UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral thesis by

Eric Eilertson

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Vipin Kumar

Name of Faculty Co-Adviser

Signature of Faculty Co-Adviser

Zhi-li Zhang

Name of Faculty Co-Adviser

Signature of Faculty Co-Adviser

Date

GRADUATE SCHOOL

A Data Collection, Storage, and Analysis Framework for Network Security

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Eric Eilertson

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Vipin Kumar, Zhi-li Zhang

October 2007

Abstract

As the number, severity and sophistication of computer network attacks increase network administrators have an increasingly difficult time identifying and cleaning up compromised computers. In this thesis some of the areas where existing tools and techniques are deficient are identified, and possible solutions are proposed and evaluated on synthetic as well as real networks. This thesis has four major contributions. The first is a lightweight semi-stateful network data capture module. The second contribution is a framework for storing and accessing raw packet information as well as meta information, such as network sessions. The third contribution is a set of analysis routines for identifying computer network attacks, and computers that have been successfully compromised.

The fourth contribution is a framework for iteratively building and analyzing the communication patterns of networked computers. This allows security analysts and researchers to identify compromised computers, as well as perform forensic analysis to answer questions like *What computer compromised this computer? When did the compromise occur? How did the compromise happen? What data was stolen or modified?*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Securing computer networks has been an active research area for the past several decades, and particularly the past few years. The necessity of this research has grown as attackers have realized that there is a lot of money to be made by stealing information from peoples home computers, or by stealing design or research work from industry or government computers. Unfortunately, the hackers seem to be winning at the moment; current estimates put the number of computers in a single botnet related to the Storm Worm at 1-10 million. Security administrators working on this botnet, or any other network security incident, all realize that current tools and techniques are not designed to cope with these types of attacks, nor can they handle the volume of traffic or alerts generated by existing tools.

The simplest of the tools used today are signature-based tools. These have a predefined set of activity that they will allow or flag as malicious. Examples of these are the firewall or the ever popular open source tool Snort [47]. Snort operates in a mode typical to most IDS systems, a mode that has existed almost as long as network security has. Snort runs on a computer that has access to all of the data transiting a switch or router, typically at the network border of an organization. Snort collects packets from the network looking for byte sequences defined a-priori, raising alerts

when a match occurs.

Realizing that having a predefined set of patterns will always be behind the curve, never able to identify new attacks, researchers have been working on techniques that use the latest research in the areas of data mining, pattern recognition, neural networks, etc. Some of this work has had good success on synthetic benchmark data, and a few have even worked well on real networks. However, the operational and research tools created thus far suffer from a number of problems, such as:

- Possible data sources are too fine grained, requiring lots of processing to prepare the data for analysis. This forces the tools to either operate on raw information, with little context, or to maintain state information about the sessions, hosts, and networks monitored.

- The current in-line mode of operation for most IDS systems imposes severe limitations on the amount and type of analysis that can be performed. Given the limited amount of CPU, memory, and disk available to the sensors, there is only a fixed amount of analysis that can be done, and only a small amount of historical information that can be stored to give additional evidence to support the declaration of a connection to be an attack.

- A problem with these tools lies in the fact that they are designed to catch intrusions as they happen, assuming that the intrusion will be stopped by some other entity once flagged. While this would be excellent if it were true, sadly this is not the case. Intrusions do happen. Some of these may have been caught by the IDS tools deployed on the network, but a delay happened before an administrator was able to look at the alerts. Or as is often the case, the intrusion was never detected.

- Once the intruder has compromised a computer, there are very few tools designed to help the administrator clean up the network and identify compromised computers. The current IDS tools do not even begin to deal with the tasks *after* the inevitable compromise has happened, answering the *who, what, when, why, where and how* questions - basic questions that must be answered to properly clean up the problem, seek legal recourse against the actors, and prevent such a compromise from happening again.

1.1 Thesis Contributions

As part of this research project a framework for network security has been created. This framework addresses the problems listed above, and modules implementing this framework have been tested on synthetic data, as well as running operationally on real networks. The main components are **Data Collection** tools that address the state-maintenance problem of current IDS tools, **Data Storage** tools that store information necessary for the forensic process, and the reduction of both false alarms and false positives. Several **Analysis Engines** have also been created that leverage the ease of access to stateful information provided by the overall system. These analysis engines either provide increased detection compared to existing tools, or fill a gap currently not addressed by any existing tools.

Many of the components developed as part of this thesis have been used for over four years by the US Army and the University of Minnesota. This system has led to increased efficiency of the security analysts and allowed for a greater understanding of security incidents and networking in general.

Chapter 2

Background, Research Challenges and Proposed Approach

2.1 Background

As was briefly alluded to in the introduction, many intrusion detection tools must, for each tool, collect raw data and organize the data into more useful bits of information for processing. This organization often involves maintaining some sort of state. This "state tracking" can take the form of following a tcp session, linking an FTP data transfer with the FTP command session, or something more complicated like chaining a series of connections together into one large event or related set of events. A typical example of IDS tools maintaining state is the snort preprocessor for tracking tcp sessions. This add-on to snort tracks the state of the tcp sessions which allows signatures to be matched to sessions, as opposed to single packets.

Adding state information has huge benefits in terms of reducing false positives and false negatives. Maintaining state often requires more CPU and memory than is available for deployed systems. These systems also have limited amounts of disk

space available, limiting the amount of historical information available. This in turn limits the ability to perform profiling of hosts, applications, and networks, restricts the type and quality of analysis that can be performed, and makes it nearly impossible to present any contextual information.

2.1.1 IDS is based on decades old technology

Perhaps more of a symptom than a basic problem is the fact that most IDS systems are based on decades old technology. The majority of IDS systems in use today are based on signature matching, which has existed for decades. A lot of research has been put into ways to automatically create what are effectively signatures. These systems look for byte sequences that correlate with "malicious" activity, or communications matching a certain set of parameters, or threshold-based schemes that wait until a count goes above or below a threshold to declare an activity malicious or benign [28].

Unfortunately, the reason these old techniques are still in use is not because they work so well, but because nothing better has come along. A major contributor to this problem is the difficulty researchers and analysts have getting access to the data to study networks and network attacks. Without the ability to understand the basics of the world in which they are operating, it is difficult develop new techniques to discover, let alone analyze, the intricacies and subtle nuances of both normal and deviant behavior.

One of the reasons for a lack of understanding of how networks work, for both normal and malicious behaviors, is that current tools are lacking the ability to analyze data for any length of time. Most of the existing tools are stateless or maintain a minimum amount of state. This severely limits the context in which a connection can be analyzed, and makes doing so extremely difficult, if not impossible. Part of the reason for the inadequate tools could be that even though CPU, disk, and memory

resources have grown considerably, so has the complexity of the networks. Another contributing factor could be that researchers do not know what it is they should be collecting to study (a chicken and egg problem).

2.1.2 Current forensic process is inefficient

The forensic process of most security analysts and organizations is at best inefficient and often non-existent. When a security alert has piqued the interest of a forensic analyst that analyst often has to determine what to collect, and then collect his own data to analyze. Aside from the obvious problem of collecting data *after the fact*, the data collected is typically raw tcpdump or netflows and may be missing elements of information. Once some data is collected the security analyst then spends the majority of his time sifting through the volumes of data looking for communications involving the IP address in question. This process can literally consume many days of a security experts precious time - time spent while the hackers are running rampant on the networks.

2.1.3 Current Data Sources

What data is collected sets the course for what analysis and research can or cannot be performed. In an ideal world all of the data would be collected and readily available. Having access to the raw payload of every network packet would enable every kind of analysis, however collecting every packet for a network of any size is simply not feasible. The other extreme is to collect nothing, this is of course not very interesting. This raises the question, "What should be collected?"

There are currently two main types of data used in the networking area - tcpdump [49] and netflows [55]. Tcpdump collects the raw payload of each packet, while netflows collects session based information. Each of these data sources has its ad-

vantages, disadvantages, and major limitations which will be discussed in detail in Chapter 3.2, and presented briefly here.

Tcpdump Tcpdump [49] is a program built around the libpcap library. This captures packets from a networking interface and brings the packets through kernel memory into user memory. There simple filters can be applied to determine if this packet should be stored. If the packet is to be stored, it is written out with a small header, followed by a set amount of bytes beginning at the link layer. The amount of bytes to be recorded is a global value set at runtime and defaults to only 68 bytes.

Netflow Netflow is a data format which can be collected from most routers. It is a format developed by Cisco, though the other major router manufacturer, Juniper, also supports this data collection format. Netflows are analogous to phone call records; they contain high level information with no details of what actually was communicated. Netflows contain the source and destination IPs, source and destination ports, protocol, number of packets, and number of bytes. This information can be very valuable for communication pattern analysis, but netflows also have several deficiencies. Most of these deficienes relate to the unidirectional nature of flows, meaning a flow only contains one side of a conversation, more detailed discussion of netflows is in Chapter 3.2.

2.2 Research Challenges

There are a number of basic problems with the way operations, analysis, and research are performed in the current sensor-based IDS paradigm. The problems are interrelated, often creating a chicken-and-egg problem to arrive at a solution. All of these problems contribute to a situation where people are not able to answer some questions, inefficient at answering others, and have difficulty developing new programs

and tools thus severely inhibited in their research.

2.2.1 Difficult to collect the right data

Much of the problem with data collection stems from the large volume of data to collect and poor tools available for data collection. Collecting all of the traffic is simply not possible given the relatively high amount of bandwidth available compared to the amount of storage available. A typical home connection is now 10+ Mbit, and could require over 100 GB a day to store its traffic if fully utilized, though these connections are rarely fully utilized. Larger organizations, like companies, governments, or universities often have orders of magnitude more bandwidth available to them then home users, and use most of their available bandwidth since they may have thousands of users on their networks.

Given that it is almost impossible to collect all of the traffic for a site, downselecting the amount of traffic collected seems like a good alternative. Unfortunately, the tools for doing this provide too course a granularity and do so on the wrong dimensions. Tcpdump [49] is discussed in section 3.2 and explains many of the shortcomings.

The alternative to analyzing payload information is usually to analyze flow information, typically in Cisco netflow format [55]. Many of the problems are discussed in section 3.2, but essentially this format is at too course a granularity and does not include any of the content that was actually transfered.

2.2.2 Current frameworks cannot answer basic questions

The fact that current IDS frameworks cannot answer basic questions is like the fact that IDS is based on decades old technology, perhaps a symptom more than a basic problem.

When security analysts or researchers observe a network connection of interest, there are a few basic questions that typically need to get answered before any understanding of that type of behavior can be found. These questions are the *who, what, when, why, where and how* questions. Unfortunately, current IDS frameworks typically 1) do not have the information necessary available to answer those questions, and 2) if present, finding the information can be cumbersome and time consuming.

Many IDS frameworks store only the alerts of possible attacks, while a few store the payload of the packets that triggered the alerts. This limited amount of information cannot give a proper context to the possible attack and often does not have enough supporting evidence to confirm that an attack was either successful or unsuccessful.

Some security experts run tools like tcpdump or netflow to collect basic raw information about what transpires on the network, [49, 55]. However, these tools create very large files which cannot be efficiently accessed, and require a significant amount of work on the analyst's part to build network sessions and identify which connections may be related to the attack in question.

2.2.3 Managing large volumes of data is difficult

One of the problems facing researchers in data mining and networking is how to effectively manage the large volumes of data available to them for research. Research is often slowed due to the time consumed by simply manipulating the data and trying to construct a testbed for performing experiments. In some cases, good research possibilities are abandoned because of the overhead associated with getting a suitable environment for conducting experiments.

This is particularly true in the area of network security. While data miners typically want as much data as they can get a hold of, the volume available in the network

arena is far bigger than most researchers can reasonably deal with. For example, the University of Minnesota typically receives about 500 million network connections per day. This requires about 30 Gigabytes of storage per day just to store Netflows [55], which contain only a minimal amount of information relating to the connections. Making the problem more difficult is the fact that in order to perform any "interesting" research, researchers will often require significantly more than one day's worth of data; often several months' worth of data will be required.

Requiring months of network data is already enough to hinder research at a single site. Compounding this problem is desire to perform cross-site analysis, which will require data of some sort to travel from one site to another.

An example demonstrating the difficulty involved with managing massive amounts of data, and of the multidisciplinary skills currently required to perform IDS research is the ongoing scan detection research occurring as part of the MINDS project. The MINDS group has completely filled a terabyte of disk with meta-information about scans. This information can be obtained in fractions of a second from the raw data if indexed properly, instead of hours as is currently required. An additional problem faced by the scan detection researchers is the massive data duplication.

2.2.4 Root Cause: Poor data access and management

The root cause for the above listed problems with the current IDS paradigm seems to come from the fact that developers, analysts, and researchers do not have access to the right kind of data, and working with the data they have access to is difficult. The creation of tools to address these problems is hindered by the fact that developers don't know what it is they don't know, and the developers need not yet created tools to understand what tools they need to create.

The goal of this research project is to create a system that enables the easy

collection of the necessary data, and allows developers, analysts and researchers to abstract away many of the issues involved in operating on very large streaming data sets, such as network data.

2.3 Motivating Example

My motivation for this whole project began in the spring of 2004. After working with the ARL-CIMP for over a year, I was beginning to understand their capabilities and needs, but had no concrete reason or example to back up my thoughts.

2.3.1 Stakkato

Analysts at the ARL-CIMP had identified a compromised computer on their networks communicating with a computer at the University of Minnesota's chemistry department. The analysts at the CIMP asked if I could coordinate with the University's network security to investigate.

We began by going to the compromised computer at the University of Minnesota to look at just what happened. We looked for rootkits and rebooted from a CD to have a clean image to work with *. After doing some digging and spending hours talking with other sites, we decided to continue to monitor the computer to see what else the hacker was doing. This choice, and the fact that the hacker didn't abandon the computer, turned into a gold mine of information which many people credit with the information necessary to arrest the hacker.

The way the University's security analyst and I monitored the hacker was done entirely manually and was very time intensive. We had a computer we knew was hacked, so we looked to see what other computers it talked to, hoping this would

*In hindsight this was a bad idea, since in general this procedure can alert the hacker that they have been caught, causing them to disappear without ever learning their intentions

give us additional insight into the intentions and would identify other compromised computers. Many of these communications were legitimate, and manual analysis of the flows (and sometimes a few phone calls) allowed us to say so in most cases. While we were busy looking for computers this chemistry computer was talking to, the hacker was very busy using this as a password collector and then hacking into thousands of additional computers all over the world (though primarily at universities and government research labs). Several times per hour we would give information to various law enforcement officials who were traveling all over the country, gathering information and meeting with other investigators.

After the first day, we began to fall into a rhythm. We would take our list of known "bad" computers and look to see whom had they been talking to. We would try to determine which of these were legitimate and which were suspect, and feed that information to people at other sites, some of whom would share what they knew, and we would perform another iteration of analysis. Unfortunately, the data required for analysis had to be collected manually. Then a series of programs was run against the newly-collected data to get it into a form which was easier for us to analyze. The process was conceptually simple, but the volume of data made it time consuming and taxing on our computational resources.

After a few days the behavior of the chemistry computer changed, it was now receiving a lot of traffic on port 53 TCP (this is typically used for DNS queries). Manually using tcpdump we were able to look into the payload to see that what was being sent to this computer (and then relayed to other computers) was in fact usernames, passwords and IP addresses. Combining this information with the files found on the hacked chemistry computer gave us the answer to how the hacker was spreading so incredibly fast. He would hack a computer and replace ssh with a trojanized version which would send the username, password and remote IP address

of every connection to a password collector. The hacker was not using exploits or buffer overflows to gain access to computers, he was logging in with real usernames and passwords!

This insight led to the realization that the way the hacker gained access to the ARL was to actually use a Kerberos ticket sitting in a users home directory. This led to an immediate change in the way users connected to all government supercomputers. Kerberos tickets would be good for only five minutes, and eventually the tickets would exist only inside a Unix pipe and never actually go to disk. This slowed the hacker down for a few days, but he eventually figured out how to steal Kerberos tickets from the Unix pipe and use them to gain access.

While investigators were figuring out how the hacker was operating, the hacker was busy hacking into Cisco where he stole the source code to their newest IOS [†].

My involvement in the investigation lasted only a few weeks before the University's security analyst and I realized the computer science department's computers were hacked. The hacker had gotten control of the domain controller, trojanized the departments ssh program, and was doing all this using the username of the analyst I was working with, (The hacker seemed to find great satisfaction in mocking administrators.) At this point the analyst had to shut down the University's Internet connection and begin the long process of cleaning up.

While administrators were cleaning up the network, I noticed an interesting computer talking to the chemistry computer. It was at arsc.edu, Arctic Region Supercomputer Center [3], which is similar to the Army High Performance Computing Research Center [1]. It was at this point I began having conversations with progressively higher and higher level managers within the DoD, which ended with the Defense Research and Engineering Network (DREN) [17] being shutdown. DREN

[†]So much for Cisco's "self-defending network"

is one of the largest networks in the world, and the only way to reach government research and supercomputer centers.

It was almost a year later that the 16 year old hacker who called himself Stakkato was arrested in Sweden. This story was covered by every major news source, but like most big events, was broken by the New York Times [52]. Other sites with information on the case can be found here [50, 51].

2.3.2 Problems in Network Analysis Methodologies

Working the Stakkato case confirmed my suspicion that current tools and techniques lack the ability to quickly determine the the scope of compromise. If analysts were able to find out what computers may be involved in a compromise, he could then determine how to handle the incident. Does it look like a worm or virus, is any data being uploaded or downloaded, does the command and control look automated or is a person driving everything? These are a few of the questions an analyst or manager would want answered before making a decision about how to handle the situation.

To answer these questions it is best to begin with what is known or at least highly suspected to be true. Given a computer suspected of being hacked, determining *to whom has the computer talked*, and *does it normally talk to these computers* is a critical first step. This information allows the analysts to determine how big the situation may be. For example, if it is a laptop which normally talks to a few external webservers and now talks to a few IRC servers, it is probably a bot reporting to a bot-net. This is a problem, but probably does not warrant shutting down the network or getting people out of bed. If, however, it is a fileserver which normally never talks to the outside world and is now doing large data transfers outside the country, waking up more senior personnel is probably a good idea.

