the Windows OS instead of offering a generalisable solution. The first weakness of this data source is that registry manipulation is reasonably simple using tools such as MSF [1], applying a similar penalty to this approach as for log file based systems. Second, registry access is not required for all activity and consequently does not provide a high quality metric of real-time system behaviour. It is more suited to root kit detection [27], for example, which makes subtle but persistent changes to the OS and consequently requires a footprint in the registry. This data source does not provide a general solution for IDS employment, and is unlikely to produce high quality results in the harsh modern cyber environment.

2.1.1 The STIDE Method

The system call approach proposed by Forrest in [23] produced an implementable algorithm known as the STIDE method. STIDE is designed to model an individual program, and usually requires retraining for each particular process. In the experiments detailed in this paper, STIDE was applied to general kernel traces to demonstrate its compatibility with other complete IDS solutions.

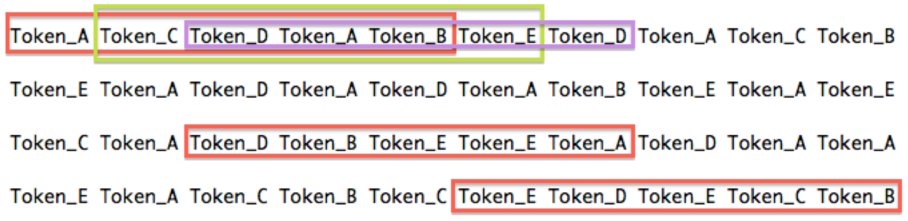
Token_A Token_C Token_D Token_A Token_B Token_E Token_D Token_A Token_C Token_B
Token_E Token_A Token_D Token_A Token_D Token_A Token_B Token_E Token_A Token_E
Token_C Token_A Token_D Token_B Token_E Token_E Token_A Token_D Token_A Token_A
Token_E Token_A Token_C Token_B Token_C Token_E Token_D Token_E Token_C Token_B

Figure 2.1: Sliding Window Example

A two-part process is used to apply the STIDE method to a given system call trace. First, a database of allowable patterns is built by using a sliding window of specified size to determine which system calls typically follow other calls. The length of this window is arbitrary and determined by the user, and is denoted as having length $k + 1$. Figure 2.1 provides a graphical example of this process for a window of length 5.

Second, once these normal patterns have been determined, new traces can be compared by repeating the process with the same window size. The databases extracted in the first step are searched for occurrences of the patterns harvested from the new system call traces. If if a pattern is not present in the existing databases, a mismatch is deemed to have occurred. Forrest defines the maximum number of mismatches possible, M_{max}, in a trace of length L using a window size of $k + 1$ as:

$$M_{max} = k\frac{L-(k+1)}{2} \tag{2.1}$$

Using Equation 2.1, any trace producing a given number of mismatches, n, can be assigned a ratio value reflecting its difference, M_n, from the training data. This scaled value can then be used to classify the behaviour as normal or anomalous based on its relation to a threshold, T, as shown below:

$$M_n = \frac{n}{M_{max}} > T \implies \text{anomalous trace} \tag{2.2}$$
$$M_n = \frac{n}{M_{max}} \leq T \implies \text{normal trace} \tag{2.3}$$

The STIDE algorithm produces good results when evaluating a single program's execution, but performs less well when considering the full suite of concurrent kernel operations[4]. The algorithm still requires training, as in other decision engines, and is sensitive to any system calls which do not appear in the initial training data. This fragility precludes easy transference between operating systems, but does provide a good level of protection for individual processes.

2.2 Detection Methods

Intrusion detection methodologies in HIDS are either based on signature recognition or anomaly detection. Signature recognition schemes operate by defining a set of identifying characteristics for each known attack type and then attempting to match real-time activity against these signatures. If a match is found, then an alert is raised and the activity classified as an attack. Signature-based approaches form the backbone of modern anti-virus (AV) systems, and are reasonably effective at detecting attacks with known characteristics. The performance of signature-based systems is a function of the comprehensiveness of the attack database; the more expansive this resource, the greater the chance that an attack will be correctly recognised. Signature-based systems have high DR and low FAR for previously seen attacks with a signature in the database, but have no zero-day attack detection capability. This is a serious weakness in the modern cyber environment, where the rapidly changing nature of the threat means that cyber entities are readily subjected to new high impact attacks. Furthermore, payload obfuscation technology and methods [46–48] are very advanced in the present day, with polymorphic code, byte code level encryption routines and DLL injection techniques [49,50] all significantly reducing the availability of clear and definitive attack signatures.

4. See Chapter 3, Section 3.6.

Whilst signature-based methods will continue to be an important element in the defensive perimeter for the foreseeable future, their inability to detect zero-day or obfuscated attacks means that they will never offer complete protection. These more exotic attack methods can only be reliably detected by anomaly-based systems, which form the second core detection method. Under this approach, the IDS establishes a normal baseline for the protected system, carefully quantifying the full range of normal activities. Any divergence from this baseline is consequently marked as a potential intrusion. Clearly, the accuracy of this family of system is strongly dependent on the accuracy of the system baseline; if natural changes in the nature of the protected system mean that the baseline is no longer accurate, FAR will consequently increase dramatically. Anomaly-based systems typically have lower DR and higher FAR than signature-based systems, but are able to detect zero-day attacks. This is a hugely significant advantage, and the driver for research in this field; if the performance of anomaly-based HIDS can be brought on par with that of signature based systems then a significant increased level of security is created as a result. Anomaly-based systems are the focus of this PhD thesis, and will consequently be the focus of subsequent discussion.

2.2.1 Decision Engine Options

The decision engine component of an IDS is clearly a critical aspect of the system as a whole, irrespective of the data source and sensor. DE selection, training and tuning are fundamental contributors to the effectiveness of the eventual IDS, and these functions can drastically impact on the overall level of protection if not correctly performed. Significant research into the use of DE's in the IDS field has been conducted over the years, with most DE algorithms having been trialled in IDS applications at the time of writing. The sophistication of the evaluating DE varies across a wide range, with simple linear algorithms such as the STIDE approach introduced by Forrest [23][5] on one end of the spectrum, offset by complicated high order algorithms such as Conditional Random Fields (CRF) [51–55], Hidden Markov Models (HMM) [56–59], and Artificial Neural Networks (ANN) [25,26,60–62] at the other end. This section concludes with a detailed examination of the relatively new Extreme Learning Machine (ELM) algorithm [63] which is used heavily in Chapters 3, 4 and 5, and an overview of the HMM which has been extensively shown to perform extremely well when classifying tokens extracted using Forrest's system call approach [23], and hence is used by this research to provide a performance baseline for comparison with the new techniques explored in the following chapters..

Clustering-like DEs such as Conditional Random Fields (CRFs) provide an excellent evaluation tool for signature-based IDS, but are often somewhat ineffective

5. Further refined by subsequent research detailed in [39–42].

in anomaly-based systems. At heart, this family of DE is optimised to separate new data into different classes. To do this, training data from each class is usually supplied to the DE during the set-up phase. This is a fundamental part of training a signature-based IDS, but anomaly-based systems must be trainable on only normal data, that is, data from one class. This limitation is a significant handicap for this type of DE, and restricts the performance of the anomaly-based IDS using these machine learning algorithms in high-end applications. Some success has been demonstrated by clustering approaches such as SVMs [28,64,65], genetic-based clustering [66], K-Nearest Neighbour (K-NN) [67–69], and the text-mining inspired "bag-of-calls" clustering approach [70] against older datasets, including the KDD98 collection [20]; whilst in some respects a proof of concept, this success is not representative of the applicability of the DE family to anomaly-based applications as a whole. As is shown in Chapter 3, Section 3.5.2, the KDD98 dataset is a very simple collection of hacking attacks by modern standards and is not representative of current technology.

As such, clustering approaches used on this data are not adversely impacted by the absence of attack data in their training routine as the simplicity of the dataset means that cluster boundaries can be easily drawn with data from only one class. This is not the case for more complex and representative datasets such as the Australian Defence Force Academy Linux Dataset (ADFA-LD) where clustering approaches are unable to provide a usable discriminator[6]. This analysis suggests that clustering algorithms are more appropriate for malware classification in a signature-based IDS, rather as the engine for an anomaly-based HIDS.

HMMs have been extensively used in intrusion detection applications, based largely on the strength of this DE in pattern analysis [71]. When coupled with the sequential system call analysis pioneered by Forrest [23,39–42], HMMs have demonstrably proven their suitability for inclusion in a HIDS; indeed, a focus in some of Forrest's later work [41] was on proving that lighter-weight DEs could provide adequate performance without the training overheads of HMMs, implicitly highlighting the benchmark performance of this DE class. Whilst most HMM-based IDS use system calls as the data source [57,58,72,73], these DEs are well-suited to any analysis of sequential information, with research such as [56] applying HMMs to privilege flows in an attempt to map local incursions. HMMs are particularly attractive in Linux systems where the linear nature of process-kernel interactions means that strict temporal chains are maintained, providing rich sequence-based information to the HMM-based IDS. Unlike clustering approaches, HMMs are able to train quite easily using data from only one class. So long as the model is defined with the possibility for two types of behaviour, training data need only come from the normal

6. See Chapter 3, Section 3.6, Figure 3.7.

baseline of the system, which allows for HMMs to be used in anomaly-based IDS. Once trained, various techniques can be used to evaluate new data passed through the DE, with *Viterbi* analysis [74] often the preferred method. HMMs are a powerful contributor to IDS accuracy, and have been used to great effect across the full scope of the IDS research sphere.

The strong performance of HMMs has led to their inclusion in more complicated hybrid systems [75, 76] with the design aim of offsetting inaccuracies in other DEs whilst simultaneously having their weaker aspects offset in turn. The benefits created by a hybridisation process, in this instance meaning the inclusion of multiple DEs in a single HIDS rather than the amalgamation of a HIDS and NIDS, are not indisputable, and potentially create inaccuracies of their own. Sophisticated decision fusion is inevitably required to combine the output of disparate DEs [77–80], which may well provide conflicting assessments of the same incident data as well as carrying an increased processing and training burden. Approaches to this problem include the use of fuzzy systems [81] and genetic-based algorithms [59], although a clearly superior approach has yet to be found. When combining HMMs in a hybrid environment, classification methods other than *Viterbi* analysis have often been used, such as the behavioural based classification used in [73], but definitive proof of a preferred classification method has not yet been reached, with each approach offering peculiar strengths and weaknesses.

In short, hybridisation is very situationally dependent and often requires a bespoke implementation. As such, hybrid IDS of this type are a valuable tool, but are not an absolute solution to classification errors and will not necessarily improve performance in all situations. Ideally, the sub-optimal nature of the hybridisation process could be avoided by a sufficiently high-performance individual IDS. Given the sophistication of modern DEs and the increasingly saturated nature of the machine learning research field, such a standalone IDS is likely to require a fundamentally richer data source or new core operating principle in order to break the evolutionary stalemate currently afflicting the research field.

ANNs are a powerful DE with excellent versatility and provide a scalable solution for most classification problems, with the ability to alter the neural net structure to best suit the particular application at hand. This type of DE has been used extensively in IDS research, largely due to its excellent pattern recognition capability [60, 82–86]. This particular strength of the ANN family lends itself to both signature- and anomaly-based algorithms, and underscores the usefulness of this DE across the whole scope of intrusion detection. Particularly strong performance has been demonstrated in using this algorithm to detect patterns in sequences of system calls[7]. In the case of anomaly-based IDS, the recognition task performed is similar

7. As discussed in Section 2.1.

to the self/non-self assessment performed by artificial immune system inspired approaches [23, 87–89]. ANNs used for anomaly-based IDS require some knowledge of the anomalous data class; as training data for this class cannot be provided under an anomaly-based regime, information is provided to the DE through various techniques such as padding the training data with a null entry and scaling the output of the DE accordingly. The ability to provide a tunable threshold to the operator is an attractive aspect of this family of DE, lending itself ably to real-world implementations which are inherently noisier than laboratory environment or collected datasets.

Table 2.1: Results from Contemporary Algorithms[8]

Algorithm	**Detection Rate** %	**False Alarm Rate** %
Data mining of audit files [90]	80.2	Not cited
Multivariate statistical analysis of audit data [31][9]	90	40
HMM and entropy analysis of system calls [91]	91.7	10.0
System call n-gram sliding window (assorted decision engines) [41][10]	$95.3 < DR < 96.9$	~ 6.0
Radial Basis Function (RBF) ANN analysing system calls [62]	96 mean	5.4 mean
Multi-Layer Percepton (MLP) on subset of KDD98 [92][11]	99.2	4.94
SVM on subset of KDD98 [92][11]	99.6	4.17
kNN with Smooth Binary Weighted RBF [93]	96.3	6.2
Rough Set Clustering [94]	95.9	7.2

The basic ANN principle has been implementing in many different ways, with machine learning researchers seeking to optimise performance through a number of techniques. Most of these implementations have been applied to IDS applications, with notable successes using RBF neural networks [62], MLPs [25, 86, 95, 96], and Self-Organising Maps (SOMs) [61]. An attractive development and machine learning field in recent years has been the discovery of the ELM [63, 97] family of ANN.

8. This table shows only results obtained by HIDS algorithms which were gained from either publicly available corpi or reliable private corpi. Unverified results have been excluded as non-representative of the current state of the art.
9. Extracted from ROC curves with DR set at 90%.
10. Whilst good, these results were obtained using a small dataset so decision engine performance may not generalise well.
11. Results potentially lack generalisability.

This relatively new algorithm avoids many of the training concerns associated with traditional MLPs, such as local minima trapping arising from gradient descent and lengthy training times, by conducting all training tasks in one pass by using the pseudoinverse function [63]. Extreme Learning Machines have much to offer the IDS sphere, and have been used extensively in this PhD research. They also seem to be well-suited to the NIDS field, with the research detailed in [98] proving their applicability.

To complete this section, comparative results for a selection of modern HIDS algorithms are provided in Table 2.1. In this table, various machine learning DEs have been used to assess assorted datasets, and as such, the results are only indicative of performance and not directly comparable. Regardless, it is clear that most decision methodologies are applicable to the HIDS task, with a reasonable conclusion being that a DR of 95% for FAR under 5% is representative of good performance against datasets of this era. Following on from the discussion in Section 2.1, there is, however, a clear superiority of system call based systems over audit- or log-based systems.

It is important to note that the results in Table 2.1 were generated using old datasets, and that performance against newer and harder datasets will naturally be worse, all other factors being equal. In conclusion, it is clear that DE performance, whilst important, is not the central factor in IDS performance; given the current high level of accuracy from most approaches, further advances must reasonably come from improvements in data and sensing, rather than DE innovation alone.

2.2.1.1 ELM Methodology

[63] contains a detailed proof and exposition of the ELM methodology. This work is summarised below for convenience.

The Moore-Penrose pseudo-inverse of matrix $\mathbf{A}$ is denoted as $\mathbf{A}^\dagger$. Assume an ELM with $\tilde{\mathbf{N}}$ hidden neurons, training data $(\mathbf{x_i}, \mathbf{t_i})$ where $\mathbf{x_i} = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in \mathcal{R}^n$ and $\mathbf{t_i} = [t_{i1}, t_{i2}, \ldots, t_{in}]^T \in \mathcal{R}^m$ and activation function $g(x)$. The output of this ELM, assuming that the underlying net structure is capable of approximation with zero error, is produced by:

$$\sum_{i=1}^{\tilde{\mathbf{N}}} \beta_i g(\mathrm{w_i} \cdot \mathrm{x_j} + b_i) = t_j, j = 1, 2, \ldots, \mathbf{N} \tag{2.4}$$

In Equation 2.4, $\mathrm{w_i}$ is the weight vector from the ith hidden neuron and each input neuron, and β_i is the weight vector connecting the ith hidden neuron and the output layer. Recalling that an ELM only modifies the weights between the hidden layer and the output layer, β, and writing Equation 2.4 in matrix notation, we obtain the compressed representation:

$$\mathbf{H}\beta = \mathbf{T} \tag{2.5}$$

where

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w_1} \cdot \mathbf{x_1} + b_1) & \cdot & g(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_1} + b_{\tilde{N}}) \\ \vdots & \vdots & \vdots \\ g(\mathbf{w_1} \cdot \mathbf{x_N} + b_1) & \cdot & g(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_N} + \mathbf{b_{\tilde{N}}}) \end{bmatrix}_{\mathbf{N} \times \mathbf{\tilde{N}}}$$

Hence, holding the input weights constant, the solution to Equation 2.5 for β is given by:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{2.6}$$

After solving Equation 2.6, the ELM can be updated with the new weights, $\hat{\beta}$ and commence assessment of subsequent data samples.

System behaviour naturally changes over time as programs evolve, updates are distributed and user behaviour adjusts to reflect new tasks. As an anomaly-based IDS relies on an accurate system baseline in order to effectively highlight anomalous deviations from this defined norm, the decision engine must allow a means of updating the system baseline after initial deployment. The ELM algorithm provides for this requirement by means of the batch training method suggested in [97], summarised below:

1. Set $\mathbf{M}_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}, \beta^0 = \mathbf{M}_0\mathbf{H}_0^T\mathbf{T}_0$.
2. Use the new training data to calculate the hidden layer output vectors $\mathbf{h}_{(k+1)} = [g(\mathbf{w}_1 \cdot \mathbf{x}_i + b_1), \ldots, g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_i + b_{\tilde{N}})]^T$
3. Calculate updated $\beta^{(k+1)}$ using

$$\begin{aligned} \mathbf{M}_{k+1} &= \mathbf{M}_k - \frac{\mathbf{M}_k\mathbf{h}_{(k+1)}\mathbf{h}_{(k+1)}^T\mathbf{M}_k}{1 + \mathbf{h}_{(k+1)}^T\mathbf{M}_k\mathbf{h}_{(k+1)}} \\ \beta^{(k+1)} &= \beta^{(k)} + \mathbf{M}_{(k+1)}\mathbf{h}_{(k+1)}(\mathbf{t}_i^T - \mathbf{h}_{(k+1)}^T\beta^{(k)}) \end{aligned} \tag{2.7}$$

2.2.1.2 HMM Methodology

Consider the system shown in Figure 2.2. This diagram represents a system with two states, $\mathcal{S}_1$ and $\mathcal{S}_2$. These states are not directly observable, however, and the state transition is a stochastic process. At each state, a token is emitted. This *is* observable, and in this example there are 4 possible tokens, denoted $\mathcal{O}_{1-4}$. The aim of an HMM is to model this process, and hence allow the user to predict the current state of the system using only the emitted tokens.

The following definitions apply to any given HMM, referred to as λ:

- N = The number of states in the system.[12]

12. In an anomaly-based IDS, such as used in Section 3.5.3, the number of states is 2 - normal, or anomalous.

Figure 2.2: Example HMM States and Observations

$$\begin{tikzcd}
\mathcal{S}_1 \arrow[rr, Rightarrow] \arrow[dd] & & \mathcal{O}_1 \arrow[dd] \\
& \vdots & \\
\mathcal{S}_2 \arrow[rr, Rightarrow] \arrow[dd] & & \mathcal{O}_2 \arrow[dd] \\
& \vdots & \\
\mathcal{S}_1 \arrow[rr, Rightarrow] \arrow[dd] & & \mathcal{O}_3 \arrow[dd] \\
& \vdots & \\
\mathcal{S}_1 \arrow[rr, Rightarrow] \arrow[dd] & & \mathcal{O}_2 \arrow[dd] \\
& \vdots & \\
\mathcal{S}_1 \arrow[rr, Rightarrow] \arrow[dd] & & \mathcal{O}_1 \arrow[dd] \\
& \vdots & \\
\mathcal{S}_2 \arrow[rr, Rightarrow] & & \mathcal{O}_4
\end{tikzcd}$$

- T = The number of symbols in an observation sequence.[13]
- M = The number of tokens.
- $S(t)$ = The state at time t.
- $\mathcal{O} = \{\text{observation tokens}\}$. [14]
- $\pi = \{\pi_0, \ldots, \pi_N : \pi_i = P(S(0) = N_i)\}$, i.e. the initial state distribution.
- $A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \cdot & \cdots & \cdot \\ a_{N,1} & \cdots & a_{N,N} \end{bmatrix}$
  where $a_{i,j} = P(S(t+1) = N_j | S(t) = N_i)$. [15]
- $B = \begin{bmatrix} b_{1,1} & \cdots & b_{1,M} \\ \cdot & \cdots & \cdot \\ b_{N,1} & \cdots & b_{N,M} \end{bmatrix}$
  where $b_{i,j} = P(\mathcal{O}_{T=t} = \mathcal{O}_{(p,j)} | S(t) = N_p)$. [16]
- $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_p, \ldots, \mathcal{O}_T\}$, where $\mathcal{O}_p$ denotes the observed token at time p.

Given these definitions, a particular HMM can be denoted by the 3-tuple $\lambda = \{A, B, \pi\}$.

To use an HMM, either the components of the 3-tuple must be known exactly, or a training process must be used to fit the model to a given set of training data. The *Baum-Welch* algorithm [99] is commonly used for this fitting task, and performs multiple training iterations of a forward and backward pass process until a given accuracy threshold is reached. This process can be extensive, and research such as [100–103] focusses on various techniques to reduce the training time without unduly sacrificing accuracy.

13. In a system call-based IDS, the window size.
14. The set of system calls allowed in a given OS, for example.
15. The state transition probability matrix.
16. The token observation probability matrix.

With an adequately trained HMM, new observation sequences can be analysed to provide the most likely hidden state transition path. This is commonly done using the powerful *Viterbi* algorithm [74], which produces good results when applied to trained models with minimal processing burden. In effect, this algorithm assesses the various probabilities contained in A and B and calculates the 'shortest' path through the field of possible states which could lead to the particular sequence of observed tokens.

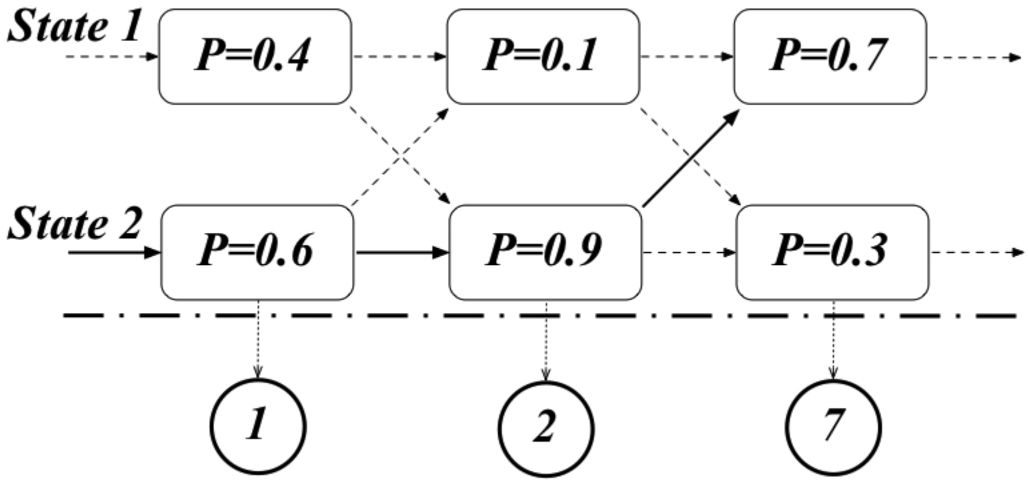

Figure 2.3: *Viterbi* Path Analysis Example

Figure 2.3 shows an example of this process, where a trained HMM evaluates a token sequence *1-2-7*. The *Viterbi* path is shown in bold arrows, and the possible state transitions are indicated in dashed arrows. This analysis indicates that the system is most likely in State 2 for the first two samples before transitioning to State 1. From this diagram, the application to intrusion detection systems is clear. To deploy this decision engine as an anomaly-based IDS, the *Baum-Welch* algorithm is used to train the HMM using known normal data. After training, the *Viterbi* algorithm is used on new data, potentially collected in real-time, to estimate the state transitions based on the new sequence of system calls, producing an overall classification of either normal or anomalous behaviour.

2.2.2 Scope of Surveillance

A central and somewhat controversial debate in the IDS field is whether IDS performance should be optimised for evaluating single processes or whole-of-system activity. Each school of thought has a compelling argument for its position, and each approach carries a unique set of advantages and disadvantages. Single process IDS stem from Forrest's first work on system call based HIDS [23] which created the STIDE algorithm. This algorithm is designed to evaluate single processes for divergences from a normal baseline for that particular process, using a technique inspired by biological immune systems. The self/non-self classifier pioneered by Forrest has been further developed by other research such as [39, 41, 59] which also use an artificial immune system approach to intrusion detection. Most bio-inspired IDS use a per-process approach to classification, comparing a program's behaviour only to a distinct and separately maintained baseline unique for that program.

A per-process analysis is not limited to artificial immune system IDS, however, and several more general DE approaches have been used in this setting [72, 104]. RBF ANNs [62], for example, demonstrated good performance when deployed in a single process environment. Similarly, clustering approaches such as [70] and the rough set theory used in [94] have also demonstrated good portability to this family of IDS. The per-process approach is also applicable to the Windows OS, with an immune system based IDS successfully implemented by the authors of [87]. The increased specificity of this type of IDS means it is particularly applicable in niche areas, such as the Remote Access Trojan focused IDS developed in [105]. This application, and others like it, are well suited to the narrow scope of a per-process IDS which enables increased accuracy at the expense of breadth of protection. Most per-process IDS are tested using datasets such as the University of New Mexico (UNM) collection [106], which provides data for a selection of Linux processes. By contrast, whole-of-system IDS are usually tested with more expansive datasets, such as KDD98 [20], KDD99 [21], and the ADFA-LD, which was created as part of this research.

Per-process HIDS have one telling advantage over whole-of-system approaches, namely an increased accuracy which arises as a direct function of the reduced decision horizon possible under this approach. By effectively training an IDS for each running process, this method reduces the amount of variation which the IDS must process, allowing for more accurate baselines and consequently more accurate classifications. This is particularly attractive in anomaly-based systems, where variation in the baseline is a direct contributor to high FAR. The second main advantage of this type of HIDS is that the training burden is also less than a whole-of-system approach. Again, this advantage is a direct result of the reduced decision horizon; with less variation in the system due to the reduced scope of analysis, it is easier to generate representative training data of smaller size, which can then be processed more quickly by the IDS DE in its training phase, simplifying the process and expediting the creation of the eventual HIDS.

These two advantages come at a steep cost, however. By reducing the decision horizon to a single process, the HIDS is inherently unsuited for any other evaluation task. This means that each running process on a host must consequently be protected by its own HIDS. In effect, this means that high security systems must spawn a HIDS for each and every program they are running, imposing a significant processing overhead on this real-world system. The consumption of system resources by numerous sophisticated DEs is significant, and drastically limits the applicability of this method in real-world environments. Furthermore, the training benefit offered by a single per-process IDS does not scale when a separate HIDS must be maintained for each program which can be run by a host. To provide complete protection, the training process must be repeated for each program installed on the computer, quickly presenting a much larger training burden then with a single whole-of-system HIDS. These marked disadvantages of the per-process method limit its effectiveness, and generally restrict the deployment of this type of HIDS to critical programs only.

The alternative approach to a per-process analysis is a whole-of-system HIDS. Under this approach, system behaviour as a whole is considered, rather than discrete processes within their own sandbox. Most IDS use this approach, based largely on the increased scalability and arguably better overall protection. The major disadvantage of this type of HIDS is that training the DE is a lengthy and involved process. First, training data representing normal system activity must be collected. This involves operating the host in all of its normal modes of activity, and collecting appropriate data during this process. The raw data must then be processed into a format suitable for analysis by the DE, and whilst this is also the case for a per-process HIDS, the scale of work required for a whole-of-system algorithm is significantly greater. Furthermore, the increased volume of training data means that the training process itself is much longer than for a per-process algorithm. This fact makes the desirability of high fidelity rapid training DEs such as the ELM [60, 63, 97, 98] readily apparent.

The critical and definitive advantage of this approach, however, is that the increased scope of the HIDS greatly enhances system security for a significantly low impact on the host's resources. Only one IDS must be run at any given time under this approach, and this IDS is able to provide detection for the full range of system activities by itself. Noting that training is usually conducted in an off-line environment, the increased training time that arises from this approach has little impact on the effectiveness of this system for the end user. Furthermore, the addition of a new program to the host does not require an immediate and complete retraining of the IDS; whilst a change in the system baseline will naturally impact on accuracy, high-end HIDS algorithms are usually sufficiently robust to withstand some variation [107]. This is in direct contrast to the per-process method where the increased accuracy is directly reliant on specific training, consequently imposing an immediate retraining burden.

When the KDD98 [20], KDD99 [21] and UNM [106] datasets were compiled, attacks usually had a large footprint in a single process. As such, per-process HIDS had all available information about an attack within their scope. Modern attacks operate in a largely distributed manner [108, 109], and are often able to spread their activity across multiple processes. This effect is particularly pronounced in Windows computers, where DLL injection attacks [49,50] are extremely effective in circumventing single process analysis. This phenomenon suggests that per-process HIDS are increasingly unsuited to the modern cyber environment, and as such, the research in this PhD thesis is focused on a whole-of-system approach.

2.3 Training Time

A secondary analysis metric for IDS efficiency is the time required to train the assessing DE. Training time is an important extension to IDS evaluation, as it directly relates to the real-world deployability of the system, but it is not an inherent part of IDS accuracy assessment. As such, it is usually regarded as an ancillary factor by academic work, worthy of comment but not a central aspect of algorithm evaluation. Regardless, there are several options with respect to the training process which inform debate in this field, necessitating the inclusion of some discussion in this chapter.

At its most abstract level, DE training falls into two categories - on-line training and off-line training. On-line training refers to the process of dynamically training the DE whilst it is actively assessing live data. This application has many attractive characteristics, including the ability to adjust the system baseline to reflect real-time changes and hence, hopefully reduce FAR. In order for an on-line training regime to be effective, the DE must be able to rapidly adjust its decision metrics in order to respond to data as it arrives in real-time. Slow training DEs such as gradient descent based MLPs [62, 86, 96] are unsuited for training regimes of this type, but rapid training DE such as the HMM proposed in [100] or the ELM methods suggested in [60, 98] present a viable option for on-line training.

Given the difficulty of on-line training, most IDS algorithms use off-line training. Under this training regime, the IDS has no requirement to dynamically update its baseline, with all training conducted prior to deployment. The benefit of this approach is that it allows large processing tasks to be done by dedicated computers without consuming resources on the protected host. This is much more appropriate than on-line training in situations where there is a large amount of training data or the decision algorithm itself requires extensive data processing or formatting. The disadvantage of off-line training is that it raises the possibility that a dynamic host may eventually outgrow the established baseline, leading to subsequent IDS assessment based on stale training. Often, this downside is offset by the increased robustness of the off-line training cycle, made possible by more expansive datasets, but it remains a distinct factor in the long term lifespan of the IDS. Periodic re-training is usually required for off-line training systems, and this must be factored into an organisation's security posture.

An important research area linked to training time is that of distributed IDS. Under a distributed approach, IDS agents such as those outlined in [108, 110, 111] are situated at key points throughout a network and share data between themselves to provide a broader level of protection. This data sharing translates effectively into distributed training as well as distributed evaluation, enabling at least some of the training burden to be spread amongst all IDS nodes. There is a ceiling to the level of distributed training possible, however, as each node will eventually require dedicated training to ensure optimal performance in its particular role.

Implementing a distributed system requires adoption of either a cooperative approach or an information sharing approach. Under the former, IDS nodes cooperate to assess activity by sharing decision verdicts as well as data. This approach has been applied quite successfully by research such as [112–114], although the long-term scalability of these systems remains untested. The advantage of this approach is that it allows for load balancing between each node, and provides a centralised protection mechanism for the surveilled network, which is consequently able to correlate disparate activity throughout the infrastructure. This correlation arguably increases the accuracy of the distributed IDS by allowing for low-footprint suspicious activity to be potentially detected, whilst also offering some saving in training time.

The second distributed approach focuses on sharing information only, leaving each individual node to make its own evaluation. This approach benefits from a reduced overall training effort, and also receives an arguably higher quality of training data due to the wider sourcing. By allowing each node to contribute to a centralised and shared database, much of the data processing and training data compilation routines can be conducted off-line and then distributed to the end user nodes. Whilst fine tuning at each node will inevitably impose a small localised on-line training burden, [88, 115] suggest that some overall efficiency gains may be possible using this scheme.

Training time is a significant issue; it has the potential to cripple the real world effectiveness of an IDS if not adequately addressed. Notwithstanding this, training time reduction is more an aspect of the IDS algorithm's implementation rather than the core methodology, and hence lies closer to the domain of software engineering than computer science. Certainly, highly accurate IDS algorithms should not be unduly penalised for requiring lengthy training in laboratory environments; rather, the challenge of reducing the length of training required should be embraced as a separate aspect of the security problem, best handled at the implementation stage rather than theoretical development stage.

2.4 Evaluating Performance

Performance evaluation of HIDS can be a somewhat controversial topic in the field. Until recently, there has been a lack of a contemporary benchmark dataset representing modern practices and operating systems which has forced the use of legacy datasets of doubtful quality and relevance or proprietary personal datasets, consequently preventing duplication of results or direct comparison with competing algorithms. Most IDS evaluation has historically been performed on the KDD98 [20] dataset, with some research also using the KDD99 [21] and UNM [106] collections. The KDD family represented an important contribution to the field at the time of its compilation, and was the first attempt to create a publicly available open source database for intrusion detection research. It has been extensively used by IDS researchers, and to some extent has been regarded as the de facto standard for many years.

Compiled in 1998-99, the KDD datasets are no longer applicable to the modern environment, with the natural evolution of technology advancing far past the scope envisaged by the datasets' authors. Arguably, the KDD datasets have been out of date for at least the last five years, which has created a significant problem for IDS research. Currency of the KDD datasets is a serious issue, and has been commented on at length in academic literature [116–118]. Lacking a common benchmark of clear relevance to the modern environment, researchers have been faced with the choice of either continuing to evaluate their work using legacy data or utilising bespoke private datasets and consequently lessening the impact of their algorithms' performance.

To some extent, the creation of the UNM dataset in 2004 addressed some of these issues, at least in the short-term. The uptake of this dataset was not large, however, based on two factors. First, the scope of the UNM dataset was much smaller than the DARPA-sponsored KDD project, resulting in a niche dataset with limited generalisability. Second, the UNM dataset was focused towards a per-process IDS regime, rather than the whole-of-system approach of the KDD collection. Noting the discussion in Section 2.2.2, this is clearly a significant disadvantage of UNM based performance evaluation.

Furthermore, and perhaps even more significantly, several researchers have commented on numerous data artefacts residing in the KDD collection [117–120]. These data artefacts are easily detectable by modern IDS, and often reside close to attacks. As such, the authors of [117–120] make a fairly compelling argument that high reported detection rates may in fact be the result of the assessing algorithm classifying the artefact rather than the actual system activity. This is particularly concerning for anomaly-based systems where sensitivity to the baseline is the fundamental enabler for the IDS; by unnaturally skewing the data stream, this class of IDS receives a significant performance benefit, potentially undermining otherwise high accuracy levels. Further compounding this issue is the reported discrepancy between various results discussed in [118, 121]. It is likely that these discrepancies are in fact a function of the underlying data artefacts, strongly highlighting the impact of this issue on the utility of the KDD family. In many ways, the continued use of the KDD datasets, at least in recent years, has been because of the lack of a viable replacement.

Noting the strong criticism of KDD-based results outlined above, coupled with the almost perfect classification of this data set by the new semantic algorithm proposed as part of this PhD research, a new dataset for Linux systems was developed by the author in order to provide more appropriate evaluation of IDS performance. This dataset, the ADFA-LD, is discussed more fully in Chapter 3, Section 3.5.2, and has been made public on the `cybersecurity.unsw.adfa.edu.au/ADFA%20IDS%20Datasets/` website with design and content details provided publicly in [122, 123]. This dataset has been designed from first principles to reflect modern hacking tactics, techniques and procedures, and uses modern patched software as its backbone. The criticisms and concerns of [116–121] were taken into account in the design of the ADFA-LD, and the completed product represents a new benchmark for Linux HIDS.

2.5 The Windows Challenge

The Windows OS is used extensively throughout the world, and enjoys the majority market share of the personal computer sector [124]. Derivatives of this family of operating systems are also used in many server applications, resulting in a high degree of market penetration in both the private and professional computing arenas. As such, security flaws in this OS present a huge attack surface to the modern hacker, with a corresponding need for layered security controls to mitigate the impact of the large number of both known and unknown security vulnerabilities.

The techniques described in the preceding sections have been modelled almost exclusively on the Linux OS. This is largely due to the different architecture present in the Windows OS, which extensively integrates DLL usage far beyond the level of linked libraries used by Linux compilers [125–127]. As such, development of Windows host-based detection security programs has generally focused on AV techniques, with most computer users at least passingly familiar with the major contemporary market offerings of this type of product. AV programs are an important contribution to security, but do not present a complete solution and remain a single facet of a *defence-in-depth* security strategy [128–131].

AV systems have a loose similarity to signature-based HIDS, but are a separate research field in their own right. For the purpose of this literature review, however, it is important to note several characteristics of AV implementations in order to differentiate them from Windows focused HIDS. AV systems generally operate by scanning a file before it is accessed in an attempt to detect predetermined virus markers [129,130]. More modern AV programs incorporate some heuristic checking, with other approaches also including various runtime protections such as port blocking and active content filtering in web browsers [129,130]. In general, the more dynamic heuristic and runtime modules rely more on a blacklist of behaviours determined by the host's security posture, rather than observed system behaviour. This is in direct contrast to an HIDS, where files in storage are not considered at all in the decision process; rather, decisions are based exclusively on observed host behaviour during program execution. These two different methodologies present the first central argument in support of the requirement for a high fidelity Windows-focused HIDS, as the inclusion of such algorithms would provide a completely novel contribution to the overall security posture in keeping with the best practice of *defence-in-depth* principles [19].

The heuristic modules included in high-end AV programs provide extremely useful protection against many hacking tactics, but are, at heart, a signature-based system. The heuristic algorithms are designed to detect behaviour which has been defined as inappropriate, admittedly using a loose and fuzzy set of logic rules, but do not employ anomaly-based methodologies to do this. As such, these systems are effective against many existing attack vectors but do not contribute credible zero-day protection. As discussed above, zero-day protection is the province of anomaly-based systems, again underscoring the need for an anomaly-based HIDS deployable in the Windows environment.

Given the demonstrated need for an HIDS, ideally anomaly-based, the design considerations introduced in the preceding sections of this chapter immediately become relevant for the Windows HIDS engineer. Noting that decision engine performance is not a key facet of IDS research, standing as a separate discipline in its own right, the major limiting factor on the efficacy of historical attempts to produce Windows HIDS is the absence of an easily accessible high fidelity data source.

Ideally, system calls would be used as the basis for Windows IDS algorithms as is currently considered best practice in Linux systems[17]. Unfortunately, the nature of the Windows OS and its heavy reliance on the incorporation of DLLs means that system calls are neither as accessible nor as representative of system activity as is the case for Linux hosts. These issues are discussed more fully in Chapter 4, where the original research of this PhD thesis proposes a solution to these problems, but the underlying cause of the issue is the multiple access paths to a single Windows system call provided as an inherent and inescapable part of the DLL-dependent architecture, in contrast to the more linear execution paths in a Linux system.

17. See discussion in Section 2.1.

In the absence of system call based data sources, IDS developers have historically used two main information types for Windows-based IDS, namely log file analysis [24] and registry analysis [26–28]. The disadvantages of these approaches when compared to system call based analysis are stark, as discussed above in Section 2.1. Regardless of this inherent disadvantage, some limited success has been enjoyed by researchers in this field, although direct comparison between the various proposed algorithms is complicated by the lack of a standardised benchmark datasets such as the Linux focused KDD98 [20] and KDD99 [21] collections.

An additional technique for both attack and defence unique to the Windows environment involves DLL *injection* or *hijacking*. This attack type leverages Window's dependence on DLLs, and seeks to replace various functions within a preloaded DLL with the attacker's code in order to achieve system compromise. This threat vector is unique to the Windows system, and research such as [49, 50] have attempted to quantify dynamic DLL rewriting in order to detect this type of attack. Whilst initial efforts have been quite promising, it should be noted that this is a niche solution to a niche problem, and not a general DLL detection methodology. As such, the search for a high accuracy generalized HIDS solution for the Windows environment is not ended by this approach, necessitating further research.

A major contribution of this PhD research is the proposal and validation of a new technique to allow access to system call-based information in Windows systems. This new concept, discussed in detail in Chapter 4, is an evolutionary step in IDS development and allows previously unseen levels of accuracy when classifying system behaviour in a Windows OS. Furthermore, a Windows dataset known as the Australian Defence Force Academy Windows Dataset (ADFA-WD) has been released to the international research community in [132] as part of this research to address the issue of benchmarking. It is hoped that this contribution will stimulate interest in this newly opened sub-field, with an overall increase in the level of security provided to contemporary Windows operating systems.

2.6 Stealth Attacks

Like any arms race, advances in cyber defence strategies and systems will soon be met by complementary developments in offensive technology. Within the HIDS realm, the increased accuracy afforded by system call based systems was offset in 2002 by the *Mimicry Attack hypothesis* [133]. Mimicry attacks relate to a proposed method of bypassing system call analysis by adding additional system calls to the hacking payload in order to increase the resemblance of the hack to normal host behaviour; in doing so, this hypothesis proposes that the mismatch between anomalous and normal behaviour can consequently be reduced, resulting in an undetected attack.

Several key terms are required to fully appreciate this theory. Hacking attacks almost invariably consist of two distinct stages. The first stage, referred to as the `exploit stage`, relates to the method of controlling execution flow in the target program. At its simplest level, this can be a stack-based buffer overflow where the instruction pointer is overwritten by an abnormally long text string [7, 134, 135], but it also includes the full suite of techniques which can be used to gain control in more complicated attacks. Note that this stage of the attack merely hijacks the control of the system in preparation to execute code; it does not actually make the program divert from its baseline. As such, it is extremely difficult to detect this stage of the attack using anomaly-based systems. There has been some limited research into detecting the actual exploit [136], but this is not nearly as effective as targeting the second stage. Indeed, from a system call perspective, the exploit is functionally identical to normal behaviour, presenting significant difficulties for HIDS which use this otherwise high quality data source.

The second stage of an attack is known as the `payload stage`. After the exploit successfully redirects execution flow to the payload, this stage is then responsible for performing the malicious activity desired by the hacker. Payloads usually consist of shellcode, so named because the focus of the payload often is to create an interactive command prompt on the hacker's computer to enable privilege escalation and post-exploitation activities [134, 135]. Once program execution is redirected to the payload by the exploit, system behaviour starts to diverge from the normal baseline, and it is at this point when anomaly-based systems begin to be effective at detecting the attack.

Given this framework, the mimicry attack hypothesis proposed in [133] suggested that the anomalous system call patterns in the payload may be able to be replaced with functionally identical sequences which form part of the normal execution flow of the target program. By doing so, anomaly-based systems would be rendered impotent and unable to differentiate between true normal behaviour and the mimicked normal behaviour. Noting the prevalence of system call based HIDS in the Linux environment, this concept is deeply troubling as it potentially nullifies the increased accuracy normally associated with a system call based analysis. Notwithstanding the significant consequences which an implemented mimicry attack would pose to HIDS algorithms, successful implementation of the concept has yet to be achieved.

The original research in [133] claimed verification of the hypothesis based on their experiments; as is discussed more fully in Chapter 5, these experiments were far from conclusive and centred on replacing a high impact payload capable of full system compromise with an extremely low impact payload design to simply add another user with a known password. Whilst the low impact payload was successful in evading the HIDS when implemented using the mimicry approach, the fact that a high impact payload was not successfully created is a significant criticism of the generalisability of this concept.

Subsequent research [137] attempted to use various techniques such as genetic algorithms to derive high impact mimicry payloads, with a measure of success. Unfortunately, the resultant shellcode is largely unusable due to its significant size, which is an unavoidable artefact of the genetic algorithm. As discussed in [134], the available space for shellcode in modern exploits is extremely limited, and large unoptimised payloads are generally undeployable in the real world. Furthermore, evaluation of existing algorithms for mimicry attack resilience is largely impossible due to the absence of high impact mimicry payloads, or indeed any benchmark dataset. Notwithstanding the somewhat unproven nature of true stealth attacks and the difficulty of implementing high impact payloads, this class of attacks represents a significant threat to HIDS.

Current mimicry attack countermeasures fall into two main groups. The first group of strategies relies on extracting more information from system call patterns, ideally in a manner which makes the disguise of attacks impossible. A popular implementation of this concept centres on a consideration of the arguments passed to the system call, with example methods for this process discussed in [138,139]. By considering the arguments of each function, this school of thought suggests that the ability to insert padding `no-ops`, a central tactic of the original mimicry attacks [133], is greatly reduced. This approach differs from traditional HIDS system call use as introduced by Forrest [23, 39–42] which only considers the positional relationships between system calls, discarding all argument information. Whilst the approaches detailed in [138, 139] present some theoretical protection against mimicry attacks, they are not without their drawbacks. Implementation of a system designed using this approach will unavoidably require the use of significant amounts of the host's resources; by designing a protection philosophy which requires the detailed examination of each interaction with the kernel, system throughput is critically reduced hence limiting the real-world protection and usefulness of these systems.

Other approaches to the search for additional information include attempts to detect the exploit stage [136]. This methodology offers some limited advancement in detection horizon, but was shown to be quite application-specific; in short, many exploit stages do not have a discernible system call footprint and hence the generalisability of this algorithm is low. Regardless, this philosophy presents a useful augmentation to traditional payload-based detection methods with minimal impact on system resource consumption. Finally, research such as [73] attempts to create a more granular decision engine in order to detect mimicry attacks as a fundamental characteristic of the DE. Whilst any improvement in DE performance will naturally have a positive impact on the utility of an HIDS, reliance on decision engines to detect mimicry attacks without fundamental changes to the provided data source provides no guarantee of broad-spectrum detection and hence does not significantly mitigate the threat.

The second major group of mimicry attack countermeasures does not seek to detect the attacks *per se*, instead aiming to complicate the mimicry process to the point of infeasibility. An example of this approach is contained in [140], where researchers demonstrate that by artificially complicating system call activity by effectively inserting 'random noise', mimicry attacks can no longer be conducted with any degree of reliability. In many respects, this is an attractive solution to the mimicry problem. It allows existing HIDS technology to remain optimised, protecting against mimicry attacks by virtue of the increased difficulty. Noting that mimicry attacks are challenging to implement, evidenced by the low impact payload used by [133], the obfuscation techniques discussed in [140] may well defeat this class of intrusion by making it computationally infeasible to implement. Unfortunately, this approach artificially slows system throughput as an artefact of the additional system calls inserted into the kernel to obfuscate genuine program activity. Effectively, these obfuscating system calls are complex `no-ops`, consuming CPU cycles to no effect. Clearly, this approach carries a steep efficiency cost and as a result is unlikely to be suitable for high-end systems.

From the discussion above, is readily apparent that stealth attacks are a significant problem and a very real threat. Current technology is unable to defeat this attack class efficiently or reliably, with most algorithms imposing a stark processing load on the protected host. An ideal solution to the stealth attack problem would consist of an HIDS which does not require artificial augmentation but is still able to detect all attacks reliably. The development of such an HIDS is irrevocably linked to the data source provided to the DE; fundamentally, current system call analysis techniques do not provide adequate information to detect mimicry attacks. Whilst system call argument consideration [138, 139] provides a potential solution, it also imposes a prohibitive on-line processing burden. With this methodology, there is no escape from the requirement to exhaustively analyse all arguments passed to each system call as it is made, which realistically necessitates examining each CPU register and the contents of the stack each time a system call is made. Clearly, this is a sub optimal solution with extremely limited relevance to real-world requirements. In order to preserve the utility of an HIDS, any additional processing required to detect mimicry attacks must be performed off-line prior to deployment. As such, there is clearly a need for a different augmented data source for modern HIDS; only by shifting the additional processing burden to an off-line arrangement will intrusion detection capabilities functionally be elevated to the required practical level for protection against mimicry attacks.

