An Extensible Framework for Active Network Intrusion Detection System

Advisor: Prof. Nen-Fu Huang

Student: Roger S.W. Chien

July 2002

A Thesis Submitted to the

Institute of Computer Science

National Tsing Hua University

Hsinchu, Taiwan

Republic of China

In Partial Fulfillment of the Requirements for the

Degree of Master in Computer Science

Abstract

The network security is getting more awareness by public because the computers are network connected and there are increasing worms and network attacks in recent years. Some network devices are introduced to help increasing security, like firewalls and intrusion detection systems (IDS). But for IDS, previous works mostly focused on passive model, which aims at detections and alerting. But it is not enough to cope with current network threats. An active network IDS (ANIDS) would help to patch the rift.

The thesis introduces that the main advantage of ANIDS is to stop attacks at first line and proactive defense. The ANIDS also has some limits like other security gateways and passive IDS. Several related works are surveyed, and a comparison among passive IDS, active IDS and other gateways is given. The issues of designing an ANIDS involve several considerations and trade-offs, such as performance and extensibility. The thesis presents a flexible, efficient, extensible framework for designing an ANIDS. An implementation based on this framework is developed on Linux 2.4 kernel. Performance and detection ability is compared with Hogwash.

Contents

1. Introduction 5
1.1 Introduction 5
1.2 Objectives 6
1.3 Related Works and Background 7
1.3.1 Firewall 7
1.3.2 Passive IDS 7
1.3.2.1 Snort 8
1.3.2.2 Bro 9
1.3.3 Protocol Scrubber 9
1.3.4 Hogwash 10
1.4 Thesis Organization 11
2. The Design of Active Network IDS 12
2.1 Design Issues 12
2.2 Limitations 13
2.3 Our Design 14
2.3.1 Packet Dispatcher 15
2.3.2 Basic Statistics 17
2.3.3 Event Dispatcher 17
2.3.4 Auxiliary Module 17
2.3.5 Detection Preprocessor Module 18
2.3.6 Control Panel 18
2.3.7 Detection Engine 19
2.3.8 Event Handler 19
2.3.9 Packet Journey 19
3. The Implementation 21
4. Performance 23
5. Conclusion 28
6. References 30

Figures

Figure 2.1: The Framework of ANIDS.....15
Figure 2.2: Packet Chains in ANIDS Framework.....16
Figure 2.3: Packet Journey in ANIDS Framework.....20
Figure 4.1: SmartBit Testing Environment.....24
Figure 4.2:SmartBit UDP Throughput Comparison Between Hogwash and ANIDS With Rules.....24
Figure 4.3: SmartBit UDP Throughput Comparison Between Hogwash and ANIDS Without Rules.....25
Figure 4.4: Catapult TCP Testing Environment.....26
Figure 4.4: Catapult TCP Throughput Comparison.....26

1. Introduction

1.1 Introduction

Security is not a young research area, but it was focused on cryptography, access control and auditing before. Until the recent decade, the computer vulnerabilities and attacks are dramatically increased; therefore, the intrusion detection is gaining more awareness than ever. Intrusion detection system is "a computer system that attempts to detect any set of actions that try to compromise the integrity, confidentiality, or availability of a resource" [1]. There are many intrusion detection systems introduced nowadays [2][3].

Because of the increased complexity of heterogeneous computing environment, the network threat prevention and recovery costs not only the value of resource but also the manpower and time spent. Therefore, the key of successful and economical network management is automation and effective reaction. Previous IDS systems are emphasized on passive detection and alarming when attacks had happened. But from the experience of the recent famous worm like "CodeRed"[4], we realize passive IDS is not enough. It can only detect its invasion but not stop it. Once a machine is infected, it quickly tries to propagate and infect the other machines. While passive network IDS generates alarm and waits for system administrators to take over, in this rift the worm has been spread widely.

The ANIDS, however, is designed not only to detect intrusions and generate alarms, but also stop the attacks at first line. It has many similarities with other gatekeeper applications like firewalls and mail filters. For CodeRed example mentioned above, why bother system administrators to patch all vulnerable hosts in a tight time? With ANIDS, system administrators can stop incoming attacks at first line and have the luxury of time to apply necessary patches or take countermeasures.

The other advantage of ANIDS is proactive detection. From experience, before real network attacks, scanning is often the first step. In [5] first mentioned about the issue of OS fingerprinting techniques. And many scanning techniques including stealthy and slow scanning are introduced [6]. Matthew and his partners proposed an idea to scrub operation system's TCP/IP fingerprint in [7] to counter this issue. An ANIDS naturally has the ability to block traffic or change the traffic content. Therefore, it can spoof and hide network and host information from attacker's view and increase the difficulties of attack, in other way, increase the strength of defense.

On the other flip, the active IDS inherently have some limits: time and resource. For example, it must check the packets in a timely fashion within constrained computing power and storage space. The issues on designation and implementation are presented later in this thesis.

1.2 Objectives

ANIDS is needed in current network environment. But its importance and limitation is not seriously studied before. The differences between various security gateway and passive IDS are highlighted to provide a clear view for deployment considering of security components. The second, in order to develop a flexible, extensible and strong ANIDS, the framework is proposed. To accommodate the limitations of gateway devices, such as avoiding performance bottleneck, the designation issues are discussed. Finally, in order to prove our design is practical, a prototype will be made. Its performance result will be compared with other ANIDS.

1.3 Related Works and Background

1.3.1 Firewall

A firewall "can restricts people to entering at a carefully controlled point, prevents attackers from getting close to your other defenses and restricts people to leaving at a carefully controlled point" [8]. Firewall acts as an active device that filters, blocks and logs passing-by packets. Therefore, a firewall "can't protect you against malicious insiders, against connections that don't go through it, against completely new threats and against viruses". And firewalls often rely on IP address for authentication, they possibly suffer from the attacks like IP spoofing described in [9].

But there is a trend that firewalls integrate partial intrusion detection functions. In such system, intrusion detection often acts as an add-on module. When an attack is detected, the intrusion detection component could reconfigure firewall to cut connection or block further access.