The entire investigation of Stakkato revolved around answering those above two

questions over and over. Very experienced analysts from all over the world worked very hard 24 hours a day to answer those questions, and the work was unfortunately done manually. The Stakkato incident very clearly pointed to a problem this thesis addresses - *analysts need the right data, and they need access to it quickly.*

If the proper data is collected, determining what computers a compromised computer has talked to is fairly straight forward, at least in theory. Simply scan through the netflows and look for every flow with a source or destination IP address of the one in question. This naive approach works well for small sites, or sites which have data for only a short period of time. However, for sites with tens of thousands of computers, this is a daunting task. For a site like the Aberdeen Proving Grounds, which hosts a root DNS server, this can quickly become an impossible task, since almost every computer on the Internet talks to the root servers at some point.

Storing the data required to answer that question can itself be a difficult task. The sheer volume will require gigabytes of data per day. However, having information for at least a month is often necessary to get back to the beginning of the compromise. Once the data repository has reached this size it is no longer reasonable to do a linear scan of the data to look for IPs of interest, so some sort of intelligent data storage needs to be employed.

Presenting the raw information of what IPs the compromised computer has talked to may be fine if the number of IPs is a few dozen, but for anything larger this would require too much manual work to determine what is legitimate and what is not. Determining what is "normal" and what is not is a very large research area, one with many interpretations and sometimes contrary viewpoints. Sometimes the concept of "normal" may even change, or how close to "normal" something has to be to be considered normal may change.

Answering the question of, *what is normal?* may require many types of profiling.

As part of this thesis, I have developed some simple but effective techniques for profiling. However, these techniques still leave much room for expansion. Fortunately, the overall system I have developed will make it easier for future work on profiling and for easy integration of these techniques into the system.

2.4 Proposed Approach

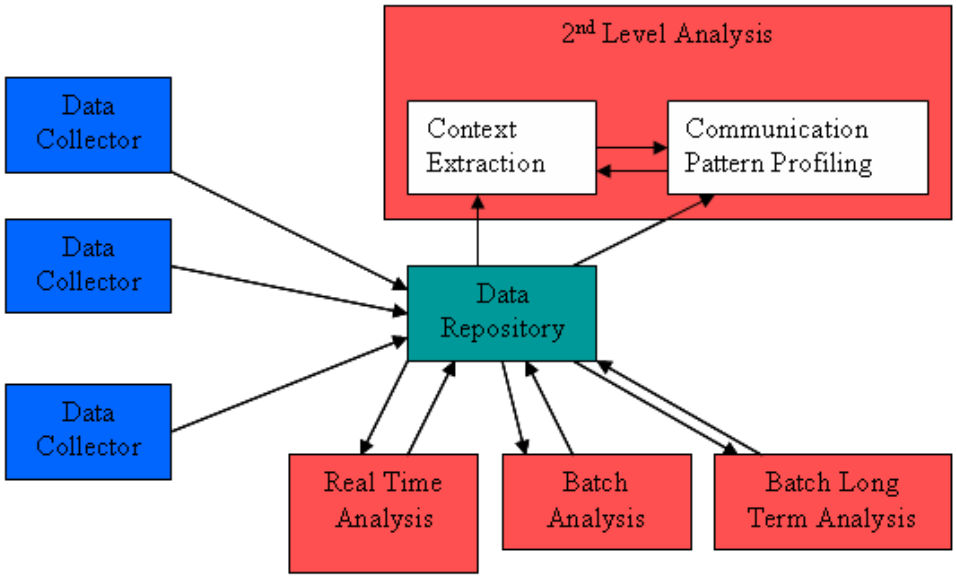


Figure 2.1. Proposed Arcitecture

To address many of the problems with current IDS systemss I propose shifting from a *sensor-based* architecture to a *centralized* architecture. There will be three key components to this new architecture, first **Data Collection**, and second **Data Storage**, these two existing primarily to enable the third component, **Analysis Engines**.

In the proposed architecture, the data collectors will be placed around the net-

work or networks in locations similar to where current IDS systems are. These data collectors will intelligently collect subsets of the network traffic and stream the data back to a centralized data repository.

At the centralized data repository the data is stored and indexed so that storage is efficient and provides quick data access. In addition to providing quick access to data, this system provides APIs for easy access to a richer set of data than simple raw packets, namely communication sessions or other semantically rich types of information.

The security applications component will encompass many types of analysis techniques. Example applications range from simple signature based tools or scan detection, to bot detection, anomaly detection, sequential pattern analysis, social network analysis and most importantly, to tools not yet even imagined.

One of the key advantages of the proposed architecture is that it allows experts to focus their attention on their areas of expertise. Data miners will no longer need to be experts in reassembling networking sessions or designing and accessing large databases. Networking researchers will no longer have to worry about how to interact with their operating system and hardware issues involved with data collection. By using a modular design and implementation, researchers are able to leverage the work of their colleagues to achieve a greater overall impact.

Another key advantage is that this proposed system will be able to easily answer questions which are currently extremely difficult or impossible to answer in the current sensor-based approach. Four of the six questions mentioned earlier, *who, what, when, and where,* can be easily answered with this proposed architecture. Answering the *how* question can be addressed with follow up analysis. This system can also give insight that may help a person answer the *why* question.

Building the above system has involved addressing and solving many challenges

in each of the three main areas. The data collector needed to be flexible to allow for different types of collection and varying quantities and granularities of collection. The data collector has to operate at line speed on high-speed networks, which requires great care in designing and implementing to achieve maximum efficiency. The data collector should ideally be reconfigurable on the fly, so updates can immediately take effect without adversely affecting performance or collection thoroughness.

The data storage system needed to be able to allow quick access to a massive amount of information. Not only must the entire system cope with potentially terabytes of new information every day, but it needs to provide very quick access to the data being analyzed or requested. Complicating the problem was the fact that much of the data access will be random accesses in support of providing contextual information for analysis.

2.5 Framework Contributions

This framework makes contributions in several areas of network security. The first area of contribution is **Data Collection**, the second is **Data Storage**, the third **Analysis Engines**.

Contributions in the area of **Data Collection** include a software-based data collection with the ability to:

- Collect network data at wire speed.
- Build and collect network sessions
- Collect meta information about a session
- Collect variable amount of payload information about a session

Contributions in the area of **Data Storage** include a system that can:

- Efficiently store large amounts of networking data
- Quickly access networking data and meta information

Contributions in the area of **Analysis Engines** include only a sampling of the potential applications that can be developed using the proposed system. Some applications developed as part of this thesis:

- A scan detector with significantly better detection coverage than existing methodologies, able to detect very slow scans, while maintaining a near zero false positive rate.
- A framework for extracting historical information about a computer or network and its network activities.
- A framework combining communication graph analysis and profiling to determine the larger context of a network event.

Chapter 3

Lightweight "Stateful" Network Data Capture

3.1 Introduction

Recent years have shown an increase in research in the area of high-speed data collection for networking and network intrusion detection. One of the most successful researchers in this area is Luca Deri, [14, 12, 16] who has created ntop for collecting netflows [55], and his most recent publication on high-speed packet filtering [15]. The other prominent researchers in this area are Vern Paxson and Stefan Kornexl with their work on Bro and building a time machine, [39, 31]. The records produced by these hardware and software-based data collectors then become the input for many research, open-source and commercial network Intrusion Detection Systems, as well as being used for network health monitoring.

A problem lies in the fact that the data collection tools tend to be either too rudimentary, e.g. raw packet capture, or too heavy machinery for high-speed networks. For example, Bro does a detailed analysis of the networking and application protocol, looking for deviations in each session. This session building can reconstruct

the session as it happened on the network, handle fragmented packets, and detect which packets were retransmitted. Unfortunately, this cannot be done on commodity hardware even for medium-sized networks and it has been suggested that special purpose hardware be designed to implement this analysis, [40].

In addition to being difficult to collect information as detailed as that available in Bro, most tools do not need that level of detail. For example, many anomaly detection tools, such as MINDS, need only the network sessions, so using Bro's session inspection to construct sessions would be overkill.

3.2 Motivation

There are currently two main types of raw data used in the networking area, tcpdump [49] and netflows [55]. Tcpdump contains the raw payload of each packet, while netflows contain higher level information about the volume of traffic transferred between a pair of computers. Each of these data types has its advantages and disadvantages, and major limitations.

Tcpdump Tcpdump [49] is a program built around the libpcap library. This captures packets from a networking interface and brings the packets through kernel memory into user memory. There simple filters can be applied to determine if this packet should be stored. If the packet is to be stored, it is written out with a small header, followed by a set amount of bytes beginning at the link layer. The amount of bytes to be recorded is a global value set at runtime and defaults to only 68 bytes.

While tcpdump and libpcap provide a lot of functionality, they lack several valuable features.

- First, recording a constant amount of bytes per packet can lead to an inefficient use of resources. For example, a given type of analysis or research project may

require access to all of the bytes for a specific service on a specific host, while only a minimal amount of information is needed from other services or hosts. This creates a significant overhead, wasted storage, and wasted processing capabilities.

- Second, there is no notion of a session; each packet is treated individually. While this may be adequate for some types of analysis, it limits the types of analysis that can be done without additional data preparation. For example, it is possible to build sessions out of the recorded packets; however, current tcpdump implementations require the full storage of every packet. So if what was needed was the first 20k bytes of every session, the full contents of every packet would need to be collected and then processed to determine what is the first 20k bytes of a session. This not only wastes storage, but also requires each researcher to know how to reconstruct network sessions, wasting not only researchers' time, but also possibly limiting the number of people working in the field. A recent paper by Kornexl discussing the heavy-tailed nature of networking traffic shows just how much storage could be saved. His data showed that keeping only the first 20 KB of a session would capture everything from 85% of the sessions, but would remove 85 or 99% of the data, [31]!

- Third, the performance of tcpdump can be quite poor. One of the main reasons for this is the fact that it is interrupt driven. There is a patch to the Linux kernel which changes how libpcap works to make it use a large ring buffer; this increases the performance significantly [61, 13]. There are also hardware cards which implement much of the libpcap work in hardware [18].

- A fourth challenge with using tcpdump-based data is that there will most likely be missing data. One of the most common reasons for this is that the data

collector simply cannot keep up. The bottleneck can be in the Ethernet card, PCI-bus, memory, kernel, CPU, or hard disk.

- A fifth challenge is that the the data for a given connection may exist in separate files or may not be available at all. This can happen due to a-symmetric routing, or fiber-based networking where each direction of the communication is observed on a different network interface.

Netflow Netflow is a data format which can be collected from most routers, in addition to software-based network sniffers which construct netflows. Netflow is a format developed by Cisco, though the other major router manufactures also support this data collection format. Netflows are analogous to phone call records; they contain high level information with no details of what actually was communicated. Netflows contain the source and destination IPs, source and destination ports, protocol, number of packets, and number of bytes. This information can be very valuable for communication pattern analysis, but netflows also have several deficiencies.

- The first is that the flows are unidirectional, so a two sided conversation will be recorded in two (or more) netflow records. For many types of analysis it is required to know if there was a reply, and how the reply happened. While it is possible to match the initiating and responding netflows, there are limitations; and in real data sets there will often be mismatched or unmatched flows.

- The second problem is that netflow sessions from a router do not fit a session in the traditional networking sense. The flows may arbitrarily be split by the router, and are always split at regular intervals when the old data file is closed and a new data file is opened. This typically happens every 5 to 10 minutes. Once again it is possible to use heuristics to merge the flows into the networking

sense of a flow, and then to try and match the initiating and responding flows so as to get both sides of the conversation linked together. However, this is a heuristic, and experience thus far has shown that there will be mis-matchings and initiator-responder will be incorrectly labeled.

- A third problem, which simply exacerbates the previous ones, is that the time-stamps of the data records may not be totally correct, and the precision is not fine enough to determine which flow happened first. This can make it very difficult to determine initiator-responder status.

3.3 Design

Given the problems associated with the two dominant raw data types and the complexity and hardware requirements of the more sophisticated collection tools, it seems prudent to develop a new data type and collection method which does not suffer from some of these same problems. However, since there is already a lot of infrastructure in place which requires netflows or tcpdump as an input, it is also prudent to maintain backwards compatibility or allow for an easy conversion to the current formats.

The primary problems the proposed approach addresses are those of the netflow data type, namely uni-directional flows and fragmented flows. In addition, this allows the amount of data collected in raw packet form to be assigned on a session level in addition to a per packet basis.

At a high level the design of a tool to construct network sessions vs network flows is fairly simple, though there are a few subtle challenges which required compromises in the design. Simply put, when a packet arrives a lookup into a hash table is done to determine if that 5-tuple, protocol, IPs and Ports, has been observed, as either an initiator or a responder. If so, the number of packets, bytes and timestamps are

updated. If not, a new record is created.

One design point where a compromise was made was in how closely to follow the protocol for building tcp sessions. One extreme is to simply aggregate based on the 5-tuple, similar to how Cisco Netflow does it. The other is to perform a detailed analysis like that done in Bro, [39]. However, the additional load and resource consumption required for this more detailed stateful inspection makes it prohibitively expensive to run on large networks. As was pointed out in Sommer's thesis [48], Bro requires several reboots everyday due to resource exhaustion. When this happens the system must be rebooted and the state of the system is lost as well as any data traversing the network during that time.

Given the problems with the two extremes and our desire to ensure our data collector was able to store the data traversing the network for follow on analysis, which could include the detailed inspection done by Bro, we opted to do some session tracking, though not as detailed or as rigorous as is done in Bro. Thus, the session records collected may not be as accurate in the networking sense as that collected in Bro, but all of it will be recorded. So, in addition to following the 5-tuple, we examine the TCP flags for new sessions that happen to use a 5-tuple already in use. We do not, however, enforce witnessing the 3-step tcp handshake, or any other state.

One other area where, due to system issues, we had to make a compromise was in how long to keep sessions in memory. After studying the raw traffic at several large networks, it was determined that a timeout of 6 minutes would cover 99.99% of the sessions, while only keeping a few hundred thousand sessions active at anytime.

3.4 Implementation on Commodity Hardware

One of the considerations that went into the design was making the code portable and easily compilable on different platforms. For this reason the code was written in C++, using the Standard Template Library [54]. The code made use of the libpcap library, [49] for the interface to raw packets. The code was throughly tested with gcc compiler version 2.9 and greater and works on Linux, solaris, and BSD collection platforms.

There are two optional performance enhancing components that the data collector can make use of. The first is a software patch to the Linux kernel which changes the way packets are handled, [61]. Essentially, this changes the kernel so that the kernel is interrupted every time a packet is received to deliver it to the collection program. Instead, the packet is placed in a ring buffer and then the libpcap operates on the data in this buffer. This enhancement can increase performance by almost an order of magnitude.

The second optional performance enhancement is the use of a network capture card that does some of the work done by libpcap in hardware. These cards, made by Endace [18], can collect on networks up to 10 Gigabit.

3.5 Pseudocode

Algorithm 3.5.1: SessionBuilding(P)

for each $p \in \mathcal{P}$

do if $p.protocol == TCP$

then HandleTcpPacket(p)

else if $p.protocol == UDP$

then HandleUdpPacket(p)

else if $p.protocol == ICMP$

then HandleIcmpPacket(p)

else HandleOtherPacket(p)

Algorithm 3.5.2: HANDLETCPPACKET(p)

$C =< SrcIP, SrcPort, DstIP, DstPort >$

$S =< DstIP, DstPort, SrcIP, SrcPort >$

if $p.syn\_flag\_only$

then
- **if** $old\_session \leftarrow tcp\_sessions.find(C)$
  - **then**
    - $Write(old\_session)$
    - $tcp\_session.remove(C)$

if **not** $tcp\_sessions.exists(C)$ **and** **not** $tcp\_sessions.exists(S)$

then
- $CreateNewRecordForSession(C)$
- $break$

if $existing\_session \leftarrow tcp\_sessions.find(C)$

then
- **if** $existing\_session.end\_time + TIMEOUT < current\_time$
  - **then**
    - $Write(existing\_session)$
    - $tcp\_session.remove(C)$
    - $tcp\_session.insert(C)$
    - $break$
  - **else**
    - $Update(existing\_session, P)$
    - **comment:** Update values like end_time, client packets, client bytes
    - $break$

if $existing\_session \leftarrow tcp\_sessions.find(S)$

then
- **if** $existing\_session.end\_time + TIMEOUT < current\_time$
  - **then**
    - $Write(existing\_session)$
    - $tcp\_session.remove(S)$
    - $tcp\_session.insert(S)$
    - $break$
  - **else**
    - $Update(existing\_session, P)$
    - **comment:** Update values like end_time, server packets, server bytes
    - $break$

Algorithm 3.5.3: HANDLEUDPPACKET(p)

$C =< SrcIP, SrcPort, DstIP, DstPort >$

$S =< DstIP, DstPort, SrcIP, SrcPort >$

if not $udp\_sessions.exists(C)$ **and not** $udp\_sessions.exists(S)$

then
- $CreateNewRecordForSession(C)$
- $break$

if $existing\_session \leftarrow udp\_sessions.find(C)$

then
- **if** $existing\_session.end\_time + TIMEOUT < current\_time$
  - **then**
    - $Write(existing\_session)$
    - $udp\_session.remove(C)$
    - $udp\_session.insert(C)$
    - $break$
  - **else**
    - $Update(existing\_session, P)$
    - **comment:** Update values like end_time, client packets, client bytes
    - $break$

if $existing\_session \leftarrow udp\_sessions.find(S)$

then
- **if** $existing\_session.end\_time + TIMEOUT < current\_time$
  - **then**
    - $Write(existing\_session)$
    - $udp\_session.remove(S)$
    - $udp\_session.insert(S)$
    - $break$
  - **else**
    - $Update(existing\_session, P)$
    - **comment:** Update values like end_time, server packets, server bytes
    - $break$

Algorithm 3.5.4: HANDLEICMPPACKET(p)

$C =< p.SrcIp, p.icmp\_type, p.DstIp, p.icmp\_code >$

if $p.icmp\_type$ **not** $DestinationUnreachable$

then $\begin{cases} Write(C) \end{cases}$

if $p.reply\_header.protocol == TCP$

then $\begin{cases} session \leftarrow tcp\_sessions.find(C.reply\_header) \\ Write(Session) \\ tcp\_sessions.remove(C.reply\_header) \\ Write(C) \end{cases}$

else if $p.reply\_header.protocol == UDP$

then $\begin{cases} session \leftarrow udp\_sessions.find(C.reply\_header) \\ Write(Session) \\ udp\_sessions.remove(C.reply\_header) \\ Write(C) \end{cases}$

else $\begin{cases} session \leftarrow other\_sessions.find(C.reply\_header) \\ Write(Session) \\ other\_sessions.remove(C.reply\_header) \\ Write(C) \end{cases}$

Algorithm 3.5.5: HANDLEOTHERPACKET(p)

$C =< SrcIP, DstIP >$

$S =< DstIP, SrcIP >$

if not $other\_sessions.exists(C)$ **and not** $other\_sessions.exists(S)$

then { $CreateNewRecordForSession(C)$; $break$

if $existing\_session \leftarrow other\_sessions.find(C)$

then {

if $existing\_session.end\_time + TIMEOUT < current\_time$

then { $Write(existing\_session)$; $other\_session.remove(C)$; $other\_session.insert(C)$; $break$

else { $Update(existing\_session, P)$; **comment:** Update values like end_time, client packets, client bytes ; $break$

if $existing\_session \leftarrow other\_sessions.find(S)$

then {

if $existing\_session.end\_time + TIMEOUT < current\_time$

then { $Write(existing\_session)$; $other\_session.remove(S)$; $other\_session.insert(S)$; $break$

else { $Update(existing\_session, P)$; **comment:** Update values like end_time, server packets, server bytes ; $break$

3.6 Performance Evaluation

In this section the performance of the data collector is presented using two different data sources. The first is the University of Minnesota where a thorough analysis of the collection capabilities and scalability was performed. The second is anecdotal

evidence from the US Army Research Laboratory.