As introduced in Chapter 1, Section 1.3, two of the research questions posed for this PhD thesis centred on the design of a general theory of stealth attacks and development of an algorithm sufficiently robust to detect the general family of stealth attacks, of which mimicry attacks are a subset. Chapter 5 details the successful answers to these questions, and also contains details of a publicly available data set generated by this research and released for use by the wider research community.

2.7 Chapter Conclusion and Contribution of this Research

This chapter has introduced the key issues affecting the intrusion detection field of today. Fundamental algorithmic performance remains a central focus of IDS research, with significant work being conducted globally to improve decision engine performance, data sensing and the fundamental source of data. Innovative achievements in all three of these foundational areas has resulted in significant increases in accuracy over the last 10 years, with modern HIDS performing extremely well against legacy datasets. Unfortunately, the issues associated with these legacy datasets, such as contemporary relevance and data artefacts, impose limitations on high-end IDS evaluation. With the introduction of more modern datasets, IDS performance can be better quantified allowing for enhanced protection against contemporary threats.

The majority of IDS algorithms have been developed and evaluated for the Linux OS, and play a significant role in hardening this set of computers. By contrast, the Windows OS is largely unprotected by IDS due to the difficulty of accessing the clearly superior system call-based data source in this OS. Given this difficulty, IDS protection has had little choice but to use second-tier data extracted from log files or registry access requests to provide a limited level of IDS protection. Windows computers have consequently been forced to rely on AV technologies, which is fundamentally a signature-based system, providing limited zero-day detection capability. The absence of effective anomaly-based HIDS for this operating system is a significant weakness, and hence eminently worthy of investigation.

As system call based techniques migrate to the Windows OS, so too does the threat of stealth attacks. Whilst fiendishly difficult to implement, the threat posed by these hand-crafted payloads is significant, with effective protection remaining elusive. Current technology attempts to address this problem by either examining arguments passed to system calls, enhancing decision engine performance in the hope that stealth attacks will be detected, or obfuscating system call traces to complicate the attackers attempts to mimic valid patterns. Each of these techniques imposes a huge processing overhead on the protected host, and is far from an optimal solution. Ideally, this problem would be resolved with a solution where any extra processing is conducted off-line, consequently allowing streamlined operation with minimal consumption of scare system resources once the HIDS is deployed. These challenges create a modern research field rich with potential for significant advances. The research questions outlined in Chapter 1, Section 1.3 are closely tied to these central issues.

Chapter 3

Syntactic and Semantic Analysis of System Calls - A New Core Theory Developed for the Linux Operating System

Intrusion Detection System (IDS) performance can be improved either by crafting a more accurate Decision Engine (DE) or by increasing the amount of information extracted from a given data source. Decision engine enhancement is not limited to the intrusion detection sphere, with bodies of study dedicated to improving machine learning algorithms and practices. Whilst a separate and distinct area of study in its own right, machine learning is a tool from the IDS perspective, and does not inherently affect the underlying methodology. A better DE will always produce better results, and as decision engine technology improves so too will the performance of the system as a whole. As such, DE performance can be considered as a constant factor in any generation of IDS, with incremental but stepped performance increases occurring as machine learning advances filter through to client disciplines such as intrusion detection.[1]

Machine learning developments do not, however, increase the boundaries of knowledge in the IDS field. Improvements in decision engine technology are significant and hard-won developments, but the DE is simply a tool for this field. Any true and extended advancement in knowledge in the IDS arena simply must be driven by the development of features with higher data content. By creating a richer feature to represent the underlying nature of activity, a truer depiction of the underlying causal links of system behaviour is provided to the assessing decision engine. By leveraging the increased fidelity of activity representation, the IDS engineer is consequently able to increase performance in all areas; advancements in DE technology, however significant in their own right, will only achieve saturated performance if the quality of data provided to them is exceptional.

1. The core theories outlined in this chapter have been published in:
G. Creech and J. Hu. A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns. *Computers, IEEE Transactions on*, PP(99):11, 2013.
G. Creech and J. Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE,* pages 44874492, 2013.

Existing IDS DE data extraction uses an expansive collection of disparate information sources to generate usable classification features. In the host-based IDS world, the widely acknowledged best data source for classification accuracy are sequences of system calls made at the kernel level. System calls represent the transformation of high level abstract user and program actions into the enabling low-level functions, and are in effect the translation of user intent to binary manipulation. The use of system calls as a data source for intrusion detection systems was first suggested by Forrest et al [23] in her seminal work, and introduced a much richer data source for analysis into the IDS design process.

System calls are a superior data source because they do not pre-interpret activity before IDS assessment, in direct contrast to approaches such as log file analysis. System call analysis also provides both frequency and positional information, which increases the inherent richness of the data source. Forrest's research [23, 39–42] conclusively proved that sequence-based analysis of system calls was a highly accurate means of assessing system behaviour, as her method allowed the syntactic positioning of contiguous system calls to be considered concurrently with the frequency appearance of individual system calls for the first time in the history of the field. This drastically increased the accuracy of her classifications, and represented a marked and fundamental improvement in performance.

The extra richness which positional information affords IDS features becomes even more significant as the classification task becomes more difficult. Up until the time of writing, most IDS performance evaluation has been conducted against the KDD family of datasets [20, 21]. Whilst KDD is an extensive collection, providing a surplus of data for training and evaluation, it is a dated representation of the contemporary cyber environment and holds little relevance in the current era. It is also an easy dataset to classify; the complexity of attack attempts was low, reflecting hacking technology at the time of generation, but is far from representative of the current state-of-the-art penetration methodologies [46, 141, 142]. The KDD datasets have come under increasing criticism in recent years, with additional negative points including extensive artificial artefacts throughout the data and labelling issues [116–121]. As part of this PhD research, a new contemporary dataset was generated to better reflect IDS performance. This dataset is discussed in more detail in Section 3.5.2, but performance of legacy algorithms on this new attack collection was extremely poor. Indeed, algorithms such as [70] which dispense with positional information entirely were unable to produce a continuous Receiver Operating Characteristics (ROC) curve on the new dataset due to a fundamental inability to classify the data due insufficient feature dimensionality.

The importance of positional information cannot be overstated. The excellent performance of decision engines such as Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) relies on the extra classification dimensions which positional analysis provides. The strong pattern analysis capability of these DEs, coupled with the sequence analysis introduced by Forrest, generally results in better performance than the frequency analysis of their competitors, as discussed in detail in Chapter 2. Notwithstanding the good performance against the KDD family, however, the performance of the existing sequential analysis algorithms against the new Australian Defence Force Academy Linux Dataset (ADFA-LD) was significantly worse than when applied to the easier KDD family, and whilst better than the discontinuous ROC curve of non-positional analysis, is far from a usable IDS.

This PhD research introduces a new feature in an attempt to extract further granularity from system call analysis by increasing the richness of the positional portion of the feature. The new feature was extracted by using semantic language theory, allowing for analysis of discontinuous system call patterns for the first time. Discussed in more detail below, the core of this new theory centres on the positional information contained in non-terminating units, as opposed to the terminating unit analysis introduced by Forrest and used extensively by current IDS solutions. By introducing discontiguous sequence analysis, the new feature developed by this research allows for higher order pattern analysis to be conducted. In effect, this extracts a larger amount of usable data from the same seemingly one dimensional sequence of system calls. When coupled with a frequency analysis of the occurrence and patterning of the non-terminating semantic units, this allows for high fidelity classification and produces superior results on both the KDD dataset and the ADFA-LD.

A summary of these results is provided as Table 3.1 where the performance of semantic and syntactic implementations of a number of different DEs, including Extreme Learning Machines (ELMs), HMMs, Support Vector Machines (SVMs) and various configurations of Forest's Sequence Time-Delay Embedding (STIDE) methodology [23], have been compared using the Area Under the Curve (AUC) of the ROC plot to evaluate the merits of each approach, a technique discussed in [30] and commonly used in the IDS community. Comparative analysis of these results clearly indicates the superiority of the semantic feature and readily illustrates the contribution of this research. A full analysis of these results along with the experimental design and discussion of the implications is provided in the later sections of this chapter, particularly Section 3.6.

The increased dimensionality of the new semantic feature is significant. Since the compilation of the KDD datasets attack methodology and hacking tools have undergone significant revision. Previously, a viable attack solution was to penetrate

Table 3.1: Summary of Results for the Linux OS

KDD98 Dataset		**UNM Dataset**		**ADFA-LD Dataset**	
HIDS	Area under Curve	HIDS	Area under Curve	HIDS	Area under Curve
Semantic ELM	99.68%	Semantic ELM	78.00%	Semantic ELM	95.32%
Syntactic ELM	99.57%	Syntactic ELM	16.08%	Semantic SVM	88.93%
Syntactic HMM	98.86%	Syntactic HMM	10.53%	Syntactic HMM	76.22%
STIDE 3	27.46%			STIDE 10	86.87%
STIDE 9	56.69%				

a single process by exploiting a remote vulnerability, but the contemporary situation is significantly different. KDD-era attacks were readily detectable largely because the act of hacking changed a single process significantly; this marked change is readily detectable using only syntactic, or terminating unit, sequence analysis and has allowed for the creation of extant IDS methodology.

By contrast, modern hacking attacks have a very small footprint in any one process. The distributed nature of modern software naturally spreads the attacking footprint across numerous processes, and this natural distribution is further amplified by the direct actions of attackers who exploit this characteristic to increase their chances of remaining undetected. Hacking tools such as `Meterpreter` [1] are able to load themselves into memory with as few as 3 system calls, making their creation extremely difficult to detect. Functionality is provided to the hacker by utilising a large number of disparate processes to accomplish each act, thus making detection of the attack as a whole extremely difficult. By spreading the attack activity through multiple processes, the signal-to-noise ratio of the detection problem is greatly reduced, making it much harder to detect this sort of breach. The higher dimensionality of data produced using a semantic analysis allows for detection of many of these attacks, and is a significant advancement. As shown in the results portion of this chapter, Section 3.6, decision engines of multiple formats and structures were unable to reliably detect modern distributed attacks using syntactic features. Once the increased information content of a semantic feature was provided, however, both Artificial Neural Networks (ANNs) and SVMs demonstrated an immediate increase in accuracy, highlighting the value of the new feature and underscoring the change in attack methodology.

The rest of this chapter is organised as follows: Section 3.1 details the theory behind the generation of the new semantic feature, with a mathematical exploration of the core hypothesis. Section 3.2 details the extraction process for the new feature, including data cleansing considerations and assumptions. DE selection is examined in Section 3.4. The implementation of the new theory for Unix based systems is detailed in Section 3.5, and the performance results for the new feature are presented in Section 3.6. Section 3.7 provides concluding remarks for this chapter.

3.1 A semantic theory of discontiguous system calls

Under semantic theory, sets of rules can be used to describe sequences of pattern units [71,143,144]. From a natural language perspective, these rules form the grammar of the language in question, and allow the construction of valid sentences within that rule set [143]. These grammars can also be used to identify invalid, or ungrammatical, sentences [71,143,145,146], and it is this fact which suggests their applicability to intrusion detection systems. The hypothesis of this section of the research proposes that system calls must inherently follow a defined grammar, as they belong to a rule-based system and have a clearly defined syntax. As such, attacks should appear as invalid sentences and hence be discernible from the normal baseline of the system.

Semantic theory defines two key classes, namely terminating semantic units and non-terminating semantic units [71, 143]. These definitions are easily applied to human language, where a terminating unit is usually synonymous with a *"word"* and a non-terminating unit is usually quoted as a *"phrase"*. Under a Context Free Grammar (CFG), phrases can always be broken down to their comprising words. This generalises to the key underpinning principle of semantic theory which states that non-terminating units are comprised of terminating units which can be reduced to their constituent terminating units by means of a simple iterative transform.

The field of semantic analysis allows for many different types of grammar, the simplest of which is a CFG. This type of grammar was selected to govern this research as it is most commonly used for computer systems, with applications ranging from programming language development through to complex artificial reasoning [71,143]. By using a CFG, the number of arbitrary rules which form the language system are drastically reduced allowing for a flexible system which is easily portable between operating systems and architecture styles.

The extension of semantic theory to system call analysis equates the actual system calls to letters, a sequence of system calls as a word, and collections of system call sequences as phrases. Under the strictures of a CFG, system calls become terminating units and sequences become non-terminating units. Previous research has only considered the sequences of terminating units, in effect performing a syntactic analysis. This research hypothesises that higher order positional information can be extracted from an analysis of discontinuous semantic units, and verifies this hypothesis experimentally.

The following subsections develop the core enabling hypothesis mathematically, concluding with an inequality which forms the basis for system behaviour classification under the new methodology. This inequality, Equation 3.3, represents a new way of extracting high order positional information from one-dimensional system call traces at a system level, and is a significant contribution to the field; as will be shown in Section 3.6, this feature allows for much higher fidelity classification and greatly improves the efficacy of contemporary IDS.

3.1.1 Definitions

1. Let $\mathcal{T} = \{\text{architecture specific system calls}\}$.
2. Let $\mathcal{N} = \{\exists x \in \mathcal{T} : y = x_i, x_j, x_k.....\}$,
   or in other words, $\mathcal{N} = \{\text{all possible sequences of system calls}\}$.
3. Let $\mathcal{G}$ represent a known normal trace, and $\mathcal{A}$ represent a known anomalous trace.

3.1.2 Syntactic development

Generalising Definitions 1-3, $\mathcal{T}$ represents the set of terminating syntactic units, and $\mathcal{N}$ represents the set of non-terminating syntactic units. Using the semantic theory applicable to a CFG outlined in [143], the syntax of a system call trace can be defined as follows:

$$\exists x \in \mathcal{T} : \mathcal{G} \rightarrow x\mathcal{G}' \tag{3.1}$$

Using Production 3.1 and extending the principle to a set of training data, $[\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_N]$, yields a new set, $\mathcal{B}$, such that:

$$\mathcal{B} = \{\exists x_i, x_j, x_k, \ldots : x \in \mathcal{T} | (x_i, x_j, \ldots) \in [\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_N]\} \tag{3.2}$$

The elements of $\mathcal{B}$ are the mathematical representations of the phrases, or discontiguous semantic units, comprised of sequences of system calls. Whilst $\mathcal{B} \subset \mathcal{N}$, there is no guarantee that any specific element of $\mathcal{B}$, denoted b_q, has been specifically and individually observed in the training data $[\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_N]$. Critically, as b_q is comprised of disparate system calls, it may not physically occur in the set of observed behaviour at all. Rather, b_q represents the feasibility of a particular and unique call sequence occurring, given the observed calls from $\mathcal{T}$ in $[\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_N]$.

3.1.3 Semantic hypothesis

From the above analysis, it is clear that an anomalous trace $\mathcal{A}$ must demonstrate the same syntactic patterns as a normal trace $\mathcal{G}$, as each trace shares the same source $\mathcal{T}$ and $\mathcal{N}$. As long as the core terminating units remain unchanged, so too must the non-terminating units. Semantic analysis, however, such as that pioneered by this research, demonstrates a strong difference between anomalous and normal traces based on the positioning and frequency of non-terminating semantic units within a system call trace of unknown provenance.

The following inequality is proposed, where $\mathcal{G}_n$ represents a previously unseen normal trace:

$$|\{\exists x \in \mathcal{B} : \mathcal{G}_n \rightarrow x\mathcal{G}_n'\}| \gg |\{\exists x \in \mathcal{B} : \mathcal{A} \rightarrow x\mathcal{A}'\}| \tag{3.3}$$

This new semantic analysis of system calls proposes the key enabling principle that a collection, or corpus, of semantic phrases should be generalisable from a training set of known normal system call traces. By dynamically varying the length of each system call semantic phrase and hence allowing the phrase length to be unbound within the individual trace length, the resulting corpus will include all normal semantic units in its membership, without the need for any given phrase to be specifically observed in the training data. This aspect of semantic analysis allows for much more complete profiling of the training data, and consequently increases the accuracy of any IDS using the new feature as a data source. This characteristic requires that the training set is expansive, however, but not necessarily exhaustive. Equation 3.3, when applied to a complete training set, provides a feature suitable for provision to a DE, and will allow classification and clustering based on this feature.

3.2 Feature extraction

This section details the process for using Equation 3.3 to turn a set of system call traces into a semantic classification feature. System call traces are direct captures of calls to the Operating System (OS) kernel made by running processes. They are usually grouped by process number, and hence it is possible to extract a list of system calls in the order that they were made. There is generally no information provided about the temporal gap between individual system calls due to recording difficulties, but the sequence of the system call trace is time sensitive and ordered according to the time of logging. As such, analysis centres on the sequence pattern from a relative perspective, rather than absolute analysis of the gap between system calls or the absolute timing.

Depending on the OS architecture, extraction of system calls has varying difficulty levels ranging from quite easy, such as using the Solaris BSM logging function, to significantly challenging, such as on a Windows system. At heart, however, the capturing of these calls is a mechanical process, and is achievable regardless of OS, given appropriately engineered tools. This is an important point, as it illustrates the portability of a system call analysis technique across OSs, with the caveat that multi-kernel or multi-DLL systems present significant challenges for information fusion. Chapter 4 explicitly discusses this point, and presents a valid system call based multi-DLL IDS for the first time.

The output of system call capture systems is usually a file with a list of the limited system call names or identifying indices. The actual format of the system call identifier is immaterial, so long as it is consistent across the architecture for a given host. The code written for this research uses the KDD convention of providing system call traces as a single line file with white space delimitation. Using these files, it is possible to translate the system call patterns into a mathematical representation and then apply semantic processing to collapse the data to a usable DE feature.

Feature extraction from a given set of training data is a two pass process. In the first pass, the training data is used to compile a dictionary of observed words. This compilation is a processing intensive task, and by necessity is performed off-line. Once established, however, real-time updates are possible and revision of the word dictionary can be performed in an on-line training environment. To populate the dictionary, each system call trace is scanned, and entries to the dictionary made based on the encountered words. Starting with the first system call in the trace, words are added with increasing length until the total length of the trace is reached. The starting point is then increased iteratively, until every sequentially possible word has been extracted.

If the current word under consideration by the extraction process does not exist in the dictionary, a new entry is created. If an entry does exist, however, then the count for that entry is incremented. The iterative nature of the extraction process means that the dictionary rapidly grows to very large sizes. On the one hand, this size represents the richness of the feature, but on the other hand, the sheer magnitude of the dictionary quickly becomes unworkable due to low count words skewing the information distribution. To reduce the size of the word dictionary to workable proportions, an arbitrary filtering level was applied, and words with counts below this tolerance were discarded. This is an artificial constraint, but has minimal impact on accuracy, as demonstrated in Section 3.6. The appropriate filter level is architecture specific; for the KDD98 dataset, a filter level of 50 was used, in contrast to a filter of 200 for the ADFA-LD.

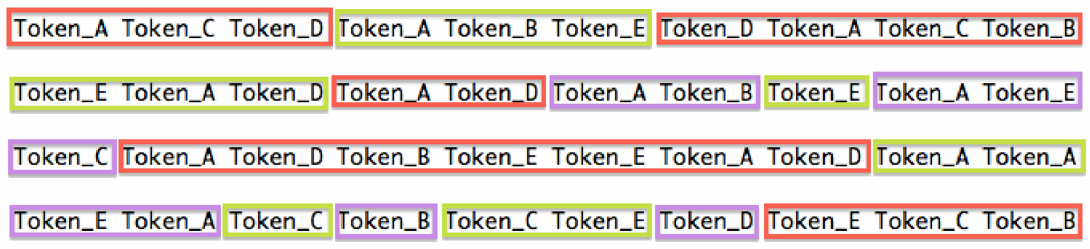

Figure 3.1: Semantic Phrase Extraction

Once the word dictionary has been compiled and filtered to the appropriate level, second pass processing is required to extract semantic phrases. As implied by the name, phrases are comprised of multiple words; in system call terms, this means that the discontiguous semantic phrase units are sequences of identified contiguous system call patterns. Second pass processing involves generating phrase dictionaries for different lengths of phrases, as illustrated in Figure 3.1. This figure illustrates the variable phrase lengths present in a system call trace, and should be contrasted with the overlapping sliding window of the STIDE method, shown in Section 2.1.1, Figure 2.1. Phrase length refers to the number of system call words in the phrase, with examples of phrase lengths 1 to 5 shown in Figure 3.1.

To generate these dictionaries, the training data is re-examined and the entries in the word dictionary are consulted to find valid combinations in the training data at the required phrase length. The compilation of phrase dictionaries is an intensive and lengthy process, as it must by necessity be performed iteratively for every word in the word dictionary. Even though some level of filtering is applied to the word dictionary during its compilation, there is still an extremely large number of words which must be tested for each portion of a potential phrase. This is not an on-line process, and requires significant processing resources. Phrase dictionaries of length 1 to 5 were compiled for the results generated in this chapter, with this complexity providing a good balance between processing overhead and result granularity. An increase in the length of phrases should increase the accuracy of the IDS DE, but with a significant processing price.

The results of the method described above provide a set of phrase dictionaries for a given series of lengths. Some examples of the entries in these dictionaries are provided in Appendix A, Section A.1, which demonstrate the use of semantic tokens as *keys* which provide access to the associated occurrence count. To some extent the selection of phrase length is arbitrary, but as the length increases so too does the dimensionality of the information. A balance must be struck between phrases so long that they occur rarely, and phrases so short that they do not significantly contribute information to the evaluation process. Empirically, as shown in Section 3.6, using phrases of length 1 to 5 provides a good balance between these two extremes and allows for efficient and accurate classification.

Once the phrase dictionaries have been established, the final step of feature extraction is to use these dictionaries to gather metrics from any provided system call trace. The method used to produce this research's results was a frequency analysis of the occurrence of each phrase for a given length in the system call trace under analysis. To perform this evaluation, phrases were extracted from the system call trace and compared against the established dictionaries. If the phrase was present in the dictionary, the frequency of occurrence of the stock phrase derived from the training data was used to weight the number of times the given phrase was seen in the sample to produce a numerical result. By repeating this process for all observed phrases in the sample trace, a weighted value representing the number of "normal" phrases encountered was calculated. This weighted value was used as the input to the DE for this phrase length. Whilst on one level this process can be classified as a frequency analysis method, positional information is inherently considered in the formulation of the final feature metric. This distinguishes the new method proposed by this research from existing frequency analysis methods such as [70], and is responsible for the significantly better results obtained using this new semantic method.

By repeating this process for all phrase lengths, a multi-element feature for each system called trace can be constructed. The format of the resultant feature is shown in Table 3.2, and sample extracts of real semantic dictionary entries are shown in Appendix A, Section A.1. This table presents example frequency counts for phrases of varying lengths, with Sample 1 representing a normal trace and Sample 2 representing an anomalous trace from the KDD98 dataset. Before provision to the DE, data elements are normalised as demonstrated by the second half of the table. Once the DE has been trained using the newly extracted features, classification can be performed by comparing the DE output against a threshold, and classifying the trace based on this result.

Table 3.2: Feature Format

	Phrase Length				
Sample Number	1	2	3	4	5
Raw value: $i_R \in \mathcal{I}_\mathcal{R}$	Dict. 1 Count	Dict. 2 Count	Dict. 3 Count	Dict. 4 Count	Dict. 5 Count
Sample 1 (Raw)	1014	1118	713	397	130
Sample 2 (Raw)	345	145	4	0	0
Normalised value $i_N \in \mathcal{I}_\mathcal{N}$	$i_R[1]/\max(\mathcal{I}_\mathcal{R})$	$i_R[2]/\max(\mathcal{I}_\mathcal{R})$	$i_R[3]/\max(\mathcal{I}_\mathcal{R})$	$i_R[4]\max(\mathcal{I}_\mathcal{R})$	$i_R[5]/\max(\mathcal{I}_\mathcal{R})$
Sample 1 (Normalised)	0.0412	0.0454	0.0289	0.016	0.005
Sample 2 (Normalised)	0.0140	0.0059	0.0002	0.000	0.000

The multi-stage feature extraction process allows the creation of rich training data, which can then be used to tune the selected DE. Semantic features are appropriate for all DEs, but the experimental results presented in Section 3.6 show a performance edge for artificial neural networks. Adaptation of the data for different DEs is discussed in Section 3.4, and pseudocode for the extraction process is provided in Figure 3.2.

The semantic algorithm proposed by this research has an inherent theoretical resilience to mimicry attacks as introduced by [133] and investigated further by [47, 64, 73, 109, 137–140, 147–149]. This type of attack, whilst serious, relies on manipulating system call traces to avoid detection. The original authors of [133] had some limited success in bypassing syntactic IDS using this technique, although notably failing to obfuscate a high impact payload. The increased dimensionality of the semantic feature detailed here means that there is no scope to amend system call traces without fracturing high order semantic chains; in doing so, the inequality presented in Equation 3.3 is reinforced, and the trace will be consequently labelled as anomalous. This theoretical resilience to mimicry attacks is further examined in Chapter 5, with particular emphasis on the Windows OS.

Figure 3.2: Semantic Feature Extraction Algorithm

```
function getWords(traces)
    for all traces do
        counter ← 1
        for system calls in trace do
            word =$systemcall$+$nextcountercalls$
            if word in wordDictionary then
                Increment count of word
            else
                Add word to wordDictionary
            end if
            counter + = 1
        end for
    end for
    return wordDictionary
end function
function genPhrases(word dictionary, length)
    Create new phrase dictionary for phrases of given length
    for all words in word dictionary do
        while current phrase length < length do
            currentPhrase ← currentWord
            for currentWord do
                currentPhrase + = next dictionary word
            end for
        end while
        Add phrase to phraseDictionary
        word list start position++
    end for
    return phraseDictionary
end function
function getPhraseCount(trace)
    featureVector = new array with length=number of dictionaries
    for all Phrase Dictionaries do
        $i$ ← phrase length for dictionary
        phraseCount ← 0
        for all Phrases in Dictionary do
            if phrase present in trace then
                phraseCount++
            end if
            featureVector[$i$] ←phrase count
        end for
    end for
    return featureVector
end function
```

Figure 3.2: Semantic Feature Extraction Algorithm (cont.)

```
44: function EVALUATESYSTEMCALLTRACE(newTrace)
45:     newFeature ← getPhraseCount(newTrace)
46:     normalise newFeature
47:     feature → trained decision engine
48:     deResult ← decision engine output
49:     if deResult > global threshold then
50:         classification ← anomalous
51:     else
52:         classification ← normal
53:     end if
54:     return classification
55: end function
```

3.2.1 Data cleansing

The new semantic feature discussed above has an inherent level of robustness due to the increased entropy and dimensionality of the positional component, but the high processing overhead means that corrupted or non-representative data has a stark real-world impact on the time taken to produce usable features. As such, and noting the impact of extended training times on system level IDS effectiveness, it is critical that data cleansing be applied to training data compiled using this feature.

The above requirement does not mean the training data should be manually selected or overly filtered; this is unrealistic, and does not represent a system usable in the real world. Data cleansing for this research was limited to evaluating the relative lengths of system called traces considered. This is a realistic method of limiting the impact of bad data on the resulting feature, as it is implementable by controlling the collection system. When evaluating the KDD data, no filtering was applied, and training data was simply selected from their provided pool. The ADFA-LD, on the other hand, required size filtering on the collected traces to prevent unworkably large file sizes. For this dataset, training data was excluded if it fell outside the size range 300 bytes to 6 kB. Validation and attack data was filtered so that each trace size was between 300 bytes and 10 kB.

Scope exists to further refine the purity of collected data prior to processing by evaluating the range of system calls presented by each case. Ideally, every system call allowed by the architecture would be represented at some stage in the training data, but realistically, this may not be the case. If system calls are not represented in training data, this can present problems for some decision engines such as HMMs and other sequential analysis engines. Additionally, careful selection of training data, if feasible, to encompass the entire suite of normal system behaviour will naturally improve the accuracy of the resulting DE. Rejection of system call traces consisting of only a few different calls will increase the variety of the training data as a whole, but is not necessarily representative of the underlying system behaviour. Care must be taken to ensure that any pruning of collective training data does not adversely or unrealistically skew the resulting set. The robustness of the semantic feature is sufficient that minor aberrations should not unduly affect the ultimate classification. As such, the only data cleansing that should be undertaken for this feature must be driven by processing requirements, rather than any manual rejection of perceived bad data.

3.2.2 Assumptions

The following key assumptions were made when implementing the new semantic theory as an Host Based Intrusion Detection System (HIDS) algorithm:

1. A filter level can be applied to the word dictionary, as introduced in Section 3.1.3, to expedite feature extraction without unduly affecting the underlying accuracy of the IDS;
2. A linkage between phrase length and accuracy of classification exists; and
3. Anomalous traces will have a lower count of phrases extracted from normal training data than that observed in previously unseen new normal traces.

The first assumption was made due to an observed large number of low count entries in the initial word dictionary compiled from both the KDD98 dataset and the ADFA-LD. Indeed, at times over 20% of the entries in the word dictionary for both data sets consisted of an item with only one occurrence throughout the training data. These low count items can safely be pruned as they are not representative of the dataset as a whole; notwithstanding the empirical veracity of this statement, there is still an underlying assumption that these low count items are not an integral part of the data representation. This assumption has been confirmed experimentally based on the strong performance demonstrated by the semantic systems trialled in this thesis when using this assumption, but there is a possibility that it may not hold for high complexity operating systems. If accuracy were seen to adversely decline during such a deployment, this assumption would need to be revisited as a potential source of inaccuracy and correcting action taken as appropriate. The removal of the low count filter would significantly increase the processing time at all stages of phrase dictionary compilation, and hence should only be considered if accuracy is untenable.

The next assumption relates to the presumed merit of high-order semantic phrases. As semantic phrases increase in length, thus containing more words, the number of occurrences of any given phrase within a sample of finite length naturally decreases. Hence, it is logical to assume that finding a long phrase length is a good indicator of semantic normalcy, and hence assign more weight to this evidence. This assumption has been empirically confirmed as valid for the KDD98 dataset, UNM collection, and

ADFA-LD, again based on the strong performance demonstrated by the various semantic systems detailed in this thesis, but may not necessarily hold for all operating systems in all environments. The increase of phrase length is analogous to considering higher-order dimensionality within the positional information of a system call trace, and as such, each distinct phrase length will represent a particular level of complexity in the supplied system called data. Data cleansing and sampling size filters may artificially limit the occurrence of high order phrases, thus unnecessarily reducing the count of these long semantic units. Conversely, continual unfiltered sampling may create an unnaturally high occurrence of long phrases. Semantically significant phrases may also be split over two temporally consecutive capture files, potentially losing critical high order data. Poor accuracy may eventuate as a result, as the underlying strength of the semantic approach is the inclusion of higher-order positional information than used by syntactic approaches. In the event of mediocre IDS performance, an increase in the phrase length under consideration may assist by introducing higher-order data into the decision engine classification process. There is a ceiling to the effectiveness of higher-order phrase use, however, as low occurrences of inordinately long phrases do not represent the underlying information content well, and may unavoidably skew classifications. The balance between phrase length and accuracy will be OS dependent, and should be empirically determined prior to deploying a semantic IDS in the wild.

As the dimensionality of the training data for any decision engine increases, so too must the sensitivity of the DE to the selection of this training data. Approaching this problem from a slightly different frame of reference, as the sensitivity of any classification system increases, the impact of outliers in the training data will be more pronounced. This characteristic of finely tuned decision engines manifests in the new semantic IDS theory developed by this research in the assumption of representative training data. Datasets such as the KDD collection have a surplus of system call traces, with hundreds of thousands of potential training elements to choose from. These traces are well labelled, but no study into the amount of variation between traces of the same label has been undertaken. Indeed, there is no guarantee that any data selected for training or validation from datasets of this magnitude will be representative of the inherent activity of the host OS. By contrast, the ADFA-LD is of much smaller size, and is targeted at the evaluation of a single host-based IDS[2]. Whilst every effort has been taken to ensure that this dataset presents a cross-section of reasonably normal system activity, it is possible that the training data is not representative of the normal system baseline. The assumption of representative data relates more to system deployment rather than algorithmic design, but is a critical accuracy input. Whilst theoretical evaluation of

2. Details of the ADFA-LD are contained in Section 3.5.2.

IDS performance using benchmark datasets will apply the same training data to all algorithms, and hence the same inherent assumptions, real-world performance does not allow the same level of control; grossly inaccurate classifications in a deployed system may consequently arise from non-representative training data.

The final key assumption is specific to the new semantic theory, and relates to the forecast frequency distribution of phrase occurrence in normal and anomalous traces. As discussed above in Section 3.1, the core proposal of this new theory is that normal traces will present a higher count of known-normal semantic units when compared to an anomalous trace. In other words, this new theory assumes that once a dictionary of semantic phrases has been compiled from normal data, these phrases will occur more often when the system is operating in a normal mode than when under attack. This principle is encapsulated in Equation 3.3, and underpins the entire semantic theory of intrusion detection. Whilst this assumption has been proven empirically, with the strong results against the KDD98 dataset and ADFA-LD underscoring the efficacy of the new theory, it is nonetheless an assumption. There is no guarantee that all operating systems will necessarily exhibit similar behaviour, but it is a reasonable assumption given the theory outlined in Section 3.1.3.

The assumptions detailed in this section are critical to the successful application of semantic theory to the IDS problem. Each assumption has been tested empirically, and found valid against all available datasets. This strongly suggests that these key assumptions can be extended across operating systems as required, but remain potential points of failure in an end-state IDS. Careful re-evaluation of the validity of these assumptions should be an ongoing process for any IDS engineer, but they are unlikely to change in the short term.

3.3 Time Required for Feature Extraction

The time required to conduct semantic feature extraction is considerable. This becomes even more lengthy as phrase length is increased, suggesting that there is a functional ceiling to the maximum phrase length possible. When compiling the semantic dictionaries used for the KDD98 data, four weeks were required to construct the dictionaries. Whilst unarguably significantly longer than the time required to produce syntactic dictionaries such as might be used for the STIDE family of approaches, it should be noted that this processing time used unoptimised and interpreted *Python* [150] code rather than the compiled code used by the various C implementations of traditional algorithms such as HMM, STIDE and clustering DEs. If the semantic extraction process was implemented in a compiled language and subsequently optimised for performance speed, this large initial processing time would be significantly reduced.

Furthermore, note that training and feature extraction time is rarely considered as important for HIDS performance as the Detection Rate (DR) and False Alarm Rate (FAR) characteristics of the IDS. There is certainly no commonly accepted baseline for performance in this area, and whilst the semantic algorithm undeniably requires longer processing time than syntactic approaches, the vastly improved classification performance of the new family of HIDS markedly offsets this longer set-up time. Finally, training a DE with a semantic feature requires no greater time than with a syntactic feature, and it is only the extraction process itself which is longer. As such, the feature extraction time required to compile semantic dictionaries is not viewed as a significant detraction from the new algorithm, and is only a minor point when contrasted with the excellent classification accuracy possible under the new scheme.

3.4 Decision engine selection and adaptation

Numerous decision engines can be applied to intrusion detection. Each particular type has its own advantages and disadvantages, but functional performance has been demonstrated by most algorithms, at least when classifying KDD era datasets. A detailed comparison of the various methods has been presented in Chapter 2, with comparative results contained in Table 2.1. Evaluation of the new semantic feature was conducted by applying two types of algorithms well-suited to the peculiarities of this new data representation, and then comparing these performance results against various other recent IDS algorithms. It is somewhat rare to find IDS performance evaluation tests conducted against an entire corpus; as discussed in Section 3.5.1, many algorithms focus on a subset of attacks, or limit the scope of testing in some other way. All algorithms used to generate the results presented in this thesis were evaluated against all attacks present in each dataset with an observable footprint in the system call traces. This accounts for the generally worse performance of many algorithms under the test conditions imposed by this research when compared to the original publications, but presents a much truer evaluation of each individual performance. A central tenet of IDS design should be the system's applicability to the real world, and artificially limited testing horizon unacceptably distorts any performance evaluation.

The first algorithm examined by this research's baseline measurement was the seminal STIDE method proposed by Forrest [23]. This method was the first attempt by researchers to combine positional information and frequency information, and was extremely successful within its testing scope. Under the definitions introduced by this thesis, STIDE uses a syntactic analysis, comparing the positioning of terminating semantic units or 'words' within an observed system call trace. STIDE operates by training the evaluating IDS on a set of known normal data from a single process. New data from this process is then scanned as it arrives at the collection point, and the syntactic units compared against the known normal corpus compiled from the training data. A percentage mismatch is calculated, and the activity deemed as anomalous if the resultant figure breaches an arbitrary threshold. STIDE was optimised for a single process level deployment, rather than the whole of system arrangement common to other IDS. Regardless, it is still able to perform better than many algorithms in a full system deployment, based largely on its consideration of syntactic positional information. STIDE was trained for this research using the same stock training data as the other algorithms. In a divergence from the other methods, STIDE requires knowledge of all system calls allowable in the system architecture to correctly compile its classification engine, and hence this information was provided as a simple text file. There is no impact by this provision on the testing fidelity, as this information is readily available for any given OS architecture, and does not represent an unfair advantage. STIDE trained more rapidly than any other algorithm, largely due to the simplicity of its classification method. The generally good performance provided by STIDE against single process attacks despite the simple proto-DE is strong evidence for the importance of positional information, with this significance reaching its natural conclusion in the new semantic theory proposed by this thesis.

Clustering algorithms have been extensively applied to the KDD98 data, with generally good results, shown in Table 2.1. When evaluating performance against the ADFA-LD, clustering algorithms were also used as part of the comparison process. Both the K-Means and the K Nearest Neighbour clustering algorithms were implemented using the frequency approach proposed by [70]. The use of clustering algorithms for anomaly-based detection is somewhat controversial. Unlike signature recognition systems, anomaly IDS only use normal training data to tune the DE. This means that the classification tool must inherently allow for two classes, normal and anomalous, whilst only receiving data about the first class. For clustering algorithms, this means that the cluster boundaries must be very well-defined, and classification is consequently based on a new data point's position with respect to this delimiter. The low complexity of KDD era data, discussed in Sections 3.5.1 and 3.5.2 and displayed visually in Figure 3.4, posed little problem for this family of classification algorithms. The ADFA-LD, on the other hand, presented an insurmountable challenge to traditional clustering approaches due to the difficulty of defining a cluster boundary with this highly complex dataset. This poor performance against the new dataset suggests that clustering algorithms require extensive modification prior to reaching effectiveness against the new wave of attack methodologies.

HMMs have been applied to intrusion detection with great success. This class of decision engine is extremely good at recognising patterns in sequences and deducing the most likely class transitions which caused the observed tokens. By equating system call sequences to tokens, HMMs are capable of modelling the transition between normal and anomalous behaviour with a high degree of accuracy. Much of this accuracy in IDS applications derives from HMMs simultaneous consideration of frequency and positional information. Unlike pure frequency analysis methods such as [70], HMMs inherently require positional information and base much of their decision making on this data source. Frequency information is considered implicitly, and is captured in the number of sightings of repeated patterns which the decision engine observes in the provided sequence. For this research, HMMs were used as a baseline performance standard for syntactic methods. HMMs are widely acknowledged as exhibiting superior performance in the traditional IDS realm [56–59, 72, 77, 78, 100], and as such can reasonably be used as a fundamental comparison metric; if HMM performance is exceeded by a new IDS, then this represents a significant contribution.

In preparing the HMM for classification, training data was provided as a series of numerical tokens representing the index of each system call within the specific architecture. The HMM was defined as having two classes, but only normal training data was provided. Viterbi analysis was then used to evaluate new traces from the validation and attack data, with traces classified according to the resultant labelling. The training data was tokenised with different window lengths, similar to the method used in the STIDE approach, and this changing window was used to evaluate performance across a range of configurations. Another sequential analysis method, CRFs, were considered for inclusion in the benchmarking tests, but eventually discarded. Whilst CRFs demonstrate extremely promising performance in some IDS applications, they are limited to signature-based systems. This arises from their fundamental reliance on conditional probability, which imposes the requirement for training data to contain samples from each possible class. Anomaly-based systems only provide normal training data, and hence this class of decision engine must unfortunately be removed from the pool of potential classifiers.

The first semantic-based DE used for the evaluation experiments on the ADFA-LD, detailed in Section 3.5.2, was a single state SVM. In some respects, the SVM shares many similar characteristics with clustering approaches, but the increased versatility and sensitivity of this methodology lends itself to an accurate semantic-based classification. A single state SVM operates by defining a multidimensional hyper plane which encompasses the supplied training data, all of which is from a single class. Classification is performed by evaluating whether or not new data lies inside or outside the hyper plane. To prepare the semantic feature for evaluation by the SVM, the feature format detailed in Table 3.2 was used. Training data was processed using the established semantic phrase dictionaries for depths 1 to 5, and a frequency occurrence count established for each training element at each phrase depth. Prior to using the data to train the SVM, all data elements were normalised using the maximum observed count as the normalisation factor. After training, the SVM was able to predict the class of new data with a reasonable degree of accuracy. Best performance was achieved using an Radial Basis Function (RBF) kernel, with extremely small very values of γ and ν. Inspection of Figure 3.4 indicates that these small values are required due to the high degree of similarity between normal and attack data in the ADFA-LD, and hence the hyper plane envelope must be tightly enforced to provide accurate classification. The single state SVM represents a viable anomaly-based IDS, however it requires careful multivariate tuning to perform, which is a potential weakness of this DE.

ANNs were the final semantic-based DE used in this research. This type of DE is well known for its strong pattern recognition capability [83], and performed extremely well in these experiments. The type of ANN used for this experiment was the ELM, a methodology proposed by [63, 97]. This method achieves the same function and behaviour as an Multi-Layer Percepton (MLP), but trains by calculating the pseudoinverse of the training data and using this to adjust the hidden layer neuron weights in one pass. This training method can be performed much more quickly than traditional gradient descent methods, and avoids local minima concerns easily. As such, it represents an excellent lightweight but high-accuracy neural network and is eminently suitable for deployment as an IDS DE. The ELM used in this research was prepared using the same training data as supplied to the single state SVM with a small amount of null data appended in order to scale the output, and resulted in five inputs to correspond to the five phrase depths. Various hidden layer sizes were trialled, with optimum performance achieved with a hidden layer size of 31 neurons. A single output from the ELM was defined, providing a numeric value scaled between 0 and 10. A threshold was applied to this output, with labels assigned based on an output's position above or below this threshold.

All decision engines require training prior to deployment. Whilst the initial training can often be conducted in an off-line environment, hence decoupling the impact of lengthy training times from the effectiveness of the end state system, a dynamic environment will quickly require retraining of the decision engine to at least some extent. Anomaly-based IDS are no exception to this requirement, and indeed, the sensitivity of an anomaly centric system to changes in the underlying system baseline is significant. Fundamentally, if the IDS does not have a precise baseline of the protected system, the false alarm rate will climb rapidly and soon become untenable. Training time for all the decision engines discussed above was not considered as a performance metric in the results in this thesis, but does play an important part in the transition to real-world deployments. Importantly, the ELM, used as the preferred decision engine for semantic features because of its superior performance, trained extremely rapidly. Whilst fundamentally an MLP, the ELM is able to rapidly adjust its underlying neuron weights in single pass training by use of the pseudoinverse function. Pseudoinverse calculations are an intensive process, and ill suited to use in lightweight systems where CPU cycles are at a premium; on a modern desktop computer, however, the transitory load on a multi core CPU has no tangible impact on system performance and hence is an extremely attractive method for rapid IDS training. By removing the incremental nature of training common to so many types of DE, the ELM allows for extremely rapid adjustments and is an ideal anomaly-based IDS decision engine.

The process of extracting a usable semantic feature from provided system call data takes a significant amount of time to complete when compiling the initial training dictionaries. As such, training is strictly an off-line activity. Evaluation, however, can easily be conducted in real-time once the phrase dictionaries have been constructed from the training data. This means that whilst a significant investment in processing time must be made when establishing the semantic-based IDS, system protection is provided in real time at the demonstrably increased accuracy level once feature extraction is complete. The impact of the extended feature extraction time is hence minimal, and more than adequately offset by the enhanced real-time evaluation provided by this approach.

IDS fundamentally rely on the use of a decision engine to provide the classification function. The complexity of the DE can be low, such as the method used by the STIDE algorithm, or complex such as CRFs or HMMs. Regardless of the decision engine used, however, the underlying feature provided to the decision engine has a critical role to play. The use of a wide range of DEs for the experimental section of this chapter indicates strongly that whilst there are some differences between individual DE performance, the major factor is the use of a semantic or syntactic feature. The classification algorithm used by any particular IDS has an important role to play in providing a high-quality high-fidelity system, but good decisions require good data, which in this setting means data which is highly representative of the host's activity. The major contribution of this research is not towards the decision engine methodology per se, but in the increase of data richness and dimensionality offered by the semantic feature. The semantic feature can be applied to any decision engine, with varying degrees of manipulation and formatting. The richness of the feature means, however, that training times may increase drastically for some families of decision engine. The ELM, which has performed better than any other DE in these trials, does not consider the high order positional information on the new semantic feature, for example. Whilst a frequency based analysis of semantic units inherently includes positional information, ANNs are not a sequential analysis tool. A full semantic feature could be applied to decision engines such as HMMs, which are optimised for sequential analysis, but the processing overhead and iterative consideration of all phrase units is significant and prohibitive for normal computers. As such, advances in the decision engine design field to simplify training processes and expedite the convergence of sophisticated positional DEs have much to offer the IDS world, and will expedite the inclusion of semantic features as the basis for a broader range of classification algorithms with a corresponding increase in decision accuracy.

3.5 Adaptation for Unix-based systems

This section provides an overview of the different data sets used to evaluate the new semantic feature, contrasted with traditional syntactic methods. The initial comparative results were generated using the KDD98 dataset, and then the robustness and portability of the semantic feature was tested by evaluating a subset of the University of New Mexico (UNM) dataset using an IDS trained on the KDD data. Whilst initially counter-intuitive, this experiment evaluates the ability of a semantic IDS to be centrally trained, and then fine tuned on individual hosts within a network, given a degree of commonality between the endpoint operating systems. HIDS are very sensitive to the training data provided, and to the training process in general. Often, large amounts of time and resources need to be dedicated to training each individual HIDS on each separate host. By introducing a feature which has sufficient data richness to allow for the bulk of HIDS training to be performed offline in a centralised and generalised fashion prior to endpoint distribution for specific training, this research makes a significant contribution to the expediency of large scale HIDS deployment.

A strong performance was demonstrated by the semantic feature when contrasted to the syntactic methods, with a clear superiority shown when classifying a different version of the OS than the version from which training data was extracted. A new dataset was generated by this research to more accurately baseline IDS performance against modern hacking threats, and the structure and format of this dataset is discussed in Section 3.5.2. Finally, implementation of the experiments and their configurations are discussed in Section 3.5.3.

3.5.1 The KDD98 and UNM Datasets

The KDD98 dataset represents a collaborative effort between MIT and DARPA, and was the first large-scale compilation of intrusion detection activity. At the time, it heralded the increasing focus on cyber security of the late 1990s, and attempted to standardise the evaluation of HIDS and Network Based Intrusion Detection System (NIDS). The importance of this dataset is clear; KDD98 represents a seminal contribution to the field, and is still used to this day. Notwithstanding the extensive criticism by contemporary authors [116–121], this dataset has shaped much of modern IDS research. The full dataset represents an extensive collection, consisting of multiple weeks of system uptime. A complete network was constructed for this project, and network traffic was gathered along with host activity. For the purposes of this research, system call traces were extracted from a machine running Solaris 2.5.1, named `Pascal`. System call traces were assembled from the raw BSM data files, and represent all processes running on `Pascal` during the sampling period.