1.3.2 Passive IDS

Inherently, IDS has more knowledge about what intrusion is than firewalls. According to their position of deployment and target of information, passive IDS are often categorized into two types: host based IDS and network based IDS. Since we focus on network IDS, only network IDS is discussed here. A passive NIDS "monitors packets on the network wire and attempts to discover if a hacker/cracker is attempting to break into a system (or cause a denial of service attack)"[10].

Based on the techniques used, IDS is divided into two mainstreams: misuse detection

and anomaly detection. The former defines what's illegal or malicious while the later defines what's good and tolerable variance. Signature based detection and protocol analysis method is often categorized as misused detection, while the statistical method is often used in anomaly detection. Both detection types can be used in active network IDS.

1.3.2.1 Snort

Snort [11] is a successful open source project, and it becomes a popular choice for non-commercial passive IDS deployment. Snort is a signature based IDS, all attack detection rules are described as signature, which are the combination of field name, operation and value.

Snort uses Libpcap [12] to read incoming packets, and checks signature by special designed OTN network. In pattern matching, it uses Boyer-Moore algorithm. According to signature type, a packet can be passed, logged and labeled as intrusive alerts. Snort works like a pure network by-stander and doesn't involve active reactions like sending out RST packets to cut off intrusive connections.

It has a quite flexible interface called plug-in to incorporate other functions. There are three types of plug-ins: preprocessor, detection engine and log module. Pre-processors include http URL string extraction, TCP stream reassembling and others. Detection engine plug-in allows statistical based analysis to co-work. Log plug-in provides various logging mechanisms, like IDMEF[13], syslog and plain file.

Though with many contributions from network community and many features are added, the major disadvantage of current snort is: it is stateless. There is no built-in TCP state inspection, thus illegal TCP packets that are not match the current state user can't be filtered out. It can't resist attacks like massive TCP packets flooding. And, it can't trace the protocol state transition and give corresponding detection.

1.3.2.2 Bro

Bro [14] is a passive IDS that works by monitoring traffic on a network link. The core component of Bro is the event engine. Network packets are parsed into events that reflect the various types of activities. Events can be low level like a connection establishment or be high level like an unsuccessfully authentication in a remote login session.

A set of policy scripts is given to analysis the generated events. Policy scripts are written with a specialized language dedicated for network and security analysis. There are events handlers in the script to specify what to do whenever a given event occurs. Event handlers can maintain and update global state information, write arbitrary information to disk files, generate new events, call functions (either user-defined or predefined), generate alerts that produce syslog messages, and invoke arbitrary shell commands. Thus, it might terminate a connection or reconfigure ACL in border router or Firewall.

For any IDS like Bro, the performance relies on filter out traffic that is uninteresting. Bro uses an efficient packet filter to capture only a subset of the traffic that transits the link it monitors and relates to the chosen analyzers.

Consequently, Bro emphasizes the use of tables and sets of values as ways to codify policy particulars such as which hosts should generate alerts if seen engaged in various types of connections, which usernames are sensitive and should trigger alerts when used, and so on. The various analyzers are written such that you can customize them by simply changing variables associated with the analyzer.

1.3.3 Protocol Scrubber

Malan proposed an idea in [15] to develop a transparent interposition mechanism called scrubber for explicitly removing network attacks at transport and application protocol layer.

The scrubber in transport layer is to convert ambiguous network traffic into well-behaved traffic that is unequivocally interpreted by all downstream endpoints. For example, different endpoints has different implementation of TCP/IP, attacker can take advantage of this ambiguity to launch attacks.

And, for attacks like insertion and evasion attacks, that use ambiguities to subvert detection on passive network-based intrusion detection systems, the scrubber can eliminate those while preserving high performance.

The application protocol scrubbing mechanism is used as a substrate for building fail-closed active network-based intrusion detections systems that can respond to attacks by eliding or modifying application data flows in real-time.

1.3.4 Hogwash

Hogwash [16] is an open source project that is the first trial to turn passive IDS into active one. It gets the packets from Libpcap and utilizes Snort as processing module. After Snort signature checking is done, it determines these packets should be dropped or forwarded to another interface.

The problems of hogwash are: first, it uses Libpcap to capture the passing packets, but Libpcap always does packet copy, this is quite unnecessary and slows the packet processing time. Second, it uses Libnet [17] to send out packets, which copies the packets again. Besides this, the packets sent by Libnet would be captured by Libpcap again; therefore, this leads to un-avoidable checking to break infinite capture-and-forward loop. Third, it uses an encrypted control messages embedded in incoming packets to control the system, which is not easy to manage.

1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 introduces the design concepts of an extensible ANIDS framework. Chapter 3 describes the implementation based on this framework and related issues. The result of performance experiment is presented and discussed in Chapter 4. Chapter 5 concludes this thesis with a brief discussion and future work.

2. The Design of Active Network IDS

2.1 Design Issues

The network based IDS has different types, including signature based, statistics based or even hybrid. To employ different detection algorithms, the modularization of detection engine is needed and each engine can be replaced or incorporate with others. The standard interface of detection engine registration and invocation is necessary. Besides, an arbitral mechanism should be given to accommodate the conflicts caused by engines.

All network IDS, no matter passive or active, are triggered by packets. According to study in [18], for Snort, packet analysis and pattern matching consumes more than 50% computing power. For ANIDS, the performance is an important issue since it must process packets in a restricted and short time. Otherwise the network performance will suffer great delay and furthermore an overload attack [14] toward IDS may have deep impact. In [18] proposed a hybrid method to have performance gain by mixing several string matching algorithm for different sizes of pattern sets. Another conventional but practical workaround is to filter out packets that are not interested for detection. Most IDS use BPF library that can capture packet that meets the given condition.

For all network IDS, the major aim is to detect the happening of attacks. Therefore getting the received packets is the minimum requirement. With different considerations, an IDS implementation may need to access more resources. A flexible interface to access auxiliary resources is necessary for different purposes to extend special IDS functions. For example, in order to prevent log overwhelming, an IDS may want to get current system loading status to determine whether an event will be logged or not.

The passive network IDS is quite similar to active ones except the reaction ability and proactive prevention. Most experiences can be borrowed from passive IDS, and the two

should have the freedom to switch to each other by using a simple control mechanism. This can increase the flexibility on deployment.