Runtime Analysis In order to understand where time is spent in the session-building tool, the software was compiled to use the Linux utility gprof. This utility samples where the program counter is as the program runs and then uses this information to determine the number of times a function is called, and how long the function runs. Table 3.2 shows the report of functions consuming most of the time while building the sessions. These performance numbers are from processing a file containing five minutes worth of data from the University of Minnesota on a Linux computer running at 2.0 Ghz.

The input data consisted of 104,468 source IP addresses, 126,332 destination IP addresses, and 484,319 sessions. The distribution of sessions for each protocol can be seen in Table 3.1.

Protocol	Sessions
IP	73771
ICMP	46891
TCP	170497
UDP	193018
GRE	18
ESP	37
SKIP	18
PIM	69

Table 3.1. Number of sessions for each protocol

This shows that finding an existing session in the tables is by far where most of the runtime is spent, accounting for 70% of the runtime. After this, it is the garbage collection routines which perform scans over all of the sessions, looking for expired sessions which use the next largest amounts of time, but these account for only a single digit percentage of runtime.

% time	Cum. Seconds	Self Secs	Calls	ms/call	Name
70.06	2.20	2.20	2980616	0.00	find_session
6.69	2.41	0.21	54	3.89	cleanup_tcp
5.41	2.58	0.17	448708	0.00	lower_bound
5.10	2.74	0.16	37	4.32	cleanup_udp
3.18	2.84	0.10	543778	0.00	process_tcp
2.55	2.92	0.08	NA	NA	main
1.91	2.98	0.06	380748	0.00	rule_matching
1.59	3.03	0.05	379079	0.00	process_udp
1.27	3.07	0.04	3	13.33	delete_session
0.96	3.10	0.03	419342	0.00	write_session
0.64	3.12	0.02	448704	0.00	insert_session
0.32	3.13	0.01	959307	0.00	process_other_flow
0.32	3.14	0.01	1	10.00	write_open_sessions

Table 3.2. Time required for session building functions on 5 minutes of University data

These experiments strongly indicate that the performance of the software based data collector is sufficient for monitoring high speed networks and that it outperforms other software and hardware based data collectors.

3.6.1 Real World Performance

If the data to be collected is sufficient for the follow on tasks, the important performance criteria of the collector are that it can work on real networks, and under real-world conditions. One such location that I was fortunate enough to use as a test site is the Army Research Laboratory's Center for Intrusion Monitoring and Protection (ARL-CIMP). Using their infrastructure and test environment, I was able to collect data on small networks with only a few tens of users, to very large multi-gigabit connections with hundreds of thousands of users. The performance of the collector was more than adequate on even the busiest networks.

On the smallest of networks, which is where the code was tested initially, the program was not even listed on the program Top, a Linux utility that displays current

programs running and their memory consumption. The CPU required on networks moving a few megabits per second was less than .1% of the CPU. The memory requirements were only a few megabytes, which is the size of almost any program running on Linux.

On the larger networks, which is where a collector that balances the rudimentary capture of tcpdump and the detailed analysis of Bro is most needed, the performance again was impressive. During the busiest times of the day the, CPU utilization would spike to 8 or 9% of the CPU briefly, then drop back to 3-4%. These spikes occur when the collector performs garbage collection and flushes expired sessions. The memory usage required to monitor the largest of networks was quite low, requiring only 100 MB of memory of maintain all of the state necessary.

Determining if any data was missed has proven to be a difficult task. Libpcap has a function that can report how many packets were dropped, though this assumes the packet makes it to libpcap and isn't lost by the network card. The number of lost packets reported by libpacp has always been 0 in every log file so far. An alternative is to compare the results to another data collector monitoring the same network to determine what is the overlap and the difference between the two.

When compared against another data collection computer on the same network, the resulting output files of the two collectors were identical. This strongly indicates that all of the data was collected. If there had been some packets missed, it would seem highly unlikely that of the billions of packets traversing the network during the day the evaluation was occurring, both collectors dropped the same packets.

Chapter 4

Storage and Retrieval of Networking Data

4.1 Background and Motivation

Working in the areas of network security and networking research often requires access to a large volume of information, and often analyzing only a small subset of that large volume of information. For example, many IDS systems need access to each packet or session crossing that network. However, follow on analysis routines may look for traffic involving only a single IP address.

There are many systems currently available that store alerts from IDS sensors. These databases and front ends work well for reviewing the alerts generated and getting aggregate information such as number of alerts of type X, or type X alerts on a given network. However, these systems typically do not have access to the information traversing the network, and in cases where the traffic is available, it is usually only the traffic directly involved in generating the alert. These systems do not provide the information necessary to determine what else that computer in question was doing, or what legitimate activities are going on on a network.

There are also systems available for looking at high level aggregate information about network activities. For example, commercial tools from Arbor Networks or open source tools from RRDtool are used for looking at bandwidth usage, number of connections in or out of a network, or number of connections on certain ports or protocols [2, 43]. The databases underlying these front end tools, however, are not well suited to providing access to the details of the communications on the network.

The type of system that is lacking in the networking area right now is a system that can access the payload of any network communication, retrieve the communications of any IP address (which IPs did this one communicate with, when, and how much data was moved), provide high level statistics about a network (how many active IP addresses, servers, clients), as well as provide bulk access to every packet or session from a network for some time window.

4.2 Design and Implementation

Designing a system that can meet these requirements is made easier by using the data collector described in Section 3. Since this collector provides information such as session level statistics, as well as the raw payload associated with those sessions, what remains to be done is to design a system capable of storing and accessing this information.

The actual storage of the information on disk requires some thought to be given as to how the data will be accessed. To support existing IDS tools, this system must provide access to all of the sessions or payload information from any sensor for any time window. For example, Snort needs access to all of the raw payload information, and ideally this would be done in real-time to shorten the response time of the analysts. Some batch analysis tools, such as MINDS, need access to an entire

time window's worth of data at once, typically an hour.

To support the analysis tools requiring direct access to the data, a simple directory structure like /site/sensor/year-month-day-hour.minute.second.filetype works well. Files for a given time window are easy to locate for any site and any sensor on that site. This structure also makes it easy to select ranges of data or ranges from multiple sites for longer term analysis.

Real-time analysis tools based on libpcap can be easily changed to operate on files fed into standard input. A simple change to the libpcap library to allow reading in from standard input is then automatically available in any tool that uses this, which is nearly all packet-based analysis tools. As the data files are written to disk, the Linux tail program can be used to take the updates and feed them into the standard input of the analysis routines. This allows real-time analysis to occur with minimal delay when compared to the tools collecting the data themselves. The only delay is the time to transfer the files from the sensor to the disk, which can be done in milliseconds even across the continent.

This directory structure also allows permissions to easily be set on a site or sensor basis. For example, at a University the head of networking security could have access to all of the data. However, networking security at a certain campus might have access to only that campus. Additionally, an administrator for a department might have access to sensors in only that department. Permissions on individual filetypes can also be set, so researchers might have access to session level statistics but not access to the raw payloads.

Now that the directory structure is set for the storage of the raw data and for access by the 1st level analysis routines, a database for storage of the sessions can be designed to provide quick access to information about specific IP addresses, ports, and protocols. One of the first questions that needed to be answered when designing

this database was, "What queries will be issued?" This can be further broken down into how often will these queries be issued, and how fast do they need to be. The answers will help guide the indexing of the data.

While working with the ARL-CIMP, it was observed that most of the 2nd level analysts' time was spent looking for all traffic involving a specific IP address. This indicates that this is one of the most common queries that will be issued and should be made to be as fast as possible. A naive and straight-forward way to solve this is to store all the sessions in a single table with indices on SrcIP and on DstIP. This may suffice for small networks, but when the number of records gets into the hundreds of millions or billions, performance can become a problem. After several design and implementation iterations, the analysts revealed that they typically were not interested in what an IP address did a month ago, and only wanted to know what happened in the last few days or weeks.

Armed with this information, the structure was redone to use the partitioning feature of Oracle. The table was partitioned based on the start of the session. This way when an analyst queried for what an IP address did today, only that day's data was examined. A query for a single day is now as fast as the initial naive design. A query for multiple days takes proportionately longer, though, since many database servers have multiple CPUs, the query can be done in parallel, forking one query for each day.

A simple table for storing the sessions in a MySQL database.

```
# Table: 'sessions'
#
CREATE TABLE `sessions` (
  `flowid` int(10) unsigned NOT NULL auto_increment,
```

```
  `start` datetime NOT NULL default '0000-00-00 00:00:00',
  `end` datetime NOT NULL default '0000-00-00 00:00:00',
  `sip` int(10) unsigned NOT NULL default '0',
  `sport` mediumint(6) NOT NULL default '0',
  `dip` int(10) unsigned NOT NULL default '0',
  `dport` mediumint(6) NOT NULL default '0',
  `cpackets` int(10) unsigned NOT NULL default '0',
  `cbytes` int(10) unsigned NOT NULL default '0',
  `spackets` int(10) unsigned NOT NULL default '0',
  `sbytes` int(10) unsigned NOT NULL default '0',
  `protocol` tinyint(3) NOT NULL default '0',
  `flags` tinyint(2) unsigned NOT NULL default '0',
  `session_closed` ENUM('0','1') NOT NULL default '0',
  PRIMARY KEY  (`flowid`),
  KEY `dip` (`dip`),
  KEY `triple` (`sip`,`dport`, `dip`)
) ENGINE=MyISAM;
```

The above table has two indexes created on it. One is of the destination IP address, the other is on multiple columns, source IP, destination port, and destination IP. Since MySQL can use partial keys, there was no need to create an index on just the source IP address, since it can use the triple index. However, if there are many queries being issued where only the source is given, creating a separate index on just that may be beneficial to keep the index size smaller.

4.3 Performance Evaluation

In this section several types of queries that would be issued regularly by network security analysts are evaluated. The evaluation was done on one day's worth of data collected at the University of Minnesota on Feb 17 2007. The data consisted of 114,945,093 sessions, with 5,469,035 unique source IP address, and 4,121,690 unique destination addresses.

One of the most common tasks performed by network security analysts while investigating a possible compromised computer is to look at the recent activity involving that computer. Accomplishing this can be achieved by executing a simple SQL query. A typical query would look like:

```
Select * from $TABLENAME where dip = $DIP
```

where $TABLENAME is replace with the name of the table, and $DIP is replaced with the destination IP address to be queried for.

In table 4.1 one can see the number of sessions returned for a query of a specific IP address, using a query like the one above. In table 4.2 the time taken to look up an IP address is listed for a given data set size. This table also shows the time taken to "grep" for an IP address in the raw data file. This "grep" method is the approach typically taken by analysts such as those at the University of Minnesota and the US Army prior to using the storage and retrieval system outlined in this thesis.

IP	5-min	1-hour	4-hour	1-day
128.101.101.101	8435	55834	213525	1267064
128.101.35.207	103	1466	5360	27157
128.101.35.35	2	9	13	184
0.0.0.5	0	0	0	0

Table 4.1. Number of sessions found with the IP address

IP	5-min	1-hour	4-hour	1-day
128.101.101.101	0.010935	0.073421	0.272299	1.582071
128.101.35.207	0.000760	0.002301	0.007166	0.036744
128.101.35.35	0.001028	0.000684	0.000667	0.001019
0.0.0.5	0.000567	0.000734	0.000599	0.000576
Using grep	6.599	58.387	191.831	1565.932

Table 4.2. Number of seconds to lookup an IP address

The IP addresses queried for were chosen to be representative of differing activity levels. One IP address was not present at all, 0.0.0.5, one is a workstation in the CS department. One is a web server in the cs department, and one is the University's name and time server. The name and time server is the busiest server at the University in terms of number of sessions per day.

From the above tables we can see that having the sessions stored in an appropriately indexed database is significantly quicker than performing linear scans of the raw data.

Another common task performed by network security analysts is to look for connections to or from a specific network. This can be acomplished by issuing a range query such as this one:

```
Select * from $TABLENAME where sip > $IPSTART and sip < $IPEND
```

or

```
Select * from $TABLENAME where dip > $IPSTART and dip < $IPEND
```

The first of the queries above retrieves connections initiated from a network, and the second retrieves connections destined to the range of computers specified.

Another activity common to analysts is to look for activity from a set of IP addresses, this could be accomplished by issuing seperate queries for each IP address, or can be done in a single query such as:

IP	5-min	1-hour	4-hour	1-day
128.101.35.*	310	4276	15385	91139

Table 4.3. Number of sessions found for a specific source network

IP	5-min	1-hour	4-hour	1-day
128.101.35.*	0.01	0.11	0.14	0.89

Table 4.4. Number of seconds to retrieve sessions from a source network

```
select * from $TABLENAME where dip in ($IP1, $IP2, $IP3...)
```

This type of query is often issued in the context extraction process, described in Chapter 5.

4.3.1 Building Activity Graphs

Researchers attempting to identify bots, worms, and black or grey IP space often would like to look at the inbound and outbound activity for IPs, or networks. One common approach when building these plots is to make linear scans through the dataset to build up this information. While this works for small amounts of data, it can become time consuming as dataset sizes increase. It is not uncommon for such scans to take tens of minutes, or even hours to complete when analyzing data from a few days on a large network like the University of Minnesota. This delay limits the number of experiments researchers can perform in a day and thus slows down research. The storage system described here is suited quite well to answering these types of questions.

Recently, researchers at the University of Minnesota were studying the inbound and outbound traffic from networks on the University, attempting to identify black and grey network space, then using this to identify bots, worms, or scanners, [27]. This work was taking days to build graphs for analysis. After putting the data being

IP	5-min	1-hour	4-hour	1-day
128.101.35.*	397	5889	20329	155162

Table 4.5. Number of sessions found for a specific destination network

IP	5-min	1-hour	4-hour	1-day
128.101.35.*	0.01	0.01	0.21	1.4

Table 4.6. Number of seconds to retrieve sessions for a destination network

analyzed into the storage system described here, and running a query like the one below, the data can be retrieved in 0.05 seconds on average, and easily made into charts like those in figure 4.2 and 4.1.

```
select sip, start, count(*), floor((extract(hour from start)
* 60 + extract(minute from start)) / 5) as window from
$TABLENAME force index (triple) where sip > NETWORK_START
and sip < NETWORK_END and start > '2006-02-08' and
start < '2006-02-09' group by sip, window;
```

The above query returns a table like that in table 4.9. The run time for a query like this is only a few milliseconds (0.05 seconds on average). The data retrieved could be displayed in a plot instead of a table which would look like figure 4.2 for outbound activity, and 4.1 for inbound activity.

Being an SQL database, there are obviously many other types of queries that can be issued. This section detailed just a few of the potential queries that are likely to be used in the course of network security operations, or research.