500 known normal system call traces were used as training data for the various algorithms under evaluation, and 4500 normal traces were used as validation data. The attack set used for this research's evaluation consisted of all attacks launched during the entire project, with the caveat that there must be a reasonable expectation of each attack having a footprint in the system call traces. 37 attacks met this criteria, belonging to the following 10 attack families:

- *format*
- *ffb*
- *ipsweep*
- *ftpwrite*
- *warez*
- *warezmaster*
- *warezclient*
- *spy*
- *loadmodule*
- *eject*

The structure of the selection of data from the KDD98 collection is important. First, the ratio of training data to validation data used for this experiment was 1 : 9. This provides a challenging task for the assessing IDS, as there is no guarantee that the training data is statistically representative of the entire scope of activities of the host `Pascal`. This is a deliberate injection of difficulty, as it readily equates with a realistic portrayal of a real-world situation. When an IDS is deployed to protect a real system, it must be able to be trained with whatever data is available. The system baseline will naturally change over time, and there is every likelihood that the training data initially provided to the IDS will quickly become stale and lose relevance to the current state of the host. As such, any IDS which requires an inordinate amount of long term system fidelity in the training data will not translate well to deployments containing any level of uncertainty. The training and validation data used for this research were selected from the fifth week of the KDD experiment, with training data extracted from the *Monday* subset and the rest of the fifth week providing the validation data. This presents a realistic but challenging data distribution, and is representative of forecast IDS training requirements.

The second important factor of the KDD data selection used for this research is the inclusion of all attacks with a footprint in the host's systems calls. Most research focuses on an unnaturally curtailed subset of the available attacks. By limiting the number of attacks that are passed to the IDS for evaluation, this approach will unavoidably skew the performance evaluation of the IDS. The research in this thesis includes all attacks which can be detected by a HIDS, hence greatly increasing the level of difficulty for the IDS but also the relevance of the resulting performance evaluation as it extends to real-world protection. IDSs used outside the laboratory are not required to only protect against a subset of attacks, and hence any experimental methodology which conveys an unfair advantage of this nature does not truly capture the performance characteristics of the test DE. This research provides the broadest possible range of attacks for evaluation, hence providing an accurate performance baseline subject to the constraints of the dataset composition itself.

A major limitation of the KDD98 dataset is that the attack generation was not approached as a standalone task, but merely as an enabling function of the data collection. Exploits were simply launched against vulnerable services, without regard for the likely methodology or process of an attacker. On one level, there is arguably little impact from this approach on the core aims of the KDD effort; the designers of these experiments sought to collect a large amount of data, with a good balance of attacks to normal activity, and were not overly concerned with the long-term survivability of the resulting dataset. As hacking has developed over the years to its current form, so too has the importance of realistic datasets upon which to base the evaluation of potential IDS algorithms. Given the distributed nature of modern hacking attacks, coupled with the incremental form of modern system compromises, the importance of an attack set which captures this process is critical. At the time in which it was compiled, the KDD approach was reasonably representative of the hacking of the era. Systems were generally porous, updates slow, and exploit code not readily available. Hence, hackers could reliably deploy exploit programs without modification with the reasonable expectation of a successful penetration. The KDD dataset controllers could thus merely activate similar exploit code and create a ready facsimile of a period hacker.

In contrast, modern systems are well patched, frequently updated and difficult to hack. Exploit code is readily available, but usually requires a high degree of modification to work on any but the older OSs. As such, modern hacking is a slow process requiring a high level of skill, focused on an incremental penetration rather than the single stage attacks of the KDD period. It is therefore clear that the attack methodology used by the KDD designers, whilst appropriate at the time, does not represent modern techniques and hence conclusions drawn from this dataset cannot be considered as a true representation of IDS performance.

The KDD98 dataset was engineered to represent a corporate intranet, with numerous hosts, network traffic, and a simulation of normal behaviour [20]. The experiment ran almost continuously for numerous weeks, and was designed to provide researchers with the broadest possible scope of data for evaluation. The UNM dataset [106], by contrast, is a collection of single processes of selected programs from predominantly UNIX-based OSs, and is focused on providing a dataset to accurately model the behaviour of single programs. This is an important distinction. The UNM approach is tailored towards providing publicly available data for these specific individual programs, and not towards a complete system level dataset. Research conducted on this data set is usually focused on modelling anomalous behaviour in single processes rather than on generic IDS development. The UNM dataset is of comparable vintage to the KDD collections, with similar criticisms, although the smaller scope of the UNM experiment has minimised much of the data artefact and labelling issues attributed to the KDD data sets. Importantly, the single process focus of the UNM collection does not represent modern hacking methodology, where malicious footprints are rarely concentrated in a single process and instead are distributed throughout numerous running programs. This point is discussed more fully in the following section, but is a critical limiter of the UNM datasets applicability to the modern environment.

One of the operating systems used to compile the UNM dataset was Sun OS 4.1.1, also known as *Solaris 1*. This OS is an earlier version of `Pascal's` operating system, which was used in the KDD98 experiment. This fortunate convergence means that portability testing of the strength of the new semantic feature was possible by leveraging the two disparate systems. As part of the results presented in Section 3.6, the DE trained using the KDD data was then used to classify the UNM data generated on the ancestor operating system. By way of baselining, other algorithms using traditional syntactic features were also measured under similar test conditions, with a clear and significant difference between the performance of the semantic feature under these conditions and the performance of merely syntactic features. All Solaris base traces in the UNM data were used for the portability resilience testing, which allowed for 618 known normal traces to be used for validation and 8 different attack traces to be used for calculation of detection rate. Whilst this does not comprise a large enough data sample to define performance in absolute terms, the relative difference between semantic and syntactic approaches is clear, with the semantic feature performing significantly better. This adds weight to the merit of the new method proposed by this PhD research, and suggests that training times for large-scale HIDS deployments can be reduced under a semantic regime by providing a semi-trained core DE image which can then be fine tuned on each individual host.

3.5.2 The ADFA Linux Dataset

Prompted by the many criticisms of the KDD datasets [116–121], discussed more fully in Chapter 2, this research developed a new publicly available dataset better representing contemporary attack methodologies [141] and operating systems [151]. Unlike the KDD attacks, the exploits used for the ADFA-LD represent a complete system compromise, from initial penetration through to privilege escalation. As such, they are a much better representation of actual hacking as it is conducted in the modern cyber environment, and present a more realistic benchmark for IDS performance evaluation.

A modern Linux distribution, *Ubuntu Linux 11.04* [151], was used as the target OS for the new dataset. This OS is freely available, and is widely used throughout the world. SSH, FTP and MySQL version 14.14 [152] were installed as running services on the target host, and the underlying operating system was fully patched. Web-based attack vectors were created by installing *Apache V2.2.17* [153] supported by *PHP V5.3.5* [154]. The web-based collaboration tool *TikiWiki V8.1* [155] was also installed to provide a realistic content server for the website. This version of *TikiWiki* was selected as it has a known vulnerability, which is catalogued in many on-line databases such as [156]. This vulnerability is small, but allows for a complex escalation process to be followed to full system compromise. The overall set-up of the target host represents a modern Linux server, providing remote access, file sharing, Web-based collaboration, web server functionality and database services. It is a well patched system, with several small residual vulnerabilities. The configuration is a good approximation of a real-world server, and presents a challenging hacking target which would require an intermediate to advanced level of skill to exploit.

The purpose of the ADFA-LD was to modernise IDS evaluation. As such, it was deemed critical that the attack methodology reflected current best practice. To provide this quality control, industry penetration testing methodologies such as that provided by [141] were used, and a new attack framework created. This attack framework, shown in Figure 3.3, links vulnerabilities to effects through the focus of the attack, accessibility of the exploit tool, attack vector and the deployability of the exploit. The new framework attempts to formalise the coverage provided by any new IDS dataset by allowing the designers to ensure appropriate representation across the spectrum of attack options. Importantly, a discrete attack is not required for each element, with functionality able to be combined to produce a key set of representative attacks which cover multiple aspects of the framework.

The *Focus* level of the framework relates to methods of distributing an attack package to a host with an identified vulnerability. Vulnerability assessment is the first stage of any attack, but once identified, a distribution focus must be selected. Client side attacks relate to connections originating on the target machine, such as

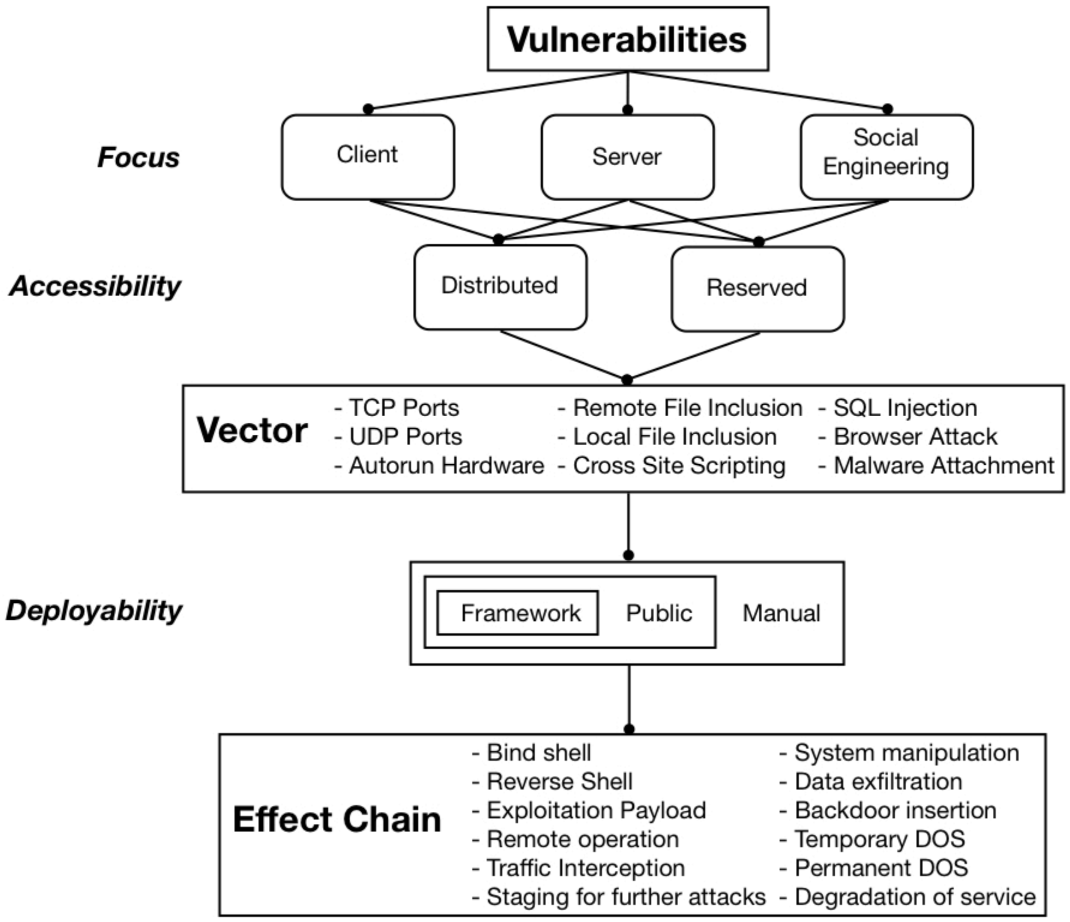

Figure 3.3: Attack Framework

from a poisoned executable or the upload of a Trojan program. Server side attacks describe the "classical" hacking method, where open services on the target are used to gain a foothold, and then a command shell. Social engineering attacks include all methods used to directly manipulate the user of the target machine to access malicious payloads and hence open an access point for an attacker. *Accessibility*, as defined for this framework, describes the availability of exploits for the identified vulnerability. Some exploits are publicly available, such as those catalogued at [157], but many are deliberately private. Availability of a public exploit is a strong indication that a vulnerability is significant, and hence this availability should naturally be considered quite important.

Numerous *Vectors* exist for transmitting an attack to a target, ranging from low-level TCP or UDP connections through to advanced cross site scripting and SQL injection. It is important that any attack dataset contains a broad range of attack vectors to ensure good coverage of the options available to a hacker.

The *Deployability* layer reflects the ease with with an exploit can be launched. The term *'framework'* refers to whether the exploit is included as part of centrally controlled hacking tool kits. Frameworks, such as the Metasploit Framework (MSF)

[1], provide an integrated package which makes coordinated attacks much easier to launch. If exploit source code is in the public domain but not integrated into a framework, it is labelled as a *'public'* exploit. Standalone single vulnerability exploits falling into this category represent a viable option for many attackers, particularly when the vulnerability being exploited is new, but do not allow a high level of integration and can reduce impact.

Privately available attack frameworks or private exploit source code exist in the *'manual'* sub-layer, often with bespoke exploit code which can allow the compromise of an otherwise impenetrable system. These suites are expensive and closely guarded trade secrets, however, and generally only used by corporate level organisations. It is pertinent to note that even a well known and publicly available exploit may effectively comprise a zero-day attack attack against a system which has no knowledge of this exploit; if signatures do not exist in the local IDS database then even a globally common attack may be a local zero-day attack from the perspective of the system under evaluation.

Finally, numerous *Effects* are possible and it is important that attack datasets contain a range of options for the final stage in the attack process. Each payload will have a different signature, and IDS must consequently demonstrate an ability to protect against a broad swathe of payloads if they are to be considered as effective general protection.

The ADFA-LD attacks cover a large section of this attack framework. All *Focuses* were covered, but reserved *Accessibility* and private *Deployability* exploits were not investigated due to availability constraints. A wide range of *Vectors* were used to distribute attacks, fully exploiting the various vulnerabilities offered by the target system. Numerous aspects of the *Effect Chain* were used, creating a large selection of host system manipulations[3]. When compared against the Attack Framework introduced here, the ADFA-LD represent a good coverage of the various options available to a hacker. By contrast, the KDD98 dataset has a very small cross-section of attack options, focusing strongly on server-side TCP attacks. Whilst this focus allowed for a greatly simplified data collection process, noting the bulk collection requirements of the KDD experimental set-up, it does not represent a viable attack methodology and has resulted in a dataset that does not achieve good balance across the spectrum of contemporary hacking options. As such, the ADFA-LD provides a much more accurate and representative baseline for evaluation of IDS systems.

Table 3.3 details the dataset structure, which was designed for HIDS evaluation. The ratio of training data to validation data is approximately 1 : 5, allowing for

3. The system manipulations referred to in this context are distinct to the term *system manipulation* as it relates to the new Virtual Kernel (VK) theory introduced in Chapter 4 and should not be confused.

Table 3.3: ADFA Linux Dataset Structure

Data Type	Trace Count
Normal Training Data	833 Traces
Normal Validation Data	4373 Traces
Attack Data	10 Attacks per Vector (See Table 3.4)

accurate evaluation of FAR. A wide range of attacks were used, and are detailed in Table 3.4. The *Hydra* attacks are remote brute force password guessing attempts, which are a common tool in a hacker's arsenal. This family of attack is regularly used [141], but leaves a large log footprint and is hence classified as noisy; it is important that any contemporary IDS is able to detect these attacks as they are often used by less skilled attackers, or if stealthier vectors prove unsuccessful. System call based approaches struggle to detect brute forcing, however, as the system is fundamentally performing an allowed task. This fact is partially responsible for the lower overall DR observed across all evaluating algorithms used on the ADFA-LD. The addition of new super users is a privilege escalation technique often used by hackers on Linux systems. Privilege escalation is a difficult process, and this common vector is frequently used, hence its inclusion in the attack set.

Table 3.4: ADFA Linux Dataset Attack Structure

Payload/Effect	Vector
Password brute force	FTP by Hydra [158]
Password brute force	SSH by Hydra [158]
Add new superuser	Client side poisoned executable
Java Based Meterpreter	Tiki Wiki vulnerability [156] exploit
Linux Meterpreter Payload	Client side poisoned executable
C100 Webshell	PHP Remote File Inclusion vulnerability

Meterpreter, used as the payload for the next attack types, is an enhanced functionality command shell. It is part of the MSF [1], and is widely used by hackers throughout the world. It greatly simplifies the compromise phase, and contains many built-in tools to facilitate various aspects of the hacking process. Meterpreter is available in several different formats, including *Java* and standalone executables, which have both been used for the ADFA-LD. Whilst the functionality presented to the hacker is largely the same irrespective of version, the interaction with the underlying OS is markedly different, hence warranting the inclusion of different Meterpreter versions in this test dataset. Finally, a `C100` web shell was included as the final payload, leveraging the remote file inclusion vulnerability of the *TikiWiki* installation [156]. Various methods were used to deliver all payloads, including simulated social engineering, client side attack, poisoned executable, Web-based attack and remote attack. This represents a broad spectrum of the various exploit vectors available to contemporary hackers, and presents a challenging range of hacking activity to the supervising IDS, rendering a more accurate performance appraisal as a result.

The ADFA-LD is significantly more complex than the KDD era datasets, and presents a much more significant challenge. The results contained in Section 3.6 clearly demonstrate the increased difficulty from a functional perspective, but a complexity analysis of the underlying data is warranted noting the novelty of this new benchmark. To provide this analysis, the methodology suggested by [70] was used to evaluate the underlying database structure. This method uses a "bag of system calls" approach to clustering, similar to that used in the "bag of words" text mining method. The approach suggested by the "bag of calls" method relies on frequency analysis alone, discarding positional information, and hence is arguably a much weaker technique. The implementation presented in [70] achieves good results against the UNM dataset [106], however, reporting 100% DR for 1.29% FAR. The UNM dataset is slightly more current than the KDD family but of more limited scope, and is generally regarded as being of the same era and suffering from the same currency issues. Figure 3.4 shows a plot of the decision engine feature extracted by this algorithm from both the KDD98 dataset and the ADFA-LD. The KDD graph clearly shows distinguishable features between the two different classes, with numerous different maxima and minima in the normal and attack data. The ADFA-LD plot, on the other hand, shows a much higher level of similarity, with attack and normal data almost identically distributed, at least in frequency. This figure conclusively shows the difference in complexity between the two datasets, and points strongly towards the high level of difficulty introduced by using the ADFA-LD as a benchmark standard.

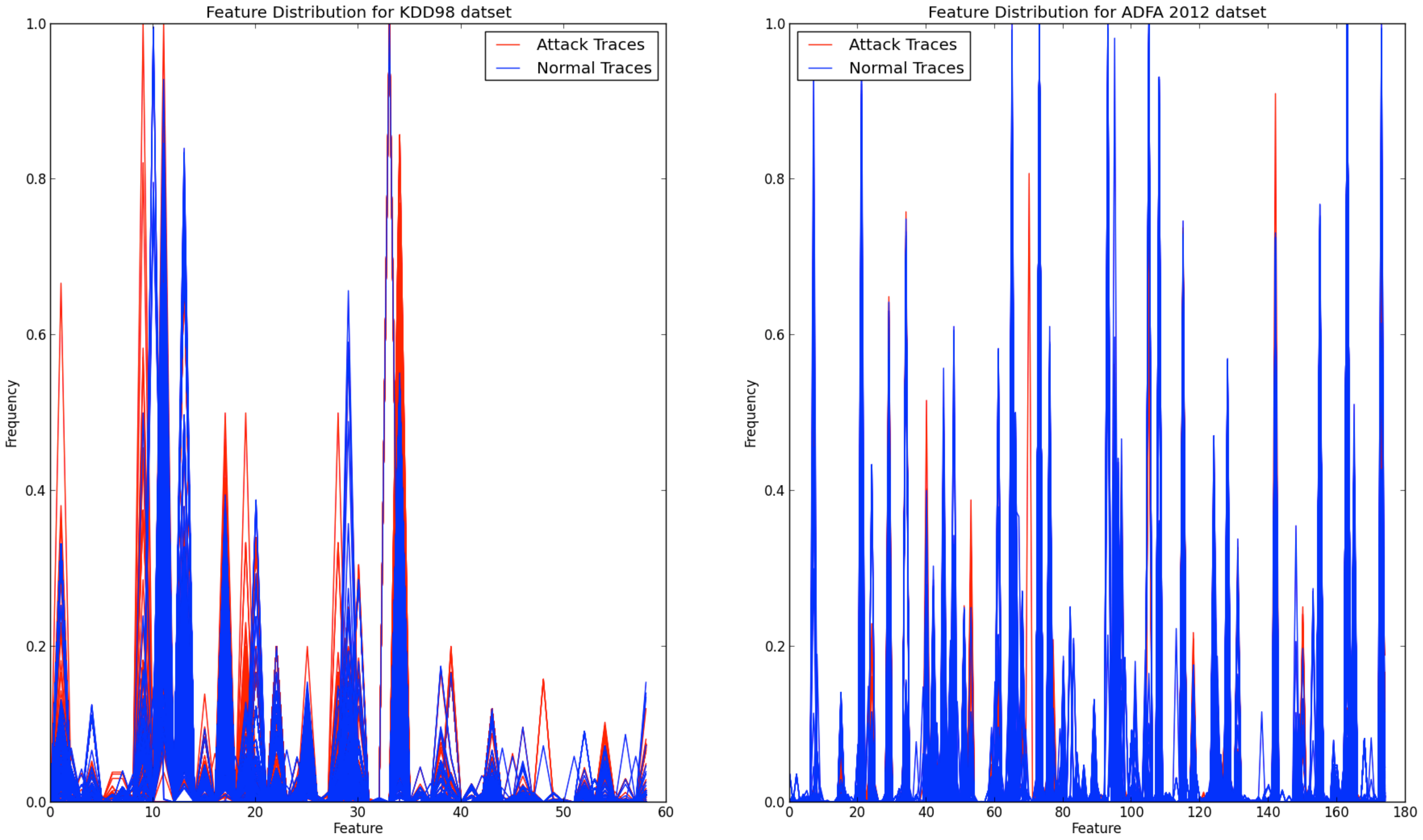

Figure 3.4: Complexity of ADFA-LD and KDD98 Dataset

3.5.3 Implementation and Experimental Methods

The first part of the experiment conducted for this chapter consisted of baselining the new semantic algorithm's performance against existing methodologies. This was done using the KDD98 dataset, and involved four separate trials. These trials enabled accurate benchmarking and consisted of the following tests:

1. The standard STIDE approach proposed by Forrest [23].
2. The use of contiguous syntactic units of varying window sizes as the input to an ELM.
3. The use of different length contiguous syntactic units as the input to a HMM[4].
4. The use of the new feature proposed in this paper as the input to an ELM, comprising a true semantic representation of the system call corpus.

Two key metrics were used for performance evaluation, namely DR and FAR, defined as follows:

$$\mathrm{DR} = \frac{\text{number of detected attacks}}{\text{number of attacks present}} \times 100\%$$

$$\mathrm{FAR} = \frac{\text{number of false alerts}}{\text{number of traces in validation data}} \times 100\%$$

Each trial was conducted under the same conditions, with 500 training elements and 4500 validation elements. Each training element represents a system call trace extracted from the host `Pascal`, and was supplied to the decision engine for each trial without filtering. The attack set for each trial was the same, and as discussed in Section 3.5.1, represented all attacks present in the dataset targeted at the individual hosts level. The nature of the KDD98 data and experimental conditions is such that it is impossible to identify exactly which process was attacked by each malware deployment. Hence, any labelling of any trace by the IDS in each trial was defined to be a detection. At times, multiple detections were registered for an attack cycle, perhaps indicating that multiple processes were affected. This effect was more pronounced when evaluating the ADFA-LD, as would be expected from the more distributed nature of modern hacking attacks.

Trial 1 consisted of the STIDE algorithm with varying window sizes. A full system call list was provided to the DE during training, as there is no other way of ensuring that the algorithm will have awareness of all possible system calls. STIDE-based IDS are particularly sensitive to any new system call not seen during training,

4. The variable structure of the semantic units is not well suited to training an HMM; as such, the HMM was used as a control to baseline syntactic performance using this powerful DE.

as this increases the mismatch ratio and consequently is usually registered as anomalous behaviour. Training data will not necessarily contain all possible system calls within a given architecture, and hence STIDE performance is affected significantly if this architecture specific list is not provided. Whilst this is a fairly significant disadvantage for real-world deployment and has the potential to drastically affect the training overhead and responsiveness of an IDS, the purpose of these experiments was to measure baseline effectiveness, and hence it is appropriate to provide this information. Of note, semantic features do not require extra information based on the inherent robustness of this new approach and consequently do not suffer from the same real-world limitations. ROC curves were generated from this trial by varying the detection threshold for the mismatch rate and plotting the resulting DE and FAR. As discussed in [30], the merit of an algorithm is directly proportional to the area under its ROC curve, and this approach forms a standard evaluation metric for IDS.

Trial 2 introduced the ELM operating on a syntactic feature. The phrase dictionary of length 1, compiled as part of the semantic feature processing, was used to provide a five element feature to the DE, which then performed a standard classification performance evaluation on the validation and attack data. The format for this feature is shown in Table 3.5, and represents an enhanced syntactic feature. In effect, this feature consists of the frequency counts for each word in the trace under consideration, and simultaneously considers multiple STIDE-like window lengths. This would allow greater accuracy than comparable low-level syntactic methods, as it is simultaneously considering a wide range of possible windows. Word lengths of between 3 and 7 were allowed for this experiment, which whilst arbitrary, nonetheless reflects the five elements used for each feature in Trial 4 and allows for some commonality between various configurations. The limits for word length were selected to avoid the trivial inclusion of words of length 1 and 2, with the upper limit 7 effectively preventing the inclusion of outliers. Due to the low count but high impact nature of outliers, they had the potential to drastically skew the results of this syntactic system, as it is not offset by the increased data content of a semantic feature. The ROC curve was generated by varying the decision threshold and recording the associated metrics.

Table 3.5: ELM Feature Format for Trial 2

	Word Length				
Sample Number	3	4	5	6	7
Raw value: $i_R \in \mathcal{I_R}$	Dict. 1 Count	Dict. 1 Count	Dict. 1 Count	Dict. 1 Count	Dict. 1 Count
Sample 1 (Raw)	10	10	10	9	7
Sample 2 (Raw)	9	6	6	5	4
Normalised value $i_N \in \mathcal{I_N}$	$i_R[1]/\max(\mathcal{I_R})$	$i_R[2]/\max(\mathcal{I_R})$	$i_R[3]/\max(\mathcal{I_R})$	$i_R[4]\max(\mathcal{I_R})$	$i_R[5]/\max(\mathcal{I_R})$
Sample 1 (Normalised)	0.1538	0.1538	0.1538	0.1384	0.1077
Sample 2 (Normalised)	0.1384	0.0923	0.0923	0.0769	0.0615

Trial 3 used HMMs to evaluate patterns in system calls. The following model parameters were used for this experiment:

- $N = 2$, with the state either being 'Normal' or 'Anomalous'.
- $M = \{\text{all allowable system calls}\}$.
- T as specified by the individual experiment iterations; equivalent to STIDE window length.

The general methodology and training schemes detailed in [72, 77, 78, 100] were used to create the final DE for this trial, with the Viterbi algorithm [74] used to evaluate the most likely state transition. A detection was recorded if the results of the Viterbi analysis indicated a probable state transition away from the normal baseline. DR and FAR were calculated from the resulting figures, and ROC curves generated from the performance of different window lengths. A complete list of architecture specific systems calls was provided to the HMM prior to evaluation and training, for similar reason as for Trial 1.

The final experiment, Trial 4, used the full semantic feature with phrase lengths of 1 to 5, with an ELM acting as the assessing DE. The phrase dictionaries compiled as part of the semantic extraction process were used to evaluate each input trace, with the resulting information condensed into a 5 element vector consisting of normalised phrase counts for each length. Different phrase lengths outside this range could theoretically be used, however the number of occurrences of each phrase at the higher lengths starts to diminish quite rapidly, suggesting that a maximum effectiveness will be reached for some phrase length, n. Empirically, $n \approx 5$ when processing KDD era traces. In a real-world deployment, where traces do not actually stop growing in size as new system calls are received almost continually when programs are running, high length phrases may be quite useful in providing an extra classification tool, but for the trace lengths under consideration for this experiment, a semantic length ceiling of 5 proved an effective compromise between data quality and statistically significant occurrence counts. Training data for the ELM was padded with a null count item, in order to scale the resulting output between 0 and 1. Decisions were made as to whether a trace was anomalous or normal based on the resulting ELM output compared to a tunable threshold, which was varied to create a performance range for plotting on the ROC curve.

Portability between operating systems is extremely desirable characteristic for HIDS. Training times are significant for this class of IDS, particularly when using an anomaly-based evaluation method. DEs are so sensitive to the changes in the system baseline that each HIDS must effectively be custom trained for each specific host. Any ability to perform centralised training, taking the bulk of the processing task away from the individual hosts, would be an extremely significant real-world advantage. This would also allow for centralised updates within a network, and a corresponding increase in accuracy once these updates are distributed to each node. The semantic methodology introduced by this thesis is sufficiently robust to allow centralised training to occur. This robustness and portability was demonstrated empirically as the final section of the KDD experiments. To explore this feature of semantic IDS theory, various IDS algorithms were trained using the same 500 elements as for Trials 1 to 4, but were then tasked with evaluating information extracted from the UNM dataset, as discussed in Section 3.5.1. As expected, performance was extremely low for traditional algorithms, but semantic-based systems performed quite well, with a marked difference between their classification and syntactic based decisions. Whilst the number of available traces from the UNM dataset which were suitable for inclusion for this experiment were too low to provide generalisable conclusions to be drawn, the comparative advantage of a semantic feature is readily apparent, strongly indicating the portability of this new approach.

3.6 Results and Discussion

The results presented in this Section clearly indicate the superiority of the new semantic feature. Three different ROC curves were generated, quantifying the performance of numerous different algorithms applied to different datasets under a broad set of test conditions. In each graph, the performance of the semantic feature is clearly superior to that of traditional algorithms, often by a large margin. Throughout all different test cycles, the combination of the ELM DE and semantic feature was unsurpassed. Whilst the semantic feature proposed by this research can be applied to any DE, albeit with some modification, the ELM's ability to consider the information from multiple semantic phrase dictionaries concurrently creates a performance edge which is clearly visible in Figures 3.5, 3.6 and 3.7.

The first test cycle involved the evaluation of IDS DEs trained on the KDD98 data and then used to evaluate previously unseen data from this same dataset. This represents a control baseline for the new semantic algorithm's performance, as it duplicates the test conditions commonly used across the IDS research sphere. Using the definitions and experimental structure defined in Section 3.5, the relative merits of each algorithm was evaluated, collated and presented in Figure 3.5. Several key observations can be extracted from this ROC curve. First, the STIDE algorithm performs poorly under these test conditions. This is because the STIDE method is designed to evaluate a single process, having been trained on that process exclusively. The experimental design for this range of tests required all IDS algorithms to evaluate from a system-wide perspective. As a result, the STIDE algorithm was unable to cope with the complexity of the task with a correspondingly poor overall performance.

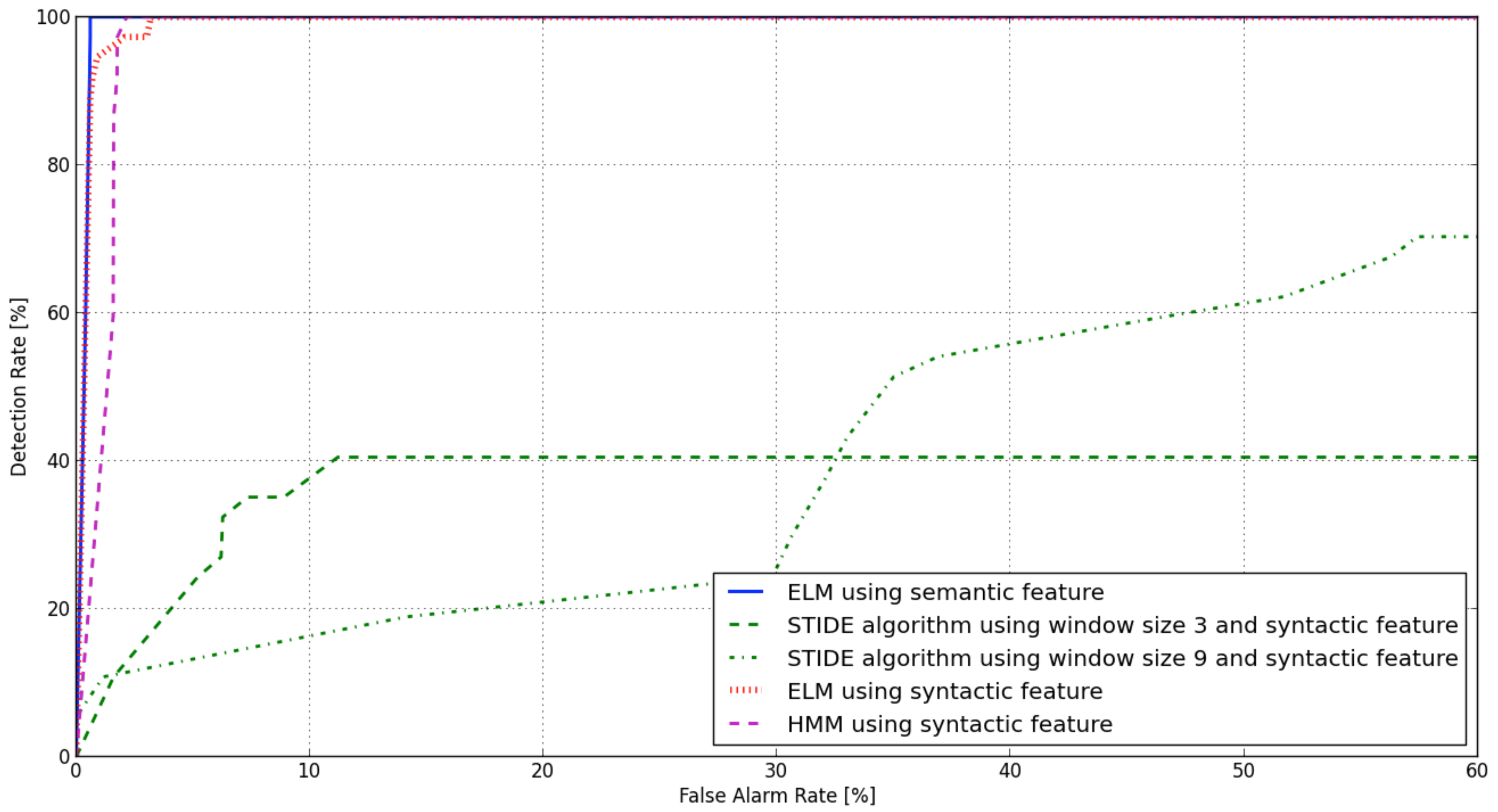

Figure 3.5: ROC curve for various algorithms applied to the KDD98 Dataset

In part, this sub-optimal performance reflects a characteristic of hacking attacks of this era; at this stage of cyber evolution, system penetration was achieved with a large footprint in a single process, which affects the ability of a STIDE-based IDS to deal with large amounts of 'noise' in the system activity. In other words, an STIDE IDS does not handle the large degree of inter-process variation well, and consequently is beset by a high FAR for only a mediocre DR. An increase in the number of training data elements would assist with improving performance for this algorithm, but would consequently violate the experimental conditions. By contrast, STIDE performed much better on the ADFA-LD due to the distributed nature of attacks in this dataset, discussed further below.

The HMM results shown in Figure 3.5 demonstrate the strong performance which has come to be associated with this family of IDS DE [56–59, 72, 77, 78, 100]. Even though using a syntactic feature, the excellent sequential analysis capability of this algorithm performs admirably for datasets of similar vintage to the KDD collection, with performance almost as good as that observed for the full semantic feature. Importantly, the HMM performance is better than that of a syntactic feature driven ELM. This strongly indicates that the superior results shown by the new semantic method when a semantic ELM is used to classify the various datasets is due to the new feature alone, rather than the decision engine or other ancillary IDS component as only the feature has changed. The comparison between a pre- and post-semantic performance for the ELM, benchmarked against the HMM results, is critical in establishing the position of the new method within the hierarchy of algorithmic effectiveness, and conclusively demonstrates the performance gains which are possible when a full semantic feature is used. The accuracy of the fully semantic ELM IDS is clearly superior to all other methods, achieving saturated performance at 100% DR for only 0.6% FAR, an almost perfect classification record. These results, made possible by the new research conducted as part of this PhD thesis, clearly demonstrate the merit of a semantic approach when applied to the KDD98 dataset.

Figure 3.6 shows the ROC curves for the second set of experiments. These trials used IDSs which were trained on KDD98 data, and then used to evaluate UNM data, as discussed in detail in Section 3.5.1. Importantly, whilst the family of operating system was the same for both KDD98 and UNM, the OS version differed significantly. This creates an extremely challenging task for the evaluating DEs, as the change in underlying structure immediately injects a large amount of noise into their classification process, greatly increasing the difficulty of their task. The only way for an accurate classification to be returned to the user under these test conditions is if the underlying data used to form the decision engine inputs is extremely robust, with an accompanying high degree of dimensionality. The purpose of running these tests was to investigate the suitability of a semantic feature as the basis for a centralised HIDS training scheme. As introduced in the preceding sections of this chapter, HIDS training is an intensive and lengthy process. Any steps towards centralised training for large networks, with a corresponding decrease in total training time, would be a significant advantage. Under this scheme, the majority of training would be conducted off-line and then distributed to the endpoint hosts, where fine tuning would occur.

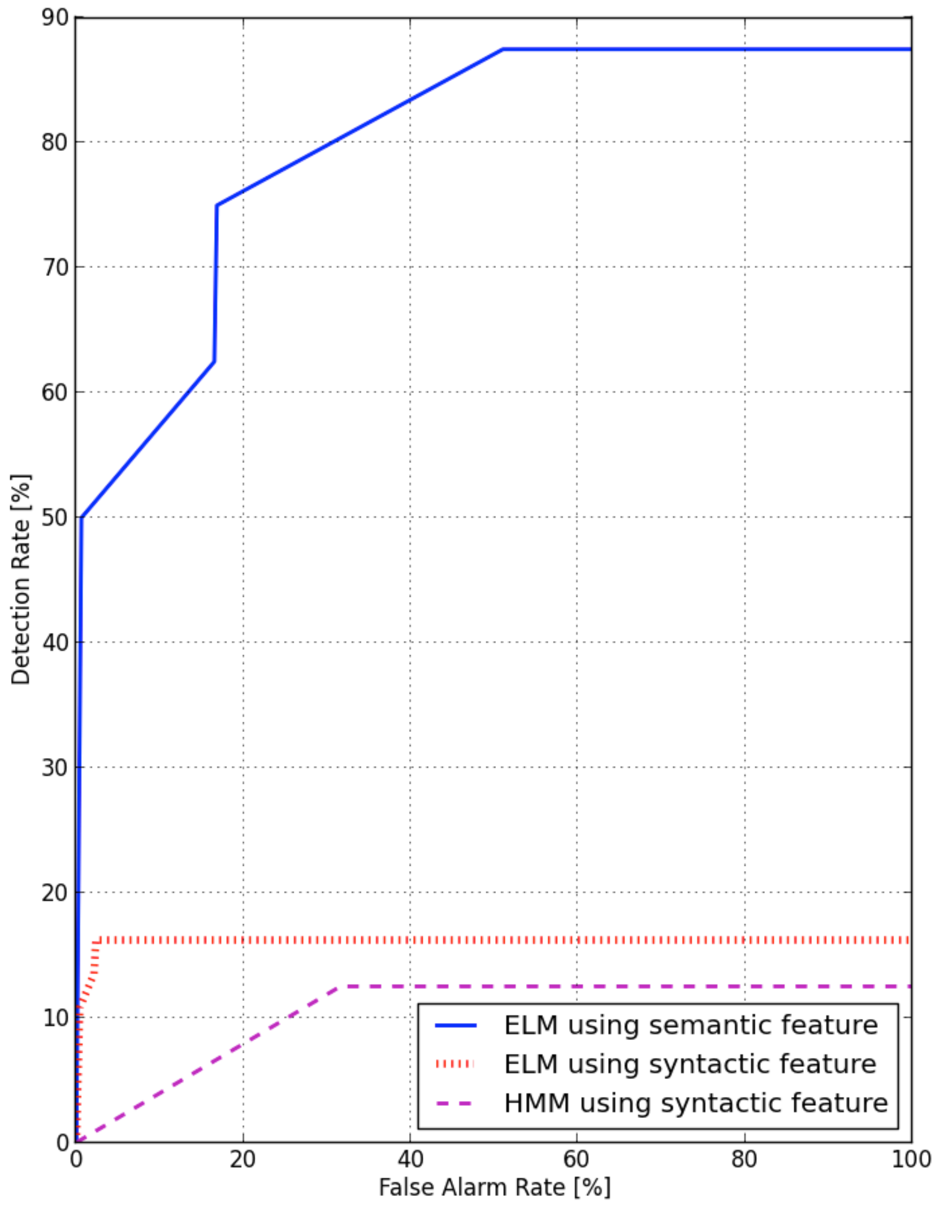

Figure 3.6: ROC curve for various algorithms trained on the KDD98 Dataset but used to classify the UNM dataset

The results obtained from this trial clearly show that a semantic approach withstands the portability testing much better then any other approach, and hence gives every indication of the accuracy required to implement a distributed training system. Both a syntactic ELM and HMM were unable to achieve meaningful classification, and the STIDE implementations in this experiment were unable to produce usable results at all. Whilst the semantic ELM does not achieve the same excellent performance under these conditions as in the KDD experimental set, the comparative merits of the semantic approach are readily apparent. Given the forecast requirement for extremely rich data to drive the assessing DE in this environment, increasing the semantic phrase lengths may improve the classification results. In some respects, an increase in the non-terminating semantic unit size may affect activity assessment by individual hosts, due to low frequency occurrence counts adversely impacting on short system call traces, but the richness of the feature will also be inherently increased. Better performance is hypothesised where phrase length is increased and the length of each system call trace is also increased to ensure a reasonable occurrence of high order phrases.

The processing overhead associated with longer semantic phrases is significant, and potentially prohibitive when considered for a single host in isolation. If a distributed training regime is used, however, then the impact of the the increased processing burden is much less, as it can be performed off-line with centralised resources. Indeed, the increased cost of longer semantic units when processed centrally may well be less than the cost of training each HIDS individually, even with shorter semantic units. The indication from this experiment is that there is a clear difference between the portability of semantic features and syntactic features. This empirical result corresponds to the forecast behaviour of the system, as the increased positional information included with a semantic frequency analysis must consequently result in greater decision accuracy.

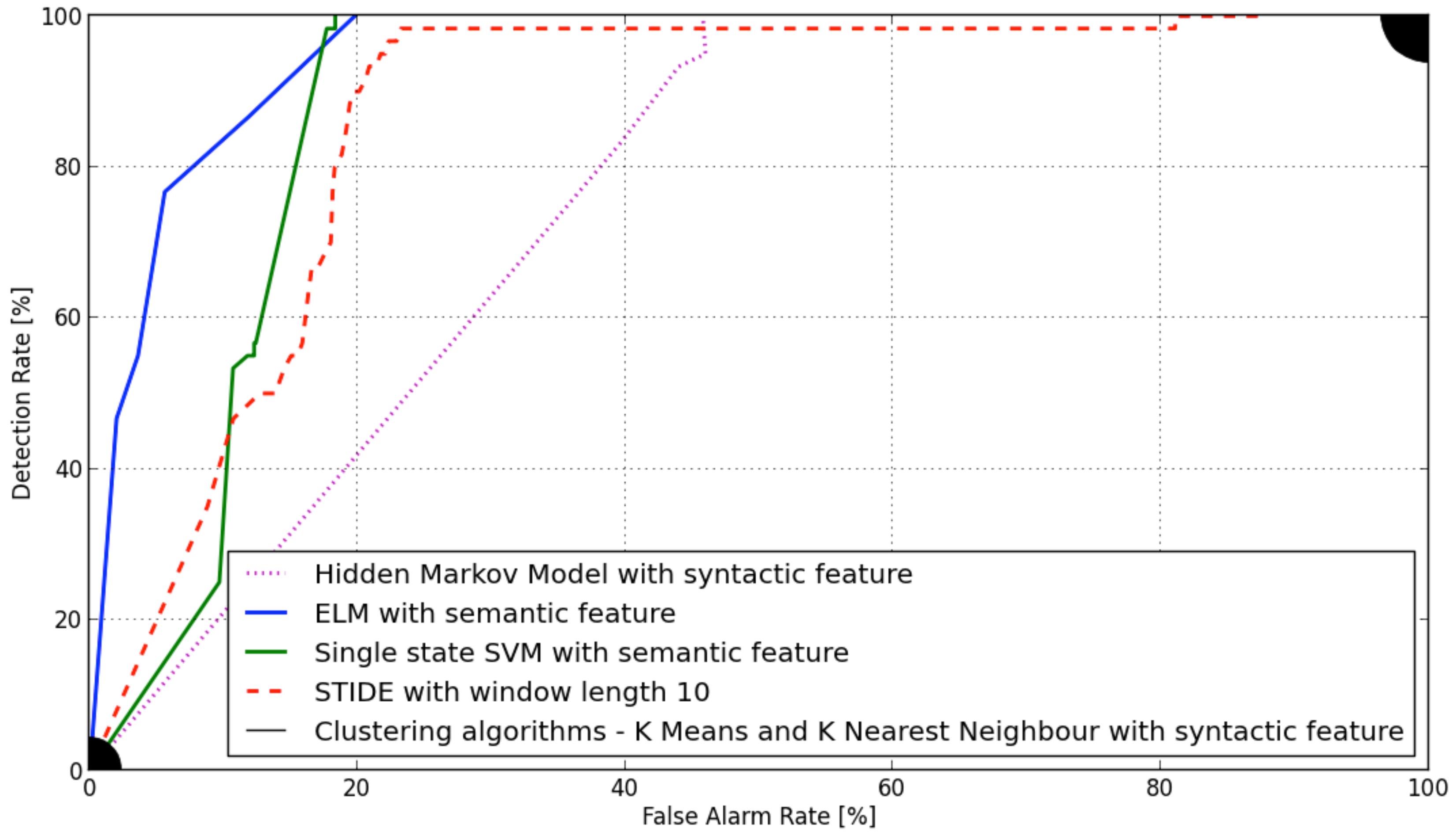

Figure 3.7: ROC curve for various algorithms applied to the ADFA-LD

The final set of results produced as part of this Chapter used the ADFA-LD to provide both training data and evaluation data. This modern dataset, representing contemporary hacking methodologies and a current OS, presented a much more significant challenge to IDS algorithms using both semantic and syntactic features. Two semantic algorithms were implemented, namely the single state SVM and the ELM. Both semantic algorithms performed significantly better than the syntactic methods, with the ELM again providing superior performance. All IDS methods failed to achieve the impressive results demonstrated on the KDD98 dataset, strongly indicating the difficulty of the ADFA-LD. A contributing factor to the generally worse performance of every algorithm is the challenging nature of detecting brute force password guessing from system calls, which may account for lower detection rates. The higher filter level applied to the semantic algorithms may also have played a part in reducing the performance of the semantic-based HIDSs. Regardless, the clear difficulty of the ADFA-LD, coupled with its currency, suggests that the relevance of the comparatively easy KDD98 dataset should continue to be questioned. The complexity of the ADFA-LD, illustrated in Figure 3.4, is such that the performance reduction is expected, but it is clear that even strong algorithms are significantly less effective in this setting.

Interestingly, the STIDE algorithm performed significantly better against the ADFA-LD than against KDD98. This likely relates to a fundamental change in the way hacking attacks compromise a vulnerable system. As introduced in preceding sections, legacy attacks such as used in KDD98 have a high footprint in a single process. Modern attacks, on the other hand, are more distributed in nature, and spread their system manipulation across many system processes. This means that process oriented IDS, such as the STIDE family, have a greater chance of detecting some manipulation in the system calls. Whilst the attack footprint is smaller overall, there is a greater statistical chance that a single process has been changed sufficiently for the STIDE algorithm to detect it and consequently raise the alarm. By contrast, if the STIDE IDS does not correctly classify the single polluted process of a legacy attack, then detection rate is significantly affected. The performance of the STIDE algorithm in this setting lends weight to the relevance of a semantic approach, as the increased accuracy strongly indicates that positional information is critical for the correct classification of modern attacks.