IDS itself can be the target of attack, if this guard force is destroyed, the invader can do anything wildly. But for convenience of remote management, the communication between user and system is inevitable and IDS can't hide its location to prevent attack. The strength of self-defense is an important consideration in design. Passive IDS often use one-way cable to listen the traffic and use another network interface to communicate with user control and log collector. For active IDS, one solution is to use private and separate network interface to do communication and hide its location from outside world. The other solution is to enhance communication privacy by encryption and authentication, and enhance resistance for DDoS attack by rate limiting.

The reporting mechanism is important for IDS. Most IDS provides attack alerts delivered in real time and detail information of attacks delivered in a period of delay. Attack alert is often sent by paging service, SNMP traps or proprietary protocols. Detail information is often sent by proprietary protocols. An emerging trend is to utilize the IDMEF, which provides a standardized message format for all IDS components to cooperate.

The reaction of passive NIDS is quite the same. They can log and send alerts and for some advanced ones, they can send out RST packet to cut off a TCP connection or incorporate with Firewall to block further access. But for ANIDS, it can log, send alerts, drop malformed packets, reject intrusions, and block hostile targets for a specific period of time. Besides, it also has the ability to modify the packet or craft new packets in order to archive the functionality of protocol scrubber or proactive defense.

2.2 Limitations

Active network IDS, like firewall, is intended to be placed in front of a set of systems

with only one connection to a larger network. The network under protection must be restricted to having one connection to the outside world because all packets traveling to and from a host must travel through this active network IDS.

Second, since active network IDS acts as a gateway, it must process packets in real time, otherwise the network performance suffers great delay. And in most cases, network IDS don't have much computing resources. Thus, for event correlation analysis or data mining, is not suitable to be placed in active network IDS core module. It is recommended to handle these types of operations by another machine.

The other limitation is the encrypted traffic. Applications utilize IPsec more than ever and most firewalls have integrated VPN. The network IDS, either passive or active ones, can't handle this kind of traffic well if packet content is important in detection.

2.3 Our Design

The objective of our design is to provide a flexible, extensible and modularized framework for ANIDS. Every component can be added, removed or incorporated with each other according to deployment consideration. This framework could be a well-facilitated infrastructure for developer to design their own detection engine but retain the freedom to extend their functionalities.

This framework is divided into several modules: control panel, event dispatcher, auxiliary module, packet dispatcher, event handlers, basic statistical module and detection engines. The relationship between modules is presented in figure 2.1.

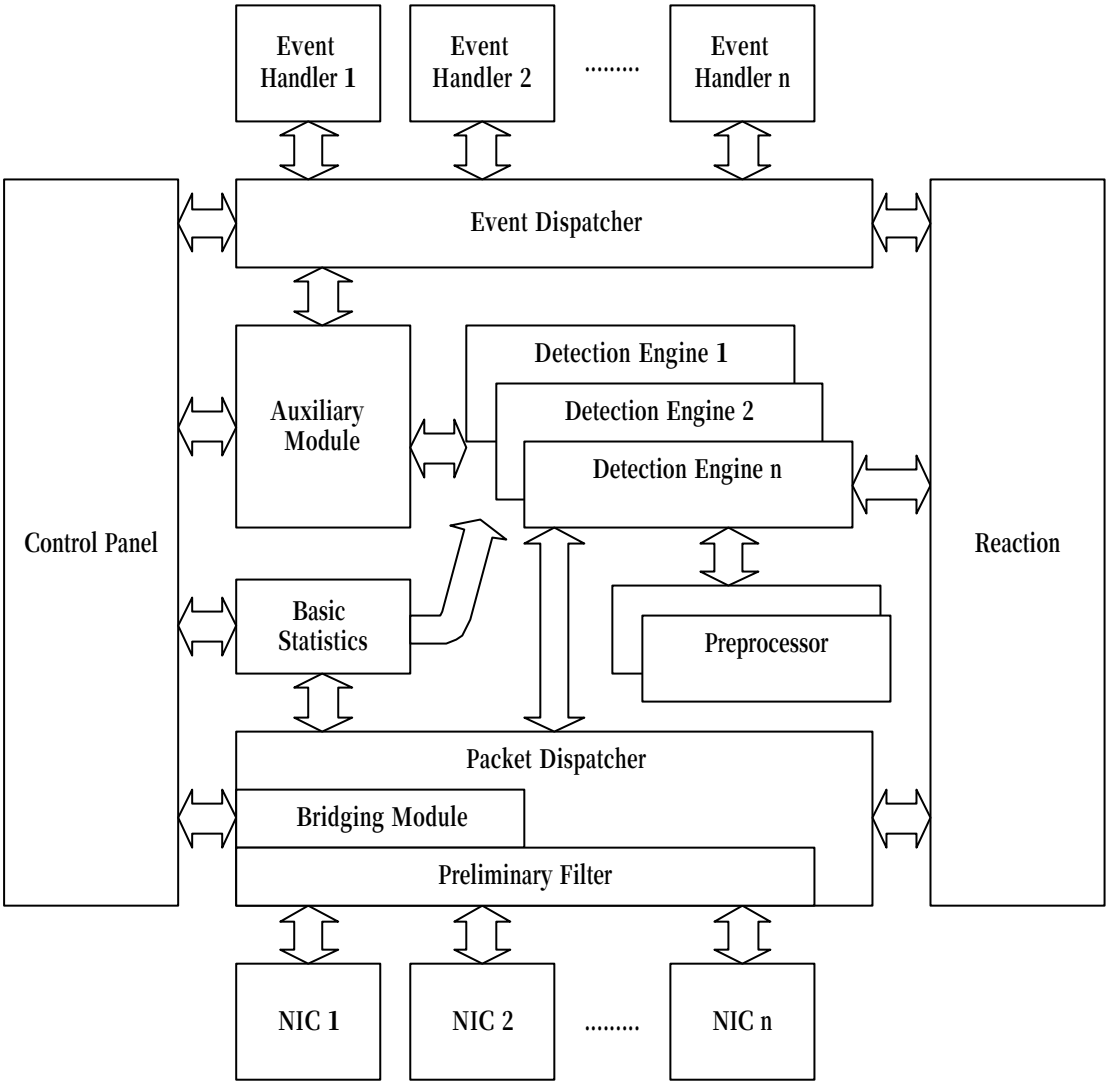

Figure 2.1: The framework of ANIDS.