IP	5-min	1-hour	4-hour	1-day
Set of 25 IPs	14	47	102	453

Table 4.7. Number of sessions found for a set of destination IP addresses

IP	5-min	1-hour	4-hour	1-day
Set of 25 IPs	0.01	0.01	0.01	0.01

Table 4.8. Number of seconds to retrieve a sessions for a set of destination IP addresses

IP	Time	Number of Connections	Window
X	2006-02-08 00:03:59	1	0
X	2006-02-08 00:08:00	2	1
X	2006-02-08 00:12:00	2	2
X	2006-02-08 00:40:00	3	8
X	2006-02-08 00:46:00	2	9
X	2006-02-08 00:57:45	1	11
X	2006-02-08 01:18:00	1	15
X	2006-02-08 01:22:17	4	16
X	2006-02-08 01:26:52	1	17
X	2006-02-08 01:31:51	1	18
X	2006-02-08 02:00:00	3	24
X	2006-02-08 02:06:00	2	25
X	2006-02-08 02:33:38	1	30

Table 4.9. Number of connections initiated by an IP address in 5 minute windows

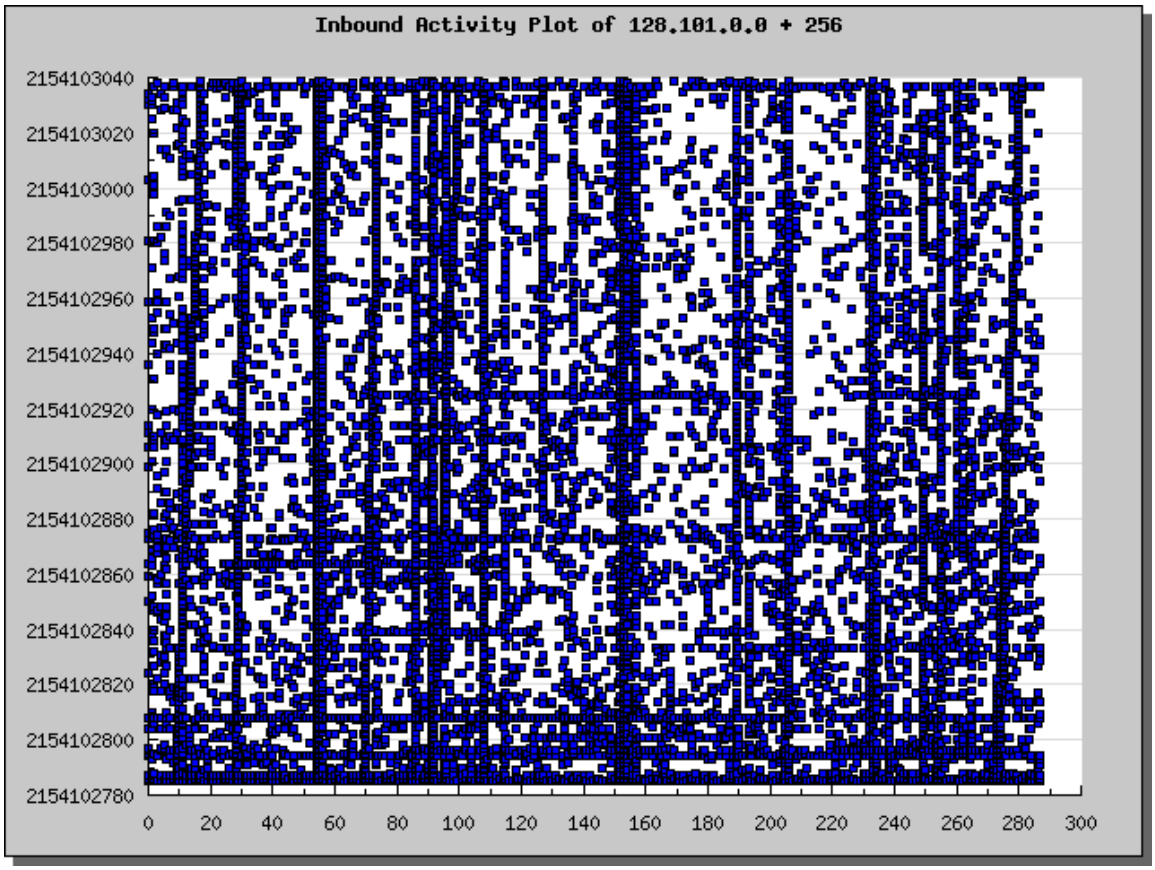


Figure 4.1. Inbound Activity Plot for the 128.101.0.X network on Feb. 8th 2006

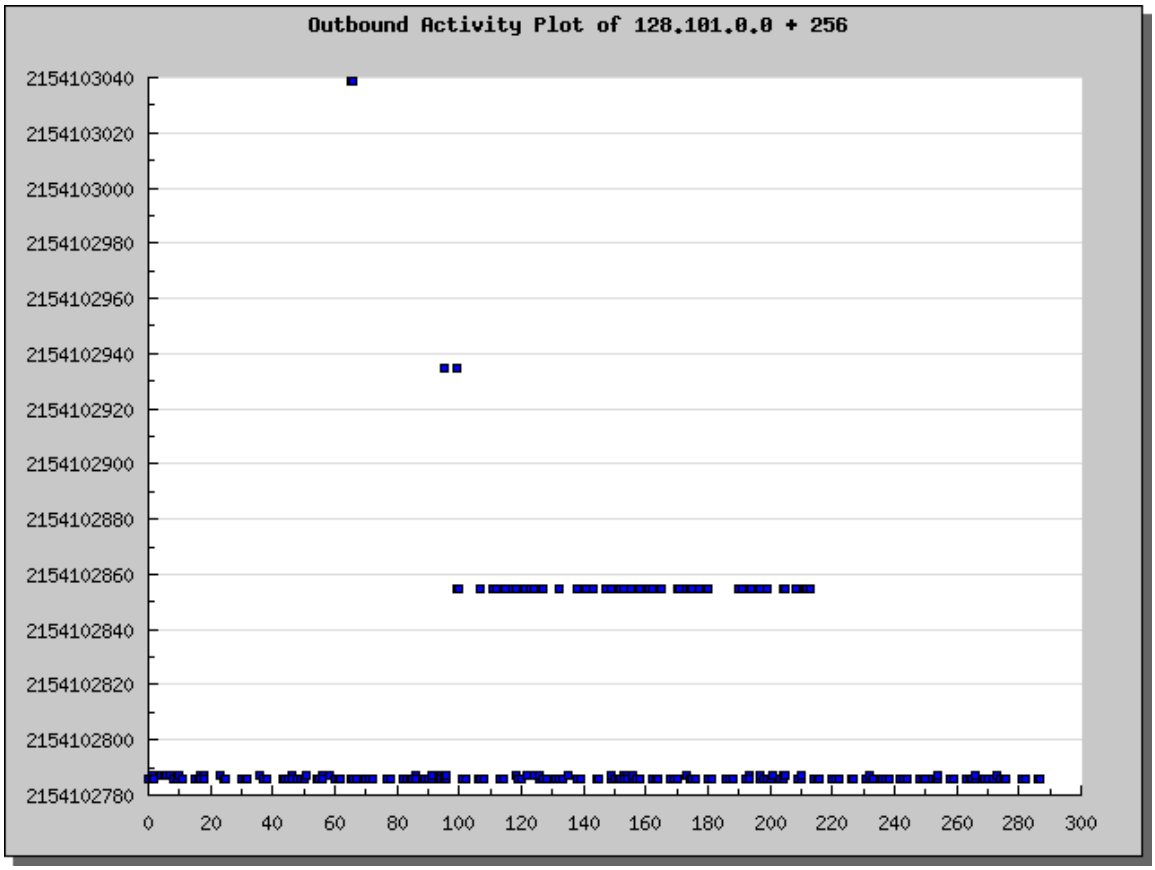


Figure 4.2. Outbound Activity Plot for the 128.101.0.X network on Feb. 8th 2006

Chapter 5

2nd level Analysis

5.1 Introduction

As the threat of attacks by network intruders increases, it is important to correctly identify and detect these attacks. However, network attacks are frequently composed of multiple steps, and it is desirable to detect all of these steps together, as it 1) gives more confidence to the analyst that the detected attack is real, 2) enables the analyst to more fully determine the effects of the attack, and 3) enables the analyst to be better able to determine the appropriate action that needs to be taken. Traditional IDSs face a major problem in dealing with these multi-step attacks, in that they are designed to detect single events contained within the attack and are unable to determine relationships between these events.

Many alert correlation techniques have been proposed to address this issue by determining higher level attack scenarios [6, 8, 35, 38, 56]. However, if the data that is being protected by the network is highly valuable, an attacker can spend more time, money, and effort to make his attacks more sophisticated in order to bypass the security measures and avoid detection. Attackers then may use techniques to prevent their attacks from being reconstructed, such as making their attacks *highly distributed*,

avoiding standard pre-defined attack patterns, using *cover traffic or "noisy" attacks* to distract analysts and draw attention away from the true attack, and attempting to avoid detection by signature-based schemes through the use of *novel attacks or mutation engines* [60]. In these more sophisticated attacks, many of these correlation techniques face certain difficulties. In the case of matching against attack models [6] or analysis of prerequisites/consequences [8, 35, 38, 56], attackers can (and often do) perform unexpected or novel attacks to confuse the analysis. In addition, the information for these schemes must be specified ahead of time, and thus the analyst must be careful to specify complete information and not miss any possible situation.

Furthermore, these correlation approaches, as well as traditional IDS techniques, suffer from a fundamental problem in that they try to achieve both a low false positive rate and a low false negative rate simultaneously. These goals, however, are inherently conflicting. If the mechanism used is set to be too restrictive then there will be many false negatives; yet if the mechanism is set to be less restrictive, many false positives will be introduced. Also, if signature-based systems such as Snort [47] are used with many rules, too much time will be spent processing each packet, resulting in a high rate of dropping packets [44]. If these dropped packets contain attacks, then they will be missed. While some of the approaches have techniques to deal with missed attack steps [6, 35, 38], they cannot handle the absence of many of the steps in the attack.

Contribution In this chapter, we propose an analysis framework that addresses this tradeoff between false positives and negatives by decomposing the analysis into two steps. In the *first step*, the analysis is performed in a highly restrictive fashion, which selects events that have a **very low false positive rate**. In the *second step*, these events are expanded into a complete attack scenario by using a **less restrictive analysis**, with the condition that the events added are related somehow to the events

detected in the first step. We describe how this framework is suitable for this problem as it addresses the tradeoff between false positives and false negatives. In addition, our framework is 1) flexible, as it allows the analyst to exercise control over the results of the analysis, 2) designed to be modular and extensible, and thus makes it easy to improve the individual components of the analysis and incorporate new sources of data. We also implemented and evaluated our framework on a dataset that contained several attack scenarios, and we were able to successfully detect the majority of the steps within those scenarios.

Organization The remainder of this chapter is organized as follows. In Section 5.2 we describe our framework. In Section 5.3 we describe our implementation of this framework and show its experimental results in Section 5.4. Next we discuss whether the framework achieves the goals set forth in the design and discuss the limitations of our approach in Section 5.5. We then describe some areas of related research in Section 5.6 and draw some conclusions and outline directions for our future work in Section 5.7.

5.2 Framework Design

The goals for our analysis framework are as follows: First, the system should address the inherent tradeoff between false positives and false negatives. Second, the system should be able to detect the majority of the steps contained within an attack and make connections between these steps to form the attack scenario. For this we assume that at least one step in the attack is visible (if none of the attack steps are visible to any lower level IDS, and thus the attack is perfectly stealthy, then we will be unable to detect the attack). Third, our analysis framework should provide high coverage of attacks (meaning that most or all of the attacks are detected). Finally, the system

should be modular by design, thus making it simple to incrementally improve our approach.

The main challenge faced in designing this kind of system is balancing false positives and false negatives. To address this problem, our analysis framework is composed of two main steps. The first step, *Anchor Point Identification*, is focused on detecting a set of events (anchor points) in a very restrictive fashion, such that the set contains very few false positives. However, this will inevitably result in a large number of missed attack steps. To deal with this, the second step, *Context Extraction*, relaxes the restrictions conditionally; for a (potential) attack step to be examined in this step, it must meet the lower requirements as set by the detection mechanism, and it must also be connected in some way to an event captured in the first step. The overall framework is shown in Figure 5.1. Note that in Figure 5.1 there are three steps, where the third step, *Attack Characterization*, is concerned with giving semantic meaning to the steps in the overall attack scenario, as detected by the first two steps. This step is out of scope for this thesis, and thus it is not addressed in the description of our framework. In addition, the analysis scheme incorporates domain specific knowledge to further refine the results, which it does by keeping a human analyst in the loop. The analyst can control the output of Anchor Point Identification and Context Extraction by specifying the sensitivity of the tools which they utilize or applying domain knowledge in the rules that are used.

In addition, the analyst can control his view in that he can specify the events that he is interested in seeing. For example, if the analyst is securing a specific machine that contains important data, he can set that machine to be the anchor point and search for relevant context that is related to that machine; or if the analyst knows about a certain activity that occurred on the network, or has a list of known bad hosts in a blacklist, he can specify the hosts involved in that activity.

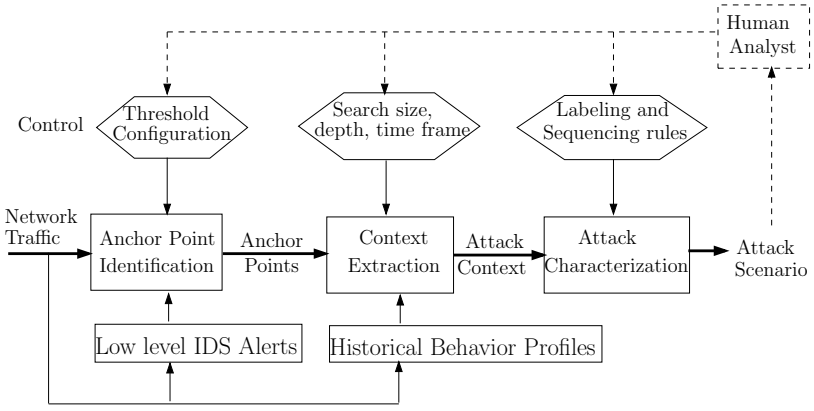


Figure 5.1. The different phases of the analysis framework

5.2.1 Anchor Point Identification

The first phase of the multi-step analysis involves the identification of starting points (*anchor points*) for analysis. This is done by taking a set of low-level IDS alerts from one or more (preferably independent) sources and selecting from this set a number of anchor points, such that we have high confidence that the set contains very few false positives. This can be done in many ways. One way is to use a single IDS configured to operate in a very restrictive manner, resulting in a high confidence yet incomplete set of attack events. Another way of doing this is through correlation techniques. It is well known that if an alert can be correlated with many other alerts, we can be more confident that this alert corresponds to a true positive [35]. Thus, in this manner, alerts from multiple sources can be combined together, where only the alerts which have high confidence are selected. However, there is a difference between the goal of this step and the goal of traditional alert correlation techniques. The difference is that we are not trying to balance false positives versus false negatives. Instead, Anchor Point Identification attempts to aggressively reduce false positives while maintaining high coverage of attack scenarios (where an attack scenario is considered "covered" if at least one attack event in the scenario is selected in this step). The low false positive

requirement is needed to ensure that the subsequent context extraction starts from a highly trusted base and thus can focus on reducing false negatives. Because high attack coverage can accommodate high false negatives, this challenge is a relaxation of the more stringent requirement on traditional techniques that require low false positives and low false negatives simultaneously.

5.2.2 Context Extraction

The anchor points generated in the previous step are comprised of events in which there is high confidence that they are part of an attack. The *Context Extraction* step generates a suspicious context around these anchor points, both temporally and spatially. This step detects events related to the anchor points which are also anomalous or suspicious, but not enough so to be detected by the previous step. The goal of this phase is to add to the context only those activities that are part of the attack, thus filling in the attack steps missed by the previous step, while keeping the low false positive rate achieved by the Anchor Point Identification. This is done by relaxing the restrictions conditionally, i.e. "lowering the bar", but only for those events that are connected somehow to an anchor point.

The major requirement for this step is some type of ranking for each network connection. One way this is accomplished is by an anomaly detection system. In this type of system, all connections are ranked according to how anomalous they are as compared to all other network connections, and this is typically done using data mining techniques. This can also be done by building historical behavior profiles for each host, determining which machines are servers and clients for particular services. When using historical behavior profiles, connections would be added to the context if they deviated from the historical behavior profiles for the hosts that they involved, for example, if a web server started initiating connections which it had never done

before. This must be done carefully, however, for example, in the case of peer-to-peer connections, which can be difficult to profile. If this type of traffic is not carefully profiled then the context can expand rapidly, effectively invalidating the result. One way to deal with this is to use peer-to-peer detection techniques [29] and ignore the peer-to-peer traffic when profiling.

This step also makes use of domain knowledge in the form of rules. Certain behavior patterns are known to be signs of malicious activity. For example, attackers often scan a network on a particular port to look for vulnerable machines. These scans most often result in failed connection attempts, as most machines will not have a service on that port. Thus, these machines will not respond (or will reject the connection attempt), and therefore are not vulnerable to being attacked on this port. This can be captured in a rule which states that all scans that do not result in a full connection (no successful reply from the scanned host) should be ignored, and all scans which do receive a successful response should be included.

5.3 Implementation Details

We implemented our framework to evaluate its effectiveness. Our framework could be instantiated in many ways; however, we chose to implement it using simple components in order to see how the framework performed even with simple components. As seen in Section 5.4, even with the simple components, our analysis framework successfully detected the attacks contained within the data on which we tested. These components, however, leave much room for improvement and, since the framework is designed to be modular, newer and more sophisticated techniques can be easily designed and inserted. In our implementation, we also utilized certain "primitives", such as low level IDS systems. The choice of these systems was driven by simplicity

and practicality and could easily be replaced by any other system that achieves the same goals.

5.3.1 Data Sources

Our framework requires certain data sources to be present in order to perform the analysis. We evaluated our framework on a specific data set (which is described in Section 5.4), and thus many of the choices for primitives were driven by this dataset. First, the network traffic was in tcpdump format which we then converted into a netflow format [55]. Thus all the analysis we performed was done on aggregated network header information. Also, along with the traffic, Snort alerts were included. Thus, our implementation used these alerts as one low-level IDS. In addition, we also used our MINDS anomaly detector [20, 21] and MINDS scan detector [19]. Note that these choices were made based on practical reasons and could be replaced by other systems. For example, Snort could be replaced by any other signature-based system, such as ISS Real Secure [25]. Also, any scan detector could be used in place of the MINDS scan detector, such as TRW [28], and the following host/service profiler could be replaced by a more systematic host/service profiler such as the port pattern anomaly detector used in the EMERALD system [42, 57].

For the context extraction, we implemented a simple historical behavior profiler (e.g. host/service profiler), which examines the network traffic and determines which machines run which services and which machines are clients for particular services. How it was used for context extraction is described in Section 5.3.3. It is based on the fact that machines typically exhibit the same behavior repeatedly. Thus, a web server might accept many connections on port 80 and 443, and rarely have any connection requests on other ports or make outgoing connections on any ports. The profiler constructs a probability distribution of services which have been accessed

on each host. The probability is calculated for each host by dividing the number of connections made to/from a particular port by the total number of connections to/from that host. If this probability is greater than a configurable threshold, then it is declared to be a server (or client, depending on the direction of the connections) on that port. In addition the profiler only profiles valid connections that have bi-directional flows (i.e. incoming flow and corresponding outgoing flow). This prevents the profiler from being skewed, for example by receiving scan packets on a port on which it does not offer any services. In our implementation, only internal hosts which have a degree of connectivity greater than some threshold (e.g. T_s for the server, T_c for client) are profiled. Once the profiles have been generated, each connection is examined and matched against the profile for the host involved. If it matches the profile (e.g. the connection is incoming on port 80 to a machine that has been profiled as a server on port 80), then the connection is assigned a score of 0, meaning normal. If the connection does not match any profile for the host, then it is assigned a score of 1, meaning anomalous.

5.3.2 Anchor Point Identification

The Anchor Point Identification step takes the output of multiple alert sensors and produces the set of events involved in attacks with higher confidence than relying on any single low-level IDS tool. In order for the alert combination to be effective, the data should be as orthogonal as possible, thus maximizing the overall information. In our implementation, we achieved this through the use of Snort alerts and the MINDS anomaly detector [20]. These two IDSs use vastly different mechanisms to flag traffic, and thus fit the requirement that the sources be orthogonal. We combined these two data sources in a simple manner, selecting Snort alerts to be anchor points if either the source or destination machine was also involved in a highly ranked anomaly. Note that

the anomalous activity need not be the same connection that was flagged by Snort. The intuition behind this mechanism to combine the data is as follows: if a machine is truly attacked and compromised, it is likely that the attacker would use the machine in a way that it normally does not behave, causing anomalous traffic to/from this host. The threshold for determining if a flow is considered to be highly anomalous is configurable. Details on how sensitive this threshold was and how effective this technique was can be found in Section 5.4.