The HMM IDS perform much worse relatively than in other experiments. This is likely because the positive impact of the distributed attack footprint on the STIDE algorithm has the opposite effect on an HMM based evaluation. The HMM approach centres on a whole of system IDS, distinct to the single process focus of the STIDE method. As such, spreading the attack footprint across a large number of processes has the effect of burying the activity in the background system noise.

Semantic-based algorithms benefit from the increased information content and data dimensionality discussed throughout this chapter, and hence perform better in this environment, despite being affected by the same background noise effect. Further accuracy gains may be possible by focusing on a single process semantic evaluation, combining the scope of the STIDE algorithm with the superiority of the semantic feature.

Finally, evaluation of the ADFA-LD dataset was attempted using two clustering algorithms. These algorithms, the K Means and K Nearest Neighbour methods, had been applied to traditional IDS datasets with a high level of success. Their performance on the ADFA-LD, however, was discontinuous and ultimately unusable. Detection rate and false alarm rate remained stable around 3% each, before jumping to 98% each, producing the discontinuous ROC curve shown in Figure 3.7. Despite tuning each algorithm using the full range of options inherent in their underlying methodology, no intermediate points were obtainable. Both algorithms exhibited the same behaviour, indicating that this performance is not related to implementation, but rather is derived from the underlying nature of the clustering approach. This statement is further supported by rigorous testing of each clustering implementation, with results duplicated by publicly available *Python* [150, 159] and *Matlab* [160] toolboxes, as well as purpose written code.

The reason for this extremely poor performance lies in the fundamental nature of clustering, which in an IDS setting dispenses with positional information by using a ‘bag of system calls’ approach [70], similar to that used in text mining. This approach can be used effectively on legacy datasets, which have a low degree of complexity, but fails badly when applied to modern scenarios. The ramification of this discovery is that positional information must be included in any system call based IDS which aims to be effective in the modern environment, as it is clear that frequency-based analysis features contains insufficient information to correctly classify these contemporary attacks. This observed characteristic of frequency centric analysis methods coupled with the sub-optimal performance of even a full semantic IDS against the ADFA-LD strongly indicates that further gains in accuracy will only be made by the provision of a rich feature to the underlying decision engine methodology, and argues strongly for the relevance of the semantic feature proposed by this research in combating modern hacking.

3.7 Chapter Conclusion

This chapter has introduced a new semantic theory proposed as the first key innovation of this thesis. Semantic theory, as applied to intrusion detection systems, defines system calls as terminating units of the underlying language, or rule set, which governs interactions between programs and the CPU. Sequences of system calls, such as those first proposed by Forrest [23], hence become a syntactic analytic method in this new framework. Forrest's approach uses contiguous groups of system calls, produced by the syntax of the language, as the basis for the decision engine feature. This research introduces a semantic method, where discontiguous syntactic units are considered. The ramifications of this theory are significant; the fully semantic approach outlined in this chapter means that any valid syntactic unit, irrespective of the originating trace, can be combined with any other valid syntactic unit, potentially from a different trace from a different process, and used to extract information about the nature of a third trace from yet another process.

The result of this new approach is a significantly richer feature for evaluation by the IDS DE, with a greatly increased level of dimensionality due to the enhanced positional information contained therein. A comparison of frequency-based methods and positional-based methods demonstrates a significant performance increase when using the latter, as this method provides a greater level of information richness. All previous methods, however, have only used a syntactic level approach, and the new semantic method of this research demonstrates significant superiority to the existing positional-based methods as a consequence. The heart of the increased accuracy lies in the ability of the semantic algorithm to combine information from different system call traces; a discontiguous analysis of valid syntactic units means that a greater proportion of the information contained in the training data can be applied each individual classification problem faced by the IDS after training. By providing a greater quantity of higher quality data, accuracy is naturally improved, as borne out by the exceptional performance of the new semantic-based algorithms when compared with traditional approaches.

Extracting semantic features from raw data requires a somewhat involved process. First, raw system call traces must be captured from the target operating system. To provide a complete baseline of system, all programs must be captured in real time, which quickly creates a large amount of raw data. Each trace within the estate consists of the system calls used by a single program, ordered sequentially. Once the sampling process is complete, the resulting raw data provides information about each individual program, and about the system as a whole. To create a semantic evaluation system, this raw data is then processed using a double pass algorithm. During the first pass, a word dictionary is created which consists of all contiguous patterns of system calls up to the length of each trace. As each system call trace is processed, the resulting words are added to a dictionary construct, and the current account for each word incremented if it is already present. In other words, the raw training data is processed to extract all contiguous syntactic units, which are catalogued with the count of each word's occurrence in the dataset. A filter is then applied to remove low count words, serving the dual purpose of reducing the processing burden and eliminating outliers which would skew the classification process unnecessarily.

Once the word dictionary has been compiled, it is then used to reprocess the training data in order to extract semantic phrases. Each phrase is comprised of a number of words from the syntactic dictionary, and has an arbitrary length. The experiments in this chapter used phrase lengths of 1 to 5, but the upper limit could well be increased if sufficient training data were available. Traces of shorter length will naturally have less phrases of larger lengths, and if the phrase length is unduly increased then this may result in an artificial lowering of performance due to the natural absence of long phrases in short traces. As the phrase dictionaries for each length are compiled, a count is kept of how many occurrences of each particular phrase are seen. This count is used as a weighting factor; if phrases are encountered many times in the training data, then the subsequent presence of this phrase in a new trace is likely to represent normal behaviour and hence should be assigned more weight than a phrase seen but rarely in the training data. After the compilation of the phrase dictionaries, traces are evaluated for the final time and an n-element feature is constructed, where n is the number of different phrase dictionaries used, and each element consists of the occurrence count of words from a dictionary in the subject trace, weighted by the factor representing the merit of that phrase. These element features are then passed to the decision engine for training, and eventual evaluation of new data.

The semantic feature generated using this method can then be applied to any decision engine. Numerous different types of DE were trialled in these experiments, ranging from artificial neural networks to the simplistic but effective STIDE algorithm. Best performance was demonstrated by the ELM, largely due to its ability to simultaneously consider multiple phrase dictionaries and hence reap the benefit of each individual data source concurrently. Whilst in some respects the implementation of the new semantic theory for the ELM moves towards a frequency-based approach, the data richness and structure of the semantic features means that positional information is inherently included in any resultant analysis. Further gains may be possible, however, by implementing a fully positional decision engine, rather than the quasi frequency-based approach used for the ANN in this chapter. Importantly, performance of the semantic feature was better than a syntactic approach regardless of DE, strongly indicating the merit of this new feature and the increased performance possible when it is used as the basis for intrusion detection classification.

Testing and validation of the new theory were conducted in three different steps. First, the KDD98 dataset was used to baseline performance of the semantic approach against existing syntactic methods. The new theory proposed by this thesis perform significantly better than any existing algorithm, achieving 100% DR for only 0.6% FAR. This exceptional performance strongly indicates that the algorithm has significant merit, and is a major contender in the cyber security field. The second test of effectiveness was conducted using the UNM dataset. This series of experiments tested the portability of various decision engine algorithms, first training the IDS on KDD98 data and then using it to evaluate UNM data. This experiment was designed to profile the fragility and sensitivity of each method, with a view towards creating a centralised training regime for HIDS using the same operating system but with different versions or applications installed. Any system that allowed centralised training, with fine tuning on the end-user machine, would allow significant efficiency gains to be made at the same time as an improvement in accuracy. Additionally, the robustness demonstrated by this experiment provides indications of the long term survivability of an IDS, as a more robust baseline will extend the time between retraining and render a marked efficiency dividend. Again, the semantic feature performed significantly better than existing algorithms, indicating distinct possibility for centralised training using this approach.

Finally, the various algorithms were tested on a new dataset, the ADFA-LD. This dataset was specifically developed to provide a modern evaluation benchmark for IDSs, and was created as part of this PhD research. The ADFA-LD used a modern Linux operating system in a well patched configurations with a few small vulnerabilities to permit incremental exploitation. It was attacked using commonly available tools [1] and methodologies currently employed by industry level penetration testers [46,142]. The resulting dataset is much harder than the KDD98 era collection, and is a much better indication of real-world performance. Whilst accuracy against this new dataset was much lower for all algorithms, the new semantic-based IDS performed much better than the syntactic options. This lends further weight of evidence to the merit of the new semantic approach, and also suggests the publicised performance of various algorithms based on legacy data sets is not a true indication of IDS accuracy.

The semantic theory introduced and empirically proven in this chapter forms a basis for a new class of IDS. For the first time, the robustness of the underlying semantic feature means that a system call-based analysis of Windows operating systems is possible. This research avenue will be explored further in the next chapter.

Chapter 4

The Virtual Kernel Concept - Mapping DLL Calls to Provide a Universal Data Source for Intrusion Detection Systems

The Windows Operating System (OS) is the most popular desktop OS in the world and enjoys the majority market share for both servers and personal computing needs [124]. This popularity has made it a sensible choice for hackers conducting zero-day attack research as well, however, and has resulted in a large number of critical security vulnerabilities being discovered, exploited, and eventually patched. Further complicating the issue is the difficulty of protecting the Windows architecture from many of these attacks with the prevalence of signature-based anti-virus (AV) systems strongly underscoring the significant challenges of implementing anomaly centric protection on this OS. The absence of effective anomaly-based Host Based Intrusion Detection System (HIDS) from the available security controls for Windows has likely been a significant facilitator for zero-day attack exploitation, and consequently a successful implementation of such an Intrusion Detection System (IDS) is likely to have a strong positive effect on the security resilience of the OS.[1]

The major factor which has steered Windows security protection towards AV systems as opposed to the HIDS methodology typically used in the Linux OS is the difficulty of accessing high-quality IDS data sources in Windows. As discussed at length in Chapters 2 and 3, system calls are widely acknowledged as the best data source available for HIDS. The linear interaction between programs and the kernel in the Linux OS means that this data source is highly representative of system activity and consequently allows accurate baselining of normal system activity as a result. The Windows OS, on the other hand, makes extensive use of the Dynamic Linked Library (DLL) concept which greatly complicates the interaction with the kernel whilst admittedly simplifying other aspects of the software engineering cycle. In effect, the DLL centric architecture of Windows means that system calls are often accessible through multiple paths with the result that decisions on normalcy cannot be made based on the system call patterns alone. As a result, Windows

1. The core theories outlined in this chapter have submitted for peer review in:
G. Creech and J. Hu. A New Virtual Kernel Theory Allowing the Application of System Call Intrusion Detection Analysis to the Windows OS. Manuscript submitted to *IEEE Transactions on Computers* for peer review.

IDS have typically been forced to use inferior data sources such as log files [24] and registry access requests [26–28] in an attempt to provide some level of anomaly-based protection. These secondary data sources are unable to provide the granularity of system calls, resulting in mediocre HIDS performance and naturally shifting the focus of Windows security vendors to AV systems [128–131].

Modern AV products are extremely effective at recognising previously seen malware, and provide a useful signature-based threat reduction strategy. Notwithstanding this, they inherently suffer from the inability of signature-based systems to detect zero-day attacks and are reliant on regular updates from the AV vendor. This inescapably limits the scope of their protection and makes their performance a function of the intelligence gathering apparatus which fuels the provision of updates to their database. Furthermore, AV systems tend to examine programs in storage rather than when loaded into memory, although high-end AV products are beginning to include heuristic checking of run-time behaviour [161]. In contrast, HIDS focus exclusively on the behaviour of the running system; both focuses are necessary to provide overlapping and comprehensive detection, and hence the absence of an HIDS has a stark impact on the porousness of the OS as a whole.

The problem of anomaly-based detection and the accompanying zero-day protection capability in the Windows OS could be solved if there was a way of extracting a similar level of information content from Windows system calls as is available in Linux system calls. If this was successfully achieved then any Linux-derived HIDS methodology should be portable to the Windows platform without further modification, bringing with it a previously unavailable protection mechanism. This is an attractive solution to the problem as it does not require adjustment of the IDS Decision Engine (DE) or data sensor which are already optimised as part of the development process for each Linux implementation. The research outlined in this chapter provides a solution to this problem and answers the relevant research questions posed in Chapter 1, Section 1.3. The proposed method permits the non-linear DLL access requests inherent to Windows to be mapped in a linear fashion consequently allowing standard HIDS algorithms to be applied to the problem.

The absence of a significant volume of Windows-focused HIDS research means that datasets for testing the new theory proposed by this PhD were not available within the existing body of knowledge. To remedy this lack, a new dataset was created reflecting modern threats and attack vectors present in the Windows family. The Australian Defence Force Academy Windows Dataset (ADFA-WD) has been made publicly available and provides a high-quality collection of DLL access requests and system calls for a variety of hacking attacks. It was used to verify the theory proposed by this chapter and represents a comparable challenge and difficulty level to that presented by the Australian Defence Force Academy Linux Dataset (ADFA-LD) detailed in Chapter 3. The ADFA-WD is an important contribution to the field as it allows the full range of existing and experimental Windows protection mechanisms to be tested against current hacking attacks for the first time. To verify the performance of the new Virtual Kernel (VK) theory outlined in Section 4.1.1, three Linux-derived HIDS employing Hidden Markov Model (HMM), Extreme Learning Machine (ELM) and Support Vector Machine (SVM) decision methodologies were used to evaluate this new dataset. As shown in Section 4.3 and as summarised in Table 4.1[2], all HIDS performed well with the new semantic method introduced in Chapter 3 achieving 99.58% Detection Rate (DR) for only 1.78% False Alarm Rate (FAR) when used to drive a SVM HIDS.

Table 4.1: Summary of Results for the Windows OS

ADFA-WD Dataset	
HIDS	AUC
Semantic SVM	99.64%
Semantic ELM	98.57%
Syntactic HMM	74.36%

The research presented in [65] uses an embryonic method similar in some respects to the VK theory proposed in this thesis. Under their approach, function calls within the Windows Native API were used to provide a limited scope of detection. There are several important distinctions between this approach and VK theory, however, despite some superficial similarity. First, considering the Native API alone excludes many critical DLLs from the classification problem. In particular, networking does not form part of the Native API; as such, any manipulation of ports which is a fundamental part of many hacking attacks is overlooked under this scheme. Indeed, the decision to use only the Native API is contrary to all of the principles of DLL selection identified by this thesis[3], and clearly differentiates the limited theory of [65] from the generalised VK theory presented here. Second, by referencing function names as opposed to memory access locations [65] consequently creates a significant security hole. If memory is called or accessed directly, function names do not feature at the sampling point used by [65]. This presents a simple bypass of any consequent HIDS, and significantly negates the efficacy of this approach. Finally, results in [65] were inconclusive. A theoretical proof of concept was conducted, but a proper dataset was not used to evaluate the algorithm and the results hence lack applicability to the general Windows problem. Notwithstanding the flaws in the approach outlined in [65], the research does present one of the few

2. This table provides comparative evaluation based on the Area Under the Curve (AUC) of the Receiver Operating Characteristics (ROC) plot as has been discussed previously in Chapter 3.
3. Discussed in Section 4.1.2.

attempts to apply a system call like analysis to the Windows problem. Whilst the implemented solution is suboptimal, [65] presents the first investigation into DLL access requests and is worthy of mention as a result.

The rest of this Chapter is organised as follows: Section 4.1 explores the difficulties of system call based analysis in Windows more fully, with the new VK theory introduced in Section 4.1.1 and DLL selection considerations outlined in 4.1.2. The ADFA-WD is discussed in Section 4.2, with the design rationale, structure and data formatting discussed in Sections 4.2.1, 4.2.2 and 4.2.3 respectively. Finally, results are presented in Section 4.3 with conclusions detailed in Section 4.4.

4.1 Using System Call based Analysis in Windows - The Migration from Kernel to Dynamic Linked Library

System calls are an important part of the modern computing environment and provide core functionality to the user. Almost every OS in use today makes a distinction between user space and kernel space. Kernel space encompasses tasks and activities which are performed with elevated privileges by the OS kernel itself, whilst user space tasks relate to ancillary functions performed by user initiated programs and processes. The kernel has unrestricted access to the lowest levels of the host, including hardware components. Granting unrestricted access to this level of the architecture to any program is fraught with danger, raising issues of non-standard access protocols, competing priorities, race conditions and data corruption, not to mention the increased security risk of having all programs operating with a high level of privilege. System calls provide a bridge between the two domains, allowing user space programs to interact with the kernel in a prescribed and codified manner in order to permit low-level but crucial tasks to be performed in a timely and efficient fashion without unduly impacting either system integrity or security. Examples of tasks usually conducted by way of system calls include file read and write operations, networking tasks and interfacing with hardware peripherals.

System calls are kernel specific, although architectures are often determined early in the life-cycle of an OS with system call specifications generally persisting for many versions prior to replacement or updating. This allows for a streamlined software development process and provides some measure of certainty about the digital environment in which third-party programs will operate. The format of a system call usually involves loading parameters into CPU registers or pushing them onto the stack, loading the system call kernel identifier into a predefined register and then calling the kernel interrupt function at which stage the system call is actioned by the CPU [134]. The effect of this protocol is to allow unprivileged programs access to important system resources through a standard interface, with the kernel providing the actual manipulation rather than the calling process itself and hence mitigating many of the potential issues outlined above.

The use of system calls and the accompanying interaction with the kernel are generally transparent to both users and programmers alike. Compiler programs are effective at producing optimised code and seamlessly implement high-level functions by using both assembly language routines and system calls. As such, is unusual for CPU registers and the stack to be directly manipulated during the system call process by the user or code author, although languages such as C provide the option for programmers to construct their own system call subroutines if they should so desire. Whilst not directly accessed by the vast majority of computer users, system calls occur frequently as they are required to perform almost every low-level enabling task upon which higher order functionality is built. This means that each discrete program function has a quantifiable system call signature which in turn accounts for the high granularity HIDS produced as a result of using this data source for HIDS [23, 39–42].

In a Linux OS, system calls form a linear data stream and can be effectively processed without expanding the dimensionality of the analysis. As each program in this OS family runs, the compiled version of the source code uses system calls as required to implement the desired programming objectives. The key point here is that each program makes it own system calls; as the particular program receives its time-share of CPU cycles, it manipulates the registers and stack as required and triggers the kernel interrupts to initiate the appropriate system call. It is rare for third-party programs to intercede in this stream, which explains much of the linearity evident in Linux system call traces. In other words, system call activity in Linux is directly attributable to the calling program. This means that system calls, as proposed by Forrest [23], provide an excellent measure of Linux system activity which is directly and irreversibly linked to the originator of the activity.

Windows, on the other hand, has a very different structure to Linux which greatly complicates system call interactions, historically precluding their use as a data source in this OS. This different structure arises because Windows system architecture makes extensive use of DLLs. Each DLL provides a set of standard functions which can be called by any program as required, so long as the relevant DLL is loaded in memory. The DLL collection can be extended dynamically by third-party developers, although core functionality is provided by Windows sponsored DLLs and the .NET framework [125–127]. By adopting the DLL architecture concept, Microsoft has attempted to create a developer friendly environment where the core OS is abstracted one layer from non-OS programs. Because of this structure, the Windows kernel can be modified at will by Microsoft without requiring new versions of third-party programs to be issued, as long as the bridging DLLs are adjusted accordingly to present the same interface to the client program. Indeed, there is no requirement for system call lists, arguments or syntax to remain static, as is the case in Linux, as standardised DLL interfaces mean that program behaviour is independent of kernel structure in Windows.

The situation is further complicated when third-party DLL development is considered. On one level, the DLL concept is an attractive software engineering feature as not only does it allow the same OS interface irrespective of version but it allows complicated and long-term software packages to utilise the same core source code for each version. Consequently, version control is greatly simplified and a significant reduction in the time and money required for incremental software updates is created as a result. This attractive aspect of DLL driven OSs means that large software houses will often create their own set of DLLs to provide a standardised interface for their various software packages, adding to the already expansive DLL set provided as part of the core Windows OS.

Unfortunately, the impact of a DLL driven OS architecture contains negative aspects as well as the positives discussed above. From a security perspective, the extensive DLL use in Windows opens attack vectors which are not present in Linux, such as DLL hijacking and injection [49, 50]. Furthermore, and directly related to this thesis's research questions, the DLL dependent structure of Windows means that system call traces cannot be used as a data source for HIDS as is the case for Linux. This arises because of the intermediary role played by the DLL corpus. In Linux, as discussed above, programs interact directly with the kernel through the use of system calls. In Windows, however, programs interact with DLLs which then interact with the kernel on the program's behalf. This layer of abstraction means that the kernel trace only contains information on the calling DLL rather than the program itself. As most DLLs are a common resource, system call traces captured at the kernel itself do not provide sufficient accountability to attribute observed behaviour to a single process.

In effect, reliance on DLLs means that each kernel system call can be accessed through multiple paths, and it is this factor which is most significant in understanding why kernel level system call analysis is unsuited to anomaly-based HIDS in a Windows environment. When writing a program, coders have the option of either using provided Windows DLLs, usually found in the Native API [125, 127] and its extensions, directly accessing the kernel system call, or writing their own DLL to act as a bespoke interface. Furthermore, legacy support means that a specific kernel system call entry may be accessed by different functions in the same DLL, adding an additional layer of non-linearity to the resulting kernel system call trace. Compounding the issue is the fact that most DLLs are a shared resource; as such, there is no guarantee that adjacent system calls from the same originating DLL belong to the same process. At a fundamental level, the combined effect of these characteristics of DLLs means that the positional patterns which are so important to Linux HIDS cannot be reliably extracted from Windows system call traces, at least at the kernel level.

The final major difference imposed by the DLL centric design of Windows is that advanced security techniques such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) have an impact on any attempt to equate DLL access requests with system call activity. ASLR is an important innovation which is designed to complicate buffer overflow attacks by randomising the various memory locations into which key OS components are loaded [162]. By doing so, attackers are forced to use more complicated techniques to access hard coded memory instructions instead of simply overwriting `EIP` with the predetermined memory address of the desired assembler sequence, such as the `JMP ESP` call used in a traditional stack-based overflow exploit [7]. In the absence of ASLR, hackers are able to simply scan memory pages in their simulated development environment and identify usable instructions in OS components which are likely to be loaded in their target. Examples in the Windows OS include the Native API, which is always loaded, and contains useful instruction such as `JMP ESP` or `CALL ESP` in a variety of locations. By randomising the address space, however, these modules will not be loaded in the same location each time and hence hard coding the addresses of these useful instructions is no longer possible. Exploitation in this environment requires a higher skill level, and the use of alternate techniques to access the overflow payload. The merits of ASLR are clear; this technique is hugely effective at hardening an OS and greatly complicates an attackers task. Whilst not an impervious defence, ASLR is a significant impediment to hackers and a marked security advantage.

The downside associated with ASLR, however, is that it makes analysis of DLL access requests difficult if not impossible in the Windows environment, as there is no way of quantifying the absolute and persistent location of a DLL in memory. In a Linux environment, ASLR has little impact on the efficacy of anomaly-based HIDS using system call traces as a data source, as each system call is identified discretely without modification by the ASLR process. Whilst the same holds true for windows system calls at the kernel level, DLLs will be loaded into different locations each time the computer is booted hence precluding an analysis of DLL access by absolute memory address. Note that this phenomenon is not limited to those models protected by ASLR, as third-party DLLs will often only be loaded when the particular program they support is started. As such, they will be written into the first available space in memory which will almost invariably be a different location each time. The combined effect of ASLR and third party DLL loading is to significantly complicate any subsequent DLL analysis, which is a contributing factor to the prevalence of signature-based AV systems and HIDS using alternate data sources in the Windows environment.

The discussion above should not be interpreted as arguing that positional information does not exist in system call traces from the Windows OS at all; indeed, logically it must persist in some form as the base interaction is the same as for Linux. Each program in a Windows environment is still interacting with the host's kernel, and system calls are still used as this interface. The major distinguishing factor is that DLLs have provided an extra layer of abstraction, and hence must be factored into the decision-making process. If the system call activity of a Windows host could be effectively linearised, then Linux-derived techniques should be applicable without fundamental revision. This would be a particularly attractive security enhancement for Windows, as the new data source would be of a sufficiently high quality to enable effective anomaly-based protection for the first time. The core theory of this chapter provides an artificial mapping which allows DLL manipulations and access requests to be considered using the same techniques as applied for a traditional Linux HIDS. By changing only the data source, the remainder of the IDS apparatus is able to function with the same efficiency as was achieved in its prime method of operation, effectively allowing algorithm optimisation independent of the target OS. This powerful new technique is discussed in detail in the following subsections.

4.1.1 The Virtual Kernel Concept

The core observation which can be made from the discussion and investigation detailed above is that the positional information comparable to that contained in Linux system call traces is still present in the Windows OS but obfuscated when measured at the kernel by the nature of the architecture. Consequently, if the high dimensionality of system call traces sampled at this level can be reduced to approach the linear nature present in Linux, existing HIDS algorithms should be directly translatable to the new OS without fundamental modification. This is a key point, and a driver for the related research questions outlined in Chapter 1, Section 1.3. By achieving this dimensionality reduction, the Windows OS would immediately be afforded the full protection of anomaly-based HIDS methodologies which have been optimised through rigorous review and improvement on the Linux OS. This portability arises because once the dimensionality of the Windows traces is reduced, the evaluating HIDS DE processes the resulting tokens rather than the raw data. Forrest's theory of system calls [23] can be generalised to include all linear kernel-level interactions, and hence the modified Windows traces can be considered in their natural format by pre-existing methodologies. The great benefit of this is that it negates any need for OS specific HIDS approaches, allowing general research to be applied to the full spectrum of OSs. A huge security weakness is significantly mitigated by bringing anomaly-based HIDS technology to Windows, closing many attack vectors which are currently effectively unprotected. The dimensionality reduction which powers the increased scope of protection is achievable by using the new VK theory developed by this research and detailed in this section.

VK theory proposes that instead of considering kernel level system calls, Windows HIDS should consider DLL access requests. This requires the introduction of a new term, namely the *system manipulation*, defined as any call by a program to a particular location within a DLL. Once the call has been made, the DLL will naturally involve other aspects of the system architecture to implement the desired action, probably using system calls and other low-level manipulations. The subsequent actions, however, are fixed and represent the effect on the kernel of calling that particular memory location. When a system call is made in Linux, the kernel performs a number of actions which are transparent to the program making the system call. A DLL access request provides exactly the same functionality but will perform a larger number of actions as it sits higher in the execution hierarchy than a system call; regardless, the resultant footprint is equally identifiable as the finer grained system call but without the dimensionality issues which have confounded system call based analysis in Windows prior to this innovation. As such, each system manipulation is comparable to a Linux-style system call elevated one level from the kernel to the boundary of DLL-space. As DLL access requests form a linear pattern, dimensionality reduction is performed inherently by this process and any consideration of these requests will provide the linearity required by existing HIDS algorithms.

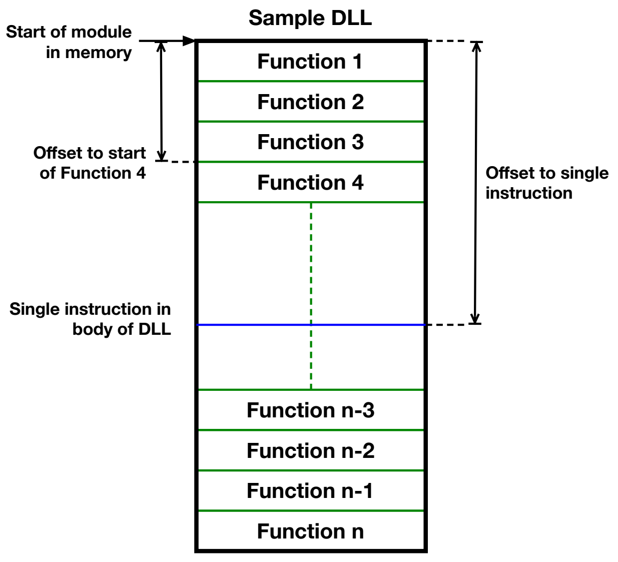
Sample DLL
Start of module in memory
Offset to start of Function 4
Single instruction in body of DLL
Offset to single instruction
Function 1
Function 2
Function 3
Function 4
Function n-3
Function n-2
Function n-1
Function n

Figure 4.1: Graphical Representation of VK Relative Offsets

The dynamism of modern Windows operating systems introduces a potentially critical level of variance in the absolute memory locations of each DLL which must be addressed by VK theory in order to produce a robust general solution. Hence, absolute referencing of system manipulations was rejected in favour of relative addressing. Under this approach, each system manipulation is now referenced by the calling module and an offset from the start of that module in memory, as shown in Figure 4.1. This means that each system manipulation is singly identifiable irrespective of the actual location in memory of each DLL, presupposing that the DLL structure itself does not change; this holds true for systems using ASLR as well as older systems without this protection. An example of a system manipulation trace using relative offsets is shown below[4], with the calling module and offset clearly identifiable:

```
... kernel32.dll+0x16d4f ntdll.dll+0x16d33 ntdll.dll+0x17113
ntdll.dll+0x15e4b ... comctl32.dll+0x11e3e comctl32.dll+0x12000
comctl32.dll+0x11e3e comctl32.dll+0x12063 kernel32.dll+0xb50b
kernel32.dll+0x19653 kernel32.dll+0x18d2c ...
```

Each of these tokens is a system manipulation and represents access to a discrete memory location, with the implied kernel level actions as a result attached as the footprint associated with that particular task. Each of the system manipulations in the example trace above uses a number of system calls, some more than others, to achieve the desired outcomes; as such, the multiple allowed paths available to access any particular individual system call are clearly evident which underscores the ambiguity inherent in simply considering Windows kernel events. Importantly, there is only one allowed path to access each system manipulation. This is not to say that there are not various methods of calling the instructions located at the specific memory location, but rather that execution of that particular component necessitate entry at the same offset from the start of the DLL module. Sampling system activity at this point in the architecture preserves the linearity of individual program action, where as sampling at a lower level introduces high dimensionality as a result of overlapping DLL activity.

In many respects, the functional Windows kernel includes loaded DLLs instead of just the kernel module itself; just as analysis of individual CPU instructions in a Linux system would present a hugely non-linear problem to an HIDS, so too does an analysis of Windows kernel system calls provide a similar situation. By conceptualising a virtual kernel consisting of loaded DLLs, the non-linearity issue is logically defeated, allowing subsequent analysis of system manipulations to provide HIDS

4. This trace was collected using the *Procmon* program [22] which supports this collection format in order to facilitate deep system analysis. Details of the collection methodology used are presented in Section 4.2.3.

protection. Examples of semantic dictionary entries compiled using this approach have been included in Appendix A, Section A.2, and clearly show the interleaved nature of DLL access requests from each constituent DLL of the VK, along with the relative offset scheme used to provide linearity in the resulting analysis.

An important distinction which must be made under VK theory is that of the *ideal* virtual kernel contrasted with an implementable *real* virtual kernel. The ideal virtual kernel would include all DLLs whether loaded or not; indeed, it should in fact include all DLLs in existence. This would provide the full range of allowable system manipulations and consequently render the greatest chance of ensuring spurious classifications do not occur. Clearly, however, this is infeasible. Even if it were possible to quantify all DLLs in existence at a particular point in time, there is no guarantee that this temporally sensitive VK would remain current for any significant length of time. Given the pace of software development, it would likely be superseded almost immediately. Furthermore, the sheer size of such a virtual kernel would preclude real-time processing on all but the fastest computers and impose a significant training burden on any HIDS. As such, a compromise must be reached between the granularity of the virtual kernel and its usability in real-world situations. This is achieved by carefully considering which DLLs should be included in the virtual kernel, and this topic is explored further in the following section. Suffice it to say at this stage that DLL selection requires careful attention by the IDS engineer at the design stage and will inherently require trade-offs between training and processing speed, performance, and lifespan of the trained HIDS. By creating an effective VK, however, accurate classifications become possible at a level comparable to those achievable in a Linux system as demonstrated empirically by the excellent results presented in Section 4.3.

In summary, this section has introduced the core VK theory which forms the main innovation of this chapter. This theory introduces the new term of *system manipulation*, which defines a DLL access request by quantifying the relative offset from the start of the loaded DLL. By using this method, HIDS algorithms developed for Linux system calls are subsequently applicable to Windows system manipulations allowing highly accurate classification as a result. The next Section addresses the considerations in selecting DLLs for inclusion in the virtual kernel, noting the impossibility of constructing an ideal virtual kernel containing the corpus of all DLLs in its entirety.

4.1.2 Dynamic Linked Library Selection

Careful DLL selection is a fundamental requirement for an effective real VK-based HIDS. As introduced in the preceding section, there is a stark difference between the theoretical ideal VK and an implementable real VK. Whilst an ideal VK should produce the best possible accuracy in a given situation, the constant state of flux in the Windows software environment precludes effective generation of any stable ideal virtual kernel. Fortunately, however, real VKs are readily implementable and are able to produce highly accurate results as shown in Section 4.3. Even if an ideal VK was feasibly implementable, however, the size of the resulting data streams would create a hugely complex situation with a correspondingly high impost on DE training time and flexibility. The impact on system resource consumption at the endpoint host would also be high, increasing the impact of implementation and resulting in a less attractive real-world solution; ideally, security protection should be transparent with minimal impact on the user experience and system performance. As such, it becomes necessary to accept a design trade-off between theoretically perfect granularity and implementable performance. Fortunately, there is considerable flexibility in the number of DLLs which can be included in a VK, allowing for effective balancing between classification accuracy, system resource consumption and DE training time in order to best suit the requirements of a particular deployment environment.

Once the decision on what DLLs to include in the VK has been made, significant effort and resources must be invested in training the DE of the HIDS to recognise the VK tokens and to tune the system baseline in preparation for anomaly-based classification. This is a lengthy process, even with a minimalist VK, and hence intervals between reprocessing or retraining should be maximised. Unfortunately, the VK is very sensitive to changes in DLL structure and consequently change management and patching schedules must be carefully coordinated within an organisation's asset pool to avoid unnecessary retraining of VK-based HIDS. As system manipulations are based on access requests to specific memory locations relative to the DLL start position, any change to the internal structure of the DLL in question will inevitably significantly affect the resulting classification accuracy. Consequently, DLLs selected for inclusion in a VK should be as stable as possible in order to avoid a significant retraining burden. This may not always be possible due to specific requirements for the protected system, but nonetheless remains a strong influence on prudent DLL selection. The sensitivity of a VK to changes in DLL structure is not a significant disadvantage when compared to similar strictures in other approaches. Linux systems, for example, require retraining of the system call-based HIDS if the kernel architecture changes and new system calls are added or existing ones removed. Indeed, the installation of new software by itself can often require a retraining of various anomaly-based protection mechanisms as there is a profound effect on the underlying system baseline upon which the HIDS has been trained. As such, the retraining burden imposed by DLL changes in a VK-based HIDS is not significantly greater than similar factors which affect traditional anomaly-based HIDS and should not be viewed as a specific disadvantage of this new methodology. Indeed, careful

DLL selection can significantly mitigate this issue and prolong the effective life of each training cycle.

Several principles guide the DLL selection process for VKs. First and foremost, the selected DLLs must be representative of system behaviour. Without representative system manipulations as a result of appropriate DLL choices, the system baseline produced as a result will be incomplete and unable to provide appropriate metrics for activity classification. This is likely to result in high FAR as the resultant baseline will not encompass the full suite of normal system behaviour and consequently will incorrectly classify normal behaviour as anomalous. Clearly, the included DLLs must be able to accurately represent the full range of normal activity if high accuracy results are to be produced. Of note, the research in [65] use a very small DLL selection consisting of only the Windows Native API; whilst this research did not conduct extensive testing of this HIDS approach, it is likely that the symptoms described above would be readily apparent in full testing against a contemporary datasets such as the ADFA-WD.

The second key principle for DLL selection is that likely threat vectors must be specifically covered by the selected modules. A good example of this is provided by considering the networking process in the Windows OS. Networking is not an inherent function of the Windows Native API, and is handled by a collection of DLLs such as `ws2_32.dll` and `mswsock.dll` [125–127]. If these DLLs are not included in the VK then networking activity will be completely transparent to the HIDS and consequently not assessed for anomalies. Fundamentally, low impact functions such as display of the graphical user interface and the presentation of data to the human user do not feature in high-level hacking activity and consequently should not form part of the VK, despite the location of the responsible DLLs within the Windows Native API. Instead, prioritisation should be given to core interfacing activities and critical system functions.

The third and final key principle is that DLL selection is fundamentally dependent on the specific activities to be conducted on the particular host in question. Certain core DLL should always be included in the VK, with the list provided below indicating a minimalist VK implementation, but a change in host behaviour may necessitate inclusion of additional DLLs. Example of this would be when mail server functionality is added to a host, perhaps to facilitate distribution of information within an intranet. In this case, the new mail server represents a significant new attack vector, and the relevant DLLs should consequently be included in the next iteration of the VK to allow appropriate protection to be provided. Conversely, including additional DLLs in the VK beyond those needed for the host's role imposes an additional training and real-time processing burden and should be avoided; careful application of this principle should allow each individual VK implementation to be optimised for the specific host to be protected, creating significant efficiency savings as a result.

The VK implementation used to prove the theory and generate the results contained in Section 4.3 consisted of the following DLLs:

- `ntdll.dll`
- `kernel32.dll`
- `user32.dll`
- `comctl32.dll`
- `ws2_32.dll`
- `mswsock.dll`
- `msvcrt.dll`
- `msvcpp.dll`
- `ntoskrnl.exe`

This selection was made by following the principles outlined above, and achieves a representative system baseline without unduly increasing the amount of data processing required by the eventual HIDS. The selected DLLs cover three main areas. `Ntdll.dll`, `kernel32.dll`, `user32.dll` and `comctl32.dll` deal with core system activity such as accessing files, reading data, accepting input, starting and stopping system services, and creating and killing processes. The set of functions covered by this DLL cluster relates to fundamental enabling activities designed to provide the modern computing experience; as such, their inclusion in the VK is clearly warranted.

Networking functions such as the creation of sockets and the transfer of data are covered by `mswsock.dll` and `msvcrt.dll`. The inclusion of this set of functionality in the VK is important to fully profile the presence of the host on the network and to enable web-based activity to be properly quantified. Without including networking, socket creation patterns and connection dynamics would not be assessed which is particularly detrimental to the overall security posture if rigorous firewall rules are not in place.

Finally, `msvcrt.dll` and `msvcpp.dll` were included to address privilege escalation attacks, which are typically used by hackers once an unprivileged foothold has been gained in a system. Many of these escalation activities require advanced coding functions found in these DLLs, and their inclusion is a significant mitigator against local privilege escalation. Similarly, `ntoskrnl.exe` itself was included to catch direct access requests to its memory locations, which is a technique used by some privilege escalation vectors. `Ntoskrnl.exe` was assessed using system manipulations, not system calls, and consequently can be viewed as a DLL for the purposes of the virtual kernel. In other words, the system calls made by DLL functions were not included in the trace for `ntoskrnl.exe`; only access requests to the various core functions provided in a DLL-like interface were used as inputs to the HIDS. If `ntoskrnl.exe` was assessed using system calls, the resultant HIDS would suffer from the non-linearity issues discussed in the preceding sections with a correspondingly significant loss of accuracy.

The selection of DLLs listed above represents the smallest VK likely to provide accurate classifications. Whilst it may be possible to remove some DLLs from this list in specific instances, this action would need to be carefully considered in light of the activities forecast for that host. Broader functionality will require additional DLLs in order to accurately baseline the system, but these decisions will need to be made on a case-by-case basis.

4.2 The ADFA Windows Dataset

The need for HIDS detection in the Windows OS has been clearly established by the discussion presented in the earlier sections of this chapter and the body of Chapter 2, and the success of the VK theory means that research into this sub-field will likely increase significantly in volume in the future. Validating the VK theory necessitated the creation of a comprehensive IDS dataset, however, and the ADFA-WD was developed to fill this requirement. In addition, comparison of research approaches requires a common standard to provide effective performance benchmarking and to facilitate discussion and analysis of the advantages and disadvantages of competing solutions. No such benchmark existed for the Windows environment prior to the creation of the ADFA-WD, with IDS datasets focused exclusively on the Linux environment[5]. Hence, the ADFA-WD represents the first attempt to provide a common evaluation standard for this OS across the research discipline, and is a valuable contribution in its own right for this reason. As will be explored more fully in the following sections, the ADFA-WD uses a collection of modern threats situated in the mid to upper band of attack sophistication. This means that the relevance of the ADFA-WD is unlikely to fade in the mid-term, allowing standardisation of performance evaluation for some time. Notwithstanding the anticipated lifespan, however, future researchers should ensure that the dataset remains relevant to the contemporary situation prior to use; the research community will not be served well by the persistent use of outdated IDS datasets, as has been strongly commented on in relation to the KDD collection [116, 117, 119–121].

The development of a dataset for the first time in a new operating system requires a more comprehensive approach than when the OS has already had IDS datasets created for it. In the Linux domain, the attack surface is well quantified from the HIDS perspective and dataset structures are consequently well-defined. As such, the ADFA-WD required a lengthier design process than the ADFA-LD[6]. Furthermore, Linux HIDS require specific datasets for each individual OS variant due to the fact

5. For example, the KDD98 [20], KDD99 [21] and the UNM [106] datasets.
6. See Chapter 3, Section 3.5.2.

that Linux is fundamentally open source and hence numerous versions and implementations are available. Whilst all Linux version share many common features which allows general performance conclusions to be made despite the highly focussed nature of Linux datasets, the eventual implementation in each Linux variant differentiates system behaviour significantly. This in turn means that generalisability between Linux systems is functionally limited. In Windows, however, the OS is controlled by a single authority and hence the different versions share a strong evolutionary trend, implying that a Windows IDS dataset should be constructed so as to be relevant to all versions of the operating system rather than focused in a single niche. This necessitated careful consideration when designing the ADFA-WD and was a significant reason for the selection of the OS version and the attack vectors.

IDS datasets must also be representative of the full range of the attack surface presented by the host. Unlike many earlier datasets in the Linux domain, the ADFA-WD covers a wide scope and aims to address the full range of activities which could reasonably be expected of a modern Windows computer. Manageability of the dataset is also an important consideration, with data size a specific concern. If the dataset is allowed to grow to very large sizes, the usefulness of the corpus is significantly reduced. Whilst on one hand the increased data allows for a more rigorous assessment, the significant processing overhead associated with training DEs on large datasets means that a subset is often selected for assessment in order to expedite evaluation. There is no guarantee that this subset is in fact representative of true system behaviour, however, and hence results obtained using this method may be skewed. A better approach is to provide a comprehensive dataset with a carefully controlled size so as to encourage use of the full dataset. By following this approach, results are better able to be compared with other researchers and the level of ambiguity in evaluations is reduced. Ideally, datasets would also be extensible in order to allow new attacks to be added seamlessly to the existing corpus as they are discovered. The ADFA-WD has been designed with this principle in mind which will assist in prolonging the effective life span of the resultant dataset.

Addressing all of the requirements outlined above was a challenging task. Data size, in particular, proved problematic due to the large quantities of data generated at the system manipulation sampling level. In order to effectively apply VK theory for different virtual kernel implementations, all DLL access requests must be captured including those not necessarily contained in the VK implementation used by this research to prove the theory. If DLL collection was limited, then the flexibility of the dataset and scope for further research would be significantly reduced. As a result, full DLL access requests were captured which in turn increased the size of the eventual dataset. Fortunately, filtering by DLL name is easy to effect which provides researchers with the ability to remove data not directly relevant to their

VK implementation, thus returning the functional dataset size to a more manageable level. Comprehensive attack surface coverage will never be possible due to the continual state of flux which the modern computing environment offers; this presented a second significant challenge when selecting the attack surface to be offered by the vulnerable host. In order to provide a representative data set, the attack surface must reflect that offered in a real-world setting without being overly trivial or overly difficult to penetrate. This balance required careful and ongoing evaluation throughout the design and collection phases, and is discussed further in Sections 4.2.1 and 4.2.2.

The design philosophy used when creating the ADFA-WD centred on creating a general test of IDS performance rather than catering to niche requirements. This meant that the selected attack surface resembles a generic endpoint terminal rather than a specific server implementation. By presenting this more generalised host to the simulated attacks, HIDS performance is arguably more representative of the true merit of the assessing algorithm. Logically, each individual method of intrusion detection will perform better in some specific applications than in others as an inherent aspect of the decision algorithm in use. Ideally, the perfect HIDS solution would perform equally well in all situations but this ideal performance is unlikely to be realised. By deliberately creating a general dataset, the performance variations of each algorithm are mitigated by removing niche advantage which consequently allows for a better comparison of the general merits of each approach. In providing this general attack surface, the host was modelled on a generic private use computer which conducts web browsing, file transfers and hosts a personal website. This mimics the configuration likely to be encountered within intranets for medium-sized organisations with a looser security posture, and presents a representative yet general target.

In keeping with this broad design philosophy, the threat vectors selected for inclusion in the ADFA-WD differ from the standard network-based attacks used in historical datasets like KDD98 [20]. These legacy datasets created an attack collection which focuses on server targeted attacks delivered from a simulated intranet environment. Whilst an important aspect of computer security, server-side attacks are only one part of the modern threat. Indeed, most modern computer users will be exposed to social engineering and browser attacks much more regularly than to traditional server attacks [10]. Furthermore, malware-based attacks are rarely considered in HIDS evaluation, largely because the triggering mechanisms for these programs often rely on a user performing a normal action to activate the program rather than the clearly aberrant behaviour of a buffer overflow server attack, for example. Malware is a significant component of the modern threat, however, and AV systems are not always effective at detecting obfuscated payloads[7]. Hence, inclusion of malware attacks is a crucial element of the ADFA-WD and represents an important threat vector in the contemporary environment. As such, the ADFA-WD includes the full range of attack types likely to be encountered in the modern cyber environment so as to provide a contemporary challenge to an evaluating HIDS. In doing so, the ADFA-WD represents a significant diversion from traditional datasets and presents a much more challenging and relevant evaluation task for IDS researchers.

4.2.1 Design Rationale

After much consideration, *Windows XP Service Pack 2* was selected as the host OS for the ADFA-WD. This decision reflects a careful evaluation of the requirements of an IDS dataset and maximises the usefulness of the ADFA-WD, despite the early location of this OS version within the Windows development time-line. Whilst the latest versions of the Windows OS have a very different look and feel, they share much of the internal structure of the earlier versions. Indeed, the fundamental Windows architecture has not changed significantly [125–127] throughout the development of the OS family, although improvements and modifications have naturally been made by Microsoft in order to continually develop and evolve their product. Particularly relevant additions to the core OS framework include security controls such as ASLR and DEP. This presented a unique challenge when creating the ADFA-WD; ASLR and DEP are significant security enhancements featured in later versions of Windows and significantly mitigate many low-end threats, but are not present in *XP SP2*.

It is possible, however, to bypass these protections using techniques such as Return-Oriented Programming (ROP) [163, 164] which presents a significant threat to otherwise solid security controls. One of the main reasons for selecting *XP SP2* was that it does not contain these protections. While initially counter-intuitive, this means that HIDS performance can be evaluated in isolation from these additional protections; as a result, artificial inflation of performance results will not occur due to assistance by security measures other than the HIDS itself. This is an important consideration as *defence-in-depth* requires overlapping security controls to be individually optimised in isolation [14–18] before combining them to provide protection as a whole. Consequently, if HIDS performance was evaluated on systems with ASLR and DEP already enabled then thorough profiling of the HIDS itself would not be possible. Indeed, if HIDS algorithms are reliant on other security measures then hackers only need to bypass the weakest link to defeat the entire suite of security protections. Hence, selection of the older OS version without advanced security

7. See the simple obfuscation routine in Section 4.2.2 which bypassed *Norton AV 2013* [161].

controls allows higher quality HIDS assessments to be made and is a convincing argument in itself for basing the ADFA-WD on *XP SP2*.