2.3.1 Packet Dispatcher

Packet dispatcher provides an interface that lets detection engine to specify what kind of traffic they want, and it will send qualified packets to detection engine. With all registrations from all detection engines, the preliminary filter in packet dispatcher can optimize filter rules and bypass detection check for those packets that are not interested by engines. This can increase the performance of IDS system since some detection time is saved. Preliminary filter maintains an allow/deny table. Therefore, it can also block further packets indicated by detection engine in a limited time of period.

The packet dispatcher maintains three packet chains: incoming packet before bridge filtering, incoming packet needs forwarding, outgoing packet. This schema presented in figure 2.2 is very similar with Linux 2.4 netfilter [19] architecture. The bridge module in packet dispatcher is given to select forwarding destinations. The detection engines can register hooks in these chains. Therefore, if any chain have packet, registered callback function of detection engine will be invoked. In most cases, passive detection engines often use the chain of incoming packet before bridge filtering, since they need to watch every packets passed by. And the chain of packets need forwarding is often used by active detection engines because they care about the traffic passed through. The outgoing chain is often used for scrubber-like detection engine to modify packets for certain reasons.

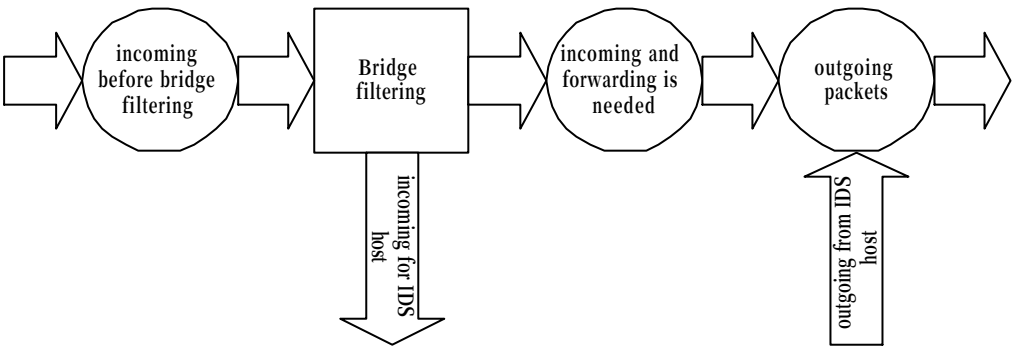

Figure 2.2: Packet Chains in ANIDS Framework.

The packet dispatcher also provides the functionality of packet forgery, which provides detection engine to craft specialized packet for active reaction and insert it in this or other chains. This gives the most advantage of ANIDS over passive NIDS. For example, a detection engine that wants to hide certain services provided by hosts behind the IDS can craft the RST packet to the source host.

Since multiple detection engines can exist in this IDS system, the reaction generated by different engines may diverse. Packet handler also has the ability to handle conflicts. For

example, one detection engine may say the packet is bad, and it should be dropped while another may say it is good. Furthermore, if a packet is labeled as something bad, will it need to be passed to next detection engine? Packet handler will make decision according to the configuration of detection engine and IDS administrator.

2.3.2 Basic Statistics

The basic statistics module is used to keep track some low level statistics, such as the number of dropped packets, the number of received packets, and layer 3 protocol distributions. This information can be used by detection engine for reference, or be retrieved by the control panel to present to user.

2.3.3 Event Dispatcher

The event dispatcher provides a standardized interface for detection engine to send events and alerts. There are four chains for event handlers: informational, low impact, serious and urgent. Each event handler registers one of the chains. If the detection engine issues an event or alert, the event dispatcher will arrange corresponding event handler to deliver this message. The decision of using what event handler can be configured through control panel.

2.3.4 Auxiliary Module

Auxiliary module is an effort to provide basic system facilities for detection engines and other components. The idea is to encapsulate the diversity of operating system of IDS host and present a unique interface for all detection engines. This will greatly help ANIDS to

port on different environments. It will provide data repository, system loading status (normalize to 1 ~ 100), a set of wrapped I/O functions to access persistence configuration and file system, and timer.

Besides these functions mentioned above, pattern matching is also suitable to place in this module. Pattern matching is a necessary function for various IDS, therefore, providing commonly used pattern match algorithms like BM and KMP in this module would save time for detection engines to implement their own pattern matching code.

2.3.5 Detection Preprocessor Module

Preprocessors are used to do some information extractions and state tracking. And different detection engines may reference its result. With preprocessor, we can make the design of detection engine easy, and save cost for duplicate work. For example, preprocessor like URL extraction is quite useful.

The TCP is a communication protocol with states. Many detection engines need to handle state lookup and update. Building a TCP state preprocessor to share among them can be efficient.

2.3.6 Control Panel

The control panel is a gateway between user and other modules. Its major function is to configure the needed parameters and present setting and status for user.

In order to accomplish this, every module must indicate what kind of parameters it needs and the meaning. It also needs to specify what kind of status it can provide.

2.3.7 Detection Engine

The detection engines are supposed to be implemented with different purposes and they can cooperate to become a hybrid system in our framework. Detection engine need to register itself to packet dispatcher. It can be signature based, anomaly detection or protocol analysis based. It can have functions and maintain its own data structure to achieve its purpose.

2.3.8 Event Handler

The event handler can adopt commonly used log module, for example: syslog, SNMP trap and other proprietary protocols. It should have following status: enable or disable, maximum number of hold event, current number of hold event, time window, event count in time window, event limit in time window. The last two properties are used to avoid log overwhelming in a short time. It needs to register methods to event dispatcher, like initialization, startup, shutdown, flush, and invocation.

2.3.9 Packet Journey

The packet in this framework has a journey between different modules. To make this clear enough, it is presented below (Figure 2.3).