5.3.3 Context Extraction

The next step in the analysis process is the *Context Extraction* step. The main goal of this step is to add events from the set of all network traffic that are related to the attack(s) represented by the anchor points detected in the previous step. The main challenge faced by this step of the analysis is to properly refine the context so as to add the maximum number of attack steps to the context, while adding the minimum number of unrelated events. As noted in Section 5.2.2, we made use of two main techniques, rules drawn from domain expertise and host/service profiling. The rules used are as follows: First, we ignored all traffic that was flagged as a scan in which the scanned host did not reply (i.e. did not successfully open a TCP connection). Conversely, we selected all scanning traffic that did result in a full bi-directional connection. Finally, each connection for which the previous rules did not apply was selected or ignored based on its host/service profiling score. If the score was above a configurable threshold, then the connection would be selected and added to the context; otherwise it would be ignored. Note that for a connection to be considered for addition to the context, it must be related somehow to the anchor points. For our implementation, we define this relation such that a connection is related to the anchor points if one of the IPs in the connection is already contained within the

context, where the initial context is the set of anchor points.

Once we have a method to define which network events are to be selected for the context, the algorithm for context extraction is quite simple. The algorithm goes through a series of iterations. At the beginning of each iteration, there is a list of all the IPs contained within the context. During the iteration, each flow is processed. If one of the IPs involved in the flow is contained within the context already, and if the flow passes the specified rules (and the flow is not already in the context), then the flow is added to the context (and any IPs not already contained within the context will be added). The iterations continue until no more flows are added to the context (i.e. the transitive closure has been reached). The Context Extraction could also be limited to add only a set number of flows or distinct hosts to the context; however, this could result in the loss of some of the attack. In addition, the threshold for the host/service profiling score can be dynamically adjusted to require, for example, that connections added in later iterations (and thus more loosely connected to the original anchor points) have higher profile anomaly scores.

5.4 Experimental Evaluation

We evaluated our proposed framework using datasets generated by Skaion corporation [46]. These datasets are simulated to be statistically similar to the traffic found in Intelligence Community. This dataset has several scenarios with attacks injected that follow different patterns. In the following sections we first describe the nature of the Skaion dataset, then discuss methods we used to evaluate our framework, and finally we show our results. As can be seen in the following results, even though our approach currently uses only simple implementations for each component, our overall analysis captures the major attack steps successfully.

Skaion Dataset

As part of the ARDA P2INGS research project, the Skaion Corporation has released several sets of simulated network traffic data. This data includes various scenarios of multi-step sophisticated attacks on resources within a protected network. The scenarios for which they have generated data include single stage attacks (a simple scan or exploit or data exfiltration scenario), bank shot attacks (where an internal host is compromised and used to attack another internal host), and misdirection attacks (where a "noisy" attack is staged on one part of the network while the true attack takes place in a more stealthy manner in another part of the network). In addition to the main attack, there are other background attacks (none of which are successful) and scans. To date, they have released three datasets, including many instances of these scenarios. However, for the sake of space, we will describe our results on one scenario in detail and present a summary of our results on other scenarios. The network topology in these scenarios is comprised of the following four domains: (i) the target protected domain, BPRD (Bureau of Paranormal Research and Defense) comprised of various servers which are the typical targets for attacks; (ii) a secondary internal domain which is not as protected as the protected domain and comprised of servers as well as clients. The hosts inside this domain have additional privileges to access the protected domain, BPRD; (iii) a set of external hosts which consists of attackers as well as normal users and (iv) a trusted domain which consists of remote users access the protected network with additional privileges over a dialup or a VPN connection. All traffic entering and leaving the entire internal network is captured by tcpdump. Snort alerts are collected for traffic exchanged between the external network and entire internal network.

Single Stage Attacks These scenarios are compromised of a simple attack made up of four steps. First, scanning is used to determine the IP addresses in the target network that are actually associated with live hosts. Typically in these scenarios, this is done by an attacker performing reverse DNS lookups to see which IPs have domain names associated with them. The next step consists of an attacker (or multiple attackers) probing these live hosts to determine certain properties, such as which OS and what version is running on the host. Then one of these hosts is attacked (possibly by a host that was not involved in any previous steps) and compromised. Finally, a backdoor is opened, to which the attacker connects and performs various malicious activities, such as data exfiltration or the downloading and installation of attack tools.

Bank-Shot Attacks These attacks are aimed at avoiding detection by using an "insider" host to launch the actual attack. In this scenario, initial scanning is done, and then an attack is launched against a host in the BPRD network. This attack fails, and the attacker then scans and compromises a host in the secondary internal domain. From this server, the attacker scans and launches attacks on hosts in the protected BPRD network. A host is then compromised, from which data is exfiltrated.

Misdirection Attacks The attacker attempts to draw the attention of the analyst away from the real attack. He does this by launching a noisy attack (one which sets off many IDS alerts) on a particular set of hosts in the protected network. Then using a previously compromised host in the trusted domain, he attacks and compromises another host in the BPRD network, from which he exfiltrates data.

Evaluation Methodology

Before discussing the results of our experiments, we first describe how we performed the experiments and the methods we used to evaluate our framework. For a given

scenario, we first ran all low-level IDS tools to generate the alerts, anomaly scores, etc. For profiling, we used ten and five connections for T_s and T_c respectively. This means that a host was profiled as a server only if it had more than 10 inbound connections. Similarly, a host was profiled as a client only if it had more than 5 outbound connections. In addition we profiled only ports with more than two connections. We then ran Anchor Point Identification using multiple rules for detecting the anchor points in order to compare the performance and sensitivity of each set of rules. First, we used Snort alone, where each Snort alert was selected as an anchor point. Next, we used the MINDS anomaly detector alone, where the connections that ranked in the top k% of anomalies were selected as anchor points. Finally, we combined Snort and MINDS in the method described in Section 5.3.2. The anchor points selected were those Snort alerts in which at least one of the IPs was involved in a highly ranked anomaly (ranked within the top k% of MINDS Anomaly Detector output). The evaluation criteria for the anchor points is twofold: first, whether it covers the attack (i.e. did it have any true positives), and second, whether it has low false positives (the lower the better). The Anchor Point Identification step generates a set of events (anchor points) which represents a connection between two hosts. An anchor point is related to the attack scenario if the connection it represents is a part of some attack step. In our results section, the results of this step are represented by the number of attack related hosts detected (true positives) and number of non-attack related hosts detected (false positives). A host is counted as attack related if it is present in an attack related anchor point. In this case we call it covered, as introduced in section 5.2. If a host is present only in non-attack related anchor points, it is counted as a false positive.

Following the Anchor Point Identification step, Context Extraction was run with each set of anchor points found by different rules utilized by Anchor Point Identifi-

cation. No other parameters were varied for this step, since the parameters mainly consist of limiting the expansion, and for our experiments this step was run until no more context was added. The goal for this step is to detect all attack-related steps (with emphasis on the more important steps, e.g. initial scanning is less important then exploits or backdoor accesses) while reducing the number of non-attack related steps. Note that there are two types of non-attack related hosts that could be added to the context. First, they could be part of background attacks, which are still interesting for the analyst. Second, there are real false positives, which are not a part of the actual attack scenario or the background attacks.

All the tables for the results follow the following notation :

AS:*Attack Steps* This represents the high level attack steps like probing (information gathering), actual exploit, backdoor access, or data exfiltration.

AH:*Attack-related Hosts* This includes all hosts related to the attack scenario including external scanners, external attackers, internal hosts scanned by the attackers for information and the eventual victims which get compromised.

BA:*Background Attack Related Hosts* This involves all hosts related to the background attacks in the traffic as attackers or victims.

FP : *False Positives* This counts all hosts that are not related to the actual attack scenario or to the background attacks but are wrongly detected by our framework.

Detailed Analysis: Skaion Scenario - 3s6

We present our detailed analysis on one of the bank shot attack scenarios. The scenario we evaluated (called 3s6) had 122,331 connections in the traffic, involving 4516 unique IPs, on which there were 6974 Snort alerts.

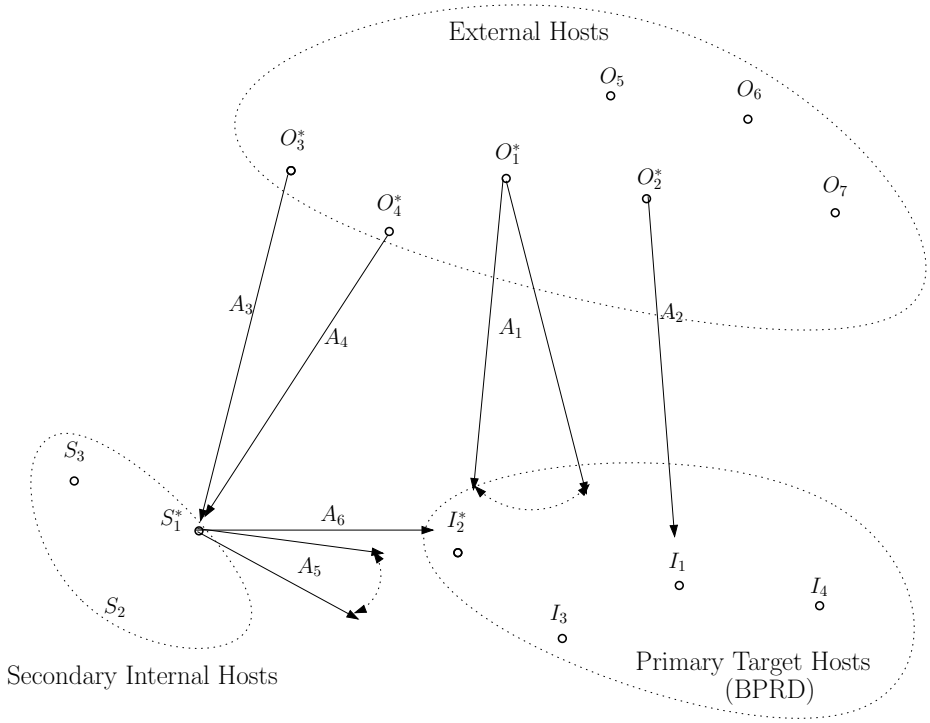


Figure 5.2. Different steps and hosts involved in the attack scenario 3s6

The attack graph for the scenario 3s6 is shown in figure 5.2. The various steps involved (in chronological order) are :

- A_1 : O_1 (74.205.114.158) scans 92 hosts *(936 flows)* inside the BPRD network.
- A_2 : O_2 (42.152.69.166) attacks internal server, I_1 (100.10.20.4) four times (*17 flows*) and fails each time.
- A_3 : O_3 (168.225.9.78) port scans *(18 flows)* secondary internal host, S_1 (100.20.20.15 *alias* 100.20.1.3).
- A_4 : O_4 (91.13.103.83) attacks S_1 *(78 flows)* using *Apache OpenSSL SSLv2 Exploit* [5] and succeeds.
- A_5 : S_1 port scans 6 servers in the BPRD network *(895 flows)* including the eventual victim, I_2 (100.10.20.8).

Table 5.1. Results for anchor point identification on bank-shot scenario 3s6

Config		AH	FP
Snort		96	169
Top k%-anomalies	0.2	5	5
	0.5	8	67
	1.0	50	114
Snort + Top k%-anomalies	0.2	93	0
	0.5	95	39
	1.0	98	83

Table 5.2. Results for context extraction on bank-shot scenario 3s6

Config		#Iters	AS	AH	BA	FP
Snort		2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	75
Top k%-anomalies	0.2	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	43
	0.5	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	58
	1.0	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	93
Snort + Top k%-anomalies	0.2	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	45
	0.5	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	47
	1.0	2	$5(A_1,A_2,A_4,A_5,A_6)$	24	3	47

- A_6 : S_1 launches attacks on I_2 using *IIS IDA-IDQ exploit* [4] and succeeds. It also browses through the files of I_2 *(4 flows)*.

The attackers try to confuse the analyst by first scanning and unsuccessfully attempting to attack the internal network (Steps A_1 and A_2). Most of the attack related Snort alerts are on this traffic. Another attacker then attacks the secondary network and compromises an internal host (S_1). This host is then used to scan the BPRD network and launches an attack on I_2. Since this traffic is internal, it is not detected by Snort. The results of context extraction in Table 5.2 show that the framework succeeds in capturing a large portion of the attack scenario (5 out of 6 attack steps). The context also captures some background attacks present in the traffic. The false alarms arise because of following reasons - 1) *Mislabeled Flows* - These arise because of errors in the data converting component due to which initiating flows might be labeled as replies and vice versa. 2) *False alarms from Our Profiler* - Host/service profiler has

an associated false alarm rate due to which some non-attack related flows are added to the context.

All configurations for anchor points result in detecting a portion of the scanning activity by O_1 as anchor points in Table 5.1. From these anchor points, the scanning activity A_1 is added to the context. Since I_1 is scanned by O_1, its traffic is analyzed. This results in adding the failed attack attempts, A_2 to the context. I_2 is also scanned by O_1. Since I_2 is attacked by S_1, this attack step A_6 is added to the context. On analyzing the traffic to and from S_1, the scanning activity A_5 is added to the context. Similarly the attack step A_4 on S_2 is also added to the context. The attack step A_3 is not captured since it involves probing of S_1 on ports on which it is a server. However, we capture all those attack steps from which we can construct the core attack scenario.

We observe from Table 5.1 that if we use a correlation of Anomaly Detector and Snort we get a lesser number of false positives as anchor points. As we relax the constraints in Anchor Point Identification step, we detect more attack related hosts, but the number of false positives also increases. However, from the context extraction results in Table 5.2 we observe that we still detect the major portion of the attack scenario even if we start with a lesser number of anchor points. Moreover, the presence of false positives in anchor points results in a high false positive rate for context extraction.

Results for Other Scenarios

The results of our analysis on other scenarios are summarized in Table 5.3. The configuration used for Anchor Point Identification was the combination of Snort Alerts and top 0.5% of MINDS Anomaly Detector Output. From the table we observe that our implementation is able to capture all important steps of each attack scenario

except for the scenario - *Five by Five*. (In this case, the volume of traffic related to the victim host was not enough to be profiled, thereby that host was not added to the context.) The attack steps which were missed in all cases involved failed attack attempts or probes before attacks. Our implementation captured all the important attack events, such as the actual exploit, data exfiltration for all but one scenario from which the core attack scenario can be generated. From the results we can observe that by using strict thresholds for Anchor Point Identification, we are able to detect some attack related events (as anchor points) while keeping the number of false positives very low. Using these anchor points, we successfully detect the core attack scenario in all but one scenario along with some background attack activity. Since the number of non-attack related anchor points are low, the false positives in the context extraction step are also very few.

A brief description of our results on each scenario is given below:

Naive Attacker All attack related steps are detected. The 7 attack-related hosts that are not detected are the hosts inside BPRD which are scanned by the attacker as part of the probe but do not reply back. Thus they do not supply any information to the external attackers.

Simple Ten All attack related steps are detected. The 240 attack-related hosts not detected are again the scanned hosts which do not reply back.

Five by Five We fail to detect any attack steps or any attack related hosts. In this scenario, the victim host inside the network was not involved in any traffic with external world apart from the attacks launched by outside attacker. There was no profile generated for this host and hence the attacks could not be distinguished from normal traffic. The attack would have been detected if there were enough traffic which would meet the thresholds related to profiling of internal

servers.

Ten by Ten All attack related steps are detected. Eleven attack-related hosts not detected include six scanned hosts which do not reply back and five external scanners who never get a reply back from the hosts which they scan. Thus effectively, these external scanners never get any information about the internal network and hence do not contribute to the actual attack scenario.

s9 All attack related steps and attack related hosts are detected without any false positives.

s10 One attack step is missed in this scenario. The missed step is a failed attack launched by one external attacker on an internal host which is not the eventual victim. Thus this step is not an important part of the whole attack scenario.

s14 All attack-related steps and attack-related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections). This occurs during the conversion of tcpdump data to netflow format.

s16 One attack step is missed in this scenario. The reason for this is the same as in scenario *s10*. We also detect two background attacks as a part of the context. The false positives arise because of two outside hosts involved in traffic on random high ports with internal servers which does not conform to the normal profile of those internal servers.

s24 In this scenario three external attackers did a distributed scanning of the internal network. One of the scanners got a reply back from the eventual victim while

the other two did not get any replies from the hosts which they scanned. These two scanning steps which did not contribute any information were missed. The false positives occurred because of the same reason as in scenario *s16*.

3s10 All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections) or due to outside hosts accessing internal servers on random high ports.

s1 All attack-related steps and attack-related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise because of external hosts accessing internal servers on random high ports.

s37 In this scenario, one of the attackers port scans two internal servers but gets reply from only one which is eventually attacked. The other server does not supply any information back to the attacker. Only this server is not detected while all other involved hosts and attack steps are detected.

s29 All attack steps except for one initial probe, which did not get any replies, were detected. The false alarms occur for the same reason as in scenario *s1*.

5.5 Discussion

In Section 5.2 we described the main design goals for our system. The first goal was that the analysis framework should address the inherent tradeoff between false positives and false negatives. We address this issue in the design of our framework by decomposing the problem into two steps. In the first, we focus on the reduction of

false positives, by selecting network events in such a way that gives us high confidence that the events are part of an attack. This was achieved in our simple implementation, through the use of Snort alerts combined with the MINDS anomaly detector. Second, we fill in the missed attack steps by extracting the context from the set of anchor points. By requiring that the anchor points be of high quality (low false positives) we can relax the restrictions on what we add to the context if they are connected to the anchor points. This was also achieved by our simple Context Extraction module, in that relatively few false positives were added to the context when the anchor points contained few false positives.

The second goal was to detect the majority of attack steps in the attack scenario. Evaluating this is not completely straightforward, since not all attack steps would be considered equal, and thus a measure such as a straight percentage of attack-related connections would not be sufficient. This is due to the fact that not all attack steps are of the same importance. For example, in the scenario described in Section 5.4, if we had detected all of the scans and nothing else, we would have detected the vast majority of network connections that were relevant to the attack (95%), but this information would be useless to the analyst. A better measure would be aggregating the connections together into steps (using techniques such as those proposed in [11, 58]), and measuring the number of attack steps that were detected. In this experiment, however, we managed to detect all major attack steps (including attacks, internal stepping stones, and data exfiltration) and many connections in the scanning. Thus we achieved the goal of detecting the majority of the attack steps.