Notwithstanding the discussion above, the *XP* default firewall was enabled for all attacks. This was done to ensure that the hacking attacks launched against the target would be sufficiently sophisticated to bypass this elementary security control. It is reasonably easy to circumvent the stateless *XP* firewall, but it is an effective counter to extremely low level attacks. As such, the enabling of this control forces the attack set to a higher level and increases the relevance of results generated from the final dataset to threats currently being faced in cyberspace. Additionally, *Norton AV 2013* [161] was used to scan certain payloads as discussed in Section 4.2.2. Many modern hackers have the ability to bypass signature-based recognition systems, and hence payload obfuscation detection is an important contribution of anomaly-based systems. By including payloads which have successfully bypassed an advanced and well-regarded AV program, subsequent detection by anomaly-based methodologies indicates an important contribution to the overall security posture of the host as a whole.

Shortly after the completion of the ADFA-WD, Microsoft announced the end of support for *Windows XP* in April, 2014 [165]. This does not limit the relevance of the ADFA-WD, however, as the similarities between this version of the OS and subsequent versions is high as has been discussed above. Whilst the natural evolution of the OS will in time limit the applicability of results generated using the ADFA-WD, this is unlikely to occur for several years. Furthermore, end of life support for *Windows Vista, Windows Server 2008*, and *Windows 7* has not been announced and is certainly not likely in the mid-term. *Windows XP* shares many marked similarities with these operating systems and consequently the ADFA-WD is certainly relevant whilst these systems continue to be used. Best practice for IDS test dataset generation would seem to require updating the dataset every 3 to 4 years, if not more frequently; the ADFA-WD will certainly remain relevant during this time frame. There is also a strong argument to be made that security controls should be optimised for older systems which do not have the defences included by default in more modern versions. As such, the next Windows test dataset should probably focus on *Windows Vista* or *Windows 7* as these versions will likely be increasingly vulnerable to the next wave of attack technology by the time the ADFA-WD requires replacement.

The Attack Framework shown in Chapter 3, Figure 3.3 was used extensively in the selection of the attacks to be used for the ADFA-WD, with the specific *vectors* and *effects* used for the ADFA-WD shown in Table 4.2. The discussion on the other aspects of the Attack Framework as they relate to the ADFA-LD[8]

8. See Chapter 3, Section 3.5.2.

also hold for the ADFA-WD and have not been repeated in this chapter. The vectors selected are particularly important for this dataset, and include a much wider range than has been typically used in Linux OS datasets. Web-based vectors are extremely relevant to the modern cyber environment with recent reporting in [10] clearly identifying the massive growth of this attack class in 2012. Previously, HIDS evaluation of this class of attack has not been tested and the inclusion of several different types of web oriented attacks in the ADFA-WD provides a unique modernisation of HIDS evaluation requirements. Defence against web-based vectors is extremely difficult for signature-based systems as accessing hosted pages and supplying input to web languages such as PHP or JavaScript are fundamentally the same for normal activities as for attacks. Additionally, the high variance in content for web activity coupled with the ease of obfuscation means that identification of clear signatures is extremely difficult. Consequently, the web presence of a given organisation often represents the most porous element of the security perimeter which accounts for the increased attack trend reported in [10]. Noting the difficulty signature-based systems have in detecting this type of attack, the importance of providing anomaly centric protection is readily apparent.

Table 4.2: ADFA-WD Vectors and Effects

Vectors	
• TCP ports;	• Browser attacks; and
• Web based vectors;	• Malware attachments.
Effects	
• Bind shell;	• System manipulation;
• Reverse shell;	• Privilege escalation;
• Exploitation Payload;	• Data exfiltration; and
• Remote operation;	• Back-door insertion.
• Staging;	

A wide selection of effects was used for the various payloads and post-exploitation activities used against the ADFA-WD target host, producing a representative selection of typical hacking activities. Whilst all attacks would ideally be detected as soon as the vulnerability is triggered, this may not always be possible; as such, a varied selection of test payloads means that post-exploitation activity may well breach the required detection threshold and trigger an alert, resulting in a defeat of the attack as a whole. Each different payload and system manipulation has been carefully identified in the dataset in order to allow future researchers to isolate the particular activity which is detected by their algorithms, resulting in increased granularity when evaluating IDS performance.

The ADFA-WD provides attacks ranging from low sophistication through to upper midrange sophistication. Of note, highly sophisticated attacks were not used when compiling the data set. The main reason for this decision was that the upper end of hacking is extremely situationally specific. Whilst generic advanced techniques exist for activities such as ASLR bypass in certain circumstances or DEP evasion using ROP techniques, complex attack vectors are almost invariably custom tuned for the target system. As a result, generic defence against the upper end of the threat is not necessarily assessable through performance against specific attacks; individual high-end attacks are inherently unable to provide a generalizable threat due to their highly specialised nature, precluding extrapolation of performance on any subset of such attacks to include the entire set. Furthermore, the more specific an attack pattern is, the more sensitive it is to small changes in the target's configuration. As such, there is clearly little merit in specifically focussing HIDS evaluation standards on protection rendered against an attack which may not be effective against all bar a very small percentage of the entire cyber domain, and which does not inherently indicate similar protection against other attacks of the same calibre.

Low to mid-range attacks, however, are most definitely generically defensible and an appropriate performance challenge was created by including a full range of difficulty levels within this band in the ADFA-WD. By presenting a challenging and reasonably advanced attack set, HIDS optimisation is easier to quantify resulting in better tuned IDSs which are capable of more accurate classification. This is, in fact, the best defence against the upper end of hacking attacks which are likely to be zero-day in nature and as discussed above, unable to be generically modelled. By optimising HIDS performance, deployed systems consequently have the best chance of detecting new attacks and demonstrating effectiveness across the entire suite of sophistication levels, particularly if anomaly-based methodologies are employed.

Noting the points raised above, the ADFA-WD is clearly relevant to the modern cyber environment and presents a significant and broad challenge. The selection of an OS version which does not already benefit from advanced security measures such as ASLR and DEP ensures that HIDS performance is evaluated without artificial inflation and in an isolated environment. Results generated against the ADFA-WD can reasonably be extrapolated to suggest a similar measure of performance in other Windows OS versions due to the core similarities at a fundamental level between the various members of the Windows family. An increased number of vectors has been included in the ADFA-WD to better model the contemporary threat, particularly noting the significant increase of web-based attacks outlined in [10]. By stepping away from datasets which focus exclusively on network-based attacks, the ADFA-WD provides a much more accurate evaluation of the ability of an HIDS to defend against the broad spectrum of threats present in the modern environment. The best protection against high-end bespoke hacking attacks is a well tuned anomaly-based system [10,13,166], and the mix of sophistication levels, vectors, and effects present in the ADFA-WD provides an appropriate benchmark for performance evaluations of this class of HIDS.

4.2.2 Structure

The software environment selected for the ADFA-WD consisted of the standard *XP* OS environment with file sharing enabled and a network printer configured. The addition of an FTP server, web server and management tool, and a streaming audio digital radio package provide a generous network-based attack surface. A mix of wireless and Ethernet networking was used to provide connectivity, with one of the more advanced attacks[9] using a fake wireless access point to provide DNS spoofing and a consequent browser attack upon redirection. The complete list of vulnerabilities and additional non-OS software used is provided as Table 4.3, with the term *Vulnerability Identifier (VID)* used hereafter to describe each of the 12 vulnerabilities listed in this table for ease of reference.

A target ratio of 1 : 10 : 1 =*normal data*:*validation data*:*attack data* was used to guide collection and structuring activities, and a complete list of the collected trace statistics is included as Table 4.4. This table also contains an entry for trace counts when split into 200 element lengths; this technique is discussed in detail in Section 4.2.3. Table 4.5 shows a detailed breakdown of the activities conducted for each specific data generation task. Of note, the selected activities have been tailored to baseline the behaviour which the particular exploit used for each VID targets. They do not represent a complete baselining of the program or OS as this level of granularity is unnecessary for a test dataset. For real-world operations, however, complete baselining of the protected system is naturally a critical aspect of HIDS deployment and must be as comprehensive as possible.

9. VID 8, Table 4.3.

Table 4.3: Vulnerability list

VID	Vulnerability	Program	Exploit Mechanics
1	CVE: 2006-2961	CesarFTP 0.99g	Reverse Ordinal Payload Injection - custom exploit.
2	EDB-ID: 18367	XAMPP Lite v1.7.3	Xampp_webdav used to upload and execute a malicious payload.
3	CVE: 2004-1561	Icecast v2.0	Metasploit Framework exploit used.
4	CVE: 2009-3843	Tomcat v6.0.20	Metasploit Framework exploit used.
5	CVE: 2008-4250	OS SMB	Metasploit Framework exploit used.
6	CVE: 2010-2729	OS Print Spool	Metasploit Framework exploit used.
7	CVE: 2011-4453	PMWiki v2.2.30	Metasploit Framework exploit used.
8	CVE: 2012-0003	Wireless Karma	Pineapple Router with an active *Karma attack* and DNS Spoofing used to automatically associate and redirect to an attacking computer hosting a browser attack.
9	CVE: 2010-2883	Adobe Reader 9.3.0	Malicious PDF file used to spawn a reverse shell.
10	···	Back-doored Executable	Reverse Inline Shell spawned.
11	CVE: 2010-0806	IE v6.0.2900.2180	Metasploit Framework Exploit used.
12	···	Infectious Media	Bind Shell spawned.

Table 4.4: Dataset Statistics

Data Type	Sampling Runs per VID	Number of Raw Traces	Length 200 Traces
Normal Data for Training	10 runs	356 traces	67,752 traces
Normal Data for Validation and FAR Calculation	100 runs	1,828 traces	590,675 traces
Attack Data for DR Calculation	10 runs	5773 Traces	374,806 traces

VID	Training Activity	Validation Activity
Background	3 minutes	30 minutes
1	- 10 MKD commands. - 10 GET commands. - 10 LS commands. - 10 PUT commands.	- 100 MKD commands. - 100 GET commands. - 100 LS commands. - 100 PUT commands.
2	- 10 uploads. - 10 downloads.	- 100 uploads. - 100 downloads.
3	10 client connections with the server already streaming.	100 client connections with the server already streaming.
4	10 separate WAR files activated once each.	10 separate WAR files activated 10 times each.
5	- 10 uploads. - 10 downloads.	- 100 uploads. - 100 downloads.
6	10 print jobs.	100 print jobs.
7	Create 10 pages.	Create 100 pages.
8	10 connections and subsequent browsing.	100 connections and subsequent browsing.
9	10 readings of a normal file.	100 readings of a normal file.
10	10 activations of the normal file.	100 activations of the normal file.
11	15 minutes of browsing activity.	150 minutes of browsing activity.
12	10 insertions of CDs, each with a different program.	10 insertions of 10 different CDs.

Table 4.5: Training and Validation Data Collection Activities

As introduced in Section 4.2.1, the Windows default firewall was enabled for all exploits. This is an important contributor to the usefulness of the ADFA-WD, as it excludes by default all trivially low-level hacking exploits which are unable to circumvent the relatively simple firewall. For the most part, the *XP* firewall was bypassed by carefully selecting ports for connect-back payloads or by hijacking existing connections through socket reuse. The default outbound posture of the *XP* firewall is to prompt the user to authorise outgoing connections, but it does this by granting permission to a program as a whole and in perpetuity, rather than using a finer grained approach to access control. Consequently, so long as the exploited program has been authorised to connect out past the firewall prior to the act of exploitation, as is likely the case for any server programs, then this authorisation will be inherited by the shellcode once it hijacks the execution flow. Similarly, commonly used ports such as 80, 21 and 22 representing web traffic, FTP traffic, and SSH respectively are allowed by default; hence, in circumstances where the exploited program does not have previously granted permission to transit the firewall then a payload constructed to utilise these default ports will not be impeded by the firewall. If a connect-back payload is not possible due to the firewall configuration, then server programs which are configured to listen on a particular port generally have permission to open ports in the firewall, thus allowing a port-bind payload to be attached to a new listening port and effectively circumventing the firewall's protection. Finally, if the exploit is prevented from implementing either a connect-back payload or port-bind payload, then a system command can be issued by the shellcode instead, causing *Internet Explorer* to connect to a specified web-page hosted by the attacker and containing malicious code, resulting in a forced social engineering style attack. Unless web browsing is completely disallowed by the firewall, this approach will always be successful within the *XP* environment. The disadvantage of this tactic is that it requires the spawning of a new process, with a corresponding system footprint and increased risk of detection.

The second allowed security enhancement enabled on the target host was the presence of a modern AV system, namely *Norton AV 2013* [161]. This product is arguably one of the best signature-based protections available and is used extensively throughout the world [161]. The inclusion of AV was a deliberate step, and is based on the fact that education levels amongst modern IT professionals and users are of a sufficient level to reasonably expect that the majority of Windows computers will have some form of AV protection installed. Whilst not an inherent part of the OS, AV is regarded as a fundamental and widely used protection and hence is a valid addition to the base OS for dataset generation; unlike ASLR and DEP, it is available for all Windows versions. Furthermore, the fact that all attacks successfully evaded the AV system highlights dramatically the need for anomaly-based protection; AV is fundamentally signature-based, and focuses on the program as it resides in storage, whereas an anomaly-based HIDS focuses on the actual behaviour of a program at runtime.

Figure 4.2: Shellcode Encoding Algorithm

```
1: function ENCODESHELLCODE(locationStart,locationStop)
2:     originalEIP ← EIP
3:     key ← 0x27182818
4:     EDX ← Key
5:     for all memory between locationStart and locationStop do
6:         XOR dword [current memory], EDX
7:         EDX++
8:     end for
9:     originalEIP → EIP
10: end function
```

Norton AV 2013 was used to manually scan the payloads for VIDs 2, 4, 10 and 12. Each of these payloads had been manually obfuscated to evade detection using the algorithm provided in Figure 4.2. This is algorithm is quite simple compared to the more advanced AV bypass techniques but was nonetheless effective at completely evading detection by *Norton AntiVirus 2013*. The algorithm is implemented by diverting the execution flow at the program entry point to an unused portion of the binary where an assembler stub has been inserted. The effect of the code contained in the stub is to XOR all memory between the specified locations with the provided key, effectively encrypting the file contents with a simple but easily reversible process. After the file has been decrypted at runtime, execution flow is restored to the original entry point and the program continues as normal. A similar process was used for VID 10, where the malicious payload was inserted into an artificially added section of the binary. Execution flow was again hijacked to run this set of instructions before returning to the normal program, effectively creating a `portbind` payload listening for incoming connections whilst presenting the same program behaviour to the user. These techniques are not hard to implement, and are widely used in the hacking community. The fact that they can trivially bypass signature recognition systems is a telling argument in favour of anomaly-based protection.

VID 1 used a custom-made exploit to compromise the FTP server. The shellcode for this exploit was generated using the `Reverse Ordinal` payload included with the Metasploit Framework (MSF) [1] and represents a fairly traditional server-side attack. VID 2 utilised default credentials in the `webdav` plug-in to upload and execute a malicious payload and was the first web-based attack against the host. VID 3 employed a standard MSF exploit to compromise a streaming media server in another server-side attack. VID 4 compromised the *Tomcat Manager* application to upload and execute a connect back payload, extending the web-facing attack surface. VID 5 and 6 attacked inbuilt aspects of the OS SMB implementation, focussing on file sharing and printer sharing. VID 7 introduced a web application attack, with a multi-stage hack against *PMWiki* resulting in full system compromise. VID 9 compromised the popular *Adobe Reader* program with a PDF containing executable malware, and VID 10 created a connect-back shell in the execution flow of an otherwise benign program. When the program in VID 10 was opened by the user, a shell was spawned on the attacker's computer whilst presenting the same interface and functionality to the user. VID 11 used a browser attack to compromise *Internet Explorer* when the user visited a malicious web-page, mimicking the increasing use of this vector by hackers "in-the-wild" [10]. VID 12 used a compromised autorun procedure on a CD to execute a `portbind` payload when the CD was inserted. This method is similar to that hypothesised for some of the `Stuxnet` infections [167–170], which used a USB rather than CD but likely exploited the same technique.

VID 8 is a special case and uses a piece of hacking hardware known as the `Pineapple Router` [171]. This device is designed to simulate any previously joined wireless access points that a host may have used in the past, and when it detects a probe request containing a specific `SSID` responds as if it was that access point. In doing so, it effectively inserts itself as a man-in-the-middle device and is able to perform traffic manipulation, redirection and modification. For the specific example used in designing the ADFA-WD, the `Pineapple` was used to automatically associate with the target host and then perform DNS spoofing to redirect web traffic from the host to the attacker's computer where a malicious web-page containing exploit code was hosted. This web-page in turn attacked the browser, and was successful in compromising the computer as a result. This multi-stage attack is particularly relevant in the era of portable computing where users are increasingly comfortable connecting to publicly available Wi-Fi. In doing so, they expose their personal computers to significant risk and this attack is an important element of the ADFA-WD as it specifically tests for HIDS ability to detect this class of attack.

After each VID was exploited, a set of post-exploitation activities was conducted. These activities ranged in scope from simple exploitation of the underlying vulnerability through to privilege escalation and data exfiltration. Table 4.6 provides details of the activities conducted for each particular VID, with elaborating remarks defining each activity code provided in the pages following this table. Of note, each vulnerability required a different post-exploitation approach based on the peculiarities of the exploit used. In particular, web-based attacks generally leveraged PHP payloads to gain the initial foothold before upgrading to a native binary payload with increased functionality. This step is unnecessary for server-side attacks, as a native binary can be used as first payload in these situations due to the nature of the exploitation environment. A range of privilege escalation techniques including token stealing and kernel exploitation were used for those VIDs which did not exploit processes running as `system`. The highest level of system effect generated was the display of the stored password hashes, which if cracked provide user credentials for the host. This activity is an important intermediate goal for hackers when compromising systems, as it potentially allows persistent authenticated access to the host on demand. By conducting a wide range of post-exploitation activities, the ADFA-WD presents a realistic set of hacking activities which closely matches observed behaviour in the real world [10]. It is hence a relevant challenge for HIDS algorithms, and is suitable for use as a performance benchmark.

VID	Activity
1	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4.
2	- 5 x P1. - 5 x P2. - 5 x P3. - 5 x P4.
3	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4..
4	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4.
5	- 5 x N1. - 10 x N2. - 5 x N4.
6	- 5 x N1. - 10 x N2. - 5 x N4.
7	- 5 x P1. - 5 x P2. - 5 x P3. - 5 x P4.
8	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4.
9	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4.
10	- 5 x S1. - 5 x S2. - 5 x S3. - 5 x S4.
11	- 5 x N1. - 5 x N2. - 5 x N3. - 5 x N4.
12	- 5 x S1. - 5 x S2. - 5 x S3. - 5 x S4.

Table 4.6: Attack Data Collection Activities

Table 4.7: Activity Code Legend

- Attack N1: Native Payload Exploit Only
  - Exploit vulnerability.
  - Cease collection.
- Attack N2: Native Payload Data Exfiltration
  - Exploit vulnerability.
  - Download target file.
  - Cease collection.
- Attack N3: Native Payload Privilege Escalation
  - Exploit vulnerability.
  - Escalate privileges.
  - Cease collection.
- Attack N4: Native Payload System Manipulation
  - Exploit vulnerability.
  - Escalate privileges.
  - Get hashes.
  - Cease collection.
- Attack P1: PHP Payload Exploit Only
  - Exploit vulnerability.
  - Cease collection.
- Attack P2: PHP Payload Data Exfiltration
  - Exploit vulnerability.
  - Download target file.
  - Cease collection.
- Attack P3: PHP Payload Privilege Escalation
  - Exploit vulnerability.
  - Upload native payload.
  - Escalate privileges.
  - Cease collection.
- Attack P4: PHP Payload System Manipulation
  - Exploit vulnerability.
  - Upload native payload.
  - Escalate privileges.
  - Get hashes.
  - Cease collection.
- Attack S1: Native Payload Exploit Only
  - Exploit vulnerability.
  - Cease collection.
- Attack S2: Native Payload Data Exfiltration
  - Exploit vulnerability.
  - Upgrade shell to meterpreter session.
  - Download target file.
  - Cease collection.
- Attack S3: Native Payload Privilege Escalation
  - Exploit vulnerability.
  - Upgrade shell to meterpreter session.
  - Escalate privileges.
  - Cease collection.
- Attack S4: Native Payload System Manipulation
  - Exploit vulnerability.
  - Upgrade shell to meterpreter session.
  - Escalate privileges.
  - Get hashes.
  - Cease collection.

4.2.3 Data Collection and Formatting

Data was collected by running the *Procmon* [22] program to capture all DLL and system call activity whilst the various training, validation, and attack activities were conducted. The output from this program can be formatted in a number of different ways, and a suitable format which supports VK analysis is one of the options. As such, the collected traces can be easily manipulated to render system manipulation data, allowing the direct application of the new VK theory. This is a convenient aspect of the *Procmon* utility, and greatly facilitated compilation of the ADFA-WD. The resulting raw output contains a large amount of data which is not directly relevant to a VK-based HIDS data source, and hence preprocessing was required to remove the extraneous information. Regardless, the raw data is included as part of the ADFA-WD dataset in case future researchers develop a need for this extra data.

Once the raw output of *Procmon* has been reduced to a system manipulation level, a second simple step of preprocessing was used to group system manipulations by PID. The result of this phase of data formatting is to create a complete list of all system manipulations for all DLLs which were loaded during the collection cycle. Hence, the resulting traces must be further filtered to apply the specific VK structure required by the virtual kernel implementation in use for any given experiment. As mentioned above in Sections 4.1.1 and 4.1.2, this PhD research used a relatively small virtual kernel in order to minimise the DE training time for the HIDS; full DLL data is nonetheless provided in the ADFA-WD to enable future researchers to flexibly select the VK structure which best suits their needs.

Care was taken to ensure that an appropriate ratio was maintained between training data and validation data when compiling the ADFA-WD. The ratio of 1 : 10 ensures that an appropriate challenge is presented in keeping a low FAR while still achieving acceptable DR. This ratio was maintained for activity conducted rather than the produced number of traces which nonetheless roughly conform to the target ratios as shown in Table 4.4. This is an important point, as examination of the trace numbers themselves may suggest statistical information about the collection activities and dataset as a whole which are not necessarily representative of the events actually conducted. The importance of the number of traces generated is not nearly as significant as the number of events performed for each given activity as background system fluctuations and automatic system activity account for significant variation in CPU activity and resource use at any given point in time. These background processes will naturally produce more system manipulations in times of high load which translates to a larger number of traces collected during an event if background activity is high. As such, the number of traces collected is more representative of overall system activity than the nature of the particular process being profiled and should not be used for statistical analysis in favour of the number of events themselves.

As has been alluded to in preceding sections, a key aspect of the ADFA-WD is that it provides traces which have been split into 200 element chunks. This is an extremely important facet of the overall design approach, and is a strong enabler for deployment of VK-based HIDS in the real world. At heart, limiting the maximum length of traces has two distinct advantages. First, it means that the processing burden when training a DE is significantly reduced. With traces of unrestricted lengths, the DE must consider all possible permutations within that trace up to and including the maximum length. If the trace is extremely long, this becomes a high complexity problem which consequently consumes a large amount of processing resources and training time. One of the disadvantages of VK theory is that it produces large amounts of data if the number of DLLs included in the virtual kernel implementation is high. The result of this is that system manipulation traces will consequently be extremely long. Processing time, complexity and load can be controlled, however, by splitting these very long traces into fixed length segments and consequently limiting the complexity which the DE must analyze. The related disadvantage of this technique is that semantic theory gains incremental advantage from processing long, unbroken semantic chains[10]. The act of splitting traces into smaller chunks fractures the higher-order semantic chains resulting in the loss of this information. Hence, it is important to select a maximum trace length which will preserve at least some higher-order semantic information whilst simultaneously reducing the processing burden to facilitate HIDS deployment.

The second advantage of splitting traces into finite lengths is that it allows near real-time evaluation by the trained HIDS. If host-based systems are trained to use only complete traces when making their evaluation then real time applicability is extremely limited. In effect, the HIDS expects to evaluate program behaviour from start to finish rather than as it is occurring. This makes it difficult to provide alerts whilst a hacking attack is underway as the anomalous activity will often be complete by the time the process terminates and the trace is finalised. This is not to say that anomaly-based systems will categorically fail to detect real-time hacking activity if they are trained only on complete traces, but the reduced efficiency which clearly eventuates from requiring full trace evaluation is logically apparent. If HIDS algorithms are optimised for performance against traces of a fixed length, however, then evaluation can occur as soon as the required number of system manipulations has been collected. Noting the required balance between reduced processing time and data content of the fixed length trace, it is likely that the selected length will be sufficiently large to preclude true real-time evaluation but is also likely that it will be short enough to facilitate *near* real-time protection. As such, researchers are encouraged to use the provided fixed length traces in the ADFA-WD when evaluating their algorithms so as to benefit from these distinct advantages. When producing the results outlined in this PhD thesis, a trace length of 200 elements was found to be the optimum balance between the competing priorities of reduced training time, maintenance of high order semantic chains and near real-time intrusion evaluation.

10. Chapter 3, Section 3.1.3.

4.3 Results and Discussion

The ADFA-WD was used to verify the new VK theory with the resulting experiments demonstrating conclusively that the VK approach is suitable for both traditional syntactic approaches and the new semantic method proposed by this PhD thesis. Three experiments were conducted to prove VK theory using three different classes of DE and HIDS methodology. All methods used a VK implementation containing the DLLs specified in Section 4.1.2 to provide the system manipulation traces and were evaluated using traces split into 200 element chunks. Each DE was trained using only normal data, with no anomalous data presented to the DE at this stage. Given this training regime, both DEs were clearly operated in an anomaly-based configuration in keeping with the research focus of this thesis. After training, the DEs were used to evaluate the validation data, roughly 10 times the size of the training data, to calculate FAR. Similarly, the attack data was used to calculate DR. In keeping with the method used in Chapter 3, Section 3.5.3, each trace segment was evaluated in isolation. If the HIDS classified the trace as anomalous, then a detection was recorded for that process as a whole. Multiple detections within a trace or in different chunks of the same process's trace were not counted as multiple detections in order to avoid double counting. DR and FAR were then expressed as a percentage using the formulae outlined in Section 3.5.3 to facilitate comparison and assist with the generation of ROC curves.

The IDS research community has become used to seeing high detection rates over 95% for an FAR under 5% based on performance assessment against the legacy KDD collection[11]. The performance difference when evaluating legacy dataset and modern datasets is profound, as shown conclusively in Chapter 3, Section 3.6, where excellent and almost saturated performance against the KDD 98 collection translates to significantly worse performance against the ADFA-LD. This is a critical point in that it strongly indicates that performance expectations rising from traditional evaluation against legacy datasets can no longer be applied to modern datasets. The ADFA-WD is a modern dataset, and consequently performance of the three HIDS methods used in these experiments should not be judged by the same standard as that expected for the KDD collection. Rather, a more accurate measure of relative merit can be gained by comparing performance between the ADFA-WD and the ADFA-LD which are both modern and contemporary datasets, presenting a significant challenge. Whilst this point is particularly important when assessing the syntactic HMM and semantic ELM, the semantic SVM performed exceptionally well and achieved performance comparable with legacy datasets, strongly indicating the superiority of the semantic approach and this DE in particular.

11. See Table 2.1 in Chapter 2.

The first HIDS method used was an HMM-based syntactic approach. This HIDS used a similar structure to that introduced in Chapter 3 for the ADFA-LD, and was the best performing syntactic method in the Linux environment. Noting this historical performance, the HMM was a logical choice to demonstrate the applicability of syntactic methods to the Windows environment using VK theory. The HMM achieved peak performance of 100% DR for 25.1% FAR. There are several important conclusions to be drawn from these figures. First, the HMM was successful at detecting all attacks including those that used obfuscated payloads to trivially bypass signature-based systems. This is an important achievement; attacks which were previously able to penetrate the host's defences undetected are now readily identified using an anomaly-based VK method. Second, the high FAR is a distinctly unattractive performance characteristic, at least on the surface. Whilst this number is high, it does fit within the performance profile expected for contemporary datasets as introduced with the ADFA-LD and discussed above. The more important aspect of the false alarm rate is that it was unable to be lowered below this point, leading to a step like shape in the ROC curve which is shown in Figure 4.3.

This is an unusual characteristic for an HIDS, and is probably an artefact of the decision methodology, DLL selection for the VK, or the longer phrase lengths required to account for the differences between system manipulations and system calls. This HMM-based HIDS was implemented using the *Viterbi* method [74] to determine the likely transition in hidden states which gave rise to the sequence of tokens observed in the system manipulation traces. It is possible that the high variance in the ADFA-WD does not lend itself well to this approach, and that other HMM decision techniques should be used instead. Additionally, the previously optimum Linux phrase length of 6 tokens [23] was demonstrably inappropriate for the analysis of VK system manipulations. The extra information injected by the inclusion of numerous DLLs meant that a phrase length of 15 was shown to produce the best performance in this experiment. It is likely that the increase in phrase length may have impacted on the effectiveness of *Viterbi* analysis, reinforcing the earlier point suggesting an alternative decision method may be required. Finally, the composition of the VK may be sub-optimal and contributing to the high FAR. As has been mentioned, the VK implementation used for this proof-of-concept experiment was relatively small in order to reduce processing time, and may hence have contributed to the step function shape in the measured FAR. Regardless of this unusual behaviour, the performance of this HIDS is significant and was able to provide protection against attacks to which the host was previously completely exposed, demonstrating that an anomaly-based VK-HIDS is able to protect against zero-day attack in the Windows environment which successfully bypass signature-based protections.

The second HIDS approach used was a semantic ELM, similar to that used in Chapter 3 for the Linux environment. This HIDS performed much better overall than the syntactic approach, as would be expected noting the superiority of the semantic theory. The semantic ELM allows a much finer grained approach to the decision problem than the first HIDS, with a more sophisticated decision threshold than the *Viterbi* analysis used in the HMM. The ELM was able to achieve complete detection with a comparable false alarm rate as for the HMM, which suggests that the HMM performance is an accurate representation of the FAR which must be accepted for 100% DR against this challenging dataset. Importantly, however, this implementation of semantic theory produced a continuous ROC curve[12] and achieved good performance of 91.7% DR for only 0.23% FAR. This performance level is eminently usable in the real world, and is competitive even when compared to performance evaluation by syntactic algorithms against legacy datasets in the Linux environment[13].

The third HIDS trialled was a semantic SVM which was implemented using the same approach as detailed for the Linux version discussed in Chapter 3. The performance demonstrated by this HIDS was exceptional, achieving a DR of 99.58% for only 1.78% FAR. These results exceed the performance demonstrated by syntactic algorithms against legacy datasets and are of sufficient quality for use in a production environment. This strong performance is a resounding proof of both VK theory and semantic HIDS theory, underscoring the relevance of anomaly-based detection methods in the modern cyber environment.

Figure 4.3 shows the ROC curves generated by each of the HIDS algorithms. In general terms, better performances is indicated by a greater area underneath the curve [30] for ROC analysis and hence the semantic approach clearly performs better than the syntactic HMM. All algorithms achieved 100% detection, however, regardless of their performance from an ROC perspective. This is particularly significant noting that signature-based systems were unable to detect many payloads once obfuscated using the simple algorithm outlined in Section 4.2.2. The ROC curve of the semantic ELM clearly demonstrates that this particular implementation is viable when compared against HIDS performance in the Linux domain when measured against a modern dataset such as the ADFA-LD, suggesting that VK theory is able to achieve portability between operating systems with minimal impact on performance. Definitive proof of this point would necessitate developing a Linux and Windows dataset of identical difficulty and then examining performance of a single algorithm in both environments, but the results presented here are sufficient to strongly suggest that VK theory is widely applicable irrespective of a given HIDS

12. Figure 4.3.
13. Table 2.1, Chapter 2.

algorithm's particular method. Regardless, the merit of the semantic SVM is beyond question, with the ROC curve for this approach demonstrating exceptional performance suitable for real-world deployment without modification.

The unusual shape of the syntactic ROC curve may well be because of the relatively small VK used for these experiments. Including a large number of DLLs in the VK will always increase the accuracy of the resulting analysis as a larger VK will provide more detail and better information to the DE. When selecting DLLs for inclusion in this VK, processing time and burden were significant factors. As such, the number of DLLs selected for inclusion was kept small to facilitate DE training and semantic phrase extraction, noting the significant time required to compile a semantic phrase dictionary[14]. Whilst the DLLs included in this particular VK implementation are representative of system behaviour and clearly indicate that a VKs-based HIDS is a valuable contribution to computer security, there is nonetheless significant scope to extend the VKs to include other important DLLs. By increasing the size of the VKs, processing overheads will also increase but so too would the richness of the data source and the accuracy of the resulting classification; the impact of increasing the VK would be a one off cost for an HIDS implementation and could be conducted in a offline training environment, consequently minimising the overall impact. It is likely that the increased accuracy would create a more traditional ROC curve for the syntactic HMM whilst also increasing the accuracy of the semantic ELM. Performance gains using this approach may also be possible for the semantic SVM, although the nearly saturated performance of this HIDS may make the increased processing burden unattractive.

14. See Chapter 3.

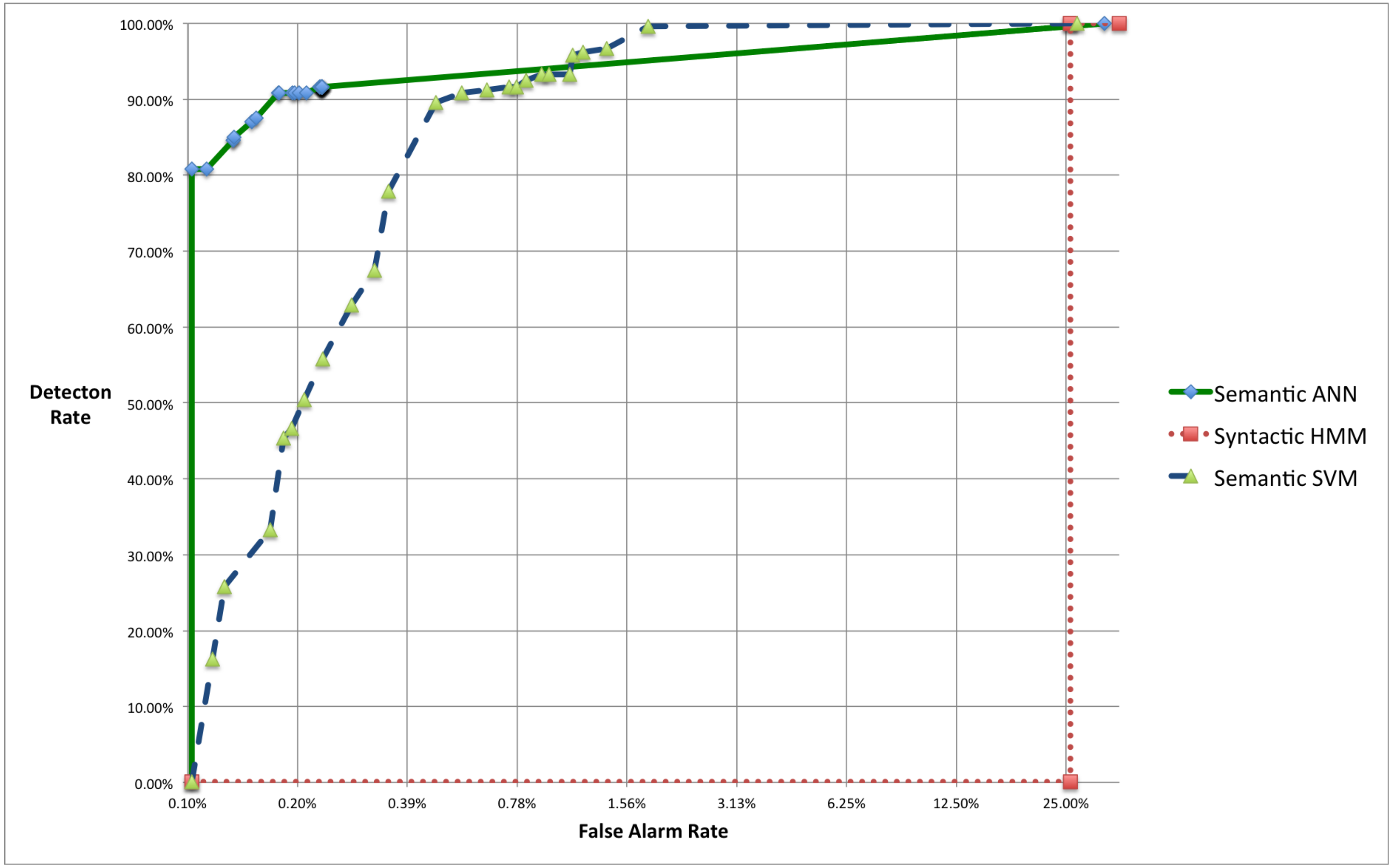

Figure 4.3: ROC Curves for a Semantic-based ANN and Syntactic-based HMM against the ADFA-WD

The division of system manipulation traces into 200 element chunks may also have a part to play in the observed performance of the various algorithms. The advantages of this preprocessing methodology for real-world implementations are clear and have been discussed at length above. Regardless, it is likely that the removal of the full trace from consideration will ultimately result in some loss of accuracy. This is particularly true for any semantic implementations, as the act of trace subdivision will unavoidably fracture high order semantic chains with a corresponding loss of the high dimensionality information which these chains normally provide to a semantic HIDS. Furthermore, HMM *Viterbi* analysis may also be impacted by this process, particularly if the break in the sequence of tokens every 200 elements is interpreted as anomalous. This may account for the high FAR observed for this method and the inability to reduce it by altering phrase length. The benefit of the subdivision process is clearly evident from a real-world perspective, however, and the utility of a semantic-based VK-HIDS is readily apparent from the strong performance of over 99% detection for less than 2% false alarm rate. As such, it seems that the negative impact of trace subdivision is adequately offset by the increased real-world usability, particularly when the inclusion of extra DLLs in the VK will likely increase HIDS accuracy overall and mitigate some of the FAR.

The use of DLL interactions as the basis for system manipulations in VK theory provides a latent possibility for HIDS bypass if DLLs are actively rewritten to contain an attacker's payload. As system manipulations are based on access request to memory locations rather than function names or some other identifier, replacing the contents of memory at that particular address would effectively bypass the HIDS. In a sense, this approach is a type of VK specific stealth attack with potentially significant ramifications. Successful implementation of such an attack, however, would require two difficult to achieve preconditions. First, and perhaps critically, the hacker must already have a foothold in the system to be able to replace the contents of a DLL already loaded in memory. Given the efficacy of a VK-HIDS as demonstrated in Figure 4.3, is unlikely that an attack would be able to gain such a foothold without being detected in doing so. Second, DLLs are not generally writeable in memory meaning that modification is difficult and consequently that this is not a general attack vector. Some DLLs are vulnerable, which has led to the development of DLL injection attacks [49,50], but most are not. Given this, replacing the content of loaded DLLs with an attacker's content is unlikely to be a realisable threat in all but the most unique of situations, assuming that a tuned anomaly-based VK-HIDS is deployed on the host and running. Regardless, the ramifications are potentially significant and hence the potential vulnerability deserves specific mention. Fortunately, as will be shown in the next chapter, semantic HIDS have a natural resilience to stealth attacks such as this, again indicating the superiority of a semantic approach to intrusion detection and mitigating the impact of all forms of stealth attacks significantly.

4.4 Chapter Conclusion

The need for anomaly-based HIDS protection in the Windows OS is clear. Current signature-based security controls are ineffective at detecting zero-day attacks, and even struggle to successfully and reliably detect known attacks if the payloads are obfuscated using simple assembler level routines. When this systemic vulnerability is coupled with the significant market share enjoyed by Windows [124], the magnitude of the threat is made clear. Windows differs from Linux in several key aspects, but the major limiter on HIDS uptake in this OS has been the difficulty of accessing data sources with a comparable granularity to Linux system calls to produce high accuracy anomaly-based classifications; Forrest's work [23, 39–42] clearly establishes system call based analysis as providing the best accuracy for anomaly-based HIDS and the absence of this data source in a usable format from Windows has severely impacted on the effectiveness of current HIDS offerings, which are forced to use registry access or log file analysis as the basis for their classifications. This data source limitation has resulted in an OS which is largely unprotected by the critical security control of an HIDS [13] and a solution of this deficiency would consequently be a substantial contribution to the field.

The new Virtual Kernel theory proposed by this thesis offers just such a solution. Under this theory, the Linux method of system call analysis is abstracted one level to include a selection of the loaded DLLs in the Windows environment. System calls are in turn replaced with *system manipulations* which refer to access requests made to specific memory locations within each DLL. In the Windows environment, DLLs are extensively used to provide software developers with a standardised interface in order to facilitate software production and prolong the lifespan of each version of a program. As such, programs rarely interact directly with the kernel and instead extensively use the standardised intermediary offered by the DLL corpus. VK theory allows system call based HIDS techniques to seamlessly shift the scope of their analysis to system manipulations by quantifying the memory access requests using a relative addressing scheme to produce a stream of tokens akin to Linux system calls. A relative referencing scheme is used to ensure that VK analysis is able to function in a dynamic environment where DLLs are loaded in new locations each time they are accessed and active ASLR is used to harden the underlying system architecture.

There is a marked difference between the *ideal* VK and an implementable *real* VK. An ideal virtual kernel would include all DLLs which exist for the Windows OS, ensuring that the maximum number of system manipulations possible is included in the analysis scope and consequently producing the most accurate classifications. This is clearly infeasible, however, due to the constant state of flux of the DLL corpus and the sheer size of the resulting system manipulation traces which would be produced as a result[15]; the processing burden and training time required to produce an HIDS which could effectively use an ideal VK is prohibitive. Fortunately, a real VK is implementable using a subset of the total DLLs available. Careful selection of the member DLLs is a critical enabler for the success of this method, however, and requires appropriate inclusions and exclusions to ensure that HIDS accuracy is not unduly or unexpectedly affected. The VK implementation outlined in Section 4.1.2 is the smallest collection of DLLs which is likely to produce effective classifications, but could readily be extended by including other important DLLs and specific DLLs for third-party programs included in the host's public facing attack surface. The flexibility of DLL selection allows VK theory to produce a highly customisable HIDS solution, and is an attractive facet of this new theory.

Evaluation of the new VK theory required the creation of an appropriate test dataset. Unlike the Linux environment, no publicly available and commonly accepted datasets have previously been created for the Windows OS. This is largely due to the fact that prior to the creation of VK theory, HIDS implementation in Windows had not been effectively achieved for anomaly-based systems. Regardless, the lack of a common benchmark necessitated the careful design of an appropriate dataset to allow for subsequent researchers to evaluate their algorithms whilst simultaneously proving VK theory. The ADFA-WD is a comprehensive dataset providing a wide attack surface which whilst hardly trivial does not require highly specific and consequently poorly generalisable attacks to penetrate. The full range of modern attack types was employed to generate the attack set for the ADFA-WD, including server side attacks, social engineering, web-based attacks, and browser manipulation. The selection of *Windows XP Service Pack 2* as the OS for the target host in the ADFA-WD means that HIDS evaluation must be conducted in the absence of additional security controls such as ASLR and DEP, ensuring that demonstrated performance is not artificially inflated by other aspects of the *defence-in-depth* solution. Whilst this version of the Windows OS is one of the older offerings, the high similarity between Windows versions [125–127] means that results generated using this dataset have a strong relevance to later versions and consequently provide a reasonable lifespan for the dataset as an evaluation entity. The ADFA-WD provides an excellent metric of the contemporary cyber threat, and is challenging dataset for

15. A comparison between the system call traces in Section A.1 of Appendix A and the system manipulation traces in Section A.2 immediately highlights the increased richness, complexity and size of the VK traces, even when only using a minimalist VK implementation.

HIDS assessment providing a comparable level of difficulty to that offered by the ADFA-LD.

The results produced when VK-based HIDS are used to evaluate the ADFA-WD are clear. Both syntactic and semantic IDS methodologies are able to detect attacks which successfully evade signature-based systems, offering significant detection capabilities which were previously unavailable for the Windows OS. Of particular note, the semantic SVM VK-HIDS produced exceptional results, providing 99.58% DR for an FAR of only 1.78%. This performance is sufficient for effective deployment in a real-world setting and categorically demonstrates that VK theory is sound and effective at providing a previously unseen level of protection against zero-day attacks. The syntactic methodology did not perform as well as either semantic approach, however, which was expected given the superiority of the new semantic theory. Notwithstanding the comparative performance difference, the measured results clearly show that VK theory is also applicable to legacy IDS methodology which rely on syntactic patterns in system calls. In effect, the new VK theory has provided a mechanism by which any system call-based HIDS algorithm may be applied seamlessly to the Windows OS. Given the clearly superior performance of semantic approaches in both the Linux and Windows environments, the weight of evidence for this new core HIDS theory is large. As attacks become more and more sophisticated, it is unlikely that the weaker syntactic approaches which drive other HIDS algorithms will be able to offer effective protection without accepting high false alarm rates as a consequence. The results presented in this chapter make it clear that a semantic anomaly-based VK-HIDS provides a completely new level of protection to the most popular operating system worldwide and effectively mitigates many currently available attack vectors.

Chapter 5

A New Method of Conducting Stealth Attacks and their Detection in the Windows Operating System

From an abstract perspective, intrusion detection is fundamentally reliant on the distinction between normal and abnormal behaviour. Whether the methodology employed is signature-based or anomaly-based, the classification decision always comes back to the base concept that attacks are different to normal system activity. If this distinction can be blurred and the difference between the two classes reduced then the ability to clearly and distinctly differentiate between normal and anomalous behaviour is markedly reduced. As has been demonstrated in Chapter 4, obfuscation techniques can be used to bypass signature-based systems relatively easily. Whilst tuning of the obfuscation engine will always be required, particularly if the anti-virus (AV) to be bypassed is a high-end system with current updates, it is unlikely that signature-based recognition will prove effective at combating a sustained attempt to bypass its recognition capabilities. Indeed, any zero-day attack will effectively bypass signature-based recognition capability as an inherent aspect of the novelty of its exploitation approach. Anomaly-based systems have generally been regarded as more resilient in the face of obfuscated payloads, but still suffer from the same theoretical vulnerability as has been demonstrably proven to affect signature-based protections. If the difference between normal and anomalous behaviour is reduced, then the magnitude of the anomaly which the decision engine must detect is consequently smaller as well; this requires a more accurate Host Based Intrusion Detection System (HIDS) to detect, resulting in a consequent increase in the likelihood of a successful penetration, assuming that deployed HIDS are already operating at peak efficiency. It is this principle which the original mimicry attacks of [133] and the more complete stealth attack theory proposed by this thesis seek to exploit.[1]

Mimicry attacks were introduced by [133] which was the first research into evading anomaly-based systems by obfuscating system call patterns. The method advocated by this seminal research suggests that it is possible to create artificial `no-op`

1. The material in this chapter will be submitted to *IEEE Transactions on Computers* for peer review once the Virtual Kernel (VK) theory introduced in Chapter 4 has been published.

system calls which have no actual effect on the executing program due to the particular arguments passed to them. By inserting these artificial `no-op` patterns into shellcode, the authors of [133] suggest that an anomalous system call trace can effectively be altered to resemble normal behaviour, hence reducing the chance of detection by anomaly-based methods. Even if the attack trace cannot be fully modified to completely resemble normal behaviour, any reduction in the difference between the anomaly and the normal system baseline will make the attack more difficult to detect.

The advantage rendered to an attacker by this reduction is particularly significant in the modern cyber environment. The research presented in this thesis in Chapters 3 and 4 has conclusively shown that modern threats are much harder to detect and contemporary anomaly-based HIDS have a much more challenging task than in previous years. As such, increasing the similarity of attacks to normal baseline behaviour by only a small amount may well defeat HIDS which are already operating at peak efficiency, particularly if using legacy syntactic methods.