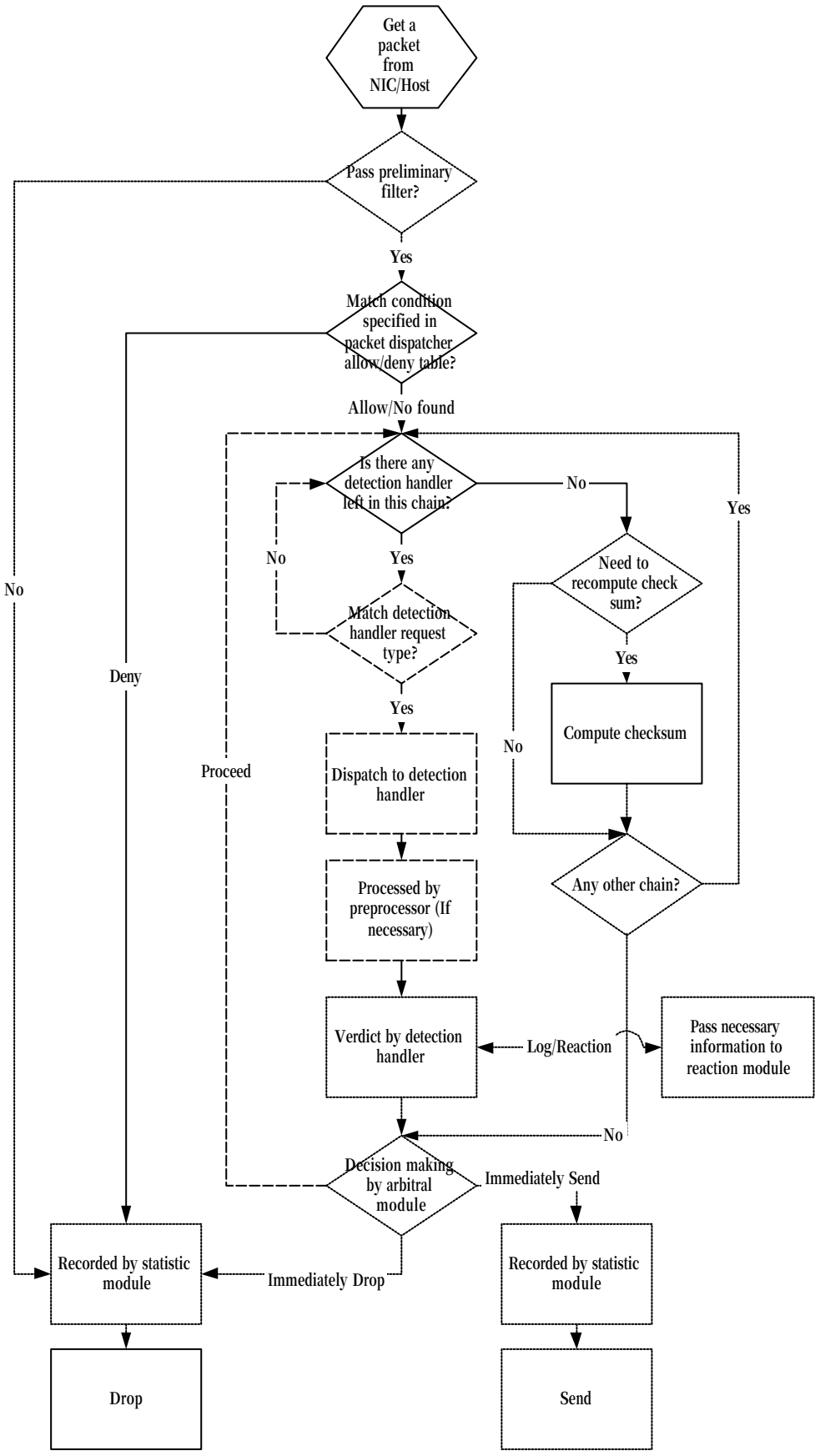

Figure 2.3: Packet Journey of ANIDS Framework.

3. The Implementation

We choose Linux 2.4 as our platform because it is the latest Linux kernel distribution at the time. There are a lot of methods to intercept packet from network, like Libpcap, TUX/TAP, raw socket or NetFilter with Libipq. But two key points laid in front. First, we need to capture all packets passed by since ANIDS must act as a transparent bridge mode device. Second, after the detection, a verdict is given. Therefore, well-behaved packets should be forwarded. The methods purposed above are good at packet interception, but need other wrapped library to send packets.

Another consideration is the easiness of programming and debugging. Of course, porting our whole framework in kernel space is the solution, but not the best one. Program written in kernel space is hard to write, verify and debug. We need program running in user space. Finally, LSF (Linux Socket Filter) becomes our best choice. It can capture the packets and also has the freedom to send any packet. It is independent with host's TCP/IP stack and provides standard user mode interfaces that are socket I/O like functions.

Besides all of above, LSF has a nice feature like BPF that can define filters to prune unnecessary packets. And its filter is fully compatible with BPF filters. As mentioned in Section 2.1, this can save precious time for processing.

In order to enhance the strength of IDS host, we utilize another tool called IPTables [19]. It is a powerful tool to limit the access from outside world to IDS host. We provide a set of wrapped functions to configure it in our framework.

Most of IDS engines will utilize file as log/report repository. But file I/O somehow slower than memory access. Nowadays, operating system and computer architecture supports DMA, that can off-load CPU computation for data transfer. In order to take advantage of this feature, our file system functions in auxiliary module will maintain a read/write buffer to collect small data and then write at once. For urgent conditions, flush

function is provided to write data immediately.

Our framework needs to handle different detection engines, event handlers and preprocessors. They are different but also have the same properties. In some cases, it's more convenient to invoke module in generic way. Object oriented model provides encapsulation, inheritance and polymorphism. Our framework is quite fit in object-oriented model. Therefore, we choose C++ as our implementation language.

Every module is modeled as a class. For detection engines, event handlers and preprocessors, the generic classes are given for each type. All implement modules of these three types must inherit from generic classes. This provides a common way to access heterogeneous objects by invoking polymorphism mechanism at run time.

4. Performance

Since ANIDS is placed in the wire, the performance issue is important and critical. Without qualified throughput, it will become the bottleneck of the network. Moreover, the IDS algorithm attack that targets the performance weak of IDS will be vital challenge for active intrusion detection systems.