The third goal set forth in Section 5.2 was high coverage of attacks. In the Skaion scenarios, however, only one attack was present in each scenario. Thus, while not fully tested, this goal was initially achieved in the fact that we were able to detect the main attack in each scenario, except for the one with insufficient profiling information.

The final goal was to make the system modular by design. This design goal was achieved as seen in Section 5.2. First, the two components in our framework are independent of each other. Thus the implementation of one can be changed without affecting the other. Context Extraction does not depend on how the anchor points are found, as long as they are of high quality. Also, Anchor Point Identification is not concerned with how the anchor points are used, and thus any algorithm can be used to implement the Context Extraction. Also, the system is not tied to any low-level IDS system. None of the design of our framework hinges on the types of alerts available. For example, in our implementation, Snort alerts were used. However, any other signature-based system could replace it. The only restriction is that the information needed by the particular implementation of the later stages needs to be present in some form. In addition, we could incorporate other types of information that could be used to detect intrusions, such as system logs [23, 24] and host based IDS alerts [10, 26, 30].

5.5.1 Limitations and Improvements

This leads us to consider the limitations of our framework. The biggest limitation is that it has greater storage requirements than most IDSs. Snort, for example, examines traffic in real time and creates alerts based on what it finds. All that needs to be stored are the alerts. However, our system needs storage of both the low-level IDS alerts as well as the actual network traffic (in some form). The more detailed the data and longer time frame for which the data is stored, the better our system will perform. Also, detecting sophisticated attacks may require the capture of traffic between internal hosts. Capturing the traffic between every host within the network would be difficult, and in many cases infeasible. This storage requirement can be greatly mitigated by storing the data in the net-flow format, where only header

information is aggregated and kept. In the University of Minnesota campus network, one year of net-flow information can be stored in 0.5 TB, whereas one week of tcpdump data requires 2-3 TB of storage. On the other hand, if complete forensic analysis is to be performed, it would be very desirable for the tcpdump data to be present, and thus our framework would pose no extra storage requirement. A operational system designed to store relevant raw network data for forensic analysis is described in [32].

One last important point to discuss is the effect of a framing attack, that is, how an attacker can attack the analysis framework itself. If an attacker knows the rules used by Anchor Point Identification, then he would be able to generate spurious anchor points. However, the amount of anchor points he can generate depends greatly upon the rules used by Anchor Point Identification. If Snort alerts alone were used, then the attacker could easily generate an arbitrary amount of anchor points involving every internal machine [22, 33]. This would basically reduce our framework to a low-level IDS system with a low threshold, flagging much of the traffic as part of an attack. If the rules used for Anchor Point Identification were Snort combined with the MINDS anomaly detector, which was shown to be effective in our experiments, then the effect that the attacker can impose on the anchor points is more limited. Again the attacker can send packets that cause Snort alerts to all the hosts in the network, but to be flagged as anchor points, each of these flows must also be in the top tier of anomalies. By the definition of anomaly, all of these packets would have to be unique with respect to the attributes used by the anomaly detector to rank the network connections. This is a difficult thing to do, since the attacker would have to know a priori what will be considered anomalous for the time period that will be examined. Also the attacker would have to be careful not to send too many packets with certain similarities, since while they may be abnormal when compared to the rest of the traffic, they may form their own cluster and be considered normal with respect

to themselves. Also, if too much abnormal traffic is sent, there could be enough abnormal traffic that the abnormal traffic becomes the "norm", making it very hard to predict what will be flagged as abnormal. For example, if the attacker sent large packets to cause the anomalies, after too many such packets, large packets will be considered normal. In addition, for the Snort and MINDS combination, there is an upper limit on the number of anomalies that will be used for selecting anchor points (due to the cutoff threshold). Thus, the Anchor Point Identification step can, through careful design and implementation, provide some measure of resistance against this type of attack. Another useful aspect of Anchor Point Identification is that it is dynamically configurable (depending on the available data sources), so if it generates too many false positives, it can be run again with a different set of anchor point selection criteria. This opens up two lines of future work on this component. First, we plan to investigate and design better approaches to combine the data sources in order to select anchor points. Second, we plan to test the designs against these types of attacks to better understand the effects that they can have on the results of our system.

Context Extraction is less resistant to this framing attack, especially when using a port-profiling technique, as it is difficult to cause a machine to act outside of its profile without actually compromising it. (To do this effectively, the attacker would need some insider knowledge of the port usage of the internal machines, such as ports on which the internal machines actually offered service but had low enough volume so as to not be profiled.) However, Context Extraction would be affected by the false anchor points generated by attacks on the Anchor Point Identification step.

5.6 Related Work

The area of research most related to this work is IDS alert aggregation and correlation. Alert aggregation has to do with taking alerts from multiple sensors and merging them into one higher level alert. Generally this is done on single events that trigger alerts across multiple sensors. For example, if a subnetwork is set up such that traffic going between it and the outside Internet would pass through two Snort sensors, then an attack that triggers a Snort alert would trigger two such alerts. If an analyst is looking at these alerts, it is more efficient if the analyst looks at the alert only once. This gets more difficult when the sensors are not the same type of sensor and report different sets of information, and often a probabilistic approach must be taken [58].

Correlation has two main aspects to it. One is the fusion of different alerts that refer to different events in an attack but are highly related. For example, if there is a DOS attack, and each probe sets off an alert, there will be many alerts from a certain source IP to a certain destination IP. Thus all of these alerts could be merged into one higher level "DOS" alert. This type of fusion can be achieved by clustering alerts based on specific fields in the alert containing matching information [41].

The second area of correlation is in the realm of relating alerts together that fit into an attack scenario. This is the most closely related work in correlation to our approach. Much of the work done in this area has been done in matching prerequisites and consequences of alerts [7, 11, 34–37]. In this approach, the analyst defines the set of actions that must take place before a given alert can occur (its prerequisites), and then once an alert has happened, what actions can subsequently take place (its consequences). By placing this information with each alert, a system can match them together (along with extra information such as IP addresses or time-stamps) to form sequences of attacks or attack scenarios. One limitation of this approach is that it

requires extensive expert domain knowledge to determine exactly what is required for an action to take place and what its consequences can be. In addition to pre-requisites and consequences, there have also been probabilistic matching approaches proposed [9], and matching detected events against attack models [6].

There have been many other approaches proposed to correlating alerts, and many of these have been incorporated in the comprehensive system in [59].

5.7 Discussion

This chapter has shown how the multi-step analysis approach can be beneficial in analyzing network traffic and IDS alerts to discover multi-step, sophisticated attacks. One of the most important directions for future work is to utilize the output of the context extraction module in a way that allows for easy analysis. This is the task of the Attack Characterization step, which was ignored in the description of the framework. Even if the output of the context extraction is 100% accurate, it is still a (potentially large) collection of raw network traffic data. Presenting this information to the analyst in an easy to use format, perhaps using visualization techniques, would be beneficial to the analysis, and could help to reduce the effect of false positives from the Context Extraction. One way to accomplish this is to use alert aggregation techniques [11,58]. A second mechanism that could be useful is attack graphs [45], which are possible paths of attack and are generated based on vulnerability assessment and network connectivity information. Matching the detected context against full attack graphs could provide more information to the Attack Characterization step. Another area of future research is to create better and more sophisticated components for the individual steps in the analysis framework. The proposed methodology worked well with the simple components, and improving them should improve the overall results.

The context extraction phase described in this chapter is made quicker by using the data retrieval methods described in Chapter 4. The sample queries presented in that chapter are the same types of queries issued while performing context extraction. The most common query executed during context extraction is looking for all activity involving a certain IP address. Using the data retrieval components of Chapter 4 this query takes less than two seconds for the most active IP address, and thousandths of a second for less active IP addresses. This is a huge improvement compared to the traditional "grep" approach which can take tens of minutes.

The quality of the results of the context extraction process is increased if the sessions used were from the data collection process described in Chapter 3. This happens because the client and server for each session is known, rather than guessed at when applying heuristics to construct sessions out of netflows. Knowing the initiator of each session also increases the quality of the profiles which are used to prune the communication graphs. This then leads to graphs with fewer spurious edges which were included only because a session looked to be initiated from a server, rather than correctly labeled as a reply from the server.

Table 5.3. Summary of results for different Skaion scenarios

	Scenario	Ground Truth					Anchor Points		Context Extraction			
		# Conn	# Hosts	# Alerts	**AS**	**AH**	AH	FP	**AS**	**AH**	BA	FP
Single Stage	Naive	1739	581	27	**4**	**10**	2	0	**4**	**3**	0	0
	Simple Ten	12040	2616	114	**4**	**246**	4	0	**4**	**6**	0	1
	Five by Five	7853	2101	177	**3**	**13**	5	45	**0**	**0**	0	5
	Ten by Ten	9459	1435	54	**4**	**16**	5	11	**4**	**5**	0	1
	s9	4833	472	53	**3**	**2**	2	3	**3**	**2**	0	0
	s10	4792	582	58	**4**	**3**	2	6	**3**	**2**	0	0
	s14	8915	1210	95	**3**	**2**	2	9	**3**	**2**	12	4
	s16	5711	368	1372	**4**	**3**	2	4	**3**	**2**	2	3
	s24	4334	699	452	**6**	**10**	2	4	**4**	**4**	1	3
	3s10	47490	3084	3150	**3**	**6**	5	21	**3**	**6**	1	5
Bankshot	s1	45161	12292	10896	**6**	**7**	4	32	**6**	**7**	11	3
	s37	23970	1517	7671	**6**	**5**	4	18	**6**	**4**	0	0
Misdirection	s29	10926	627	451	**7**	**6**	5	1	**7**	**6**	0	4

Chapter 6

Scan Detection

6.1 Introduction

A precursor to many attacks on networks is often a reconnaissance operation, more commonly referred to as a scan. Identifying what attackers are scanning for can alert a system administrator or security analyst to what services or type of computers are being targeted. Knowing what services are being targeted before an attack allows an administrator to take preventative measures to protect the resources they oversee, e.g. installing patches, firewalling services from the outside, or removing services on machines which do not need to be running them. In addition, these computers, scanners and those replying to scans, can be used as anchor points in the 2nd level analysis framework described in Chapter 5.

Intrusion analysts, whether monitoring networks for the government or the private sector, are required to perform far more analysis tasks then time will allow. Because of this, only a small subset of the alarms that various intrusion tools produce can be investigated further. Analysts quickly discover that it is most effective to investigate intrusion alerts that indicate immediate and catastrophic compromise. This leaves very little time to analyze alerts such as scans that are essentially informative

in nature but do not necessarily indicate immediate compromises. Hence, despite the importance of scan detection, its value is somewhat overlooked in the security community. One reason is that there is a lack of good tools for doing proper scan detection.

The existing scan detection schemes essentially look for IPs that make more than X connections in Y seconds. These schemes are very good at picking out disperse noisy scans. Unfortunately, tools based on these techniques are quite bad at detecting stealthy scans or scans targeted specifically at the monitored enterprise - the type of scans that analysts would really be interested in. Stealthy scans can be defined as scans that would normally not trigger typical scan alert technology. The adversary is keenly aware that most scan detectors are set to alert based on the number of connections attempted by a given host over a predefined period of time. It is fairly trivial for an adversary to adjust his scans to evade detection by slowing down the frequency of his transmissions. Intrusion analysts and several IDS vendors tried to counter this threat by either reducing the connection threshold and/or increasing the time window threshold in their scan detection technologies. This created an untenable situation where chatty yet benign protocols (e.g. protocols such as netbios that talk to many hosts in a relatively short time) generating numerous alerts and effectively disguising the presence of the stealthy scans. The result is the current situation where analysts either do not run scan detectors or essentially ignore their outputs in the interest of time.

This section presents new scan detection techniques that have much lower false alarm rate and much higher coverage than existing techniques. A key strength of these new scan detection techniques is their ability to detect stealthy scans. This section presents some heuristics to reduce false alarm rates of the existing schemes, as well as a new way of detecting scans which makes use of usage information. The section also

presents a detailed experimental comparison of existing and the proposed new scan detection methods on a large size network data from the University of Minnesota. These experimental results show that the proposed methods have much better recall than the basic time window based schemes with a near-zero false positive rate. These new techniques have been in production use at the University of Minnesota and at the US Army Research Laboratory Center for Intrusion Monitoring and Protection (CIMP) which analyzes traffic from many DoD sites around the country. The ability of the new schemes to detect very low volume scans has lead to the identification of several compromised computers used as stepping stones inside the Army's network, compromises that were not detected by any other method.

6.2 Related Research

Scan detection has been often thought of as the process of counting X number of events in Y number of seconds. The most widely used scan detection method counts the number of unique destination IPs talked to by each source on a given port in a given time window. If this count within a time window of Y seconds is greater than a user specified threshold X, then the source IP is considered to be scanning. This section will refer to this as the basic scheme. This method requires storing information about the destination IP and port as well as the time stamp. This method has a reasonable false alarm rate. This basic method is used in SNORT's [47] scan detection module, which detects host scans and port scans. However, this method is not suited for detecting slow scans. To do so would require having a large time window while keeping the threshold low (which also increases memory requirements considerably) to increase the recall. But this will also result in a higher false alarm rate. Typical false alarms generated using this scheme are web browsing, crawlers,

sending e-mails to large mailing lists, and peer-to-peer applications.

One standard way to handle slow scans is to add a connection window which keeps a history of the most recent N connections made by each source to a destination port. Using only a time window to detect slow scans, requires storing significant amount of data, most of which are related to legitimate sources that make many connections. Using a connection window allows us to store enough information about low volume talkers to detect slow scans while minimizing the storage for high volume talkers.

The basic method and its derivatives do not take into account the usage information for the hosts that are talked to. An external source connecting to a server contributes just as much to the scan score as another external source trying to connect to an internal machine where the service doesn't exist. Obviously, the latter one should be weighted more than the former. The method used in SPADE [53] which is available as SNORT preprocessor, makes use of the usage statistics. It maintains the joint probability distribution of destination port and destination IP, which they have found to work the best among different methods. It raises an alarm on a packet if the negative log likelihood of the destination IP / port combination for a packet exceeds the threshold. * Using the methodology described above, SPADE raises too many alarms, and it is turned off by default in SNORT. For example, SPADE will raise an alarm on the first few connections to previously unused IP / port combination, even if this is the only activity from that source. In such cases, connections from a single source to a rarely-used service on a single IP will be declared as a scan simply because it is rare. This problem is exaggerated due to the fact that the history cannot be kept for long periods of time. As a result, every occurrence of low intensity periodic events can potentially be declared as scans. In addition to the false alarms, SPADE may also miss very popular scans (i.e., the scans that are performed by many sources)

simply because the corresponding counts in the usage statistics will be high.

Another common approach tries to address the false positive problem in scan detection by only looking at the ICMP replies to packets sent to unallocated/dark IP spaces. No legitimate connection should involve a dark IP address. Other than misconfigured machines and rare occurrences of users typing IP addresses wrong, the rest of the connections to dark IPs should be due to scans. This approach is good for detecting random scans, as they have a high probability of hitting unallocated IP ranges. However, this methods cannot detect smarter scans (e.g. those that avoid unallocated IP ranges by using BGP information). Other problems with this method include not knowing which allocated IPs were scanned on the inside network and the inability to identify computers that responded to scans. In addition, scanners that randomly pick IPs to touch will become less effective and thus less common with the move from IPv4 to IPv6; hence the need for new techniques to be able to detect smarter scans.

In [28] Jung presented a new technique for detecting network scans based on a random walk on a stochastic process. The authors proposed their system in response to the need for quick detection of network scans. In their evaluation on a portion of MIT's network they detected approximately 14,000 unique scanners in a 24 hour period, while maintaining a false alarm rate of less than 0.3%. Although their work is impressive, it suffers from two problems; first it suffers from a potential state explosion problem in trying to keep track of information for every possible scanner. They also require five to six connections before a scan can be declared. Compared to other

*Maintaining statistics on the IP / port combinations require considerable amount of memory. For example, for a class B network, the number of IP / port combinations inside the network is $2^{16} * 2^{16} = 2^{32}$. Storing 4 bytes per entry would require 16 GB of memory for a relatively small sized network. To address this problem, in SPADE, usage statistics are stored in a sparse matrix representation and the counts are aged and an entry is deleted when the count falls below a certain threshold.

existing methods for detection scans, this is considerably faster considering they have a low false alarm rate. However, for similar precision and recall our proposed usage based method requires as little as two connections.

6.3 Proposed Method

This section presents some techniques that address the limitations of the basic method.

Incorporating Heuristics in the Basic Method To address the high false alarm rates of the basic threshold-based schemes, the proposed scheme makes use of two characteristics that differentiate scanning behavior from normal traffic. First, most scan traffic consists of connections to ports on which no service is running. These connections will not contain more than three packets, as the three-way-handshake cannot be established. Even for ports on which a service is running, the scanner will try to terminate connections quickly to maximize the number of machines it can scan in a given time. In contrast, the majority of legitimate traffic tends to last longer than three packets (approximately 60% of all connections involved more than three packets). Hence, the first heuristic used was to consider only incoming flows that are less than 4 packets. This heuristic helps eliminate a large fraction of normal traffic in scan detection, which potentially decreases false alarms.

Second, normal traffic can consist of short connections from one IP to many other IPs (raising false alarms on basic threshold-based schemes discussed earlier). For example, during typical web browsing, a user might do a web search and visit many sites returned in the search results in a short period of time. However, most of these connections tend to be on random IPs not necessarily concentrated on a specific subnet. In contrast, most scan traffic tends to make connections to many machines within same subnets. This is particularly true for targeted scans that look for vulnerabilities

in a specific organization's network. Hence, the second heuristic used restricts scans into subnets. This is done by counting the number of connections by a source to IP/port combinations on a network. This reduces false alarms by not alerting on typical normal traffic.

6.4 Usage Information

This section presents a scheme that incorporates complementary strengths of the basic scheme and the scheme used in SPADE, while avoiding their weaknesses. As discussed earlier, SPADE can raise an alarm on a packet / connection regardless of what else is done by the source IP, but it does give higher importance to destination IP / port combinations that are rarely used. On the other hand, the basic scheme considers all the traffic generated by a source IP in a given time window before declaring it a scanner. However, the basic scheme does not differentiate between connections to heavily used servers and previously unseen traffic. The proposed scheme not only looks at recent connection history for each source, but also weighs each of the touches to unique destination IP / port combinations differently in its scoring according to the usage information.