[133] also makes mention of the fact that system call arguments are rarely considered by HIDS in their evaluation. This means that the replacement of system call arguments with malicious entries may be able to achieve the attacker's aims without requiring further modification of the system call traces. These *replacement attacks* were not investigated in detail in [133], and likely have a limited scope due to the architecture of modern CPUs. Arguments are passed to system calls in one of two ways - either CPU registers are loaded with the required values, or the values are pushed on to the stack prior to triggering the kernel interrupt which invokes the system call process [134]. Given the nature of the exploitation process which generally leverages memory corruption, is unlikely that individual system call arguments could be effectively and reliably modified in isolation to achieve an effective system compromise using the replacement attack methodology. Even though the threat of this type of attack is largely mitigated through the inherent technical controls present in modern CPUs, it is still a latent exploitation vector which may become more significant in the future. Compared with mimicry attacks and other attacks with a controlled system call trace, however, replacement attacks present little realisable threat.

The experimental results presented by [133] were inconclusive, however, demonstrating only partial proof of the authors' theory. Some success was shown in creating low impact payloads which were capable of escaping detection, with the authors successfully modifying an `adduser` payload to circumvent an HIDS which had previously successfully detected the natural form of the payload. They were unsuccessful, however, in obfuscating a high impact command shell payload which is a telling weakness of the mimicry attack process. Furthermore, there remains a lack of publicly available data to independently verify results and to conduct further research into mimicry attacks. Generation of these attacks is a difficult process, often limited by the particular nature of the payload when contrasted with the available system call patterns in the target program. Given this difficulty, there have been several attempts to automatically generate high impact payloads using the mimicry attack method which have been discussed at length in Chapter 2, Section 2.6. The key weakness of all these approaches is that the size of the resultant shellcode is significant, limiting applicability to real-world tasks where payload space is a premium [134, 135]. This is particularly evident in the brute force genetic generation approach used by [137], with the resultant payload arguably too large for effective deployment. The inability of the mimicry attack method to generate high impact payloads allowing full system compromise critically limits the relevance of this attack school to the modern cyber environment.

The research detailed in this chapter introduces a general theory of stealth attacks which has allowed the generation of high impact command shell payloads for two different server programs included as part of the Australian Defence Force Academy Windows Dataset (ADFA-WD). Stealth attack theory defines a superset of all attacks which have a deliberately modified system call trace in an attempt to evade detection by HIDS. Mimicry attacks are a subset of the superset of stealth attacks, and this PhD research has defined a further three classes of stealth attack which are outlined in Sections 5.1.2, 5.1.3 and 5.1.4. Each of these new class of stealth attacks is capable of providing full system compromise through the delivery of high impact command shell payloads to the target host, unlike the limited payloads offered by mimicry attack theory. Furthermore, an addendum to the ADFA-WD, the Australian Defence Force Academy Windows Dataset: Stealth Attacks Addendum (ADFA-WD:SAA), has been made publicly available to allow other researchers to evaluate the resilience of their HIDS algorithms to this new class of potent cyber attack, with details of the new addendum provided in Section 5.2.2. By implementing stealth attack theory in the Windows domain, the relevance of these attacks is significantly increased. As was discussed in Chapter 4, Windows enjoys the majority market share of computing systems deployed worldwide [124] and hence is the most likely target for contemporary hackers. By providing publicly available data to allow Windows HIDS evaluation against this new class of attack, the benefit to global computer security is significantly enhanced particularly when considered in light of the research outlined in Chapter 4 which allows the creation of anomaly-based HIDS for the Windows Operating System (OS) for the first time.

The impact of this new attack methodology is significant. Fundamentally, stealth attack procedures allow bypass of both signature-based AV systems and anomaly-based HIDS implementations. If successfully executed in the real world, the high impact nature of the available payloads means that undetected and extensive target compromise is likely. The impact of this new methodology must not be underestimated. Table 5.2 in Section 5.3 lists 47 of the most advanced commercial AV programs available as of June, 2013 [172]; all were completely bypassed by a malware implementation of a Type III *Chameleon* attack[2], including several AV programs which use advanced heuristic methodologies. Unlike earlier AV bypass techniques[3], the payload in this evaluation was provided in raw, un-obfuscated form; no attempt was made to actively bypass the AV evaluation. Fundamentally, this malware is undetectable using existing technology and hence demonstrably represents a critical threat, unlike the latent but unrealised possibility of mimicry attack methods. Furthermore, this is but one of three new attack types which underscores the significant size of the attack scope offered by the new theory.

The significance of the new stealth attacks is clear. Fortunately, the new semantic and VK theories introduced by this PhD research are able to effectively detect and prevent attacks which use this new attack technology, as shown by the Detection Rates (DRs) of an Extreme Learning Machine (ELM) and Support Vector Machine (SVM) VK-HIDS detailed in Table 5.1. The higher dimensionality of semantic analysis means that the simple shaping of system call traces which effectively bypasses both signature-based and traditional syntactic Intrusion Detection System (IDS) is ineffective at circumventing semantic HIDS detection. As demonstrated conclusively in the results section of this chapter, semantic theory offers excellent protection against all classes of stealth attack. This feature of the new semantic methodology cements its relevance to the field, and underscores the importance of the contribution made by this research. Whilst the stealth attack theory presented here is original work, there is at least a reasonable possibility that similar techniques are used by malicious hackers in furtherance of their nefarious aims. As such, the protection now possible through the use of an anomaly-based semantic VK-HIDS is clearly a critical defence against the escalating cyber threat.

Table 5.1: Summary of Results for evaluation of the ADFA-WD:SAA

Attack Type	Semantic ELM DR	Semantic SVM DR
Type I	80.0%	100.0%
Type II	100.0%	100.0%
Type III - Network	100.0%	95.0%
Type III - Malware	100.0%	100.0%

The rest of this chapter is organised as follows: Section 5.1 presents a mature and proven theory of IDS evasion, including subsections covering core principles

2. Detailed in Section 5.1.4.
3. See Chapter 4, Section 4.2.2 for an example bypass algorithm.

(5.1.1) and the three new classes of attack defined by this research (5.1.2, 5.1.3 and 5.1.4). Attack development and implementation are covered in Section 5.2 and 5.2.1 respectively, with the addendum to the ADFA-WD outlined in Section 5.2.2. Detection performance of the new semantic algorithms against the new stealth attacks is outlined in Section 5.3 with conclusions detailed in Section 5.4.

5.1 A Theory of IDS Evasion

The mimicry attack theory detailed in [133] was the first work of its kind, but fails to provide a general theory of IDS evasion largely due to its inability to obfuscate high impact payloads. This is a critical point, and warrants further examination. A high impact payload can be considered to be any payload which does not require artificially lax security protocols in the host to provide full system compromise. A clear example of a high impact payload is a `reverse shell` payload. This type of payload functions by forcing the compromised target to initiate a TCP/IP connection back to the attacker's IP address with input and output redirected to provide command shell functionality running at the privilege of the compromised process. Once this level of access has been granted to the hacker they merely need to follow a post-exploitation privilege escalation process to gain root access to the target.

In contrast, a low impact payload provides an incremental compromise of the target but the overall success of the hacking attempt is contingent on particular conditions being met by the host's security configurations. The low impact payload used in [138], for example, was an `adduser` payload which merely created an additional authorised user on the target host with known credentials. This is far from an unconditional compromise of the target, however, as further exploitation relied on remote access services being already active on the target in order to allow the hacker to login with the newly added credentials. Clearly, such a payload is trivially defeated if remote access protocols such as SSH are disabled by default or more secure authentication protocols such as digital certificates are used to authenticate remote connections as opposed to simply supplying a username and password. As such, it becomes clear that any theory of stealth attacks must be generalisable to include high impact payloads in order to be truly relevant to the contemporary cyber situation.

The next key aspect of a generalisable theory of IDS evasion is that the evading activities should not increase system activity more than is absolutely necessary. This is a necessary constraint as increased system use is readily detectable through a number of monitoring techniques and tools. Unfortunately, many of the techniques required to avoid HIDS detection unavoidably increase system resource use by at least some margin. The original mimicry attacks, for example, rely on the addition of artificial *no-ops* to alter the appearance of the system call trace itself. These *no-ops* must be created using system calls which are part of the normal baseline of the target program, and hence are not a true `no-op` but merely an instruction which does not directly hinder the payload execution. CPU cycles will thus be expended on processing these spurious instructions with a corresponding system footprint; if the artificial `no-ops` require more complex system calls then resources in addition to CPU cycles may well be consumed in addition, such as socket use, file descriptor manipulation and process spawning. This effect extends to some of the new attacks proposed by this PhD research as well, with Type II *Chimera* attacks in particular having a large system footprint by virtue of the two-stage exploitation process. Whilst at times unavoidable, increased system presence should be minimised where possible in order to maximise the stealthiness of the attack.

Efficient reuse of system resources can significantly mitigate the impact of increased system activity as a result of exploitation, and should be a design aim of any stealth shellcode. Examples of system resources reuse include using established sockets for connect back payloads instead of creating new ones, utilising already loaded memory modules and avoiding creating new file descriptors where possible. Furthermore, the creation of new processes should be limited in scope to avoid dramatically increasing CPU load and hence telegraphing the intrusion in a non-standard manner. In Windows systems, registry modifications should be avoided as existing methodology in Windows HIDS often focusses on monitoring the registry in an attempt to detect root kit installation and other post-exploitation activities [26–28]. In high security systems, modifications to files stored on disk should also be avoided as changes to monitored files will likely be detected, particularly if white listing programs are used to conduct integrity checks by comparing historical file hashes with current ones. This limits some of the scope for stealth shellcode, as not all programs will present system call patterns which can be used to create high impact payloads whilst avoiding the secondary tell-tale signs. Given that most system protection is provided by signature- or anomaly-based systems, however, the benefit to an attacker of obfuscating their payload's system call trace likely outweighs any incidental increase in the secondary effects outlined above. Regardless, implementations of stealth attack theory should seek to avoid unnecessary secondary compromise signs while still effectively masking the attacker's primary activity.

Stealth attack theory is generalisable across all aspects of the attack surface, ranging from traditional server side exploitation through to web-based attack. Irrespective of the particular vulnerability which a hacker has selected to exploit, the target process will create a quantifiable system call or system manipulation trace. Whilst system call traces are easier to manipulate in a standard server focused buffer overflow, other attack vectors can also be modified using the new stealth attack types described in the following sections, albeit with significantly more effort. As stealth attacks rely on the system calls which a program uses as part of its normal baseline, not all programs will be stealthily exploitable in all forms. This does not mitigate the threat completely, but it does elevate the skill level required to effectively implement stealth attacks and does mean that not all programs which are vulnerable to standard exploitation will necessarily be vulnerable to a stealth payload. The ADFA-WD:SAA uses two server programs to provide stealth attack system call traces in an attempt to provide some variety, and to demonstrate the generalisability of the new theory.

Much of this chapter is, by necessity, written from the perspective of the attacker. This is not to suggest that the author endorses malicious black hat hacking activity, but is designed to fully profile the new threat uncovered by this research. Whilst system call traces have been made publicly available to assist other researchers in developing defences against this serious new attack class, working shellcode has not been and will not be made available in order to prevent malicious reuse of the new attack technology. The system call traces themselves provide no assistance to entities seeking to create their own stealth payloads, but it is the fervent hope of the author that their public release will greatly assist the research community in creating stronger defences against the next generation of cyber weapons.

5.1.1 Core Principles

Shellcode space is extremely limited in modern exploitation vectors [134,135]. Basic exploit methodology involves passing a user supplied string to overflow a buffer, simultaneously uploading shellcode in the content of the buffer and overwriting the instruction pointer in the CPU to redirect execution to the shellcode. This imposes a significant limitation on the size of shellcode which can be included in the buffer; often, shellcode must be placed before the instruction pointer overwrite which drastically limits the available space. It is not unusual for shellcode space to be limited to between 150 and 300 bytes [134,135] which makes the inclusion of high impact payloads somewhat challenging. This limitation is further exacerbated as most payloads must be encoded to prevent the use of characters which are filtered out by the target program, generally limiting the usable assembler `opcodes` to those which reside within the subset of ASCII printable characters. These usable `opcodes` must then be used to self-modify the actual payload of the shellcode, which has been encoded for transmission using characters from the allowed set, in order to reconstruct the desired functionality, such as a `reverse shell` payload. As such, the addition of a decoder stub before the encoded payload necessitates the use of precious shellcode space and increases the requirement for small raw shellcode as a result. This fact is a major reason why mimicry shellcode produced by brute force genetic methods [137] is of limited utility in the real world. The fundamental mimicry method of adding extra system calls to pad the shellcode trace [133] means that shellcode size grows rapidly and consequently limits the applicability of this method to those rare exploitation cases where buffer space is large or unlimited. Clearly, obfuscation techniques which require large shellcode space are of limited threat in the modern environment and any attempt to generate stealth payloads must make the reduction of the size of the shellcode a principal concern.

The new stealth attack theory achieves this principal concern by using a fundamentally different approach to system call trace modification than that used by mimicry attack theory. Mimicry attacks start with a desired payload and then modify that payload by increasing its size using padding system calls. Stealth attacks, on the other hand, started by profiling the target program and extracting all system call sequences used in its normal operation. These sequences are then used to construct a completely bespoke payload for that program which creates high impact command shell functionality using only the available system calls in the target.

The whole process is similar to that used in advanced Return-Oriented Programming (ROP) hacking where `gadgets` [163,164,173,174] of assembler instructions are extracted from a binary and used to construct a payload which bypasses Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP), but with the key difference that stealth attack payloads use system calls and system manipulations instead of assembler instructions. Furthermore, the entire stealth payload must conform to a sequence of valid system call patterns in order to bypass system call based HIDS which imposes a much more stringent set of restrictions on the available system calls than in traditional ROP payloads, where any `gadget` may be used at any time. This critical difference for mimicry methodology means that there is little to no increase in shellcode size between a stealth payload and a normal generic payload. Furthermore, increased system resource use is minimised by avoiding unnecessary system calls in the payload trace. As such, the superiority of this method becomes readily apparent when considered against the criteria discussed in the general theory section above.

Successful implementation of high impact payloads under both the stealth and mimicry methodologies is contingent on certain key system calls being available within the target program. Generally speaking, command shell functionality is usually rendered through the invocation of either the `system()` or `exec()` family of system calls in Linux or the `CreateProcessA()` family in Windows. Whilst some scope exists to provide high-level interactive functionality without using these system calls, the most efficient path to a command shell relies on these core elements. Linux programs have little cause to use the `system()` or `exec()` system calls in their standard mode of operation, however, which complicates building high impact payloads for this OS. Windows, on the other hand, uses the `CreateProcessA()` family for many different tasks and hence this system call abounds in programs written for Windows computers, meaning that high impact payloads using stealth technology are relatively easier to build for this OS. It is possible to mitigate the need for these key system calls by using more complicated payloads, but this strategy requires the acceptance of greater payload size and a more convoluted interactive path with the compromised system as a result.

When generating stealth shellcode as discussed above, the Dynamic Linked Library (DLL) intensive nature of Windows systems means that care must be taken to use system manipulations rather than system calls, as would be the case for Linux. Windows programs are reliant on DLL functionality to provide the end-user experience, as discussed in Chapter 4, and hence clearly system manipulations should be selected as the base unit for stealth payloads instead of system calls. A good example of the impact of different DLL versions and structures on stealth payloads is given by consideration of the `socket()` and `WSASocket()` system calls. Whilst functionally similar in that both calls open a raw socket for subsequent use in networking, `WSASocket()` redirects *standard input* and *standard output* by default whilst `socket()` does not, requiring additional piping to be constructed using subsequent system calls. By considering system manipulations, however, the entire process of establishing a bidirectional TCP/IP connection can be captured by using the access request to a specific section of DLL memory which then performs the full establishment process on behalf of the calling program. Stealth shellcode is able to leverage system manipulations in Windows systems to provide significant functionality and flexibility without adversely increasing shellcode size, and the three new attack classes described below use this principle to great effect.

The modified system call and system manipulation traces of a stealth payload have a high resemblance to the normal baseline of the target program but there will still be subtle discontinuities in the artificial trace as an artefact of the generation process. Syntactic systems are unlikely to detect these artefacts unless there are a significant number which lie within the same sliding window typically used by these methods [23, 39–42]. Semantic methods, however, naturally consider higher dimensionality information present in seemingly linear system manipulation traces and consequently have a natural resilience to attacks with artificially modified system manipulation traces. By using this new class of HIDS technology, stealth attacks are able to be defeated as there is simply no way to create a semantic stealth attack without abandoning the malicious purpose of shellcode in the first place; at heart, the high dimensionality of semantic analysis means that stealth attacks can be readily detected as they only obfuscate in the syntactic domain. This is a powerful aspect of the new semantic theory, and a significant mitigation of the threat presented by stealth attack theory.

The generation of usable stealth payloads under the new theory relies on the target program having at least some usable system manipulation sequences which can be extracted and rejoined to form the desired high impact shellcode. This may not always be possible, although it is likely that at least some form of stealth payload can be made for any given program. The diversity present in the Windows OS, however, means that it is likely that system manipulations will allow effective payloads to be constructed in most circumstances; DLLs provide a powerful range of functionality, and this can be exploited to great effect in the new theory. Stealth payloads will never be generalisable, however, and must be created anew for each target program. Furthermore, changes in DLL structure as new versions are released may also necessitate re-generation of the corresponding payload as a system manipulation level payload must also demonstrate fidelity at the system call level. The best stealth performance will occur when the modified payload trace is as close as possible to the normal baseline for the target program, and system manipulation matching should occur not only at the syntactic level but also at the semantic level. Due consideration of these factors will prolong the life of each stealth payload, and these principles have been applied in generating the proof of concept data used in this research in order to present a realistic challenge.

Finally, the actual endpoint environment may also necessitate custom shellcode even if the target program is the same version as that used for previous stealth payloads. This requirement arises because the process of generating obfuscated system manipulation traces is extremely sensitive to any changes in the target system baseline, and consequently very fragile. This is a natural aspect of stealth technology as the entire purpose of launching a stealth attack is to mimic the normal baseline as much as possible; consequently, any changes from endpoint to endpoint will require modification in order to effectively conduct a stealthy attack on the target. Different endpoints within an organisation may well be at different stages in the update cycle, use different DLLs which whilst compatible from a software point of view may have very different system manipulation patterns, and may have different security postures associated with programs installed in common. All of these factors mean that stealth shellcode must be generated for each individual program *and* each individual endpoint in order to be truly effective. Whilst this is an exhaustive task requiring a high level of skill, the results of performing the process correctly offset the investment; as shown by Table 5.2, stealth attacks are able to bypass all currently available commercial signature- and heuristic-based methods and consequently are a significant threat.

5.1.2 Attack Type 1 - The Doppelganger Attack

Figure 5.1: *Doppelganger* Attack Algorithm

```
1: function doppelgangerAttack(targetIP)
2:     Exploit targetIP.
3:     Add new superuser.
4:     Activate remote access service.
5:     Bypass firewall.
6: end function
```

The first new attack type is the *Doppelganger* attack, so named because this type of attack provides malicious functionality by only using normal system functions. The methodology for this attack is simple at heart, and is similar in some ways to the implemented `adduser` payload used in the original mimicry attacks. The algorithm for the *Doppelganger* attack is detailed in Figure 5.1. As can be seen from the steps in this algorithm, a *Doppelganger* attack performs three distinct activities in order to provide full system compromise. First, a new user is added to the target. This user will have the same privileges as the exploited process, but if this process is running with elevated privileges then a second command can be used to elevate the newly created user to share the administrative privileges of the host process. Second, the *Doppelganger* shellcode activates a remote access service to allow the attacker to connect to the target computer using the new credentials added in the first part of the shellcode. Finally, any active firewall software must be bypassed in order to allow external entities to connect to the remote access service. This can be done in several ways, but often is unnecessary as remote access protocols such as `RDP`, `VNC`, `Telnet` or `SSH` are usually allowed access through the firewall by default if they are regularly used by the organisation. If not, there are various technical methods to bypass the firewall and if need be, the now compromised host can be induced to initiate a connection to a server hosted by the attacker using the newly created superuser account.

A *Doppelganger* attack was implemented for inclusion in the ADFA-WD:SAA by attacking the *Icecast V2.0* program used as part of the original attack surface in the mainline of the ADFA-WD. This program was thoroughly profiled by reverse engineering the binary using methods similar to those outlined in [8], with particular attention paid to the use of system calls. From an examination of the machine code, it became apparent that this program uses the `ShellExecuteA` system call to perform key set-up functions immediately after receiving a new connection from a client. This program idiosyncrasy creates a situation rife for exploitation using a *Doppelganger* attack payload as it allows discreet command line instructions to be performed each time a client connects if the execution flow is redirected from the native call to `ShellExecuteA` to the system call included in the shellcode. The rest of the shellcode is then constructed to terminate the connection using the normal sequence of system calls used by the *Icecast* server. Effectively, this discovery allows an attacker to execute a single command for every connection. Implementation of the *Doppelganger* attack algorithm outlined above is achieved by issuing the following commands using this method:

- `net user /add l l`[4]
- `net localgroup administrators l /add`
- `net start telnet`

The `Telnet` server on the ADFA-WD inherited firewall access as a result of the default security posture, but if circumvention was necessary then the following command would create a reverse `Telnet` connection to the attacker's computer:

- `telnet <attacker's IP>`

The system footprint of a *Doppelganger* attack is very low, particularly if a reverse `Telnet` connection is not required for firewall bypass. The methodology required for this class of stealth attack uses only transitory system interactions to create the new user and elevate privileges, and uses inbuilt OS functionality to provide remote access. A system service is started in the form of the `Telnet` server, but no new, distinctive process is spawned as a result of this shellcode. The remote connection will consume some further networking resources as socket reuse is not easily implementable for this particular type of stealth attack. Unless the security posture of the target system is extremely high, it is unlikely that the addition of a new user will create an alert in any registry monitoring tools, at least in the short term. Indeed, when the attacker is finished the post-exploitation process they can remove the created user as the final act prior to terminating the hack. As such, it is clear that the noise level of this stealth attack is very low, and whilst some system resources are inevitably consumed the magnitude of resource use is such to be easily subsumed in background system activity. The use of more intensive remote access services such as graphical `RDP` connections will consume larger amounts of bandwidth than a simple command shell, however, with a corresponding increase in the risk of secondary detection.

This type of stealth attack has several key strengths. First, shellcode size is extremely small with final payloads generally consisting of less than 200 bytes. Most of this size is taken up with the command strings themselves, which could easily be passed elsewhere in the buffer reducing the overall size to under 100 bytes. The inclusion of the command strings as part of the shellcode proper was a deliberate

4. The username and password *"l"* were selected to minimise shellcode size; only a single character is needed for each field.

decision made for the ADFA-WD:SAA as it provides a standardised stack-based detection challenge rather than a more dynamic heap-based overflow. The second key advantage of a *Doppelganger* attack is that most of the exploitation actively consists of the system performing its normal functions, hence making detection extremely difficult. In contrast to traditional exploitation where programs are manipulated to spawn `reverse shell` connections or other highly aberrant behaviours, this type of attack merely changes some system settings in order to use the host in a normal albeit unauthorised mode of operation. Finally, this method of attack provides a built-in persistent back door in the form of the new user credentials. Whilst not as stealthy as some `Remote Access Trojans`, known user credentials for an administrative level account are often sufficient for short-term persistence and negate the need for further system manipulation if the temporal horizon of the hacking activity is limited.

The weaknesses of this type of stealth attack centre on the fact that it is dependent on core OS functionality to provide the hacker with access to the system. Whilst this characteristic of the doppelgnger attack has some attractive aspects, as discussed above, it is also the greatest flaw in this attack methodology. If remote access services are not available then *Doppelganger* attacks simply cannot function. They can still be used as part of an incremental system compromise, however, but they no longer provide a high impact payload and lose much of their potency. Furthermore, if a new user cannot be added due to tightly implemented privilege control then this attack is of limited use. Fortunately, over privileged users are common in the modern cyber environment [13] meaning that this is unlikely. The final major weakness of this type of attack is that it requires the presence of a system call similar to `ShellExecuteA`. Whilst this system call is regularly used in many programs, its functionality can be implemented more safely if programmers use more focussed code rather than giving access to such versatile flexibility as allowed by this particular system call. This weakness is an inherent aspect of all stealth and mimicry attacks, however, and cannot be easily mitigated.

Doppelganger attacks share some commonalities with the payloads implemented by [133] in the original mimicry attacks, but are a complete attack in and of themselves. Whilst both mimicry attacks and *Doppelganger* attacks both add a new user, mimicry attacks cannot be viewed as a stand-alone payload in their current incarnation. Successful use of the mimicry attack payload presented in [133] is contingent on the presence of an already running and connectible remote access service. *Doppelganger* attacks, on the other hand, activate the required remote access service as part of their inherent functionality, hence clearly differentiating themselves as a stand-alone payload in contrast to the highly dependent mimicry attack payload presented in [133]. Both types of attack share similar disadvantages, however, and whilst stealth attacks are much more capable than mimicry attacks, they still require core elements of the target's OS in order to function as designed. This is an unattractive aspect of the methodology, but offset by the low system footprint and significant stealth characteristics of the *Doppelganger* attack.

In summary, the *Doppelganger* attack has an extremely low footprint and provides a high level of stealth as a result. Successful implementation, however, is contingent on the most prescriptive set of OS requirements and characteristics out of all of the new stealth attack types and shares a similar fragility with the original mimicry attacks [133]. This type of attack is implementable in both Linux and Windows, and several system calls are able to provide the key functionality required. Notwithstanding this, finding the correct sequence of system calls in a target program to provide full syntactic payload stealth is a challenging task and not always implementable for all programs.

5.1.3 Attack Type 2 - The Chimera Attack

Figure 5.2: *Chimera* Attack Algorithm

```
1: function CHIMERAATTACK(targetIP)
2:     Exploit targetIP.
3:     Create new process.
4:     Embed access request into new process.
5:     Compromise Chimera process when it connects to the attacker's machine.
6: end function
```

The second subset of the new stealth attack theory is the *Chimera* attack. This type of attack is optimised for firewall bypass, and whilst its algorithm is somewhat convoluted the success of its methodology stands on its own merits. The name of this attack class refers to the fact that the exploit shellcode creates a full system compromise by using parts of multiple processes to conduct the whole-of-system penetration. The algorithm for this class of attack is presented in Figure 5.2.

The key to this type of attack is that the original process which is exploited initially by the hacker creates no more system effect other than to spawn a new process which is then forced to connect back to the attacker. This provides an extremely stealthy method of bypassing both stateful and stateless packet inspection protections as all previous knowledge of the attacker's original connection to the vulnerable program is lost when the new process is created. Once the new process has connected back to the attacker's IP address, there are several options for further exploitation. The simplest is to exploit a secondary vulnerability in the connecting program to perform a tiered hacking process, but more subtle effects are possible as well. A particularly nefarious option offered by this attack type is the possibility of inserting a man in the middle style server into the networking protocols of the target.

This option is discussed further below, and represents a significant and ongoing type of attack suited to the high-end of cyber warfare or corporate espionage.

The *Chimera* attack was implemented for the ADFA-WD:SAA using the *CesarFTP V0.99g* program, which contains a known vulnerability as discussed in Chapter 4. This known vulnerability formed the required exploitation vector for the implementation, but the previous shellcode was replaced with a custom *Chimera* attack. The new shellcode was generated by following a similar process as for that outlined in Section 5.1.2 for the *Doppelganger* attack. First, the server program of the *CesarFTP* suite was analysed at a binary level to extract useful sequences of system calls. Conveniently, the `ShellExecuteA` system call used to great effect in the *Doppelganger* attack implementation was also present in a useful pattern in this program and hence these shellcode development process was simplified somewhat. Had this system call being absent, however, a *Chimera* attack would still have been possible by using the `CreateProcessA` system call which is also present in a usable pattern.

It is important to note that the presence of a system call by itself does not necessarily mean that the program is vulnerable to a stealth attack. Rather, the critical system call must be part of a usable sub-trace where bracketing system calls can either be made into `no-ops` by careful argument selection or preferably used as value adding parts of the eventual shellcode. It is not sufficient to merely create shellcode from a subset of system calls used by target program without paying due regard to the pattern of the system calls in the program's natural baseline. Having verified that the `ShellExecuteA` system call was usable as part of a stealth attack, the arguments for this system call were amended to spawn a new instance of the *Internet Explorer v6.0.2900.2180* program installed by default on the host. The new process was forced to connect to the attacking machine's IP address by specifying a command line argument as part of the system call, allowing secondary exploitation and completing the *Chimera* attack cycle.

The secondary exploitation vector implemented above allows the new process to be compromised in turn, rendering the hacker able to interact with the target by means of the secondary payload. Whilst this is a serious threat, the particular nature of a *Chimera* attack allows for extremely subtle and long-term effects to be created. This characteristic of the *Chimera* attack arises because the `ShellExecuteA` system call can be used to force almost any program with networking capability to connect to a computer of the attacker's choice. Combined with the ability to issue `taskkill` commands, these two characteristics allow the attacker to effectively redirect network traffic to a server of their choice and then shut down the original service connecting to the genuine external server.

The effect of this process is to allow the attacker to insert a monitoring station in any network traffic, allowing man-in-the-middle attacks against the host's traffic. For example, outgoing web traffic can be redirected to a computer under the control of the hacker and then forwarded on to its eventual destination. The act of providing a web proxy, however, allows the hacker to read all traffic prior to forwarding it. This means that identification credentials, confidential information and private email is all accessible in real time to the attacker. Tools such as `sslstrip` [175] can be used to provide encrypted traffic monitoring capabilities as well, effectively laying bare all communications from the host. If the hacker launches an attack like this against a router or firewall computer, all traffic which passes through that choke point will consequently be accessible with potentially critical ramifications for organisational security. Clearly, the scope offered by this aspect of a *Chimera* attack is significant with consequences potentially much more serious than those offered by simple secondary exploitation.

The *Chimera* attack has many strengths to recommend it. Firstly, it is significantly flexible and able to provide a wide range of effects on the target system. Whether this be the provision of a secondary exploitation vector for the eventual creation of command shell level interactivity or the range of subtle effects possible through the creation of man-in-the-middle traffic redirection, the *Chimera* attack is able to significantly and deleteriously effect its victim. By giving the hacker the ability to selectively force installed programs to connect to a specified target, a *Chimera* attack provide a high level of inherent firewall bypass. Unless outgoing connections in general are blocked, the firewall will be unable to differentiate between normal outgoing program requests and those originating from the *Chimera* attack. Furthermore, stateful analysis is also effectively bypassed by a *Chimera* attack payload as the new process has no link to the exploited process, effectively severing any causal link between the attacker's original exploitation and the new outgoing connection. The *Chimera* attack also is somewhat easier to implement than either a *Doppelganger* attack or *Chameleon* attack, as it can use a wider range of critical system calls to implement the required functionality in either Linux or Windows systems.

Perhaps most critically of all, the invention of the *Chimera* attack effectively ends any relevance of per-process IDS methodologies in the modern era. As discussed in earlier chapters, IDS typically either function as a whole-of-system or per-process evaluation tool. Per-process methodologies are easily bypassed by a *Chimera* attack as the creation of the new secondary process completely sidesteps any evaluation of the original process itself. Given that the stealth attack methodology used to derive the *Chimera* attack means that the active exploitation has a high resemblance to the normal program baseline, it is unlikely that the injection of the shellcode will be detected. As all secondary activities occur in the new, unprotected process which has been forced to connect to the attacker's machine, the per-process HIDS has been completely bypassed by this class of attack. Regardless of the other disadvantages of the per-process methodology, the ease of bypass possible using new stealth attack tactics means that research focus should clearly be directed towards whole-of-system solutions to provide some level of mitigation for this significant new threat.

Fortunately from a white hat perspective, there are several weaknesses associated with the *Chimera* attack technology. Most significantly, this type of stealth attack has the largest system footprint of all of the new attack classes. By spawning a new process and initiating network activity, a *Chimera* attack unavoidably creates a significant level of connectivity, consuming networking resources, CPU cycles and potentially displaying tell-tale signs on the user's desktop. All of these factors mean that the secondary footprint of the *Chimera* attack is significant, although the potential system effects possible mean that hackers may well be willing to take this risk.

Secondly, the *Chimera* attack does not necessarily allow the stealthy exploitation of the secondary process. This means that any transitory benefits gained by using the stealth technology for the prime process may be quickly lost if the second process is compromised using traditional payloads and consequently detected by a whole-of-system HIDS. If the secondary process can be well enumerated, however, then creation of a stealth payload for the second stage of exploitation may well be possible. If man-in-the-middle attacks are used in favour of simple secondary exploitation, however, then the ongoing traffic redirection may well create a significant level of activity and constantly raise the risk of detection. Note that web browsers offer a large selection of system calls due to the high level of functionality expected of this type of program in the modern era of active content. The chances of being able to successfully construct a stealth payload for these programs is consequently good, meaning that two stage stealth exploitation may well be possible if a *Chimera* attack is used to gain the initial foothold.

5.1.4 Attack Type 3 - The Chameleon Attack

Figure 5.3: *Chameleon* Attack Algorithm

```
1: function CHAMAELEONATTACK(targetIP)
2:     Exploit targetIP.
3:     Inject Chameleon shellcode.
4:     Receive interactive command shell on pre-designated port.
5: end function
```

The final class of the new stealth attacks is the *Chameleon* attack, which easily presents the greatest threat to cyber security due to the extreme stealthiness and high-level impact of this payload type. The *Chameleon* attack provides an entirely self-contained sequence of machine code which creates an interactive command shell without requiring any system services or additional processes. The algorithm for this attack is as detailed in Figure 5.3.

This type of attack effectively camouflages extremely malicious activity as the normal activity of the exploited program's baseline, hence its name. Firewall bypass is an integral part of this payload as well, with an attacker able to effectively leverage existing firewall permissions granted to the target server program in order to provide unfettered access to the newly created illicit command shell. When a *Chameleon* attack is deployed against a target, the result is critical; the sequence of system calls used in the payload mimics the normal behaviour the server program almost exactly, effectively obfuscating all malicious activity while simultaneously creating a `bind` or `reverse shell` and hence immediately allowing privilege escalation and post-exploitation activities to begin. Furthermore, this type of payload is well-suited to deployment as stand-alone malware with a corresponding high level of risk associated with this variant.

Two different *Chameleon* attack implementations were conducted for the ADFA-WD:SAA. The first of these was a server side attack remote exploit payload. The target program was *Icecast V2.0*, given the strong networking nature of this server program and the corresponding prevalence of useful system call and system manipulation sequences. The binary was re-analysed with a particular focus on the networking system calls used to accept new clients.

The networking process for establishing new connections is simple at the algorithmic level, consisting of the following fundamental steps:

- create socket;
- bind the socket to a port;
- listen for incoming connections on the port;
- accept connections, spawning child processes as required; and
- direct incoming and outgoing information in accordance with the server program's coding.

This sequence is a fundamental part of all server programs, and hence it is highly likely that the enabling system calls and system manipulations required to establish this functionality exist in usable sub-traces. After examination of the binary, it was no surprise that *Icecast* provides the required system manipulation sequences in the process of accepting a normal client connection. Indeed, this program provides a complete sub-trace with the following members:

`WSAStartup`, `WSASocketA`, `bind`, `listen`, `accept`, `CreateProcessA` and `ExitProcess`.

This sequence, executed each time the program starts with the final three system manipulations repeated for each new client, provides all the functionality required to implement a high impact *Chameleon* attack implementation of a `bind shell` payload. The `CreateProcessA` system call in particular is a critical enabler for this functionality, allowing the seamless spawning of a command prompt for subsequent manipulation by the hacker.

The `WSASocketA` call is another key element of this sequence. This system manipulation creates a new socket with the specified characteristics dictated by its arguments, but provides integrated redirection of input and output to this socket without further manipulation or piping. In contrast, the `socket` system call merely creates a socket without establishing any input or output redirection. This characteristic of the socket call prevented the implementation of a *Chameleon* attack payload for the *CesarFTP* program, as this server never uses the `WSASocketA` and does not establish input and output piping. As such, there is no way of creating interactive command shell functionality for this program without diverting from the normal baseline, hence precluding a *Chameleon* attack against this server. This is a critical illustration of the requirement to consider system manipulation sub-traces rather than discrete system calls when creating stealth payloads, as the required system calls for input redirection exist in *CesarFTP*. Unfortunately, they are not accessed in sequences suitable for inclusion as part of a remote shell and consequently prevent the deployment of this payload type against this program.

The second *Chameleon* attack implementation conducted for this research involved the development of a standalone malware variant. The self-contained nature of a *Chameleon* attack is well-suited to an implementation in malware form, noting that this stealth attack methodology requires no system service or secondary process to support its functionality. An example of this variant has been included as part of the ADFA-WD:SAA, as discussed further in Section 5.2.2. Once working *Chameleon* shellcode has been developed for a given program, adaptation for malware deployment simply requires inserting the shellcode into the execution flow of a target executable, creating a back door using a similar process to that described in Chapter 4, Section 4.2.2. More complicated malware implementations are possible, of course, but a back-doored executable is an elegant proof of concept and suitable for the requirements of academic research. The malware variant was implemented in the ADFA-WD:SAA by back-dooring the *Icecast* program itself with the shellcode from the first variant. This executable was then run in order to collect the malicious system called crisis for subsequent valuation by HIDS algorithms. By structuring the malware in this way, full advantage was taken of the provided functionality in *Icecast's* bespoke DLLs which allowed for both smaller and more robust shellcode as a result.

The *Chameleon* attack has several clear strengths. The first of these is the small shellcode size. After encoding to avoid non-transmissible characters, the total size of the *Chameleon* shellcode is only 169 bytes as implemented for *Icecast*. This is extremely small when compared to standard shellcode sizes, with the equivalent Metasploit Framework (MSF) payload taking up 368 bytes when encoded [1]. This small size is a powerful attribute of the methodology, enabling the deployment of this attack class in situations where other payloads may be unusable. The second strength of *Chameleon* attacks is that they require no further support than the target process itself. Unlike the system service required for a *Doppelganger* attack or the secondary process exploitation used in a *Chimera* attack, the *Chameleon* attack is able to provide high impact functionality with no further resources other than itself. This results in a correspondingly smaller system footprint, particularly when log file analysis is likely following a breach. The final strength of a *Chameleon* attack from a black hat perspective is its ability to be weaponised in malware form. Whilst the other two types of stealth attack could theoretically be packaged in this form, neither one provides the streamlined and self-contained robustness of the *Chameleon* attack and consequently their suitability for malware-based deployment is significantly less.

The major weakness of the *Chameleon* attack is that it requires distinct and lengthy patterns of system manipulations in the target program's baseline in order to be effective. Whilst this condition is not impossible to satisfy, as evidenced by the successful implementation presented as part of this research, it is not a general attack option. Furthermore, a high skill level is required to effectively implement a fully matched *Chameleon* attack and the process of writing the shellcode is labour intensive. This limits the likelihood of encountering *Chameleon* attacks in-the-wild, and whilst this is a weakness from the black hat perspective it is a significant threat mitigation from the white hat viewpoint. A second technical weakness in the *Chameleon* attack methodology is that the ability to implement `reverse shell` payloads is significantly rarer than the ability to create a `bind shell` payload. This arises because server programs generally listen for connections and then perform an action, meaning that the system manipulation sub-traces necessary for a `bind shell` are readily available. `Reverse shell` payloads, on the other hand, require a sub-trace which provides the required system manipulations to pro-actively connect externally. Whilst firewall bypass is possible with `bind shells`, their resilience to robust firewall rules is much less than that of a `reverse shell` which has the distinct advantage of usually falling into the *"allow by default"* rule applied to outgoing connections. This is not an insurmountable problem for the payload, but is an unfortunate side-effect of the usual behaviour of remote server targets.

The impact of this attack class is clear. The possibility of HIDS bypass first hypothesised in the original mimicry attacks of [133] has been realised for the first time in the *Chameleon* attack. This stealth attack class is capable of providing high-level functionality through interactive command shell provision whilst completely circumventing system call-based analysis. Unlike the *Doppelganger* attack and *Chimera* attack, the *Chameleon* attack is able to provide this capability to the hacker without reliance on system services or secondary process exploitation. In short, the *Chameleon* attack is an extremely potent cyber weapon with significant ramifications. The high skill level required to implement a *Chameleon* attack coupled with the prescriptive preconditions required for successful deployment limit the scope of the threat somewhat, but the residual risk is significant. Fortunately, semantic analysis provides effective *Chameleon* attack detection in particular, along with a general detection capability against stealth attacks as a whole, as is conclusively demonstrated in Section 5.3.

5.2 Attack Development and Dataset Compilation

Proof of the stealth attack theory outlined in the earlier sections of this chapter required the demonstration that the new theory was capable of generating high impact payloads in multiple scenarios. These high impact payloads have successfully been created for two different programs, namely *Icecast* and *CesarFTP*, and stealth attack theory has consequently been demonstrated as valid. Mimicry attacks [133] have not yet successfully created high impact payloads, resulting in only a partial proof of this approach to HIDS evasion. The *Chameleon* attack is the most telling evidence of the validity of the stealth attack theory, as it provides a completely stand-alone payload which provides high impact interactive command shell functionality. The *Doppelganger* attack uses a similar approach to the original mimicry attacks [133] in that it leverages the ability to create a new user on the target, but differentiates itself as high impact by activating the remote access services required for privilege escalation and post-exploitation itself, rather than simply relying on these services already being active on the target as is the case in [133]. All of the new stealth attack types provide full interactivity with the target, and whilst the process used is sometimes convoluted for Types I and II, the provision of three separate high impact attack classes conclusively proves that the new approach is a significant new threat.

The Windows OS provides a much more permissive environment for stealth attacks than does the Linux OS, mainly as a function of the increased flexibility provided to an attacker by the numerous DLLs in this family of computer. Stealth attacks in a Linux computer require the consideration of system calls and the consequent shaping of system call traces to avoid detection using the methods discussed Section 5.1. If a key system call is missing in a usable sub-trace element in a Linux setting then functionality unique to that system call simply cannot be duplicated;

an example of this is the `execve` system call, which is rarely used in Linux programs but provides extremely powerful options for manipulating the host's OS. The same is true of the Windows OS, but system manipulations provide much greater flexibility when developing shellcode, given that any given system call is generally accessible through a range of different system manipulations. As such, it is more likely that a usable system manipulation will occur in a program's normal baseline as part of a possible stealth attack sub-trace, with the result of an increased probability that high impact payload generation will be possible. This important difference between the two families of computers means that the stealth attack threat is significantly greater in Windows than in Linux.

Notwithstanding the easier implementation task in Windows, stealth attack theory remains equally valid for the Linux environment, as well as other computing environments such as OS X. At heart, stealth attack theory is not purely limited to system manipulations, but can be applied at the system call level as well. The key to effectively creating stealth attacks in other operating systems lies in identifying usable system call sequences in the normal baseline of the target program. An important consideration for implementing stealth attacks in any operating system is the window length used by the HIDS which the stealth arrack is designed to bypass. If, for example, a window length of 6 such as that proposed by Forrest [23] is used, then the matching effort required is significantly less than if a window length of 15 was used. The key to stealth attack implementation in any operating system is thorough target program baselining to identify usable *" tokens"*, whether they be system calls or system manipulations, followed by the grouping of these *tokens* into valid sub-traces which will successfully disrupt syntactic window-based analysis. Finally, the *token* sub-traces must then be assembled to create a useful high impact payload using one of the methodologies outlined in Sections 5.1.2, 5.1.3 or 5.1.4.

The three classes of stealth attack provide a wide coverage of the *Attack Framework* presented in Figure 3.3. *Chimera* attacks provide coverage of the client *Focus*, and malware implementations of *Chameleon* attacks provide a social engineering *Focus*; all three classes are capable of providing server side attack *Focuses* when used in the main mode of operation, resulting in good coverage of this level of the framework. *Accessibility* will generally be reserved for any stealth shellcode, given the potency from a white hat point of view and the general difficulty of creating functioning stealth shellcode. The *Vector* used for all four implementations provided in the ADFA-WD:SAA used a TCP *Vector* to attack the target, but any of the listed *Vectors* could be used given the required vulnerability on the target host. Stealth attacks can be used at any level of *Deployability*, but are likely to be most common in manual exploits given the bespoke nature and individualised content of each particular payload. Finally, as has been discussed in relation to the *Chimera* attack in particular, a full range of effects is possible when using attacks across the spectrum of all three classes. When viewed from this theoretical viewpoint, it is clear that stealth attacks present a broad threat across all aspects of the typical attack surface.

Stealth attacks can be used in many different settings, and without trivialising the difficulty of the development process, are generally limited only by the availability of system manipulation or system call sub-traces which comprise the normal baseline of a target program. The proof of concept shellcode created as part of this research has focused on the TCP *Vector*, directly exploiting server programs to produce the required effects for each class of attack. There is no reason, however, that all *Vectors* listed in the *Attack Framework* cannot be used to deliver the required system effects by stealth attack. Consider, for example, a web server running on a target. Each time a web page is requested, a given set of system manipulations is conducted by the server program in order to deliver the content to the client. As such, there is an ability inherent in this relationship to force the server to perform certain actions and generate certain traffic. When stealth attack principles are used in conjunction with traditional web-based attack vectors, there is no reason that the resulting exploit cannot implement one of the different attack classes outlined above, but be delivered by a web-based *Vector* instead of a network *Vector*. The ramifications of this general nature of stealth attack methodology is that there is significant breadth associated with the threat, necessitating the development of robust and generalised protections.

5.2.1 Theory Implementation

Implementation of the stealth attack theory hypothesised in Section 5.1 is fundamentally contingent on usable system manipulation sub-traces being present in the target program's baseline. High impact payloads generally require the provision of a remote command shell or similar level of interactivity, and this in turn means that networking functionality must be activated by the stealth payload. In order to do this, networking sub-traces must be present in the target program to allow the shellcode to access this aspect of the core OS functionality. The implementations presented in this thesis targeted server programs in order to facilitate the extraction of networking sub-traces, with successful creation of high impact remote access payloads.

Targeting programs which already provide extensive networking functionality has a secondary advantage in the Windows OS, namely that networking establishment tasks have likely already been conducted by the target program when it was first spawned. This principle is clearly illustrated by the sub-trace presented in Section 5.1.4. This sub-trace commences with the `WSAStartup` manipulations, which is a vital element in establishing access to the networking subroutines required for the Windows implementation of the TCP/IP protocol. Standalone server programs typically include this system manipulation or a functionally identical substitute in order to provide the required flexibility to implement their specific network-based tasks. By comparison, non-server programs generally access networking functionality by using built-in routines hosted by the OS itself, such as those provided by the COM interface which underpins much of Windows functionality [126]. As these non-server programs do not have a requirement for custom networking tasks, there is little call for them to take full advantage of the flexibility provided by invoking the `WSAStartup` call; the provision of generic routines is generally adequate for their well-defined requirements. The impact on stealth payloads, however, is significant as the high impact remote functionality provided by this class of shellcode does not fit easily into the standard interface provided by the OS. Consequently, stealth shellcode has a much greater chance of successful implementation if it targets programs which provide a means of instantiating the full networking suite as a stand-alone entity as part of the normal baseline, as is commonly found in the case of server applications.