With the framework implemented above, we develop a detection module that is quite similar to Snort core module. Rules have directions: it can be outgoing, incoming or bi-directional. Numerous fields are given to specify the signature of an attack. Field is composed of type, comparison operator and value. For different types of fields, it can have different operators. Scalar values like port numbers and TCP sequence number have operators like equal, not equal, larger than, lesser than and no operation. Flags like IP flags and TCP flags have operators like equal, not equal, in, not in and no operation. And we provide four types of pattern matching: case sensitive, case insensitive, URI comparison, white space ignored comparison. Our rule can cover all specification in Snort rule except the preprocessors. In fact, a tool is written to automatically translate Snort rules into ours.

The testing environment is a Pentium-III 700 machine with 256MB RAM. It has two Intel i82559 10/100 network interfaces. The installed operating system is RedHat 7.3 (Linux kernel version 2.4.18) with default configuration, and the OS kernel is not specially optimized. We run Hogwash and our ANIDS test bed with the same 950 rules. The preprocessors in Hogwash rule were removed for sake of fairness. We use SmartBits to generate saturated and random UDP traffic. The burst time is set to 10 seconds. The network architecture is presented in Figure 4.1, and the testing result is presented in Figure 4.2.

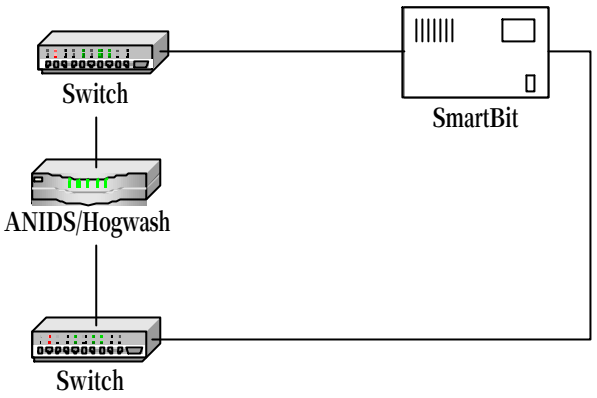

Figure 4.1: SmartBit Testing Environment.

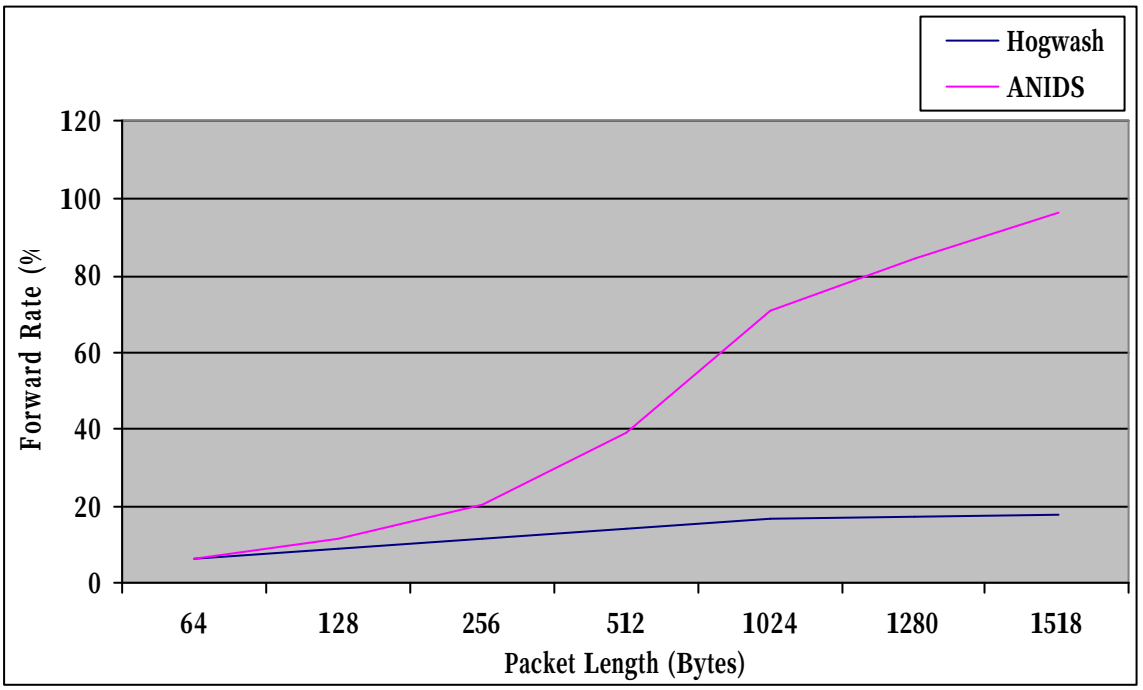

Figure 4.2:SmartBit UDP Throughput Comparisons Between Hogwash and ANIDS With Rules.

From the result, we discovered that our ANIDS works much more efficient than Hogwash. The throughput is quite similar when the packet size is small, that the computing utilization is mostly wasted on massive interrupts. But when the packet size increases, the difference is quite huge.

In order to compare performance of the underlying framework, rules in both systems are removed. The testing environment is the same. The result is represented in Figure 4.2. From this result, it shows that the ANIDS framework with LSF is superior to Hogwash with Libpcap and Libnet.