In the proposed scheme, just like in SPADE, statistics are maintained for destination IP – destination port combinations. It is also very desirable to detect outbound scans as they may point to infected hosts in the network. Therefore, statistics have to be maintained for outside hosts as well. These statistics are used to improve on the basic method. Instead of incrementing the scan count for a host by one every time it touches a new IP on a given port, the scan score is incremented by

$$\frac{1}{1 + lg\left(count_{IP/port}\right)} \tag{6.1}$$

Doing so reduces the false positives that might be generated by the basic method. For example, local users browsing various websites to get the daily news will be weighed lower since the volume of the traffic on that IP / port combination will be much higher compared to a random IP/port combination. It would take many more touches on commonly used combinations as opposed to few touches on infrequent combinations for a host to be declared a scanner.

As was stated earlier, the basic method cannot detect low volume and stealthy scans. This scheme partly addresses this by making use of usage statistics. Moreover, it also extends the detection capabilities of the basic time-window method by using the concept of a connection-window; that is, it not only scores connections made by a host in a given time window, but also scores last N connections made by a host. Using the connection window, it is possible to keep history information for low volume hosts longer. This makes it possible to detect low volume scans as well. A host is declared to be a scanner if the sum of the scores according to equation 6.1 for the unique IP addresses touched on a given port in a time / connection window exceeds the scan threshold.

Representing and storing the usage information exactly for each IP / port combination requires too much storage and is not feasible. Specifically, even the complete representation of port / IP combinations within a class B network will require 16 GB of memory. Complete representation of port / IP combination statistics for the entire Internet will require a staggering $2^{32} * 2^{16} * 4 = 2^{50}$ bytes. To bring down the memory requirements to store these statistics, this scheme does not keep statistics about each individual host and port, but groups hosts and ports into blocks as can be seen in figure 6.1. When grouping IP addresses into blocks, it is desirable to preserve the geographic locality by keeping high order bits. It is also important to preserve some low order bits in order to be able to detect scans on subnets. Therefore, IP addresses

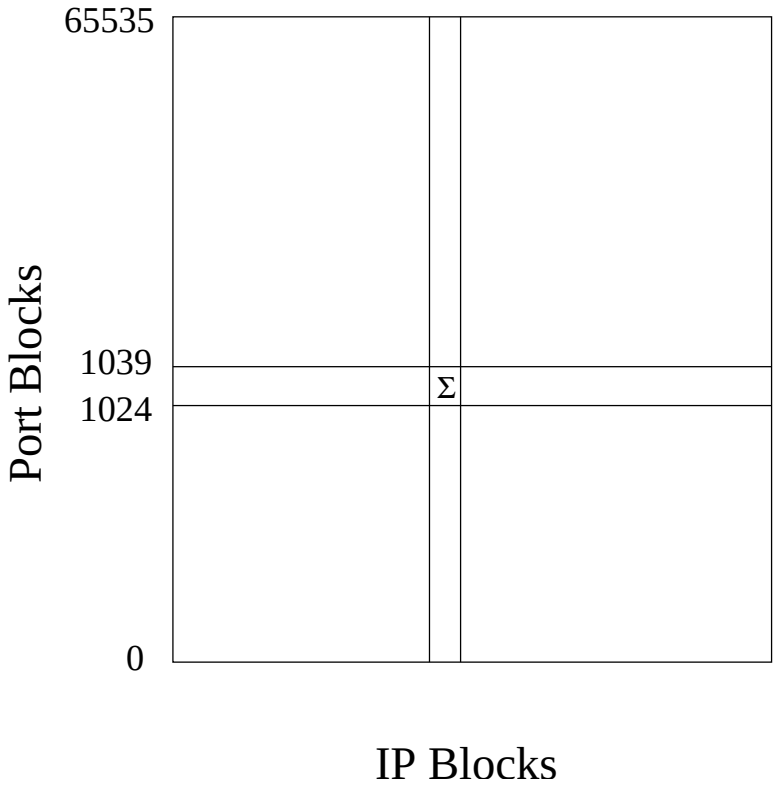


Figure 6.1. Aggregated Usage Statistics

were grouped by keeping *num_high_bits* plus *num_low_bits* bits. This scheme also groups the unprivileged ports into blocks of *port_resolution* and treated privileged ports individually. Depending on the memory available, these parameters can easily be adjusted to bring down the memory required for maintaining the statistics.

The next section presents experimental comparisons between the basic scheme, the basic scheme improved using heuristics and the proposed scheme.

6.5 Experimental Evaluation

In the experiments, netflow data collected at the border router at the University of Minnesota was used, as well as data collected using the data collector described in section 3 at sites from the Army Research Laboratory. However, due to security and publication concerns only anecdotal results can be presented from the ARL-CIMP

data. The evaluation in this section was done on a half hour's worth of data from midnight to 12:30 AM on the University of Minnesota data. The main reason why the evaluation wasn't done for a time window during day time is the amount of manual effort needed to label the scan detection output. The number of flows recorded during day time can be as many as three times larger than the number of flows collected during night time. Since this evaluation considers only inbound scans, the data from the users at the University would not have affected the results much. The data collected consisted of 3,943,410 netflows, the data transfer rate at this link was 221 Mbps. The breakdown of data transferred per protocol is given in table 6.1.

Protocol	Flows	Octets	Packets
TCP	2317461	48030621617	71794984
UDP	1525076	1557138198	10280879
ICMP	98885	25944042	244081
IPv6	1175	2920908	7714
ESP	567	257380176	271262
PIM	121	319666	3596
IGMP	47	4172	149
OSPF	26	208980	2540
AH	24	39032	189
GRE	22	299411	1053
IP	6	920	23

Table 6.1. Protocol Distribution

The first step on the University of Minnesota data was a pre-processing step due to the nature of the netflow data. This step was not needed on the ARL-CIMP data since the records are actually sessions, with the client and server already correctly identified. Flows are unidirectional; a TCP session will result in two unidirectional flows. Pre-processing paired up the TCP flows into sessions using the 5-tuple (IPs, ports and protocol) and the time stamp information. After determining the client-server relationship for the TCP sessions, only clients are considered in scan detection. Not considering servers in scan detection eliminates the possibility of declaring the

servers to be scanning. This implementation has a hard timeout on the connection window; connections from sources are removed from the history after the hard timeout even though it may not have reached N connections. Also, scans detected using the time window are not considered for evaluation in the connection window.

This section will use B_k to denote the basic scheme, H_k to denote a scheme that incorporates both of the previously described heuristics, and U to denote the usage based scheme, where k is the threshold. This evaluation applied different methods of scan detection with different thresholds. The first two methods correspond to the basic method which simply looks for X events in Y seconds. We used a ten second time window, as well as the connection window extension with a width of 256 connections. Both of the thresholds were set to ten in the first experiment and five in the second. The next two experiments used the previously described heuristics, considering only scans to a network and the number of packets in a flow less than four, using the same thresholds in the first two experiments. The last experiment again used the heuristics as well as the usage table as previously described. This experiment combined eight high bits and four low bits of the IP to define the IP blocks, and used single ports below port 1024 and groups of 16 for high ports for which the counts are collected. For this experiment a threshold of 1.01 was used. This meant that the scan had to touch at least two unique IPs as the contribution from a first touch to a never-before-used block in the usage matrix is one.

In these experiments, only inbound scans were considered due to privacy considerations. Reported outbound scans were filtered out before the analysis of the output. In the production mode at the University of Minnesota and at ARL CIMP, it is used to catch outbound scans as well, many of which identify compromised machines not easily identifiable by other means.

After running the experiments, the security analyst at the University of Minnesota

labeled the output. Many of these were obvious scans where a source touched many ports on many IPs. Many of the scans were easily labeled because they attempted to touch multiple IPs where either port was blocked at the border or the IP was not allocated. After labeling the obvious scans, the security analyst was left with approximately 20% of the scans which required investigation. This investigation involved a number of steps, including checking of the IP address to see if it resolves to a name, checking if the service is running on the destination that is scanned, and investigating other IPs touched by the suspect source IP over a larger time window. Traffic from several web crawlers (e.g. googlebot, inkotmi) were detected as scans in some experiments. They were labeled as false alarms since they are not trying to connect to machines in order to verify the existence of a service. It is interesting to note that almost half of the scans found were due to Slammer worm still in existence.

The definition of a single scan adopted for this evaluation is a source IP attempting to connect to the same service. Using the heuristics, scans were reported based on a network block (obtained from the router, available in netflows). If multiple scans from the same source to the same service were reported for multiple network blocks, these were aggregated into a single scan. In the following tables, the number of scans found using time and connection windows may add up to more than the total number of scans reported. This is due to the fact that scans by one source for a particular service to one network block may have been detected using the time window and another using the connection window.

Table 6.2 shows the number of scans detected correctly and number of false alarms by each method broken down by time and connection window. Detections using time window has a much lower false alarm rate but has significantly less coverage. Using connection window between five to seven times as many correct scans were reported. However, the false alarm rate increased substantially.

If the basic scheme was used, without the connection window extension, it would have detected only 250 scans correctly with five false alarms. If the threshold was lowered from ten to five the coverage increases to 315, but the number of false alarms increase to 70. Using time and connection window with a threshold of ten, detected a total of 1626 correct scans with 198 false alarms, and with a threshold of five, detected a total of 2459 scans correctly with 3638 false alarms. Applying the heuristics to the basic method, reduced the coverage slightly while greatly reducing the number of false alarms. With the usage based method, the coverage increased primarily due to the detection of stealthy scans while maintaining a low false alarm rate.

Most of the false alarms in the basic scheme (with a threshold of five) are due to Blubster (1470), Gnutella (1122), Web browsing / crawlers (561) and E-mail (202) connections. Increasing the threshold to ten reduces the number of false alarms to only 142 for Web and 18 for E-mail and completely eliminates the Blubster and Gnutella alarms for this data set. The drawback of raising the threshold is a significant drop in coverage. By using the heuristics, most of the false alarms can be avoided as can be seen in table 6.2. Several of the false alarms for each detection method were due to replies to Network Time Protocol (NTP) requests from distinct sources. These false alarms can easily be avoided by matching these requests and responses, which is already done by using the data collector described in section 3. The number of false alarms which this would have removed are Usage:6, H_5:3, H_{10}:3, B_5:16 and B_{10}:9.

This project also compared the false alarms generated by H_5 and Usage. The false alarms generated by the usage method do have low counts (less than five) and they do not overlap with the false alarms generated by H_5. Even though the sources of scans reported by H_5 touched at least six unique IPs, the usage method was able to eliminate the false alarms since the connections were to commonly used IP / port combinations.

Method	TP_{time}	FP_{time}	TP_{conn}	FP_{conn}	Total Correct	Total False*
U	292	2	2294	20	2561	22
H_5	314	1	2105	6	2419	6
H_{10}	256	0	1329	4	1585	4
B_5	315	70	2144	3568	2459	3638
B_{10}	250	5	1376	193	1626	198

Table 6.2. Detections and false alarms

Table 6.3 shows the comparison in coverage of scans between different experiments. Entry in row i and column j represents the number of correct scans found in experiment i and not found in experiment j. For example, there are 252 scans correctly detected by the Usage method and not detected by H_5. The greatest coverage is obtained by making use of the usage statistics. However, this method and the methods using the heuristics miss some of the correct scans detected by the basic scheme with a threshold of five. This is due to sources hopping from one network to another and only touching very few machines in each block within the University. In addition, the usage method can miss real scans if the corresponding entries in the usage statistics table have high counts for the connections involved in the scan. For example, a web server will not only cause connections to that server to have very low contributions but also will make connections that map into the same entry in the usage table have low contributions. One way to address this problem is to maintain multiple usage tables with different grouping functions, i.e. using every other bit or using 8 high bits and 8 low bits or some other combination. Legitimate connections will usually fall into blocks with high counts because they tend to connect to highly used servers. Scan connections may avoid detection in one mapping because of the blocking (e.g. by web servers), but will most likely be detected using one of the many alternate mappings.

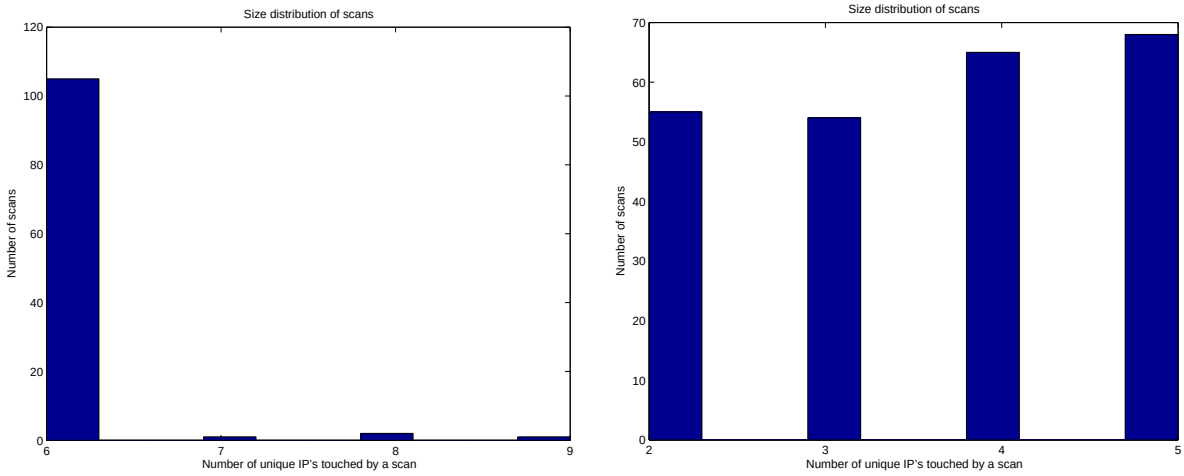
*Some of the false alarms are due to not matching UDP requests with replies. These can trivially be removed by building "UDP sessions, which is done in the data collector described in section 3".

Method	U	H_{10}	H_5	B_{10}	B_5
U	0	976	252	988	267
H_5	109	833	0	848	20
H_{10}	0	0	0	14	13
B_5	164	886	60	856	0
B_{10}	53	55	56	0	24

Table 6.3. Coverage comparison of different methods

Some scans detected by the H5 method are not detected by the usage method and vice versa. Figure 6.2 shows the size distribution of scans detected by only one of the methods. All of the scans detected by the usage method and not by the H5 method are slow scans; they involved less than or equal to five IPs. A separate experiment was performed to investigate what it would cost for the H5 method to catch those scans as well. By reducing the scan threshold from five to four the number of correctly detected scans went up from 2419 to 2801. However, the number of false alarms increased dramatically from 6 to 660. One reason for this increase is that commonly used servers at the University do not have a single IP, but there can be 5 IPs mapping to the same DNS name. Even though a user is using a service on a particular URL, the user might actually be touching five IPs. This shows that without making use of the usage information, it is difficult detect slow scans with a low false alarm rate.

Even though the coverage of the usage method is better than any other methods tried in this project, there are still scans detected by the other methods and not by the usage method. For example, there are 164 scans correctly detected by the basic method that are missed by the usage method. Since usage statistics are not stored in exact detail, there will be some cases where a scanner is touching IP / port combinations that map to frequently-used blocks in the table. In such cases, those scans will effectively be masked. To avoid this problem, another experiment was performed where two different mappings for the IPs were used and maintained in two separate tables. If the score for an IP / port combination rises above the thresholds



a.–Size Distribution of Scans found by H5 and not by Usage

b.–Size Distribution of Scans found by Usage and not by H5

Figure 6.2. Coverage comparison of Usage and H5

using either of the usage tables, then it is declared to be a scan. By doing so, the number of correctly detected scans increased from 2561 to 3082 and the number of missed scans that are detected by the basic method decreased from 164 to 60. In the original experiment, the usage method generated 22 false alarms, six of which were due to NTP. By using two different mappings at the same time, the number of false alarms increased from 22 to 35. Six of these new false alarms are due to ICMP replies which can safely be ignored (host unreachable messages cannot be a part of a scan). However, there were no false alarms due to ICMP replies on the ARL-CIMP data since this is paired with the initiating connection and is known to be a response.

Table 6.5 shows the commonly scanned ports, how many times they have been scanned for, the average size of the scan and their detection method. These statistics are calculated using the results of the usage method as it has the broadest coverage.