The networking requirement discussed above underscores the fact that not all programs will be vulnerable to high impact stealth attacks. The new theory is, after all, only able to provide high impact payloads if usable sub-traces are present in the program's baseline. This is an incontrovertible requirement of stealth attack theory, and also extends to the original mimicry attacks of [133]. Indeed, it is possible that high impact mimicry attacks may be implementable for some programs; the absence of usable sub-traces from the original program attacked in [133] is partly responsible for the low impact payload eventually generated in this work. If networking sub-traces are not present, high impact payloads are still possible if there is some other method of passing commands to the stealth shellcode rather than direct real-time interactions as is typically used in a command shell; an example of an alternate interaction mechanism is the encoded HTML command and control vector allegedly used by much of APT-1's malware, as discussed in [12]. An important distinction must be made between full networking capability and partial networking capability; an example of this has been discussed above in relation to the absence of the `WSASocketA` system manipulation from *CesarFTP* which precludes redirection of `standard input` and `standard output` without additional piping. If, however, sufficiently versatile networking system manipulations occur in usable sub-traces, then a significant impediment to high impact payload development is removed.

Even without the networking obstacle, implementing stealth attack theory to produce high impact payloads is still extremely difficult. Each type of attack defined in Section 5.1 was successfully implemented, however, and specific notes on the method of implementation for the ADFA-WD:SAA have been presented in the narrative vignettes in the relevant subsections. There are several methods for actually implementing a given stealth attack methodology; selection of a particular approach is largely stylistic, however, and any development process which result in usable high impact shellcode is a valid solution to the implementation problem. The theoretical implementation process is as follows.

First, a copy of the target program which is identical in version and patch level must be obtained. Several analysis tasks are they necessary to accurately profile the behaviour of this program. The program must be run in all modes of normal operation whilst sampling software captures traces of system manipulations so as to produce a set of usable system manipulation patterns. It is unlikely that the initial profiling of the target will capture complete and usable sub-traces, however, and a detailed reverse engineering analysis of the executable binary itself must accompany the empirical baselining process in order to provide important information on which activities generate which particular system manipulation patterns. It is no use, for example, if a critical system manipulation for the desired payload only occurs in the shut-down sequence of the target program; in this example, the critical system manipulation will effectively be unusable as it could only occur at the end of any given payload effectively negating much of its contribution to the shellcode. Effective extraction of the required system manipulation sub-traces is best achieved by an iterative process of system manipulation trace capture followed by analysis and evaluation of the binary itself until a suitable suite of valid system manipulation patterns has been collected.

After this first critical step has been successfully conducted, the second phase of stealth attack generation involves combining these collected sub-traces into a functioning payload and it is this particular step which presents the greatest challenge to the would-be shellcode author. When normal shellcode is produced by a hacker, a process similar to that outlined in [134,135] is used; fundamentally, most of the shellcode which is active 'in-the-wild' is written as an `Assembler` program before being encoded to remove characters such as `0x00`, `0x0A`, and `0x0D` which cause problems when transmitted to a remote target. This approach is not usable at all, however, if a stealth payload is to be created. In this situation, the hacker must only use the system call and system manipulation patterns which are allowed by the identified sub-traces which were collected in the first part of the payload development process. These system manipulation patterns will rarely if ever allow direct translation of a raw assembler payload, necessitating a high level of coding ability to restrict the payload's instructions to those allowed by the usable sub-traces and to ensure that the allowable instructions are only used in sequences which occur in the program's natural baseline. This is a difficult process, and is not necessarily implementable for all targets.

After implementation, the effectiveness of the new stealth payload can be tested by evaluation against a comparable HIDS to that employed on the target. Furthermore, if a stand-alone malware version of a payload is created, such as that demonstrated for the ADFA-WD:SAA by using the *Chameleon* attack, then AV programs can also be used to evaluate the efficacy of the new shellcode. This was done for the *Chameleon* attack payload generated as part of this thesis, and the resulting malware bypassed all 47 high-end AV systems hosted by [172] at the time of writing. A list of these AV programs is provided as Table 5.2 and the original third-party scan is available at [176][5], and many of these security packages provide heuristic runtime checking as well as simple signature-based analysis of files in storage. Note that the *Chameleon* attack which bypassed all of these existing detection technologies was not obfuscated at all; the raw binary shellcode was simply converted to an executable format without any polymorphic or encryption techniques[6] used to complicate the detection task. The fact that all existing AV software was bypassed by an un-obfuscated version of a new stealth attack payload suggests a chilling impact of the severity of this new technology. In light of this circumvention of existing protection, it is fortunate that a semantic-based VK-HIDS is readily able to detect all classes of the new stealth attack theory, as shown by the excellent results catalogued in Section 5.3 of this chapter.

5.2.2 Extending the ADFA-WD

After implementing each class of stealth attack to prove the validity of the new theory, a collection of system manipulation traces was collected in order to extend the ADFA-WD with a Stealth Attacks Addendum (SAA). This SAA is an important contribution to the field as it provides publicly available stealth attack data for the first time, consequently allowing future researchers to test the resilience of their algorithms to this new type of attack. To generate this SAA, an implementation of each attack class was used to attack the host of the original ADFA-WD, detailed in Chapter 4. The two targeted server programs were *Icecast V2.0* and *CesarFTP V0.99g*. Details of the implementation process have already been presented in Sections 5.1.2, 5.1.3 and 5.1.4, and the result of these specific implementations was used to generate the SAA itself. Importantly, the malware variant of the *Chameleon* attack which has been discussed above was included as a fourth element to the SAA, allowing direct comparison between anomaly-based evaluation techniques and

5. A mirror has been provided at `http://www.cybersecurity.unsw.adfa.edu.au/ADFA%20IDS%20Datasets/` in case the original becomes unavailable.
6. Such as those use in the compilation of the mainline of the ADFA-WD. See Section 4.2.2.

Table 5.2: AV Programs Bypassed by a malware implementation of the *Chameleon* attack

AV Program	Update	AV Program	Update
Agnitum	20130608	Kaspersky	20130609
AhnLab-V3	20130608	Kingsoft	20130506
AntiVir	20130608	Malwarebytes	20130608
Antiy-AVL	20130608	McAfee	20130609
Avast	20130609	McAfee-GW-Edition	20130608
AVG	20130609	Microsoft	20130609
BitDefender	20130609	MicroWorld-eScan	20130609
ByteHero	20130606	NANO-Antivirus	20130608
CAT-QuickHeal	20130607	Norman	20130608
ClamAV	20130609	nProtect	20130608
Commtouch	20130608	Panda	20130608
Comodo	20130608	PCTools	20130521
DrWeb	20130609	Rising	20130607
Emsisoft	20130609	Sophos	20130609
eSafe	20130606	SUPERAntiSpyware	20130608
ESET-NOD32	20130608	Symantec	20130609
F-Prot	20130608	TheHacker	20130608
F-Secure	20130608	TotalDefense	20130607
Fortinet	20130609	TrendMicro	20130609
GData	20130609	TrendMicro-HouseCall	20130609
Ikarus	20130608	VBA32	20130608
Jiangmin	20130608	VIPRE	20130609
K7AntiVirus	20130607	ViRobot	20130608
K7GW	20130607		

signature-based techniques such as those listed in Table 5.2 and successfully bypassed by this particular payload in raw form. As has been mentioned previously, the shellcode for each implemented attack has not been and will not be released publicly; the system manipulation traces provide more than adequate information for HIDS algorithm evaluation and hence release of the shellcode itself represents an unwarranted risk.

The structure of the attacks used to create the ADFA-WD:SAA was based on replacing generic, non-stealth shellcode in an existing exploit skeleton with the various stealth payloads outlined above. Hence, upon successful exploitation of the target server program, execution flow is redirected to the stealth payload which is then executed. This is an important aspect of the SAA, as the only difference for testing purposes between conventional cyber attacks and stealth attacks should be the payload itself. By keeping the mechanics of the overflow stage identical for both conventional and stealth attacks, the only difference presented to anomaly- or signature-based detection systems is the stealth shellcode itself. In this way, the ability of the protection system to detect the new type of attacks is isolated from any skewing of accuracy caused by tell-tale flags and markers in the overflow process itself.

10 attacks against the target host were launched for each of the four implementations of the stealth attacks, with system manipulation traces again collected by the *Procmon* [22] utility. Sampling was started after the server program had been launched to prevent collection of the initiation activity for each process. This is a mild diversion from the sampling methodology used for the main ADFA-WD, where start-up processes were captured for the normal and validation data. For the SAA, only attack data was collected and hence it is more realistic to only sample program activity from just before the exploitation act to just after. In this way, the characteristics which make the payload stealthy are isolated from erroneous masking information and evaluated solely on their own merit. For the malware variant of the *Chameleon* attack, however, a slightly different methodology was used because of the requirements of this particular implementation. Sampling for this attack was started just before launching the malware program itself, hence simulating the likely actions of a user activating the malicious program in the midst of otherwise benign system activity.

The principle of evaluating only the stealth portion of the attack was extended to post exploitation activities. Unlike the main line of the ADFA-WD, system manipulation collection was ceased as soon as the payload had activated. No subsequent data exfiltration, privilege escalation, or similar activity was conducted during active sampling. There are two main reasons for this design decision. First, performance against stealth attacks is a separate metric to core performance analysis using DR and False Alarm Rate (FAR). By their very nature, stealth attacks are designed to avoid detection by standard intrusion detection systems, and post-exploitation activity can at times be noisy. Whilst it is possible to provide tools for use after the principal exploitation is finished which have been designed using stealth technology, such tools are not readily available and would need to be purpose-built for each target. The same theoretical considerations outlined for the initial stealth payload would apply in these cases, but it is clearly inappropriate to claim successful detection of stealth attacks if traditional post-exploitation activity is featured within the system manipulation trace. The second reason is that post-exploitation activity is increasingly focused on specific aims rather than general system compromise, particularly in high-end attack situations [10, 12]. Whilst the general attack sophistication in the main line ADFA-WD is mid-range, hence attaching relevance to generic post-exploitation activities as outlined in Table 4.6, stealth attacks are highly sophisticated. Consequently, post-exploitation activities are unlikely to be readily generalisable at this level and as such there is little benefit to including generic stealthy post-exploitation in the SAA.

The SAA is designed to be used as an extension to the ADFA-WD. Only attack data is provided in the Addendum, and the assumption is that core HIDS performance evaluation will be conducted using the data provided in the main line ADFA-WD. Once a Decision Engine (DE) has been trained using the provided data to evaluate normal attacks, performance against stealth attacks can be easily and quickly evaluated by taking the trained HIDS and using it to evaluate the Addendum. If a known HIDS which has been evaluated against the central data is then used to evaluate the SAA, FAR will not change but important insight into the resilience of the algorithm against the new class of attack will be provided. This approach also allows natural extension of the ADFA-WD if more stealth attack classes are developed, and provides a rapid means of validating historical HIDS algorithms against contemporary threats as they are developed. If a future researcher only wanted to evaluate performance of an algorithm against stealth attacks, they would need to first train the new algorithm against the provided ADFA-WD training data and then evaluate DR against the SAA whilst also calculating FAR against the mainline ADFA-WD validation data. Regardless of the approach selected, the ADFA-WD:SAA is designed as an extension of the central dataset and is not intended for use in a stand-alone format.

5.3 Results and Discussion

HIDS detection capabilities against the new stealth attacks were tested by evaluating the HIDS implementations presented in Chapter 4 against the SAA. Each class of HIDS was trained using the normal data provided in the main line of the ADFA-WD, and FAR was calculated using the validation data. Detection criteria used to evaluate the attacks followed the same method as that outlined in Equation 3.5.3 and used previously in Chapters 3 and 4. As outlined in Section 5.2.2, no post-exploitation activity was conducted after the prime exploitation event when compiling the ADFA-WD:SAA and hence the detection statistics presented here represent successful identification of the hacking payload itself, indicating a credible detection capability against stealth attacks themselves rather than some ancillary mechanism associated with post-exploitation.

Detection results for the two semantic VK-HIDS implementations presented in Chapter 4, Section 4.3, are shown graphically in Figure 5.4. Each algorithm was operated with the decision threshold which provided peak DR against the main ADFA-WD without drastically increasing FAR to the 25% level generally associated with 100% DR in this data set[7]. As is noted in the legend in Figure 5.4, this meant that the semantic SVM was operating with an FAR of 1.78% and the semantic ELM with an FAR of 0.23% when classifying the SAA. Each HIDS was able to provide good detection against all classes of stealth attack, particularly when the inability of the 47 traditional Windows protection systems catalogued in Table 5.2 to detect these attacks is considered. If performance against the entire SAA as a whole is considered, the SVM achieves a mean DR of 97.5% and the ELM achieves a mean DR of 95.0%. These good results clearly indicate that semantic theory coupled with the VK methodology provide a credible detection capability against the full range of currently known stealth attacks whilst simultaneously providing extremely low false alarm rates. As such, semantic VK-HIDS empirically provide credible defence against the new attack technology.

7. Saturated DR was achieved at approximately 25% FAR for all three approaches tested in Chapter 4. See Figure 4.3 for details.

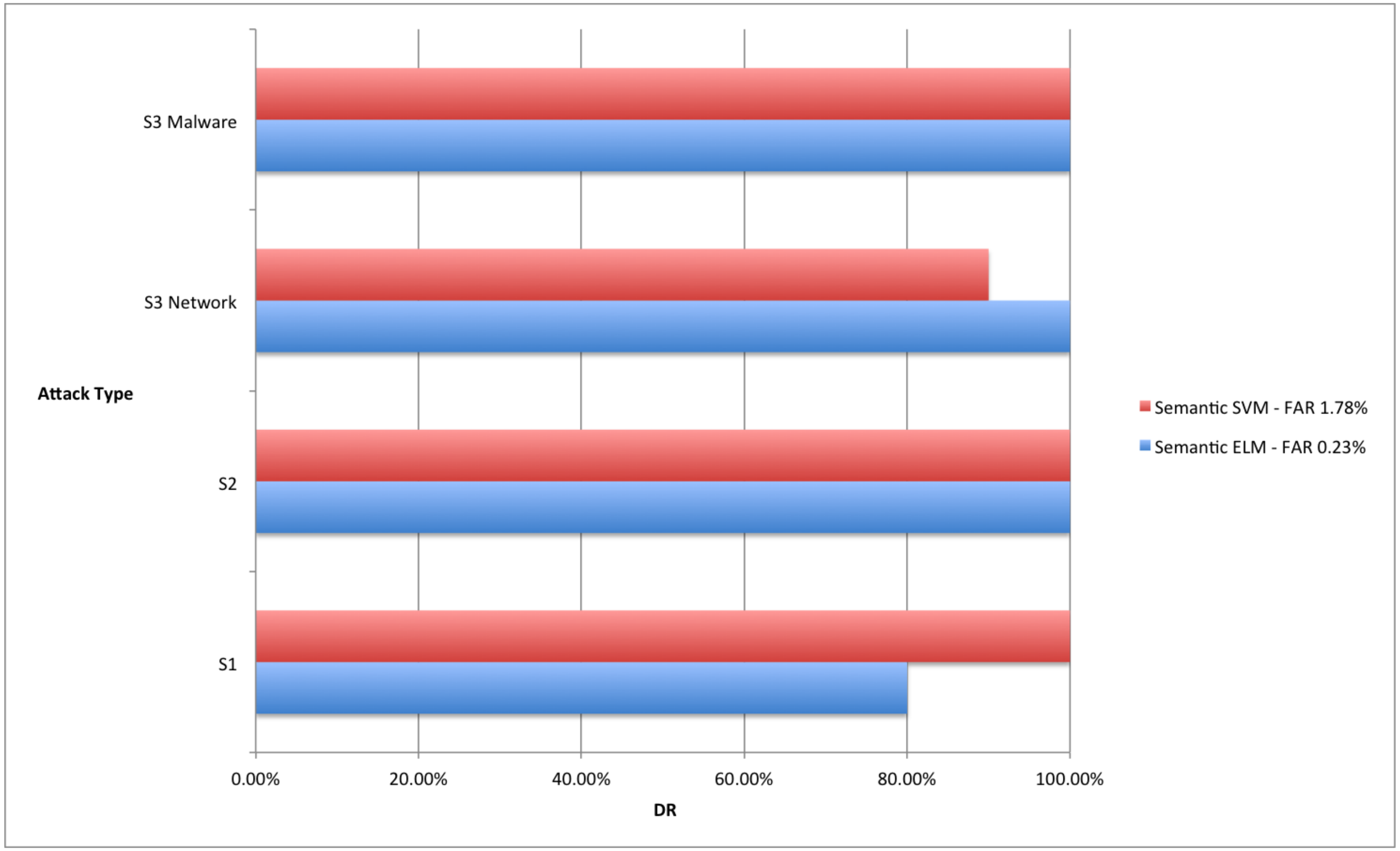

Figure 5.4: Detection Rates for semantic HIDS against the ADFA-WD:SAA

Importantly, however, the results in Figure 5.4 do not indicate perfect detection by either semantic VK-HIDS. Whilst overall an extremely credible new method of defence is presented by semantic VK theory, it is not a perfect solution which reinforces the need for an overall *defence-in-depth* strategy as outlined in [10, 14–18]. The fact that no post-exploitation activity was conducted when compiling the SAA means that it is clearly possible for a stealth attack to avoid even a semantic VK-HIDS occasionally. This is likely due to the specific activity which the target server program was conducting at the time of exploitation masking the payload even more than usual, leading to an increased similarity to the normal baseline and consequently preventing detection. Interestingly, no single attack was able to avoid both HIDSs at the same time, as evidenced by the 100% detection recorded by the alternate HIDS type each time its partner missed an attack. This suggests that the reason for the failed detection lies with the DE itself, rather than either the semantic feature or the VK system manipulation evaluation technique; if the failed detection was a result of either of these new theories, then it is reasonable to expect that there will be some instances where both HIDS implementations fail to detect the same extremely stealthy attack. The absence of this double misclassification strongly points to the DEs themselves as the origin of the classification error, and is a strong endorsement of the contribution of each of the core new theories.

These results conclusively show that per-process HIDS implementations have little place in the modern cyber environment. The major benefit which exponents of this school of thought cite for this approach to the intrusion detection problem is the increased accuracy which is possible when a specific process has a custom-made HIDS attached to it. The exceptional results observed when using a semantic VK-HIDS implementation negate much of this argument, however, and the implications of stealth attacks such as the *Chimera* attack make it clear that evasion of the per-process analysis regime is increasingly possible with potentially disastrous results. Both semantic VK-HIDS algorithms used to produce the results in this section achieved perfect detection against Type II *Chimera* attacks, strongly indicating that the whole-of-system approach used by these implementations is extremely effective at combating this attack type. A per-process HIDS, on the other hand, would likely be unable to detect this class of attack due to the nature of its methodology and the significant impact of the new stealth attacks would likely result in complete system compromise if evasion was successful. As the general performance of HIDS implementations which use the new semantic theory is so superior to traditional syntactic approaches, the classification accuracy long associated with per-process analysis is no longer readily apparent, consequently suggesting that this approach in general is no longer relevant for contemporary protection systems.

A significant limitation of the four payloads implemented in the SAA and evaluated here is that they all use `bind shell` payloads instead of `reverse shell` payloads. A `bind shell` creates a listening port on the target, and provides command shell functionality once that port is connected to by the attacker. The major limitation of this payload type is that rigorous firewall rules may prevent incoming connections on the new port, thus effectively defeating the attack. A `reverse shell` payload, on the other hand, actively initiates an outgoing connection back to the attacker's host, leveraging the fact that most firewalls tend to allow outgoing connections by default. As stealth attacks must use available system manipulation traces in the target program, however, there are significant limitations imposed as a result on the available payload types which can be created. Server programs rarely initiate outgoing connections and hence finding usable system manipulations which can provide reverse shell functionality is problematic. This means that detection rates catalogued against the SAA do not include the important `reverse shell` class of payload, which is a significant class of attack and hence somewhat limits the generalisability of these results.

The dependence on usable system manipulation patterns being present in the target program's normal baseline is the major weakness of stealth attacks, and this limitation is the reason why stealth attacks in general should not be considered an absolute bypass of system manipulation or system call based HIDS. Whilst the potential presented by this methodology is significant, stealth attacks are not always implementable and even when implemented are not necessarily in the optimum form for other aspects of the attack process, such as firewall bypass per the discussion above. Furthermore, gaining a command shell on the victim is seldom an end in and of itself. Without stealth post-exploitation tools, the attacker is left with little option but to increase the noise level of the activity at this stage of the penetration which in turn allows subsequent opportunities for detection. Consequently, gaining a foothold in the system is only ever going to be one part of the scope of detection opportunities and as this part of the process become stealthier, the focus of intrusion detection may well turn to other aspects of the attack cycle as a whole. Regardless of the latent issues associated with maintaining a stealthy posture for all segments of the attack process, it is still important to maximise the chance of detection at each critical point on this time line. The ability to detect the initial exploitation event itself is a powerful aspect of semantic VK-HIDS, and an important contribution to the protection of the system as a whole.

The absence of reverse shell payloads and other more complex high impact options from the SAA does not mean that it is impossible to implement these types of payload using stealth methodologies. Stealth attack theory provides for general solutions, and is limited only by the available system manipulations which occur in usable patterns in sub-traces extracted from the normal baseline the target program. As such, any behaviour which can be mangled to suit the needs of the hacker can be included in the stealth payload for that particular program. An example of a system manipulation sub-trace which may facilitate the development of connect back functionality while still targeting server-based programs in order to take advantage of the ease of extracting networking sub-traces from this class of application is that caused by an automatic update request. This type of network traffic is regularly caused by modern programs which poll a remote server in order to determine if updates are available. If this behaviour has been captured in the normal program baseline, and consequently forms part of the HIDS's training data, then the attacker is provided with usable system manipulation patterns to initiate an outgoing connection, thus providing the required functionality to implement a `reverse shell` payload. The critical factor in this scenario is whether or not the training data includes the critical system manipulation pattern for the outgoing connection; without this collection of system manipulations, outgoing connections are unlikely to be possible under stealth methodologies.

Syntactic anomaly-based HIDS provide some measure of protection against stealth attacks, but are unable to provide anything close to the measure of combined performance demonstrated by both semantic methods profiled here. As syntactic HIDS are able to benefit from the detection capabilities of anomaly-based methodologies, it is reasonable to expect that they would in some cases be able to detect particularly noisy stealth attacks but generally would need to accept a higher false alarm rate as their detection threshold is reduced. The syntactic method trialled in Chapter 4 presented an extremely high false alarm rate of 25.1% which could not be reduced below this level. Compared to the performance of the semantic HIDS implementations, with an FAR more than an order of magnitude less than this level, the superiority of the semantic approach is readily apparent. Given that no increase in FAR in the semantic systems is required to nonetheless provide almost perfect detection of the stealth attack implementations contained in the SAA, the overall merit of this next-generation HIDS technology clearly outweighs the legacy approaches.

The stark difference between 0% detection for signature-based approaches and 95.0% and over for anomaly-based methods when classifying the new stealth attack implementations strongly indicates the need for anomaly centric protection in the modern cyber environment. As is clearly shown by the results in this chapter, signature-based systems are unable to provide effective detection of zero-day attacks whereas anomaly-based approaches provide an almost perfect solution. Given that attack sophistication is only going to increase in the coming years [10,12], it is clearly imperative that effective implementations of anomaly-based HIDS are deployed in the Windows environment as soon as possible.

5.4 Chapter Conclusion

The research detailed in this chapter introduces a new theory of HIDS evasion. This new stealth attack theory uses a complementary approach to the original mimicry attack method presented in [133]. Mimicry attacks aim to produce a payload with high degree of similarity to the normal baseline of the target program by amending a given payload's system call pattern by inserting artificial `no-ops` so as to mimic a valid sub-trace contained in the training data extracted from the normal behaviour of the exploited program, hence evading detection when analysed by a HIDS. Stealth attacks use the converse approach by constructing a payload using valid patterns of system calls or system manipulations found in the target's baseline in a manner similar to that used by ROP exploitation techniques at the `opcode` level [163, 164, 173, 174]. This offers several major advantages, including smaller payload size, and has been soundly proved to be implementable in a way that the mimicry attacks of [133] have not been. Using stealth attack theory, high impact interactive shell functionality is provided to the attacker using a number of different techniques while completely bypassing traditional protection mechanisms. Mimicry attacks, on the other hand, have not yet successfully obfuscated a payload of this impact level, with a limited proof of concept demonstrated by the successful obfuscation of an `adduser` payload.

Three types of stealth attack have been defined by this research, with all types successfully implemented in a high impact payload. Type I is the *Doppelganger* attack, which provides high-level interactivity by utilising legitimate system services, such as `telnet`, `SSH`, or `RDP`. The difference between a *Doppelganger* attack and the `adduser` payload of mimicry attacks [133] is that *Doppelganger* attacks activate the service as part of the inherent payload functionality, rather than relying on the service already being running as in [133]. The second type of attack is the *Chimera* attack, which creates a secondary process such as *Internet Explorer* and forces this new process to connect back to the attacker's IP address. Type II attacks are particularly well-suited to bypassing firewalls and evading per-process HIDS evaluation schemes. Type III attacks provide a completely stand-alone high impact payload. The *Chameleon* attack utilises system manipulation patterns present in the target program to provide remote command shell functionality without invoking either system services or spawning a new process. Furthermore, this type of attack is well suited to deployment as malware based on the self-sufficiency of its attack structure. All three types of stealth attack have been successfully implemented and the system manipulation traces have been provided as an addendum to the ADFA-WD.

Not all programs are vulnerable to stealth attacks, however, as successful implementation of the new methodologies is contingent on usable patterns of system manipulation sub-traces being present in the program's normal baseline. The presence of critical system calls such as `CreateProcessA` by themselves is not sufficient, as anomaly-based HIDS tend to evaluate sequences of system manipulations or system calls, rather than looking at discrete events in isolation. As such, a hacker must apply a similar criterion when creating stealth payloads. In addition, post-exploitation tools must also be crafted with stealth principles in mind if HIDS evasion is to be continued into this phase of the attack. The natural behaviour of the targeted program is hence a critically limiting aspect of the stealth attack methodology, and if the program does not provide usable patterns of system manipulations then ultimately it will not be exploitable using a stealth attack.

The danger presented by stealth attacks is clear. This new attack methodology is capable of creating payloads which completely bypass most if not all existing Windows protections, with a full list provided in Table 5.2. Given that effective anomaly-based protection to the Windows OS has proved elusive until the development of the VK model presented in Chapter 4, many of those protections listed in Table 5.2 only use signature-based methodologies. There are, however, a number of these existing protections which use heuristic runtime-checking algorithms of significant potency which approach a level of anomaly-centric detection capability. Regardless, these more advanced AV products were also bypassed by un-obfuscated stealth payloads. This significant threat is particularly relevant to the modern cyber environment, where hacking activity seems to be on the rise [10, 12] and computer security is increasingly important. The potential for stealth payloads to provide high impact compromises of targeted systems without the risk of detection by existing protection systems is a hugely critical vulnerability of significant scope. Effective mitigation strategies for this new type of attack are somewhat limited at present, and it is imperative that robust defences are developed as soon as possible.

Fortunately, semantic theory has proven to have a demonstrable natural resilience to stealth attacks as shown by the strong detection results presented in Figure 5.4. Indeed, the viability of the VK methodology has provided the Windows OS with a completely unprecedented level of protection through the development of high accuracy anomaly-based HIDS for the first time. When semantic features are combined with the VK theory, much of the threat presented by stealth attacks is mitigated by the resultant HIDS. It is reasonable to expect that this protection would also extend to post-exploitation tools created with stealth methodologies, although this has not been tested by the current body of research. Furthermore, the Linux OS remains largely unexplored from a stealth attack perspective. Many of the conclusions drawn here about the Windows environment can reasonably be directly translated to the Linux OS, noting the underlying principles allowing VK theory to translate system call methodologies seamlessly into system manipulation techniques. It is important, however, that detailed investigation of the feasibility of stealth attack implementation in the Linux OS is conducted to verify or disprove the stealth attack threat for this family of computers. Finally, whilst semantic VK-HIDS are clearly able to offer high fidelity decisions even in the face of the threat posed by stealth attacks, there are undoubtedly many other advanced attack vectors which will be discovered in the future; it is imperative that the research community continually evaluates existing defence technology against the cutting edge of attack techniques in order to provide credible defence for the global cyber community.

Chapter 6
Conclusion

This thesis has made a significant contribution to the field of Intrusion Detection System (IDS) research, introducing three new high impact theories. As a result of this research, a new semantic approach to traditional Host Based Intrusion Detection System (HIDS) design has been created building on the syntactic approach first suggested by Forrest [23]. The new semantic theory outlined in Chapter 3 is a significant advance in technology, allowing for the construction of HIDS implementations which perform up to 10 times better than existing technology when peak performance is used as the ranking metric. As part of this evaluation process, a new contemporary dataset[1] has been publicly released which allows evaluation of any HIDS algorithm against hacking attacks which are representative of the current threat.

The second major contribution of this thesis was the development of a new theory which allowed the use of high fidelity system call like data sources in the Windows Operating System (OS) for the first time. This Virtual Kernel (VK) theory allow direct application of traditional Linux-based HIDS algorithms to the Windows OS without significant performance penalty. Semantic VK-HIDS were shown to perform excellently when both original theories were combined, resulting in the provision of effective zero-day attack protection for this OS for the first time. A publicly available dataset has also been released for this work[2], marking the first time a Windows anomaly-based HIDS evaluation dataset has been made publicly available.

Finally, this thesis produced a new theory of HIDS evasion which is capable of bypassing traditional Windows protections including 47 contemporary anti-virus (AV) and heuristic protection packages [176]. Notwithstanding the potency of this new attack type, semantic VK-HIDS were shown to be able to detect these attacks with excellent accuracy, significantly mitigating the threat posed by this powerful new hacking technology. The Australian Defence Force Academy Windows Dataset:

1. The Australian Defence Force Academy Linux Dataset (ADFA-LD).
2. The Australian Defence Force Academy Windows Dataset (ADFA-WD).

Stealth Attacks Addendum (ADFA-WD:SAA) has been publicly released to allow other researchers to investigate this new threat.

The rest of this chapter is organised as follows: Section 6.1 details the major contributions of this research, with Section 6.2 summarising some of the key limitations of the three new theories. Open questions and suggestions for future research directions are outlined in Section 6.3, and concluding remarks are presented in Section 6.4.

6.1 Contributions of this Work

The major contributions of this research are detailed below:

Creation of a new theory of HIDS analysis allowing superior accuracy. The semantic theory outlined in Chapter 3 is a significant contribution to the field. Forrest's original syntactic analysis of system calls [23] allowed the current generation of anomaly-based HIDS technology to be derived and revolutionised algorithmic design. The semantic data feature developed and tested in Chapter 3 is the successor to this original syntactic approach, and represents the next stage of anomaly-based HIDS evolution.

The performance of various different Decision Engines (DEs) using a semantic feature was compared against syntactic systems in this chapter, and a significant increase in accuracy was demonstrated by the HIDS implementations which were driven by the semantic feature. Indeed, the performance of the new semantic feature against legacy datasets such as the KDD98 collection was so close to the theoretical saturated performance of 100% Detection Rate (DR) and 0% False Alarm Rate (FAR) that the design of a new, contemporary and challenging baseline dataset became necessary to accurately profile the performance of the new method. The ramifications of this new theory are significant, and anomaly-based HIDS designed using this new approach will offer a previously unattainable level of protection to Linux and UNIX-based OSs. When the excellent performance of the new semantic theory is considered along with the inherent zero-day attack capability of anomaly-based systems, the extremely positive effect on endpoint security is clear.

Creation of a new theory allowing high accuracy anomaly-based analysis in Windows. The Windows OS enjoys majority market share of the world's endpoints [124], but protection for this OS has traditionally been limited to signature-based AV offerings. This limitation is in no small part due to the difficulty of finding a high fidelity data source upon which to base an anomaly centric analysis. An appropriate data source was identified by Forrest in the Linux OS [23] as sequences of system calls, but the architecture of Windows is dependent on Dynamic Linked Librarys (DLLs) which precludes linear analysis of kernel system calls in this OS.

The VK theory introduced in Chapter 4 provides a method of creating a comparable data source to Linux system calls by monitoring DLL access requests using relative offsets from the module memory load point. By treating the resultant system manipulations as Linux-style system calls, anomaly-based HIDS algorithms designed for Linux data sources can be applied seamlessly and without loss of accuracy to the Windows OS. This significant contribution allows true anomaly-based protection for this popular OS for the first time. The research detailed in this chapter also demonstrates that a semantic VK-HIDS is able to provide exceptional accuracy when classifying attacks against Windows, indicating that the value of the semantic algorithm introduced in Chapter 3 is independent of OS and provides a general solution to this security problem. As a result of this research, the Windows OS now has a credible zero-day attack protection mechanism.

Design of new publicly available HIDS datasets. Evaluation of any machine learning algorithm is only possible if there are appropriate datasets available to provide training, validation and anomaly data. The HIDS research community has been reliant on the KDD collection [20, 21] for many years, and the relevance of these legacy dataset to the modern era has been soundly questioned in recent times. Part of the contribution of this thesis is the provision of modern datasets for both Linux and Windows to the research community. Both the ADFA-LD and ADFA-WD were designed using current industry best practice and involve a range of exploit vectors and attacks.

These datasets are not simply a collection of arbitrary attacks, but rather represent a carefully formulated simulacrum of contemporary black hat hacking tactics, techniques and procedures. As such, results generated using these datasets naturally should carry significantly more weight than those using arbitrarily constructed or legacy datasets such as KDD98 or UNM [106]. These new dataset are also much more challenging than legacy datasets, with a performance comparison such as that conducted in Chapter 3 clearly indicating the much higher difficulty level of the ADFA-LD in particular. The public release of these datasets is a significant contribution to the community as it allows for algorithm performance to be standardised against a common and relevant benchmark for the first time in some years.

Creation of a new attack theory allowing bypass of existing Windows protection technologies. Much of security research is reactive, particularly in signature-based protections schemes. Notwithstanding the prevalence of reactive research, however, it is critical that new attack methodologies are actively investigated by white hat organisations so as to profile the threat and increase awareness of new developments prior to zero-day deployment by malicious entities; put simply, every zero-day attack discovered by a white hat and consequently published closes a vulnerability which may have been exploited by a malicious entity with devastating consequences. This thesis has made a significant contribution in this area through the creation and exposition of a new theory of stealth attacks.

This new theory, detailed in Chapter 5, allows the creation of shellcode which has been deliberately constructed using sequences of system manipulations taken from a program's normal baseline. In doing so, detection of the resulting shellcode is nearly impossible using traditional means, with shellcode generated using this new theory successfully bypassing 47 state-of-the-art Windows AV programs [176]. This new approach is fundamentally different to the mimicry attacks proposed by [133] which obfuscate existing shellcode by padding with artificial `no-ops`; the new stealth attack theory proposed by this thesis uses a technique similar to Return-Oriented Programming (ROP), which is used to bypass Address Space Layout Randomization (ASLR), but targeted at system manipulations rather than assembly level instructions; in doing so, the new technique creates a stealth payload without the use of any template. This original method allows for a very powerful attack, and by revealing it here and making the corresponding system manipulation traces publicly available, it is hoped that the research community will be able to harden all aspects of the defensive perimeter to resist this new threat.

The proof that semantic VK-HIDS can detect the new stealth attacks. The threat posed by the new stealth attacks is significant, but the final major contribution of this thesis is the demonstration that semantic VK-HIDS are capable of detecting them with high accuracy. This fact significantly mitigates much of the threat, as a defensive countermeasure is consequently shown to be available. The ability of semantic HIDS to detect these new attacks is largely due to the high dimensionality of a semantic feature, which significantly hardens the resulting HIDS against linear spoofing techniques. Syntactic analysis, on the other hand, is much less effective due to the linearity of this data source, and is hence consistently vulnerable to this new threat. The inherent robustness of semantic HIDS against the new attack is another strong point in favour of this new decision methodology, and strongly reinforces the superiority of the new semantic theory of intrusion detection.

6.2 Limitations of the New Theories

The major limitations of the new theories at their current level of development are detailed below:

Derivation of semantic dictionaries is computationally intensive. The process for creating semantic dictionaries from any selected system call or system manipulation trace is intensive and requires the investment of significant computational resources. This is an unavoidable aspect of the new theory as it currently stands, and arises because of the high dimensionality of semantic data. As part of the compilation process, numerous permutations of non-terminating semantic units must be considered and then evaluated for frequency to allow ranking and pruning of low frequency items to occur. As the system call or system manipulation set increases in size, so too does the complexity of the semantic compilation problem.

Fortunately, semantic dictionary compilation can be conducted in an off-line format and once compiled, the semantic dictionaries impose no additional training burden on HIDS DEs than is the case for a syntactic feature. Indeed, there is no need for dictionary compilation to occur on the endpoint which has provided the training data and this allows for semantic processing to occur in a centralised manner on a dedicated high capability computer, with processed dictionaries then distributed to the client end nodes. Whilst this approach requires a dedicated semantic processing unit, it does effectively mitigate the computational impact on client endpoints and consequently allows for real-world deployment of semantic HIDS.

Extraction of semantic phrases requires much more time than syntactic extractions. The process of creating a semantic baseline for a target endpoint is not only computationally intensive but also temporally lengthy. As noted above, however, semantic compilation can be conducted off-line which also mitigates the time required to conduct the intensive semantic derivation. The major additional burden imposed by the time required to perform semantic extraction is that it limits the ability of the resulting HIDS to respond to changes in the baseline, which will naturally occur over time. This is a potentially significant limitation as it means that flexibility of the eventual endpoint protection system is low, which may in turn limit the installation of new software, dynamic endpoint role redefinitions and the addition of new user accounts to the machine.

These limitations are significantly mitigated, however, by two key points. First, the semantic algorithm has excellent robustness as demonstrated by the superior results when evaluating UNM data after training on KDD98 data[3] which indicates that it is unlikely that the baseline will lose fidelity in the mid-term. Second, centralised processing as discussed above will allow for semantic dictionaries to be pre-derived for new software packages and distributed as updates from the central semantic extraction unit at the time of installation. This means that real-time changes to the endpoint's baseline are possible if semantic processing of the new software packages has been previously conducted.

The creation of a centralised semantic processing scheme is an important open question and forms an key future research avenue.

Identification of crucial DLLs for inclusion in VKs is somewhat ambiguous. The VK introduced in Chapter 4 use the smallest number of DLLs possible to create

3. See Figure 3.6, Chapter 3.

its system manipulations. These DLLs were deemed essential due to their core role in providing functionality for the endpoint's configuration, but do not provide a complete mapping of the underlying OS behaviour, as would be the case for Linux system calls. Indeed, the programs installed as part of the ADFA-WD did not use extensive third-party DLLs and relied almost exclusively on the core Windows OS functionality to provide much of their activities. As such, if a particular endpoint required extensive third-party software programs to be installed them it is unlikely that this minimalist VK would be able to capture a high fidelity measurement of this program's normal behaviour. Rather, an expanded VK including the specific third-party DLLs used by this program would need to be used to baseline the system after the software installation.

This fact can be logically extended to suggest that a much larger VK than that used in this research would be required to create a VK-HIDS in a complex real-world environment. This point does not negate the excellent results produced using a VK-HIDS in this thesis, however, but does suggest that future research should investigate DLL selection for VK inclusion as a priority with a view to quantifying the optimum inclusions.

VK methodologies create large quantities of data. The VK approach generates very large amounts of data very quickly. This is an inescapable aspect of the methodology, and the effect is exacerbated when more DLLs are included in the VK. Fundamentally, the addition of an extra DLL to the VK is akin to adding an extra kernel call to a Linux system. Hence, as the number of DLLs becomes large so too does the corresponding system manipulation trace which is now effectively encompassing activity from numerous kernel-like interfaces. This analogy is not perfect, as some DLLs will be used more frequently than others and no single DLL provides the full complexity of a stand-alone kernel, but the comparison is sufficient to underscore the huge amount of data which will be collected by even a modest VK implementation.

Dealing with this large data is a separate problem to the core IDS design focus of this thesis, but it is nonetheless a significant issue necessitating further investigation. As the system manipulation traces grow larger and more numerous, so too does the semantic processing time and effort increase accordingly which compounds the limitations discussed above. As such, any ability to limit the rapid data growth observed as VK size increases would be a significant enhancement of the theory presented in this thesis.

Stealth attack implementation is dependent on available system manipulation patterns. The stealth attack theory outlined in Chapter 5 provides for incredibly powerful attacks to be launched with an extremely low chance of detection when traditional technology is used to provide the defensive perimeter. Generation of these attacks, however, is contingent on strict conditions being met in the target host. When a stealth attack is designed, the hacker is reliant on using available system manipulation sub-traces in the target program's baseline to construct their payload. Functionality beyond that offered by these available sub-traces is simply not possible under this new theory, which means that the impact of the stealth attacks is potentially limited for some programs. This limitation should not be viewed as a mitigation of the threat, however, as stealth attacks are clearly implementable, as shown in Chapter 5, and devastatingly bypass the majority, if not all, existing Windows protection mechanisms [176].

The reliance of stealth attacks on available sub-traces does, however, provide an important research avenue for the future. If the ability of attackers to locate usable system manipulation sub-traces is somehow reduced or blocked completely, so too is the threat of this new hacking technology mitigated to a large extent. As such, it is clear that further research into offensive technology in general and stealth attacks in particular is necessary to fully profile these latent threats and develop strong defences against them.

6.3 Open Questions and Future Work

Whilst this research has produced some significant advances in the body of knowledge, further work is required in order to take full advantage of the new theories as presented in this thesis. The key open questions are summarised below:

- How can training time for semantic algorithms be reduced?
- What is the best way to adapt the semantic theory to provide a high quality decision feature for algorithms other than Extreme Learning Machines (ELMs) and Support Vector Machines (SVMs)?
- Can a semantic feature be developed for the Hidden Markov Model (HMM) class of DE?
- What is the best way to update an established semantic baseline in order to respond to changes in the protected host's configuration?
- Can a semantic baseline be updated without complete recalculation from first principles?
- How can semantic processing be centralised within an organisation's intranet?
- Which DLLs must be included in a VK to ensure full coverage of the attack surface by the resulting semantic VK-HIDS?
- What is the best way to deal with the rapid data growth which occurs when DLLs are added to those under consideration by a minimalist VK, such as that used in Chapter 4?
- Is the performance of VK-HIDS against the existing corpus of stealth attacks the same for all stealth attacks?

- Can the stealth attack generation process be automated in order to profile the vulnerability of programs to this new threat and consequently identify risk at design time, rather than after release of the program?
- Can the stealth principles outlined in Chapter 5 be used to create rootkits?
- If so, can stealth rootkits be detected using semantic methodologies?
- Can the stealth principles outlined in Chapter 5 be used to create stealthy post-exploitation tools?
- If so, can these tools be detected using semantic methodologies?
- Parameters such as complexity, processing time and dataset size are somewhat poorly investigated as they apply to IDS-specific research; as such, how can these parameters be better quantified throughout the research discipline?

Future work will seek to address these open questions, with particular emphasis on profiling the full scope of the stealth attack risk.

6.4 Final Remarks

The current cyber threat faced by society is significant and characterised by extensive variety in attack vector, the nature of the vulnerability, and exploitation technique. No one security control will ever be able to be optimised to combat all possible attacks, meaning that the importance of overlapping security controls as outlined by the *defence-in-depth* principle [19] is a critical facet of an effective security posture in the contemporary environment. Notwithstanding the difficulty of combating the current variety of attacks, effective endpoint protection is a key aspect of any modern *defence-in-depth* implementation and provides some measure of protection against most attack types. Commentary by leading industry entities [10, 12] makes it clear that the developing trends in the nature of the modern cyber threat means that endpoint protection will be critical for the foreseeable future. There remain, of course, several notable attack types which endpoint protection is unable to counter, such as distributed denial of service attacks, traffic manipulation and redirection, and authentication cookie stealing, but many of the high-end core hacking activities such as data theft, persistent system compromise, and corporate espionage are significantly mitigated by the presence of high accuracy anomaly-based endpoint protection.

The importance of endpoint protection underscores the significant contribution of this thesis's research. Not only has this research introduced the most accurate published HIDS algorithm to date, but this algorithm is fundamentally capable of detecting zero-day attacks due to its anomaly-based design methodology. Furthermore, the strength of the semantic algorithm has also been extended to the Windows OS, bringing effective zero-day attack detection capability to this OS and consequently providing significant enhancement to the defensive posture of the most popular OS in the world [124]. The new algorithm has also been demonstrated to provide exceptional resilience to deliberate attempts to circumvent it, which is particularly important given the increasing skill level of black hat hackers worldwide and the increased activity of well funded and resourced organisations such as Advanced Persistent Threats (APTs) [12]. It is also pertinent to note that by developing and then exposing a new and original attack methodology, this thesis allows security researchers worldwide to investigate a critical new threat type and develop a wide range of controls in keeping with the *defence-in-depth* principle. All data used to evaluate the algorithms discussed in the core chapters of this thesis has been made publicly available, which is an important contribution in its own right as it allows HIDS researchers to evaluate their own innovative new work against datasets which represent the current threat, rather than the legacy threat encapsulated in the KDD collection.

In conclusion, it is important that security research be viewed as an ongoing evolutionary process rather than as a question with a discrete and permanent answer. All indicators suggest that the importance of the cyber domain will continue to increase for the foreseeable future, and as such it is clear that the threat posed by malicious hacking is unlikely to dissipate in either the short- or long-term. The techniques and innovations described in this thesis are powerful tools against the current threat, but there is no guarantee that they will retain their potency against subsequent waves of attack innovation. Just as attackers continually search for new zero-day attack options, so to must defensive specialists seek to pro-actively identify and respond to high impact new threats in order to produce mitigating security controls as rapidly as possible, ideally before attackers are able to exploit newly discovered vulnerabilities and techniques. The importance of active offensive research cannot be overstated; purely reactive defensive security development is fast proving to be unable to mitigate high-end hacking attacks, and the consequences of each hacking event is growing along with our increasing dependence on cyber resources. It is thus imperative that researchers investigate both offensive and defensive technology in the years to come, as only by following this approach will the white hat community be able to effectively counter the growing threat.

Appendix A

Examples of Entries in Semantic Dictionaries

A.1 Linux System Call Semantic Dictionary Entry Examples

The following entries have been extracted from the semantic dictionaries used to produce the results in Chapter 3. The numbers in the `KEY` field represent the system call numbers associated with the particular pattern and the `COUNT` field provides the number of occurrences of the `KEY` pattern in the training corpus. The system call numbers are processed as representative tokens of the actual system call, and this notation is irrelevant to the analysis process itself. By way of contrast, the Windows dictionary entries detailed in the following section use system manipulation tokens rather than individual numbers, yet still apply an identical semantic analysis process.

Linux Semantic Dictionary Entries

```
{KEY='3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3', COUNT=110884}

{KEY='3 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54
54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54
54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54
309 3', COUNT=65}

{KEY='180 180 180 240 240 240 240 240 240 240 240 240 240 240 240 240
240 240 240 240 240 240 240 240 240 309', COUNT=561}

{KEY='309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54
309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3
54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54
54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 309 3 54 54 54 54 54
309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3
54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54
54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54
54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54
309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3
54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54
54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54
54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54
309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3
54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309 3 54 54
54 54 54 309 3 54 54 54 54 54 309 3 54 54 54 54 54 309', COUNT=1}
```

A.2 Windows System Manipulation Semantic Dictionary Entry Examples

The following dictionary entries below are examples of semantic units under a VK analysis regime. In these examples, `KEY` uses patterns of system manipulations to identify the unit, with `COUNT` again representing the number of observations of each pattern in the training data. Of note is the interleaved nature of access requests to each member DLL in the VK.