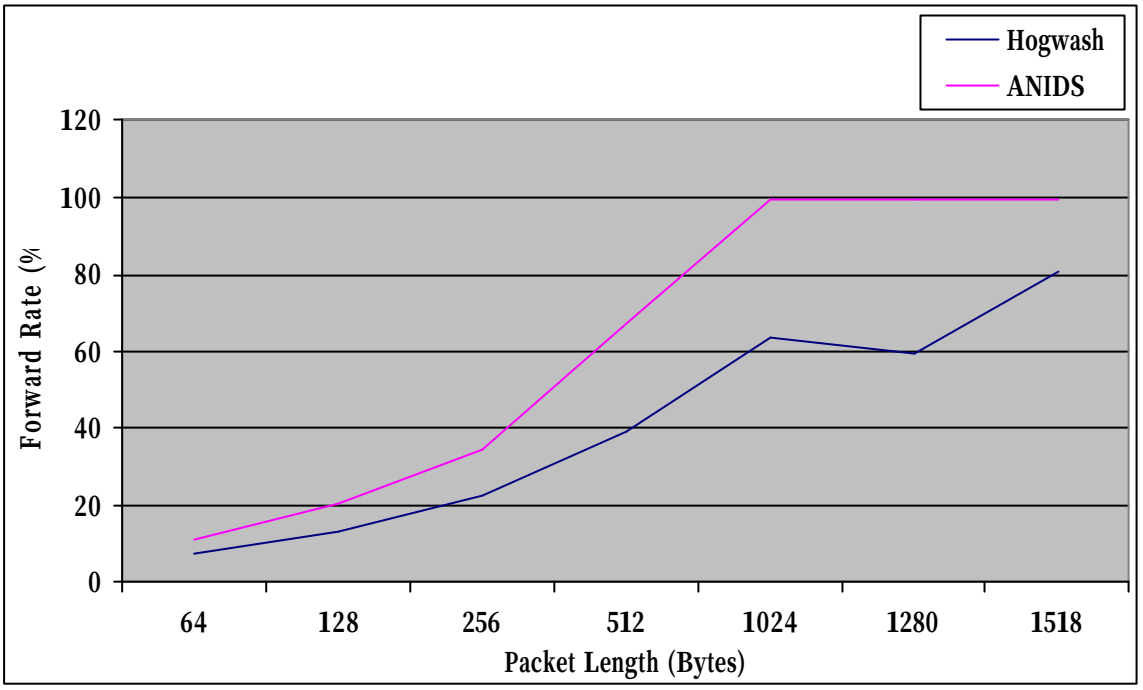

Figure 4.3:SmartBit UDP Throughput Comparisons Between Hogwash and ANIDS Without Rules.

Though our framework performs better than hogwash, with the number of rules or detection engines increasing, the performance may suffer impact. Even in this prototype, the throughput of common 256 and 512 bytes packets is between 20% to 40%. And this result is for UDP traffic, we convince that, for TCP traffic, the throughput may even lower because TCP rule number is greater than UDP rule number.

Thus, we using another benchmark program called Catapult to measure the TCP throughput. It could create real TCP connections in a best-effort fashion. The whole Catapult system needs a client to create connections and send data, and a server to listen connections

and summarize throughput data. The testing environment is presented in Figure 4.4. Two pairs of PCs are used, one for one direction transfer and the other for opposite side. This arrangement can avoid the bias since that the data transfer from client to server in Catapult always dominates. The testing procedure is quite the same as above, we compare our ANDIS and Hogwash with rules, and then, without rules. The result matrix is presented in Figure 4.5.

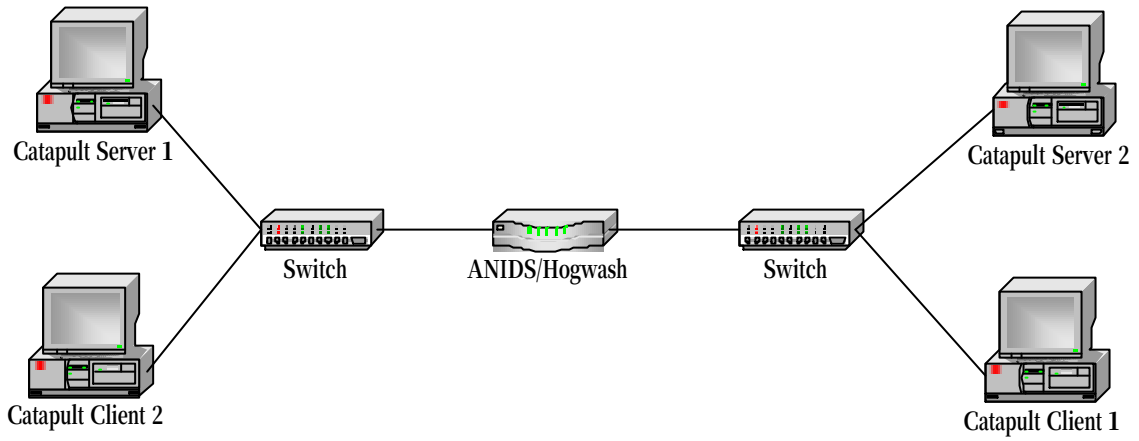

Figure 4.4: Catapult TCP Throughput Testing Environment.

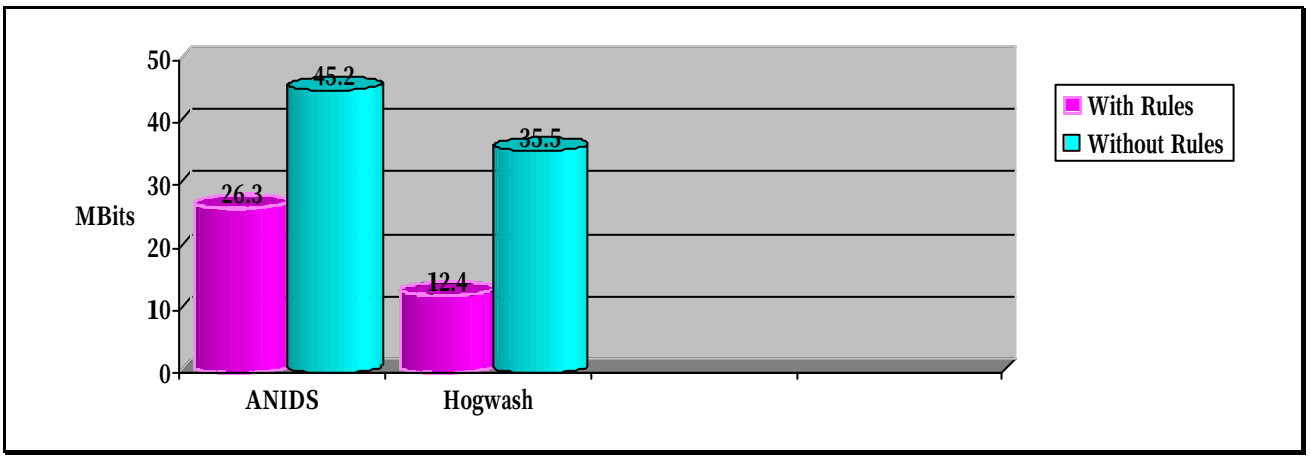

Figure 4.5: Catapult TCP Throughput Comparison.