Source IP	Port / Protocol	Number of unique IPs touched	Score	Detection method	Start time	End time
63.231.x.245/12	389/udp	2	1.18	time window	0306.00:07:32.777	0306.00:07:32.877
63.239.x.6/24	9370/udp	4	1.01	connection window	0305.23:59:41.382	0306.00:09:55.625
66.41.x.22/20	389/udp	2	1.18	time window	0306.00:18:57.268	0306.00:18:58.268
67.154.x.2/14	524/tcp	3	1.68	connection window	0306.00:06:31.320	0306.00:25:43.315
67.154.x.120/14	524/tcp	4	2.60	connection window	0306.00:06:52.288	0306.00:25:01.178
128.111.x.12/16	123/udp	6	1.58	connection window	0306.00:00:25.382	0306.00:25:44.767
132.163.x.101/16	123/udp	5	1.56	connection window	0306.00:02:17.479	0306.00:26:40.871
132.163.x.103/16	123/udp	5	1.62	connection window	0306.00:02:17.479	0306.00:26:40.871
155.68.x.128/16	6346/tcp	2	1.18	connection window	0306.00:18:24.244	0306.00:23:14.466
158.152.x.222/16	6346/tcp	3	1.36	connection window	0306.00:12:44.682	0306.00:24:21.886
163.120.x.97/16	6346/tcp	3	1.02	connection window	0306.00:02:05.643	0306.00:22:39.261
166.90.x.130/16	2048/icmp	3	1.07	connection window	0305.23:59:05.061	0306.00:23:21.102
169.237.x.1/16	2048/icmp	2	1.18	connection window	0306.00:13:35.355	0306.00:16:30.627
170.215.x.18/21	524/tcp	4	4.43	connection window	0306.00:02:39.163	0306.00:28:16.883
192.5.x.41/24	123/udp	43	11.80	connection window	0305.23:33:18.719	0306.00:27:49.799
192.5.x.41/24	123/udp	11	3.44	connection window	0305.23:57:51.262	0306.00:23:44.762
192.5.x.41/24	123/udp	9	2.39	connection window	0305.23:58:35.609	0306.00:27:51.295
192.5.x.41/24	123/udp	10	3.14	connection window	0305.23:47:34.156	0306.00:23:36.243
192.5.x.209/24	123/udp	40	11.46	connection window	0305.23:40:03.485	0306.00:27:49.799
192.5.x.209/24	123/udp	12	3.06	connection window	0305.23:50:09.081	0306.00:27:38.831
192.5.x.209/24	123/udp	11	3.36	connection window	0305.23:33:18.315	0306.00:26:32.219
192.5.x.209/24	123/udp	9	2.29	connection window	0306.00:04:37.903	0306.00:27:33.975
192.102.x.251/24	0/icmp	3	1.26	connection window	0306.00:08:06.281	0306.00:16:51.992
199.109.x.13/24	781/icmp	2	1.18	connection window	0306.00:22:14.289	0306.00:25:08.290
203.197.x.129/24	2048/icmp	3	1.15	connection window	0306.00:06:23.288	0306.00:22:56.429
204.176.x.5/24	2048/icmp	3	1.07	connection window	0306.00:23:09.285	0306.00:26:39.163
207.46.x.100/18	123/udp	12	2.04	connection window	0306.00:00:00.358	0306.00:18:36.404
207.46.x.100/18	123/udp	11	1.87	connection window	0306.00:01:28.183	0306.00:26:45.619
207.46.x.100/18	123/udp	25	4.26	connection window	0306.00:08:00.773	0306.00:16:19.615
221.233.x.244/14	113/tcp	2	1.18	connection window	0306.00:25:10.402	0306.00:25:25.563

Table 6.4. False Alarms - Usage Method

6.6 Discussion

This section presented new techniques for scan detection that address the shortcomings of the traditional scan detection methods. These schemes, implemented in MINDS, effectively increase the time-window threshold over which scans are detected, reduce the number-of-connections threshold which triggers an alert, while at the same time eliminating many of the false positives that benign protocols yield. The experiments performed at the ARL-CIMP also showed that by using the proposed session data collector, detection precision increased to the point that no false alarms have been observed. Analysts can be reasonably sure that if MINDS alerts on a scan it is indeed a scan. Analysts no longer need to ignore the indication of a slow scan. For

Port	Number of Distinct Scanners	Average Size of Scan	Distinct $Scanners_{time}$	Avg. Size of $Scan_{time}$	Distinct $Scanners_{conn}$	Avg. Size of $Scan_{conn}$
1434	1061	44.0386	23	1290.65	1038	16.42
445	549	32.0055	2	2433	547	23.23
3127	326	89.6472	9	103.33	318	88.98
135	246	148.76	9	3381.67	237	26
137	213	143.174	197	154.51	16	3.56
1080	37	118.946	2	1013	37	64.18
3128	35	62.2286	0	0	35	62.22
139	16	1499.56	2	10925	14	153.07
2048	12	1810.75	10	2155.9	4	42.5
80	8	49.125	1	32	8	45.1
9898	3	193.667	3	193.67	0	0
443	3	6997	2	10431.5	2	64
1433	3	4294.33	2	6401	1	81
8080	2	1064	2	1015.5	2	48.5
79	2	1078	2	1016	2	62
6000	2	1067.5	2	1019	2	48.5
53	2	1065.5	1	2006	2	62.5
515	2	1064.5	1	2047	1	82
514	2	42	1	8	1	78
3389	2	1066.5	1	2008	2	62.5
3306	2	1065	1	2004	2	63
23	2	1076.5	1	2006	2	73.5
22	2	1066.5	2	1033	2	33.5
161	2	1068	1	2054	1	82
143	2	1062.5	1	2004	2	60.5
111	2	1080	2	1080	0	0
110	2	59	2	20	1	78

Table 6.5. Commonly scanned ports

example, an indication by MINDS that one of Microsoft's netbios ports is involved in a stealthy scan can be taken more seriously. Previously an alert for a Microsoft scan was often dismissed, as normal network traffic on these ports frequently triggered the scan detection tool. Now analysts using MINDS at the ARL CIMP and University of Minnesota find that though some indications of scanning activity on Microsoft's netbios ports are still benign, the majority of the time they are not. The benign activity involves only a small set of computers while the set of targets scanned extend beyond that set. Since the analyst is not likely to take the time to consistently investigate the output of scanning tools, having an implicit trust in the output of the scanning tool is of critical importance.

The process described in this chapter is built around using netflows for analysis. However, if the data used comes from the data collector described in Chapter 3 the

precision of the results is increased. The false alarms due to incorrect merging and matching of netflows into sessions are removed. The coverage can also be increased because the threshold to be declared a scanner can be lowered. This can happen because the error introduced by not having precise labels of client and server on each session is removed.

Another way the scan detection process can be improved is by not aggregating the IPs and Ports, but rather keep a sparse table of the counts for each IP and port. While it is possible for this to become prohibitively large, the number of combinations observed at the University of Minnesota for an entire day's worth of data, involving the entire Internet, is only 500,000 destination IP/port combinations. By maintaining precise counts rather than aggregates the problem of busy servers hiding scans of computers that mapped into their bin is removed.

Since the false alarm rates of the scan detection techniques described in this chapter are quite low the alerts make good candidates for anchor points in the 2nd level analysis process described in Chapter 5. These make good anchor points because computers scanning networks are likely compromised computers, and computers replying to scans are likely targets for attacks.

Chapter 7

Conclusions and Future Work

In this thesis several problems relating to network security were identified, and a framework which addresses these problems was proposed, and evaluated on synthetic networking data, as well as data collected from real networks. This framework addresses some of the important problems currently faced by network security analysts, and can be extended through additional modules, or improvements to current modules to address additional problems, and new problems as they arise. A few areas for future work are described later in Section 7.1.

The first problem identified is that network analysis tools do not have access to information at the correct granularity. The data tends to be too course, as in tcpdump or netflow, or very detailed like the sessions created with Bro. The solution for the data deficiencies, discussed in Chapter 3, was to create a data capture tool that was lightweight and semi-stateful. This provides information with sufficient detail for tools such as signature based tools, but also provided higher level information which is appropriate for anomaly detection, scan detection, or communication graph analysis. This technique scales well, and is able to collect data on gigabit networks

with no detectable packet loss, and only minimal CPU and memory requirements.

The second problem addressed was how to provide access to historical information. This information is needed for building profiles, communication pattern analysis, and for forensics. The solution, detailed in Chapter 4, requires a relational database for sessions level information, as well as a disk structure for storing the packet contents. This storage framework has been tested on extremely large networks, and is able to store and provide very quick access to months worth of networking data with simple function calls.

The third major contribution is a framework for determining the larger context of a network event, Chapter 5. This framework leverages the easy access to high quality data provided by the data collection and storage components of this thesis. It has been shown that when used to investigate network attacks this iterative technique has as very high detection rate, and detection coverage of multi-step attacks.

A fourth contribution is a technique for detecting network reconnaissance operations, Chapter 6. This scan detection algorithm has been shown to have a high detection rate, with a very low false alarm rate using netflows from the University of Minnesota. When combined with the data collection component of this thesis the false alarm rate drops to near 0.

7.1 Future Work

Data Collection Since the performance of the data collection component seems to be more than sufficient for monitoring high-speed networks, one logical question is to ask what other capabilities could be added to this collector.

The first is to allow the collection software to receive updated collection rules on the fly. This would allow updates to the collection rules without losing the current

state of the network. In an operational setting, this will allow the collector to be told to collect everything from a certain IP address if some analysis routine determines it is suspicious, without losing any current information on that IP or any other IP address.

The second enhancement would be to add more detailed protocol validation. Since the first foray into a compromise between raw data capture and detailed analysis erred on the side of storing the data above all else, perhaps a more reasonable approach would be to perform a little bit more analysis, but without the full inspection load Bro would incur - something like a Bro-lite.

One of the first new protocol validation techniques to try would be following the network protocols, like TCP or UDP. This seems like the most logical place to begin enhancements since the TCP and UDP protocols are fairly well defined, and do not vary too much. They also do not need much state to be maintained. By following the network implementation of a session the data collected will have less errors, so analysis can be more precise.

Tracking network protocols more in more detail will also make more information available to applications using the data. New fields to add to the structure then would be the number of packets retransmitted, or an estimate of the number of packets dropped. Depending on the implementation, it can be possible to determine this from tcp sequence numbers. However, care must be given to make sure this technique does not suffer from a state explosion problem. This could happen by expecting a session to proceed in some pre-defined way, and missing a step, so the sequence never terminates.

Another area of future work in collection would be to perform some application protocol analysis, for example, studying the setup and transfer of a HTTP or FTP connection. In addition to making new data fields available, such as the HTTP

headers, or if the transfer was text, an image, audio or video file, this can also study if the data transfered in the session changes. For example, if the session begins as an HTTP session, and then becomes an interactive session. If such a deviation were detected, the entire payload of the session could then be collected for analysis.

Protocol analysis can also be used to determine if the traffic being transfered on port 80 is really HTTP web traffic, or is the service masquerading as a web service to get around firewall rules. This technique has become quite common in recent years as administrators have adopted the policy of blocking traffic to ports unless explicitly allowed, which web traffic typically is.

Duplicate Records Bringing together data from multiple networks, or multiple views on the same network as proposed here brings with it the possibility of duplicating, or nearly duplicating, some of the data. For example, if a collector existed in the computer science department at the University of Minnesota, and at the border of the University of Minnesota the data from the CS department leaving the University and on to the Internet could be captured twice. However, due to some fragmentation or different network types (Ethernet, ATM, Token Ring) it is possible the data seen at each collector is slightly different. This prompts the questions, *How to identify duplicate sessions?* and *How to handle duplicate sessions?*

If the sessions are truly duplicates, simply deleting duplicate records may be sufficient. However, if the data is used by people with different roles and different data access permissions selecting the correct record to delete may not be straight forward. For example, if the administrators at the computer science department were not permitted to see the data from the university level collector, removing the record from the CS departments view could make it difficult for the CS department security analysts to perform their job. While giving access to a single connection from the University

level collector may be difficult, for technical, privacy, or policy reasons.

If the session records describe the same network conversation, but some of the features describing the sessions are different, identifying these duplicate sessions and determining how to handle them could be quite difficult. These variations in features could happen if the network path changes from Ethernet to ATM, or if a router in the middle of the path fragmented the packets.

Profiling Now that networking researchers and analysts have quick and easy access to historic information about a network it is possible to make great improvements in the types and quality of models and profiles developed for networks. In Chapter 5 some simple profiling techniques were used that essentially built histograms of previous activity, and labeled everything that happened more than X% of the time to be "normal". While these techniques worked, it is possible to develop much more sophisticated models with better performance.

One of the simplest improvements would be to use multiple features at once instead of a single features at a time like the current histogram profiles. By making use of profiles such as these it would then be possible to detect deviations from a previously observed behavior even if no single feature had anomalous values. Some areas of difficulty in developing these profiles will be in dealing with different types of features, numerical or categorical, at the same time, and how to properly scale and weight each feature.

Another area of work that could hold promise is to perform sequential pattern mining on the network sessions database. The sequences identified through this could possibly be used to identify normal login sequences, those performed automatically and a users typical manual process when logging in. Deviations from the normal operations, such as not checking email within one minute of logging on, could indicate

that the operator of the computer is not the usual operator.

These sequences may also give insight into the communication protocols of an application. These protocols are often ill-defined, which may lead to differences in implementation or changes with different versions, or are proprietary. Understanding how the protocols work can give insight into potential vulnerabilities, or areas for improving either the application or the network to support the application.

Sequence mining might also allow insight into the type of communication happening through an encrypted channel. For example, interactive shell sessions tend to not move very much data, the data movement is sporadic, and the data movement tends to be initiated by the client. A data transfer session tends to move a lot of data in one direction, with only small acknowledgment packets as replies. A video or audio stream tends to move a lot of data in one direction, with no replies. It may be possible to identify sequences that commonly occur in each of these, and other types of network activities. While this will not tell the analyst what is actually being communicated inside of the encrypted session, it can give insight into what types of services are being used, and if new services are provided.

Bibliography

[1] Army High Performance Computing Research Center. http://www.ahpcrc.org.

[2] Arbor Networks. http://www.arbornetworks.com/.

[3] Arctic Region Supercomputer Center. http://www.arsc.edu.

[4] IIS IDA-IDQ Exploit, Cert Advisory CA-2001-13. http://www.cert.org/advisories/CA-2001-13.html.

[5] Apache OpenSSL SSLv2 Exploit, Cert Advisory CA-2002-23. http://www.cert.org/advisories/CA-2002-23.html.

[6] S. Cheung, U. Lindqvist, and M. Fong. Modeling Multistep Cyber Attacks for Scenario Recognition. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, 2003.

[7] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[8] F. Cuppens and R. Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2000.

[9] O. Dain and R. Cunningham. Building Scenarios from a Heterogeneous Alert Stream. In *IEEE Transactions on Systems, Man and Cybernetics*, 2002.

[10] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion detection systems. In *Computer Networks*, 1999.

[11] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion Detection Alerts. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.

[12] Luca Deri. nprobe: an open source netflow probe for gigabit networks. In *Terena TNC 2003*, May 2003.

[13] Luca Deri. Improving passive packet capture: Beyond device polling. In *SANE 2004*, October 2004.

[14] Luca Deri. nflow: Monitoring flows on ipv4/v6 networks. In *TNC 2004*, June 2004.

[15] Luca Deri. High-speed dynamic packet filtering. In *Journal of Network and System Management*, March 2008.

[16] Luca Deri, Rocco Carbone, and Stefano Suin. Monitoring networks using ntop. In *Proceedings of IM 2001*, May 2001.

[17] The Defense Research and Engineering Network. http://www.hpcmo.hpc.mil/Htdocs/DREN/.

[18] Endace Network Card. http://www.endace.com/.

[19] L. Ertoz, E. Eilertson, P. Dokas, V. Kumar, and K. Long. Scan Detection - Revisited. Technical Report 127, Army High Performance Computing Research Center, 2004.

[20] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, P. Dokas, J. Srivastava, and V. Kumar. Detection and Summarization of Novel Network Attacks Using Data Mining. Technical report, University of Minnesota, 2003.

[21] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, and P. Dokas. *Next Generation Data Mining*, chapter 11. MIT Press, 2004.

[22] C. Giovanni. Fun with packets: Designing a stick. http://www.eurocompton.net/stick/papers/Peopledos.pdf.

[23] S.E. Hansen and E.T. Atkins. Automated System Monitoring and Notification With Swatch. In *Proceedings of the Seventh Systems Administration Conference (LISA'93)*, 1993.

[24] D. Hughes. Tklogger. *ftp://coast.cs.purdue.edu/pub/tools/unix/tklogger.tar.Z*.

[25] ISS RealSecure. http://www.iss.net.

[26] R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru, and A. Valdes. System Design Document: Next-Generation Intrusion Detection Expert System (NIDES). Technical report, SRI International, 1993.

[27] Yu Jin, Zhi-Li Zhang, Kuai Xu, Feng Cao, and Sambit Sahu. Identifying and tracking suspicious activities through ip gray space analysis. In *Proceedings of the 3rd Workshop on Mining Network Data*, June 2007.

[28] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings IEEE Symposium on Security and Privacy*, 2004.

[29] T. Karagiannis, A. Broido, M. Faloutsos, and KC Claffy. Transport Layer Identification of P2P Traffic. In *Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference*, 2004.

[30] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: a file system integrity checker. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*. ACM Press, 1994.

[31] Stefan Kornexl, Vern Paxson, Holger Dreger, Anja Feldmann, and Robin Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic, 2005.

[32] Kerry Long. Catching the Cyberspy: ARL's Interrogator. In *Army Science Conference*, 2004.

[33] D. Mutz, G. Vigna, and R. Kemmerer. An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. In *Proceedings of the Annual Computer Security Applications Conference*, 2003.

[34] P. Ning, Y. Cui, and D. Reeves. Analyzing Intensive Intrusion Alerts via Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2002.

[35] P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2002.

[36] P. Ning, D. Reeves, and Y. Cui. Correlating Alerts Using Prerequisites of Intrusions. Technical report, North Carolina State University, Department of Computer Science, 2001.

[37] P. Ning and D. Xu. Learning Attack Strategies from Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2003.

[38] P. Ning, D. Xu, C. Healey, and R. St. Amant. Building Attack Scenarios through Integration of Complementary Alert Correlation Methods. In *Network and Distributed System Security Symposium*, 2004.

[39] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, pages 2435–2463, Dec 1999.

[40] Vern Paxson, Krste Asanovic, Sarang Dharmapurikar, John W. Lockwood, Ruoming Pang, Robin Sommer, and Nicholas C Weaver. Rethinking hardware support for network analysis and intrusion prevention. *1st Workshop on Hot Topics in Security (HotSec'06)*, 2006.

[41] P. Porras, M. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2002.

[42] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Distrurbances. *National Information Security Conference*, 1997.

[43] RRDtool. http://oss.oetiker.ch/rrdtool/.

[44] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the Performance of Network Intrusion Detection Sensors. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2003.

[45] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[46] Skaion corporation. http://www.skaion.com/news/rel20031001.html.

[47] Snort - The Open Source Network Intrusion Detection System. http://www.snort.org.

[48] Robin Sommer. *Viable Network Intrusion Detection in High-Performance Environments*. PhD thesis, TU Munchen, 2005.

[49] Open Source. Tcpdump. http://www.tcpdump.org.

[50] Cico source code theft part of 'mega-hack'. http://www.theregister.co.uk/2005/05/10/cisco_hack_investigation/.

[51] Swedish hacker penetrates cisco and nasa. http://www.hackwire.com/comments.php?catid=1&id=166.

[52] Internet attack called broad and long lasting. http://nytimes.com/2005/05/10/technology/10cisco.html.

[53] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans, 2002.

[54] Standard Template Library Programmer's Guide. http://www.sgi.com/tech/stl/.

[55] Cisco Systems. Netflow services and applications. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm.

[56] S. Templeton and K. Levit. A Requires/Provides Model for Computer Attacks. In *Proceedings of New Security Paradigms Workshop*, 2000.

[57] A. Valdes. Detecting Novel Scans Through Pattern Anomaly Detection. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, 2003.

[58] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.

[59] F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. In *IEEE Transactions on Dependable and Secure Computing*, 2004.

[60] G. Vigna, W. Robertson, and D. Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2004.

[61] Phil Wood. Memory maped libpcap. http://public.lanl.gov/cpw/.