Linux Semantic Dictionary Entries

```
{KEY='ntdll.dll+0x162da kernel32.dll+0x1bb9 kernel32.dll+0x1d6e
kernel32.dll+0xb50b ntdll.dll+0x1cd1b ntdll.dll+0x16071
ntdll.dll+0x162da kernel32.dll+0x1bb9 kernel32.dll+0x1d6e
kernel32.dll+0xb50b ntdll.dll+0x1cd1b ntdll.dll+0x16071
ntdll.dll+0x162da kernel32.dll+0x1bb9 kernel32.dll+0x1d6e
kernel32.dll+0xb50b ntdll.dll+0x16071 ntdll.dll+0x162da
kernel32.dll+0x1bb9 kernel32.dll+0x1d6e kernel32.dll+0xb50b
ntdll.dll+0x16d33 ntdll.dll+0x16f03 ntdll.dll+0x1ce16 ntdll.dll+0x1ccd2
ntdll.dll+0x16071 ntdll.dll+0x162da kernel32.dll+0x1bb9
kernel32.dll+0xace4 user32.dll+0x2645c user32.dll+0x2a116
kernel32.dll+0xb50b ntdll.dll+0x16d33 ntdll.dll+0x16f03
ntdll.dll+0x1ce16 ntdll.dll+0x1ccd2 ntdll.dll+0x16071 ntdll.dll+0x162da
kernel32.dll+0x1bb9 kernel32.dll+0xace4 user32.dll+0x2645c
user32.dll+0x2a116 kernel32.dll+0xb50b ntdll.dll+0x1cd1b
ntdll.dll+0x16071 ntdll.dll+0x162da kernel32.dll+0x1bb9
kernel32.dll+0xace4 user32.dll+0x2645c user32.dll+0x2a116
kernel32.dll+0xb50b ntdll.dll+0x1cd1b ntdll.dll+0x16071
ntdll.dll+0x162da kernel32.dll+0x1bb9 kernel32.dll+0xace4
user32.dll+0x2645c user32.dll+0x2a116 kernel32.dll+0xb50b
```

```
ntdll.dll+0x1cd1b ntdll.dll+0x16071 ntdll.dll+0x162da
kernel32.dll+0x1bb9 kernel32.dll+0xace4 user32.dll+0x2645c
user32.dll+0x2a116 kernel32.dll+0xb50b ntdll.dll+0x1cd1b
ntdll.dll+0x16071 ntdll.dll+0x162da kernel32.dll+0x1bb9
kernel32.dll+0xace4 user32.dll+0x2645c user32.dll+0x2a116
kernel32.dll+0xb50b ntdll.dll+0x1cd1b ntdll.dll+0x16071
ntdll.dll+0x162da kernel32.dll+0x1bb9 kernel32.dll+0xace4
user32.dll+0x2645c user32.dll+0x2a116 kernel32.dll+0xb50b
ntdll.dll+0x1cd1b ntdll.dll+0x16071 ntdll.dll+0x162da
kernel32.dll+0x1bb9 kernel32.dll+0xace4 user32.dll+0x2645c
user32.dll+0x2a116 kernel32.dll+0xb50b ntdll.dll+0x1cd1b
ntdll.dll+0x16071 ntdll.dll+0x162da kernel32.dll+0x1bb9
kernel32.dll+0xace4 user32.dll+0x2645c user32.dll+0x2a116
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b',
COUNT=9943}

{KEY='kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
```

```
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b
kernel32.dll+0xb50b kernel32.dll+0xb50b kernel32.dll+0xb50b',
COUNT=3418}

{KEY='user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
```

```
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5', COUNT=586}

{KEY='user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32.dll+0x89a5 user32.dll+0x89e8 kernel32.dll+0xb50b
user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8
kernel32.dll+0xb50b user32.dll+0x8709 user32.dll+0x87eb
user32 user32.dll+0x8709 user32.dll+0x87eb user32.dll+0x89a5
```

```
user32.dll+0x89e8 kernel32.dll+0xb50b user32.dll+0x8709
user32.dll+0x87eb user32.dll+0x89a5 user32.dll+0x89e8', COUNT=644}
```

Appendix B

Pseudocode

Figure B.1: Semantic Feature Extraction Algorithm

```
function getWords(traces)
    for all traces do
        counter ← 1
        for system calls in trace do
            word =$systemcall+nextcountercalls$
            if word in wordDictionary then
                Increment count of word
            else
                Add word to wordDictionary
            end if
            counter $+=$ 1
        end for
    end for
    return wordDictionary
end function
function genPhrases(word dictionary, length)
    Create new phrase dictionary for phrases of given length
    for all words in word dictionary do
        while current phrase length < length do
            currentPhrase ← currentWord
            for currentWord do
                currentPhrase $+=$ next dictionary word
            end for
        end while
        Add phrase to phraseDictionary
        word list start position++
    end for
    return phraseDictionary
end function
```

Figure B.1: Semantic Feature Extraction Algorithm (cont.)

```
function getPhraseCount(trace)
    featureVector = new array with length=number of dictionaries
    for all Phrase Dictionaries do
        $i \leftarrow$ phrase length for dictionary
        phraseCount $\leftarrow$ 0
        for all Phrases in Dictionary do
            if phrase present in trace then
                phraseCount++
            end if
            featureVector[$i$] $\leftarrow$phrase count
        end for
    end for
    return featureVector
end function
function evaluateSystemCallTrace(newTrace)
    newFeature $\leftarrow$ getPhraseCount(newTrace)
    normalise newFeature
    feature $\rightarrow$ trained decision engine
    deResult $\leftarrow$ decision engine output
    if deResult $>$ global threshold then
        classification $\leftarrow$ anomalous
    else
        classification $\leftarrow$ normal
    end if
    return classification
end function
```

Figure B.2: Shellcode Encoding Algorithm

```
 1: function ENCODESHELLCODE(locationStart,locationStop)
 2:     originalEIP ← EIP
 3:     key ← 0x27182818
 4:     EDX ← Key
 5:     for all memory between locationStart and locationStop do
 6:         XOR dword [current memory], EDX
 7:         EDX++
 8:     end for
 9:     originalEIP → EIP
10: end function
```

Figure B.3: *Doppelganger* Attack Algorithm

```
1: function DOPPELGANGERATTACK(targetIP)
2:     Exploit targetIP.
3:     Add new superuser.
4:     Activate remote access service.
5:     Bypass firewall.
6: end function
```

Figure B.4: *Chimera* Attack Algorithm

```
1: function CHIMERAATTACK(targetIP)
2:     Exploit targetIP.
3:     Create new process.
4:     Embed access request into new process.
5:     Compromise Chimera process when it connects to the attacker’s machine.
6: end function
```

Figure B.5: *Chameleon* Attack Algorithm

```
1: function CHAMAELEONATTACK(targetIP)
2:     Exploit targetIP.
3:     Inject Chameleon shellcode.
4:     Receive interactive command shell on pre-designated port.
5: end function
```

References

[1] Rapid7. Metasploit Penetration Testing Software. Retrieved 24 Apr. 2012, from `http://www.metasploit.com`, 2012.

[2] Ben Grubb. Hacked security firm leaves Aussies vulnerable. Retrieved 23 Jun. 2013, from `http://www.smh.com.au/technology/security/hacked-security-firm-leaves-aussies-vulnerable-20110321-1c2i4.html`, 2011.

[3] Shane Richmond and Christopher Williams. Millions of internet users hit by massive Sony PlayStation data theft. Retrieved 23 Jun. 2013, from `http://www.telegraph.co.uk/technology/news/8475728/Millions-of-internet-users-hit-by-massive-Sony-PlayStation-data\-theft.html`, 2012.

[4] Andrew Fowler and Peter Cronau. HACKED! Retrieved 19 Jun. 2013, from `http://www.abc.net.au/4corners/stories/2013/05/27/3766576.htm`, 2013.

[5] John Naughton. How the Flame virus has changed everything for online security firms. Retrieved 23 Jun. 2013, from `http://www.guardian.co.uk/technology/2012/jun/17/flame-virus-online-security`, 2012.

[6] Benedict Evans. Smartphones Are Eating the World - The spread of mobile computers, in numbers. Retrieved 13 Aug. 2013, from `http://www.technologyreview.com/photoessay/511791/smartphones-are-eating-the-world/`, 2013.

[7] Elias Levy (Aleph One). Smashing The Stack For Fun And Profit. Retrieved 15 Aug. 2013, from `http://www.phrack.org/issues.html?id=14&issue=49`, 1996.

[8] E. Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley, 2011.

[9] Symantec. Symantec Internet Security Threat Report - Trends for 2010. Retrieved 17 Jun. 2013, from `https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf`, 2011.

[10] Symantec. Internet Security Threat Report 2013. Retrieved 17 Mar. 2013, from `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf`, 2013.

[11] Zack Whittaker. Cybercrime costs $338bn to global economy; More lucrative than drugs trade. Retrieved 19 Jun. 2013, from http://www.zdnet.com/blog/btl/cybercrime-costs-338bn-to-global-economy-more-lucrative-than-\drugs-trade/57503, 2011.

[12] Mandiant. APT1: Exposing One of China's Cyber Espionage Units. Retrieved 27 Jun. 2013, from http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, 2013.

[13] Australian Signals Directorate - Cyber Security Operations Centre. Top 35 Mitigation Strategies for Targeted Cyber Intrusions. Retrieved 13 Mar. 2013, from http://www.dsd.gov.au/publications/Top_35_Mitigations.pdf, 2012.

[14] C.L. Smith. Understanding concepts in the defence in depth strategy. In *Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on*, pages 8–16, 2003.

[15] D.K. Hitchins. Secure systems-defence in depth. In *Security and Detection, 1995., European Convention on*, pages 34–39, 1995.

[16] S. Groat, J. Tront, and R. Marchany. Advancing the defense in depth model. In *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pages 285–290, 2012.

[17] Shengjian Liu, Ping Zhang, and Huyuan Sun. Research on Defense In-depth Model of Information Network Confrontation. In *Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on*, pages 267–270, 2012.

[18] E.B. Talbot, D. Frincke, and M. Bishop. Demythifying Cybersecurity. *Security Privacy, IEEE*, 8(3):56–59, 2010.

[19] H.F. Tipton. *Offical (ISC)2® Guide to the CISSP® CBK®*. (ISC)2 Press Series. Auerbach Publishers, Incorporated, 2012.

[20] International Conference on Knowledge Discovery and Data Mining-98. KDD98 Intrusion Detection Dataset. Retrieved 03 Mar. 12, from http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1998data.html, 1998.

[21] International Conference on Knowledge Discovery and Data Mining-99. KDD99 Intrusion Detection Dataset. Retrieved 03 Mar. 12, from http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html, 1999.

[22] Windows Sysinternals. Process Monitor v3.03. Retrieved 01 Mar. 2013, from http://technet.microsoft.com/en-au/sysinternals/bb896645.aspx, 2013.

[23] S. Forrest, S.A. Hofmeyr, A. SoMayaji, and T.A. Longstaff. A sense of self for Unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128, May 1996.

[24] F.A. Bin Hamid Ali and Yee Yong Len. Development of host based intrusion detection system for log files. In *Business, Engineering and Industrial Applications (ISBEIA), 2011 IEEE Symposium on*, pages 281–285, Sep. 2011.

[25] Lin Ying, Zhang Yan, and Ou Yang-jia. The Design and Implementation of Host-Based Intrusion Detection System. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 595–598, Apr. 2010.

[26] M. Topallar, M.O. Depren, E. Anarim, and K. Ciliz. Host-based intrusion detection by monitoring Windows registry accesses. In *Signal Processing and Communications Applications Conference, 2004. Proceedings of the IEEE 12th*, pages 728–731, Apr. 2004.

[27] Dong Hwi Lee, Jae Myung Kim, Kyong-Ho Choi, and K.J. Kim. The Study of Response Model and Mechanism Against Windows Kernel Compromises. In *Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on*, pages 600–608, Aug. 2008.

[28] Xueqin Zhang, Chunhua Gu, and Jiajun Lin. Support Vector Machines for Anomaly Detection. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 1, pages 2594–2598, 2006.

[29] A. Josang and S. O'Hara. The base rate fallacy in belief reasoning. In *Information Fusion, 2010 13th Conference on*, pages 1–8, 2010.

[30] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[31] N. Ye, S.M. Emran, Q. Chen, and S. Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *Computers, IEEE Transactions on*, 51(7):810–820, Jul. 2002.

[32] W. Khreich, E. Granger, R. Sabourin, and A. Miri. Combining Hidden Markov Models for Improved Anomaly Detection. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–6, 2009.

[33] W. Khreich, E. Granger, A. Miri, and R. Sabourin. Boolean Combination of Classifiers in the ROC Space. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 4299–4303, 2010.

[34] M.J.P. Castanho, L.C. Barros, A. Yamakami, and L.L. Vendite. Fuzzy Receiver Operating Characteristic Curve: An Option to Evaluate Diagnostic Tests. *Information Technology in Biomedicine, IEEE Transactions on*, 11(3):244–250, 2007.

[35] Tapas Kanungo and R.M. Haralick. Receiver operating characteristic curves and optimal Bayesian operating points. In *Image Processing, 1995. Proceedings., International Conference on*, volume 3, pages 256–259 vol.3, 1995.

[36] J.M. DeLeo. The receiver operating characteristic function as a tool for uncertainty management in artificial neural network decision-making. In *Uncertainty Modeling and Analysis, 1993. Proceedings., Second International Symposium on*, pages 141–144, 1993.

[37] Vik Tor Goh. *Intrusion detection framework for encrypted networks.* Thesis, Queensland University of Technology, 2010.

[38] Duanyang Zhao, Qingxiang Xu, and Zhilin Feng. Analysis and Design for Intrusion Detection System Based on Data Mining. In *Education Technology and Computer Science, 2010 Second International Workshop on*, volume 2, pages 339–342, 2010.

[39] Stephanie Forrest, Steven A. Hofmeyr, and Anil SoMayaji. Computer immunology. *Commun. ACM*, 40(10):88–96, Oct. 1997.

[40] Steven A. Hofmeyr, Stephanie Forrest, and Anil SoMayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151, 1998.

[41] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145, 1999.

[42] Steven A. Hofmeyr and Stephanie Forrest. Architecture for an Artificial Immune System. *Evolutionary Computation*, 8(4):443–473, 2000.

[43] Jun Wang and Yiming Hu. PROFS-performance-oriented data reorganization for log-structured file system on multi-zone disks. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 285–292, 2001.

[44] D.F. Bacon. File system measurements and their application to the design of efficient operation logging algorithms. In *Reliable Distributed Systems, 1991. Proceedings., Tenth Symposium on*, pages 21–30, Sep. 1991.

[45] Peter D. Hipson. *Mastering Windows 2000 registry.* Sybex, San Francisco, CA, 2000.

[46] Offensive Security Ltd. Offensive Security Certified Expert. Retrieved 14 Nov. 2013, from `http://www.offensive-security.com/information-security-certifications/osce-offensive-security-certified-expert/`, 2012.

[47] K. Borders, Xin Zhao, and A. Prakash. Siren: catching evasive malware. In *Security and Privacy, 2006 IEEE Symposium on*, May 2006.

[48] Stig Andersson, A. Clark, G. Mohay, B. Schatz, and J. Zimmermann. A framework for detecting network-based code injection attacks targeting Windows and UNIX. In *Computer Security Applications Conference, 21st Annual*, pages 10 pp.–58, Dec. 2005.

[49] Moonsu Jang, Hongchul Kim, and Youngtae Yun. Detection of DLL Inserted by Windows Malicious Code. In *Convergence Information Technology, 2007. International Conference on*, pages 1059–1064, Nov. 2007.

[50] Taeho Kwon and Zhendong Su. Automatic Detection of Unsafe Dynamic Component Loadings. *Software Engineering, IEEE Transactions on*, 38(2):293–313, March-April 2010.

[51] Yunmeng Tan, Shengbin Liao, and Cuitao Zhu. Efficient intrusion detection method based on Conditional Random Fields. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 181 –184, Dec. 2011.

[52] K.K. Gupta, B. Nath, and K. Ramamohanarao. Conditional Random Fields for Intrusion Detection. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 1, pages 203 –208, May 2007.

[53] K.K. Gupta, B. Nath, and R. Kotagiri. Layered Approach Using Conditional Random Fields for Intrusion Detection. *Dependable and Secure Computing, IEEE Transactions on*, 7(1):35 –49, Jan. 2010.

[54] Jianping Li and Huiqiang Wang. A Quantification Method for Network Security Situational Awareness Based on Conditional Random Fields. In *Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on*, pages 993 –998, Nov. 2009.

[55] Aiping Lu, Jianping Li, and Lin Yang. A New Method of Data Preprocessing for Network Security Situational Awareness. In *Database Technology and Applications (DBTA), 2010 2nd International Workshop on*, pages 1 –4, Nov. 2010.

[56] Sung-Bae Cho and Hyuk-Jang Park. Efficient anomaly detection by modeling privilege flows using hidden Markov model. *Computers and Security*, 22(1):45–55, 2003.

[57] Anup Ghosh, Christoph Michael, and Michael Schatz. A Real-Time Intrusion Detection System Based on Learning Program Behavior. In *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 93–109. Springer Berlin / Heidelberg, 2000.

[58] Wei Wang, Xiao-Hong Guan, and Xiang-Liang Zhang. Modeling program behaviors by hidden Markov models for intrusion detection. In *Machine Learning*

and Cybernetics, 2004. Proceedings of 2004 International Conference on, volume 5, pages 2830–2835, Aug. 2004.

[59] S. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: a bioinformatics approach. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 24–33, Dec. 2003.

[60] Chi Cheng, Wee Peng Tay, and Guang-Bin Huang. Extreme learning machines for intrusion detection. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, Jun. 2012.

[61] P. Lichodzijewski, A. Nur Zincir-Heywood, and M.I. Heywood. Host-based intrusion detection using self-organizing maps. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1714–1719, 2002.

[62] U. Ahmed and A. Masood. Host based intrusion detection using RBF neural networks. In *Emerging Technologies, 2009. ICET 2009. International Conference on*, pages 48–51, Oct. 2009.

[63] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990, 2004.

[64] R. Perdisci, Guofei Gu, and Wenke Lee. Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 488–498, Dec. 2006.

[65] Miao Wang, Cheng Zhang, and Jingjing Yu. Native API based Windows anomaly intrusion detection method using SVM. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, volume 1, Jun. 2006.

[66] H. Bahrbegi, A.H. Navin, A.A.A. Ahrabi, M.K. Mirnia, and A. Mollanejad. A new system to evaluate GA-based clustering algorithms in Intrusion Detection alert management system. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 115 –120, Dec. 2010.

[67] M. Govindarajan and R. M. Chandrasekaran. Intrusion detection using k-Nearest Neighbor. In *Advanced Computing, 2009. ICAC 2009. First International Conference on*, pages 13–20, 2009.

[68] Te-Shun Chou and K.K. Yen. Fuzzy Belief k-Nearest Neighbors Anomaly Detection of User to Root and Remote to Local Attacks. In *Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC*, pages 207–213, 2007.

[69] Xuedou Yu. Research on Active Defence Technology with Host Intrusion Based on K-Nearest Neighbor Algorithm of Kernel. In *Information Assurance and Security, 2009. IAS '09. Fifth International Conference on*, volume 1, pages 411–414, 2009.

[70] Dae-Ki Kang, D. Fuller, and V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 118 – 125, Jun 2005.

[71] Eugene Charniak. *Statistical language learning.* MIT Press, Cambridge, Mass, 1993.

[72] Jiankun Hu, Xinghuo Yu, D. Qiu, and Hsiao-Hwa Chen. A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection. *Network, IEEE*, 23(1):42–47, Jan.-Feb. 2009.

[73] Debin Gao, M.K. Reiter, and D. Song. Beyond Output Voting: Detecting Compromised Replicas Using HMM-Based Behavioral Distance. *Dependable and Secure Computing, IEEE Transactions on*, 6(2):96–110, Apr.-Jun. 2009.

[74] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, Apr. 1967.

[75] C. Raman and Atul Negi. A Hybrid Method to Intrusion Detection Systems Using HMM. In *Distributed Computing and Internet Technology*, volume 3816 of *Lecture Notes in Computer Science*, pages 389–396. Springer Berlin / Heidelberg, 2005.

[76] Xiuqing Chen, Yongping Zhang, and Jiutao Tang. HMM-based integration of multiple models for intrusion detection. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 2, pages 137–140, 2010.

[77] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. A Multi-Layer Model for Anomaly Intrusion Detection using Program Sequences of System Calls. In *Proc. 11th IEEE Intl. Conf*, pages 531–536, 2003.

[78] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. *Journal of Network and Computer Applications*, 32(6):1219–1228, 2009.

[79] Chuan Feng, Jianfeng Peng, Haiyan Qiao, and Jerzy W. Rozenblit. Alert Fusion for a Computer Host Based Intrusion Detection System. In *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, pages 433–440, Mar. 2007.

[80] A. Siraj, R.B. Vaughn, and S.M. Bridges. Intrusion sensor data fusion in an intelligent intrusion detection system architecture. In *System Sciences, 2004.*

Proceedings of the 37th Annual Hawaii International Conference on, page 10, Jan. 2004.

[81] Yongzhong Li, Rushan Wang, Jing Xu, Ge Yang, and Bo Zhao. Intrusion detection method based on fuzzy hidden markov model. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*, volume 3, pages 470–474, Aug. 2009.

[82] Sang-Woo Moon and Seong-Gon Kong. Block-based neural networks. *Neural Networks, IEEE Transactions on*, 12(2):307–317, Mar. 2001.

[83] Josef Kittler. *Papers presented at the BPRA 4th International Conference on Pattern Recognition.* Springer-Verlag, Berlin ; New York, 1988.

[84] S. Merchant, G.D. Peterson, Sang Ki Park, and S.G. Kong. FPGA Implementation of Evolvable Block-based Neural Networks. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 3129–3136, 2006.

[85] Tomas Hrycej. *Modular learning in neural networks : a modularized approach to neural network classification.* Wiley, New York, N.Y, 1992.

[86] M. Abdel-Azim, A. I. Abdel-Fatah, and M. Awad. Performance analysis of artificial neural network intrusion detection systems. In *Electrical and Electronics Engineering, 2009. ELECO 2009. International Conference on*, pages 385–389, 2009.

[87] Fei Xu, Chengyu Tan, Yi Zheng, and Ming Geng. The Method of Classified Danger Sensed for Windows Process Intrusion Detection. In *Management of e-Commerce and e-Government, 2009. ICMECG '09. International Conference on*, pages 469–472, Sep. 2009.

[88] F. Hosseinpour, K. Abu Bakar, A. Hatami Hardoroudi, and A. Farhang Dareshur. Design of a new distributed model for Intrusion Detection System based on Artificial Immune System. In *Advanced Information Management and Service (IMS), 2010 6th International Conference on*, pages 378 –383, Dec. 2010.

[89] Zhao Linhui, Fang Xin, and Dai Yaping. A novel adaptive intrusion detection approach based on comparison of neural networks and idiotypic networks. In *Nonlinear Dynamics and Synchronization, 2009. INDS '09. 2nd International Workshop on*, pages 203–208, 2009.

[90] Wenke Lee, S.J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132, 1999.

[91] DY. Yeung and Y. Ding. Host-based intrusion detection using dynamic and static behavioural models. *Pattern Recognition*, 36(1):229–243, 2003.

[92] Wun-Hwa Chen, Sheng-Hsun Hsu, and Hwang-Pin Shen. Application of SVM and ANN for intrusion detection. *Computers and Operations Researc*, 32(10):2617–2634, 2005.

[93] Alok Sharma, Arun K. Pujari, and Kuldip K. Paliwal. Intrusion detection using text processing techniques with a kernel based similarity measure. *Computers and Security*, 26(78):488–495, 2007.

[94] Sanjay Rawat, V. Gulati, and Arun Pujari. A Fast Host-Based Intrusion Detection System Using Rough Set Theory. In *Transactions on Rough Sets IV*, volume 3700 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin / Heidelberg, 2005.

[95] M. R. Norouzian and S. Merati. Classifying attacks in a network intrusion detection system based on artificial neural networks. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 868–873, 2011.

[96] Guangjun Song, Jialin Zhang, and Zhenlong Sun. The Research of Dynamic Change Learning Rate Strategy in BP Neural Network and Application in Network Intrusion Detection. In *Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on*, pages 513–513, 2008.

[97] Guang-Bin Huang, Nan-Ying Liang, Hai-Jun Rong, P. Saratchandran, and N. Sundararajan. On-Line Sequential Extreme Learning Machine. In *Computational Intelligence (CI 2005), The IASTED International Conference on*, 2005.

[98] Gideon Creech and Frank Jiang. The application of extreme learning machines to the network intrusion detection problem. *AIP Conference Proceedings*, 1479(1):1506–1511, 2012.

[99] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[100] X.A. Hoang and J. Hu. An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls. *IEEE International Conference on Networks (ICON 2004), Singapore*, 2:470–474, Nov. 2004.

[101] B. Kan-Wing Mak and E. Bocchieri. Direct training of subspace distribution clustering hidden Markov model. *Speech and Audio Processing, IEEE Transactions on*, 9(4):378–387, May 2001.

[102] C Tarnas and R Hughey. Reduced space hidden Markov model training. *Bioinformatics*, 14(5):401–406, 1998.

[103] R. Pieraccini, C.-H. Lee, E. Giachin, and L.R. Rabiner. Complexity reduction in a large vocabulary speech recognizer. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 729–732, Apr. 1991.

[104] Andreas Wespi, Marc Dacier, and Herv Debar. Intrusion Detection Using Variable-Length Audit Trail Patterns. In Herv Debar, Ludovic M, and S. Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 110–129. Springer Berlin / Heidelberg, 2000.

[105] NaiQi Wu, Yanming Qian, and Guiqing Chen. A Novel Approach to Trojan Horse Detection by Process Tracing. In *Networking, Sensing and Control, 2006. ICNSC '06. Proceedings of the 2006 IEEE International Conference on*, pages 721–726, 2006.

[106] Computer Science Department. University of New Mexico Intrusion Detection Dataset. Retrieved 27 Mar. 2012, from `http://www.cs.unm.edu/~immsec/systemcalls.htm`, 2004.

[107] Kapil Kumar Gupta. *Robust and efficient intrusion detection systems.* Thesis, University of Melbourne, 2009.

[108] Dayong Ye. *An agent-based framework for distributed intrusion detections.* Thesis, University of Wollongong, 2009.

[109] C. Arackaparambil, S. Bratus, J. Brody, and A. Shubina. Distributed monitoring of conditional entropy for anomaly detection in streams. In *Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, Apr. 2010.

[110] B. Khosravifar, M. Gomrokchi, and J. Bentahar. A Multi-agent-based Approach to Improve Intrusion Detection Systems False Alarm Ratio by Using Honeypot. In *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, pages 97 –102, May 2009.

[111] A.F. Farhan, D. Zulkhairi, and M.T. Hatim. Mobile agent intrusion detection system for Mobile Ad Hoc Networks: A non-overlapping zone approach. In *Internet, 2008. ICI 2008. 4th IEEE/IFIP International Conference on*, pages 1 –5, Sep. 2008.

[112] Wei Zhang, Shaohua Teng, and Xiufen Fu. Scan attack detection based on distributed cooperative model. In *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on*, pages 743 –748, Apr 2008.

[113] Chi-Chun Lo, Chun-Chieh Huang, and J. Ku. A Cooperative Intrusion Detection System Framework for Cloud Computing Networks. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 280 –284, Sep. 2010.

[114] M. Migliardi and V. Resaz. Distributed Intrusion Detection: Simulation and Evaluation of Two Methodologies. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 42 –48, Jun 2009.

[115] Sho Ohtahara, T. Kamiyama, and Y. Oyama. Anomaly-Based Intrusion Detection System Sharing Normal Behavior Databases among Different Machines. In *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*, volume 1, pages 217 –222, Oct. 2009.

[116] Philippe Owezarski. A Database of Anomalous Traffic for Assessing Profile Based IDS. In *Traffic Monitoring and Analysis*, volume 6003 of *Lecture Notes in Computer Science*, pages 59–72. Springer Berlin / Heidelberg, 2010.

[117] S. Petrovic, G. Alvarez, A. Orfila, and J. Carbo. Labelling Clusters in an Intrusion Detection System Using a Combination of Clustering Evaluation Techniques. In *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 6, page 129b, Jan. 2006.

[118] John McHugh. Testing Intrusion Detection Systems: a critique of the 1998 and 1999 DARPA Intrusion Detection System evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, Nov. 2000.

[119] C. Brown, A. Cowperthwaite, A. Hijazi, and A. SoMayaji. Analysis of the 1999 DARPA/Lincoln Laboratory IDS evaluation data with NetADHICT. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–7, Jul. 2009.

[120] Matthew Mahoney and Philip Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *Recent Advances in Intrusion Detection*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer Berlin / Heidelberg, 2003.

[121] Vegard Engen, Jonathan Vincent, and Keith Phalp. Exploring discrepancies in findings obtained with the KDD Cup '99 data set. *Intelligent Data Analysis*, 15(2):251–276, 2011.

[122] Gideon Creech and Jiankun Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4487–4492, 2013.

[123] G. Creech and J. Hu. A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns. *Computers, IEEE Transactions on*, PP(99):1–1, 2013.

[124] Bogdan Popa. Windows Running on Nearly 62 Percent of Computers Worldwide. Retrieved 27 May 2013, from `http://news.softpedia.com/news/Windows-Running-on-Nearly-62-Percent-of-Computers-Worldwide\-324595.shtml`, 2013.

[125] E. Russinovich and D.A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Developer Series. Microsoft Press, 2005.

[126] J. Templeman and J.P. Mueller. *COM Programming With Microsoft .NET*. Developer Series. Microsoft Press, 2003.

[127] D.S. Platt. *Introducing Microsoft® .NET, Third Edition*. Developer Series. Microsoft Press, 2003.

[128] J.R. Harrald, S.A. Schmitt, and S. Shrestha. The effect of computer virus occurrence and virus threat level on antivirus companies' financial performance. In *Engineering Management Conference, 2004. Proceedings. 2004 IEEE International*, volume 2, pages 780–784 Vol.2, 2004.

[129] P. Bishop, R. Bloomfield, Ilir Gashi, and V. Stankovic. Diversity for Security: A Study with Off-the-Shelf AntiVirus Engines. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 11–19, 2011.

[130] E. Chamorro, Jianchao Han, and M. Beheshti. The Design and Implementation of an Antivirus Software Advising System. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 612–617, 2012.

[131] O. Sukwong, H.S. Kim, and J.C. Hoe. Commercial Antivirus Software Effectiveness: An Empirical Study. *Computer*, 44(3):63–70, 2011.

[132] G. Creech and J. Hu. A Semantic Intrusion Detection System Theory for the Windows OS using DLL Interactions and System Calls as a Data Source. *Manuscript submitted for peer review*, 2013.

[133] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 255–264, 2002.

[134] C. Anley, J. Heasman, F. Lindner, and G. Richarte. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Wiley, 2011.

[135] J. Erickson. *Hacking: The Art of Exploitation*. No Starch Press Series. No Starch Incorporated, 2008.

[136] H.G. Kayacik and A.N. Zincir-Heywood. On the Contribution of Preamble to Information Hiding in Mimicry Attacks. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 1, pages 632–638, May 2007.

[137] H.G. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. Automatically Evading IDS Using GP Authored Attacks. In *Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symposium on*, pages 153–160, Apr. 2007.

[138] S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–62, May 2006.

[139] U.E. Larson, D.K. Nilsson, E. Jonsson, and S. Lindskog. Using system call information to reveal hidden attack manifestations. In *Security and Communication Networks (IWSCN), 2009 Proceedings of the 1st International Workshop on*, pages 1–8, May 2009.

[140] D. Bruschi, L. Cavallaro, and A. Lanzi. An Efficient Technique for Preventing Mimicry and Impossible Paths Execution Attacks. In *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE International*, pages 418–425, 2007.

[141] Offensive Security Ltd. Offensive Security Training and Services. Retrieved 22 May 2013, from `http://www.offensive-security.com`, 2012.

[142] Offensive Security Ltd. Offensive Security Certified Professional. Retrieved 14 Nov. 2013, from `http://www.offensive-security.com/information-security-certifications/oscp-offensive-security-certified-professional/`, 2012.

[143] Ralph Grishman. *Computational linguistics : an introduction.* Cambridge University Press, Cambridge [Cambridgeshire] ; New York, 1986.

[144] Kathleen Dahlgren. *Naive semantics for natural language understanding.* Kluwer Academic Publishers, Boston, 1988.

[145] Donald Davidson, Gilbert Harman, and Council for Philosophical Studies. *Semantics of natural language.* Reidel, Dordrecht, 1972.

[146] Paul Ziff. *Semantic analysis.* Cornell University Press, Ithaca [N.Y.], 1960.

[147] U. Premaratne, A. Nait-Abdallah, J. Samarabandu, and T. Sidhu. A formal model for masquerade detection software based upon natural mimicry. In *Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on*, pages 14–19, Dec. 2010.

[148] H. Gunes Kayacik, A.N. Zincir-Heywood, M.I. Heywood, and S. Burschka. Generating mimicry attacks using genetic programming: A benchmarking study. In *Computational Intelligence in Cyber Security, 2009. CICS '09. IEEE Symposium on*, pages 136–143, Apr. 2009.

[149] N. Srinivasan and V. Vaidehi. Reduction of False Alarm Rate in Detecting Network Anomaly using Mahalanobis Distance and Similarity Measure. In *Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on*, pages 366–371, Feb. 2007.

[150] Python Software Foundation. Python Programming Language Official Website. Retrieved 12 Feb. 2013, from `http://www.python.org`, 2012.

[151] Canonical. Ubuntu Linux. Retrieved 19 May 2012, from `http://www.ubuntu.com`, 2012.

[152] Oracle Corporation. MySQL. Retrieved 19 May 2012, from `http://www.mysql.com`, 2012.

[153] The Apache Software Foundation. The Apache Software Foundation. Retrieved 19 May 2012, from `http://apache.org`, 2012.

[154] The PHP Group. PHP: Hypertext Processor. Retrieved 19 May 2012, from `http://www.php.net`, 2012.

[155] Tiki Software Community Association. TikiWiki: CMS groupware. Retrieved 19 May 2012, from `http://info.tiki.org/Tiki+Wiki+CMS+Groupware`, 2012.

[156] EgiX. Tiki Wiki CMS Groupware Remote PHP Code Injection. Retrieved 19 May 2012, from `http://www.exploit-db.com/exploits/18265/`, 2012.

[157] Offensive Security Ltd. Exploit database. Retrieved 19 May 2012, from `http://www.exploit-db.com/`, 2012.

[158] The (Freeworld) Hacker's Choice. THC-Hydra. Retrieved 19 May 2012, from `http://www.thc.org/thc-hydra/`, 2012.

[159] NumPy Developers. Numerical Python. Retrieved 12 Feb. 2013, from `http://numpy.scipy.org`, 2012.

[160] MathWorks. Matlab Programming Language. Retrieved 12 Feb. 2013, from `http://www.mathworks.com.au/index.html`, 2012.

[161] Symantec. Norton AntiVirus 2013. Retrieved 03 Apr. 2013, from `http://au.norton.com/antivirus/`, 2013.

[162] Microsoft. Windows ISV Software Security Defenses. Retrieved 27 May 2013, from `http://msdn.microsoft.com/en-us/library/bb430720.aspx`, 2013.

[163] Limin Liu, Jin Han, Debin Gao, Jiwu Jing, and D. Zha. Launching Return-Oriented Programming Attacks against Randomized Relocatable Executables. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 37–44, 2011.

[164] M. Prandini and M. Ramilli. Return-Oriented Programming. *Security Privacy, IEEE*, 10(6):84–87, 2012.

[165] Microsoft. Windows XP SP3 and Office 2003 Support Ends April 8, 2014. Retrieved 03 Jun. 2013, from `http://www.microsoft.com/en-us/windows/endofsupport.aspx`, 2013.

[166] P. Garca-Teodoro, J. Daz-Verdejo, G. Maci-Fernndez, and E. Vzquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(12):18–28, 2009.

[167] James P. Farwell and Rafal Rohozinski. Stuxnet and the Future of Cyber War. *Survival*, 53(1):23–40, 2011.

[168] T.M. Chen. Stuxnet, the real start of cyber warfare? *Network, IEEE*, 24(6):2–3, Nov.-Dec. 2010.

[169] R. Langner. Stuxnet: Dissecting a Cyberwarfare Weapon. *Security Privacy, IEEE*, 9(3):49–51, 2011.

[170] T.M. Chen and S. Abu-Nimeh. Lessons from Stuxnet. *Computer*, 44(4):91–93, 2011.

[171] HAK5. WiFi Pineapple Mark IV. Retrieved 01 Sep. 2013, from `http://wifipineapple.com`, 2012.

[172] VirusTotal. Free Online Virus, Malware and URL Scanner. Retrieved 09 Jun. 2013, from `https://www.virustotal.com/`, 2013.

[173] Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Laszlo Szekeres, Stephen McCamant, Dawn Song, and Wei Zou. Practical Control Flow Integrity and Randomization for Binary Executables. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 559–573, 2013.

[174] V. Pappas, M. Polychronakis, and A.D. Keromytis. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 601–615, 2012.

[175] Moxie Marlinspike. Software - sslstrip. Retrieved 08 Jul. 2013, from `http://www.thoughtcrime.org/software/sslstrip/`, 2013.

[176] VirusTotal. VirusTotal scan for Chameleon Shellcode. Retrieved 09 Jun. 2013, from `https://www.virustotal.com/en/file/3f937c422c66327023b4844c94bcfc5065d63c974f8469081649bb6426eaf518/analysis/`, 2013.

Glossary

Address Space Layout Randomization A security technique where OS modules are loaded into different memory locations each time the computer is started. Designed to complicate buffer overflows by preventing trivial manipulation of EIP. 89, 92, 100, 103, 107, 121, 122, 131, 164, 199

ADFA-WD:Stealth Attacks Addendum An extension to the ADFA-WD containing system call traces for the three new classes of stealth attacks created by this PhD research. x, 126, 127, 130, 134, 136, 138, 141, 142, 145, 148–150, 152, 162, 199

Advanced Persistent Threat A term used to describe a technologically advanced and well funded group of hackers. Usually used to refer to state-sponsored groups such as APT1 [12]. 4, 5, 169, 199

Anti-Virus A family of malware detection programs generally used by computers running the Windows OS. Historically, signature-based recognition schemes have been employed by this class of program, but more advanced market offerings have begun to include runtime heuristic protection in recent times, in a step towards anomaly-based detection methodologies. x, 6, 7, 10, 17, 31, 32, 37, 83, 84, 90, 99–101, 107, 108, 124, 126, 127, 149, 150, 159, 161, 162, 199

Artificial Neural Network A type of Decision Engine renowned for its pattern recognition ability. Artificial Neural Networks (ANNs) have many different topologies, and whilst effective in pattern recognition suffer from a high training overhead. 14, 18, 20, 21, 26, 42, 57, 59, 81, 194, 196, 197, 199

Australian Defence Force Academy Linux Dataset A new anomaly-based HIDS dataset produced as part of this PhD research, reflecting contemporary attack methodologies against a modern operating system. xi, 10, 19, 26, 31, 41, 46, 50–53, 55–57, 63, 65–69, 74, 77–79, 82, 84, 97, 101, 115–117, 123, 161, 163, 199

Australian Defence Force Academy Windows Dataset A new anomaly-based HIDS dataset produced as part of this PhD research, reflecting contemporary attack methodologies against a modern OS. This dataset is the first publicly available dataset for the Windows OS. 10, 33, 84–86, 95, 97–104, 107, 109, 110, 113–116, 122, 123, 126, 128, 134, 135, 149, 151, 152, 158, 161, 163, 166, 199

***Chameleon* attack** One of the new stealth attacks developed as part of this PhD research; operates by creating a high impact payload from the allowed system calls in the target process without further system interaction. 127, 139–145, 149, 151, 158

***Chimera* attack** One of the new stealth attacks developed as part of this PhD research; operates by creating a new process which is forced to connect back to the attacker and is then exploited in turn. 129, 137–140, 143–145, 155, 158

Conditional Random Fields A DE which uses positional sequences to extrapolate the most likely transition path between pre-defined classes. Unlike an HMM, Conditional Random Fields (CRFs) require training data from all classes and hence are limited to signature based applications. They provide exceptional accuracy in those roles, however, and are a valuable tool for this family of Intrusion Detection System. 18, 41, 56, 58, 195, 199

Context Free Grammar A type of semantic grammar commonly applied to computing systems. Forms the basis for this PhD research's new core theory. 43, 44, 197, 199

Data Execution Protection A security technique provided in many modern operating systems which aims to prevent execution of memory which is marked as non-executable; designed to prevent attacks such as stack-based buffer overflows. 89, 100, 103, 107, 122, 131, 197, 199

Decision Engine A descriptive term used to describe a machine learning algorithm applied to a specific task. 7–9, 11–15, 18–22, 26–28, 35, 36, 39–42, 45–48, 50, 52–59, 61, 62, 69–72, 74, 76, 80, 81, 84, 90, 94, 98, 113–115, 118, 152, 155, 162, 165, 167, 194–197, 199

defence-in-depth Defined by the *(ISC)2* as "the practice of applying multiple layers of security protection between an information resource and a potential attacker" [19]. 6, 7, 31, 32, 100, 122, 155, 168, 169

Detection Rate A measure of how successful a methodology is at detecting true positives. Performance in this area is a trade off with FAR. 7, 8, 11, 12, 17, 18, 21, 22, 54, 66, 67, 69, 71, 74, 82, 85, 113, 115–117, 123, 127, 151–153, 162, 196, 197, 199

***Doppelganger* attack** One of the new stealth attacks developed as part of this PhD research; operates by creating a new process using allowed system calls to permit remote access. 134–139, 143, 144, 158

Dynamic Linked Library An interface between high level programs and the CPU under Windows architecture. Roughly comparable to the Unix kernel when considered as a whole. 15, 17, 28, 31–33, 83–98, 112–116, 118, 120–122, 132, 133, 142, 144, 162, 163, 165–167, 171, 199

exploit stage The first stage of a hacking attack, in which the execution flow is redirected to the payload. 34, 35

Extreme Learning Machine A type of ANN able to train rapidly through the use of the pseudo-inverse matrix operation. x, 18, 21–23, 27, 28, 41, 57, 58, 69–72, 74, 76, 78, 81, 85, 115, 117, 118, 127, 153, 167, 199

False Alarm Rate A measure of how many false alarms are generated by a given methodology. Most usually applied as a performance metric for anomaly-based IDS. Performance in this area is a trade of with DR. 7, 8, 11, 12, 17, 18, 22, 26, 54, 66, 67, 69–71, 74, 82, 85, 95, 113, 115–117, 120, 123, 151–153, 157, 162, 195, 197, 199

Hidden Markov Model A type of Decision Engine which is very effective at analysing sequences. 14, 18–21, 23–25, 28, 41, 50, 53, 56, 58, 59, 69, 71, 74, 76, 78, 85, 115–118, 120, 167, 195, 199

Host Based Intrusion Detection System An IDS designed to monitor the activity on a single computer. More advanced versions may analyse incoming network packets, and share information with other HIDS within a given network. 4, 6–10, 12, 13, 17–22, 25–28, 30–37, 51, 54, 59–61, 65, 71, 75, 76, 78, 82–85, 87–97, 99–104, 108–110, 113–118, 120–129, 131, 132, 140, 142–145, 149, 150, 152, 153, 155–169, 194, 196, 199

Intrusion Detection System A device designed to detect unauthorised activity in a cyber environment. Usually categorised as either host based or network based, and relying on a signature or anomaly detection methodology. 1, 7, 9, 11–16, 18–23, 25–33, 37, 39–42, 44, 45, 47, 48, 50–63, 65–67, 69–72, 74, 78–80, 82–84, 90, 93, 97–103, 115, 123, 127, 128, 139, 161, 166, 168, 195–199

K-Nearest Neighbour A clustering-based DE. 19, 199

Metasploit Framework An open source freely available exploitation tool kit [1] which is widely used by hackers worldwide. 1, 5, 15, 16, 64, 66, 108, 143, 200

Mimicry Attack A type of hacking attack which uses a specially modified set of system calls to attempt to evade a system call based HIDS, using the method hypothesised by [133]. 33

Multi-Layer Perceptron A type of ANN, consisting of an input layer, an output layer, and one or more hidden layers. 21, 22, 28, 57, 58, 200

Network Based Intrusion Detection System An IDS designed to monitor the network traffic of a protected network. Increasing use of end to end encryption has dramatically reduced the amount of information available in contemporary networks. 12, 13, 20, 22, 60, 200

no-op Short for *No Operation*. A system call which has no functional impact on the state of the kernel, whilst still requiring a clock cycle to execute. Used by genuine programs to allow time for other parallel tasks to complete, by mimicry attacks to alter the payload's system call footprint, and by obfuscated payloads to allow space for shellcode extraction.. 35, 36, 124, 125, 129, 138, 164

non-terminating semantic unit A linguistic token in a Context Free Grammar (CFG) which can be further reduced. 43

payload stage The second stage of a hacking attack, in which the shellcode contained in the payload is executed by the hijacked program. 34

Receiver Operating Characteristics A graph plotting DR against FAR for various thresholds to demonstrate the performance of the system in different configurations. xi, 12, 21, 40, 41, 70–75, 77, 79, 85, 115–118, 200

Return-Oriented Programming An attack technique where code execution is achieved by using functions and *opcodes* located in globally available libraries; designed to bypass Data Execution Prevention (DEP). 100, 103, 131, 158, 164, 200

Self-Organising Map A type of ANN, designed to operate in an unsupervised learning mode. 21, 200

Sequence Time-Delay Embedding A seminal IDS methodology proposed by Forrest et al [23] which considers positional information as well as frequency information. 8, 14, 16–18, 25, 41, 47, 53–56, 58, 69, 70, 72, 74, 76, 78, 79, 81, 200

shellcode The contents of the payload stage. So named in reference to the tendency of payloads to create a remote shell or command prompt through which the hacker can interact with the compromised target. 34

Stealth Attack The superset of all payloads which have a system call sequence which has been modified in order to bypass an HIDS. Mimicry attacks are a subset of stealth attacks. 13

Stealth Attacks Addendum See ADFA-WD:SAA. 149–153, 155–157, 200

Support Vector Machine A type of Decision Engine, which provides a classifying function under supervised learning conditions. 14, 19, 21, 41, 42, 56, 57, 78, 85, 115, 117, 118, 123, 127, 153, 167, 200

terminating semantic unit A linguistic token in a CFG which cannot be further reduced. 43

Virtual Kernel Theory A new theory proposed as a central part of this research which allows the application of Linux-derived anomaly-based HIDS theories to the Windows OS for the first time. Detailed in Chapter 4. 10, 65, 85, 86, 91–99, 112–118, 120–124, 127, 149, 152, 153, 155, 156, 159–161, 163–167, 171, 200

zero-day attack A previously unseen type of attack, hence with no signature in any IDS database. 1, 3, 5, 7, 9, 13, 17, 18, 65, 83, 84, 116, 121, 123, 124, 157, 161–163, 168, 169

List of Acronyms

ADFA-LD	Australian Defence Force Academy Linux Dataset.
ADFA-WD:SAA	Australian Defence Force Academy Windows Dataset: Stealth Attacks Addendum.
ADFA-WD	Australian Defence Force Academy Windows Dataset.
ANN	Artificial Neural Network.
APT	Advanced Persistent Threat.
ASLR	Address Space Layout Randomization.
AUC	Area Under the Curve.
AV	anti-virus.
CFG	Context Free Grammar.
CRF	Conditional Random Field.
DE	Decision Engine.
DEP	Data Execution Prevention.
DLL	Dynamic Linked Library.
DR	Detection Rate.
ELM	Extreme Learning Machine.
FAR	False Alarm Rate.
HIDS	Host Based Intrusion Detection System.
HMM	Hidden Markov Model.
IDS	Intrusion Detection System.
K-NN	K-Nearest Neighbour.

MLP	Multi-Layer Percepton.
MSF	Metasploit Framework.
NIDS	Network Based Intrusion Detection System.
OS	Operating System.
RBF	Radial Basis Function.
ROC	Receiver Operating Characteristics.
ROP	Return-Oriented Programming.
SAA	Stealth Attacks Addendum.
SOM	Self-Organising Map.
STIDE	Sequence Time-Delay Embedding.
SVM	Support Vector Machine.
UNM	University of New Mexico.
VK	Virtual Kernel.