From the result, our ANIDS framework still performs better than Hogwash. But, obviously, it can't reach wire speed. The possible workarounds are: change to more power platform or SMP architecture, optimize operating system, and rewrite the system kernel for packets zero-copy to gain better performance.

5. Conclusion

The ANIDS framework presented in this thesis is intended to give a solid infrastructure for developing ANIDS. It provides a set of well facilities and gives the flexibility to replace or insert a detect engine into IDS. It also encapsulates the diversity of underlying operating systems. This can make ANIDS developer concentrate on detection algorithms and make ANIDS porting easily toward various operating environments.

In implementation, we demonstrate a prototype that uses our framework and has core detection engine similar to Hogwash's but its performance is much better. With careful design and implementation considerations provided in this framework, detection engine designer can take advantage of efficient environment and gain more computing power for detection.

This framework presented in this thesis is quite extensible. Different implementations of scrubbers or ANIDS can be incorporated together. A possible function is URL scrubber. There are lots of URL attacks such as IIS Unicode attack [20], directory traversal attack [21,22] that utilize malformed URL strings to accomplish attack. Once a HTTP request is issued, the request can be passed to this scrubber. It extracts the URL string and removes ambiguity and illegal portion to normalize it. In such way, there is no chance for URL attacks to take place.

Another possible application is information keeper. As mentioned in section 1.1, an anti-OS-fingerprinting function is easily fitted in our framework. It can further extend to hide/spoof other information that may reveal network topology and host weakness. For example, it can masquerade application banner strings, block illegal SNMP queries, and etc. For more elaborate and complex interactions, it can redirect specific traffic toward Honey-pot [23,24] or Honey-net [25].

Our implementation currently is limited to single system scope. For mature and large

scaled deployment, this is not enough. Multi-hierarchical event exchange and incorporation control can be added in event handling module and incorporation mechanism can be added in control module.

Another possibility to gain performance is to utilize the power of multi-processors. Most current operating systems support the multi-processors. ANIDS can take advantage of this feature to speed up detection and reaction. In our framework, it is easy to parallelize detection engine since we provides an arbitral mechanism to solve conflicts. Of course, packet dispatcher needs to do extra work for synchronization of tasks. And every service function needs to take care about the problem like reentrance and synchronization.

6. References

[1] R. Heady, G. Luger, A. Maccabe, M. Servilla, "The Architecture of A Network Level Intrusion Detection System", Technical Report CS-90-20, Dept. of Computer Science, University of New Mexico, August 1990.

[2] S. Axelsson, "Research in Intrusion-detection Systems: A Survey", Technical Report 98—17, Dept. of Computer Engineering, Chalmers University of Technology, December 1998.

[3] R. Barber, "The Evolution of Intrusion Detection Systems - The Next Step", Computers & Security, Volume 20 Issue 2, pp.132-145, April 2001.

[4] eEye Digital Security, ".ida Code Red Worm", http://www.eeye.com/html/Research/Advisories/AL20010717.html, July 2001.

[5] Fyodor, "Remote OS Detection via TCP/IP Stack Fingerprinting", http://www.insecure.org/nmap/nmap-fingerprinting-article.html", April 1999.

[6] M. de Vivo, E. Carrasco, G. Iserm, and G. O. de Vivo, "A Review of Port Scanning Techniques", Computer Communication Review, Volume 29, No. 2, April 1999.

[7] M. Smart, G. Robert Malan, and F. Jahanian, "Defeating TCP/IP Stack Fingerprinting", Proceedings of the 9th USENIX Security Symposium, August 2000.

[8] D. B. Chapman and E. D. Zwicky, "Building Internet Firewalls", pp. 17, O' Reilly & Associates, Inc. 1995.

[9] M. de Vivo, G. O. de Vivo, R. Koeneke and G. Isern, "Internet Vulnerabilities Related to TCP/IP and T/TCP", ACM SIGCOMM Computer Communication Review, Volume 29, No. 1, pp.81-85, January 1999.

[10] R., "FAQ: Network Intrusion Detection Systems", http://www.robertgraham.com/pubs/network-intrusion-detection.txt, 1999.

[11] Snort, http://www.snort.org

[12] Libpcap, http://www.tcpdump.org

[13] D. Curry and H. Debar, "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition", IETF IDWG Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt, December 2001.

[14] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Computer Networks, 31(23-24), pp. 2435-2463, December 1999.

[15] G. R. Malan, D. Watson, F. Jahanian and P. Howell,"Transport and Application Protocol Scrubbing", Proceedings of the IEEE INFOCOM 2000 Conference, Tel Aviv, Israel, March 2000.

[16] Hogwash, http://hogwash.sourceforge.net/

[17] Libnet – Packet Assembly System, http://www.packetfactory.net/Projects/Libnet/.

[18] M. Fisk, G. Varghese, "Fast Content-Based Packet Handling for Intrusion Detection," UCSD Technical Report CS2001-0670, University of California San Diego, May 2001.

[19] NetFilter, http://netfilter.samba.org/

[20] MITRE, "CVE-2000-0884," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884

[21] arachNIDS, "IDS297/WEB-MISC_HTTP-DIRECTORY-TRAVERSAL1," http://www.whitehats.com/IDS/297

[22] arachNIDS, "IDS298/WEB-MISC_HTTP-DIRECTORY-TRAVERSAL2," http://www.whitehats.com/IDS/298

[23] B. Cheswick, "An Eventing with Berford in Which a Cracker is Lured, Endured, and Studied," Firewall and Internet Security, Chapter 10, Addison-Wesley, 1994.

[24] D.Klug, Honey Pots and Intrusion Detection, SANS Institute, September 2001.

[25] The Honeynet Project, Know Your Ememy: Honeynets, http://www.honeynet.org/papers/, April 2000.