SANDIA REPORT

SAND2003-8624
Unlimited Release
Printed October 2003

Adaptive Network Countermeasures

Jamie Van Randwyk, Eric Thomas, Anthony Carathimas, Randy McClelland-Bane

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2003-8624
Unlimited Release
Printed October 2003

Adaptive Network Countermeasures

Jamie A. Van Randwyk, Eric D. Thomas, Anthony G. Carathimas, Randy T. McClelland-Bane
Information Security
Sandia National Laboratories
P.O. Box 969
Livermore, CA 94551
{jvanran, edthoma, agcarat}@sandia.gov, randy@frenzy.org

Abstract

This report describes the results of a two-year LDRD funded by the Differentiating Technologies investment area. The project investigated the use of countermeasures in protecting computer networks as well as how current countermeasures could be changed in order to adapt with both evolving networks and evolving attackers. The work involved collaboration between Sandia employees and students in the Sandia - California Center for Cyber Defenders (CCD) program. We include an explanation of the need for adaptive countermeasures, a description of the architecture we designed to provide adaptive countermeasures, and evaluations of the system.

This page intentionally left blank

Contents

List of Figures

This page intentionally left blank

Nomenclature

ANC	Adaptive Network Countermeasures
ANCP	Adaptive Network Countermeasures Protocol
GI	general information [modules]
HENC	Honeyd Enabled Network Countermeasures, countermeasure component of Adaptive Network Countermeasures
IDS	intrusion detection system
Athena	decision-making component of Adaptive Network Countermeasures
Bro	an open source intrusion detection system
datagram	also called a "packet"
Honeyd	open source virtual honeypot software
ipfw/ipfw2	the FreeBSD packet filter / firewall
mon	a Sandia-developed intrusion detection system
Nessus	an open source network scanning tool
Nmap	an open source network scanning tool
Snort	an open source intrusion detection system
Xprobe 1 / Xprobe2	an open source operating system fingerprinting tool

This page intentionally left blank

1 Introduction

Internet attacks are on the rise as shown by the recent Blaster [1] worm. Our operational experience has shown an increase in the time and manpower required to defend against attacks and clean up in their wake. Along with a general increase in attacks, our experience has also been that targeted attacks are becoming more frequent. Malicious users are now tailoring their attacks for maximum effect against specific target machines.

Targeted attacks are made possible by advance reconnaissance conducted by malicious users. Attackers use tools such as `Nessus` [2] and `Nmap` [3] to conduct their network surveys. These tools rely on the fact that the TCP/IP protocol suite was designed to be an open suite of protocols. The protocols freely give information about the computer on which they are running rather than limiting that information.

Efforts have been made to mask certain network characteristics in order to prevent malicious users from tailoring their attacks. Perimeter defenses such as firewalls and proxy servers have been deployed to limit the information available to attackers. Honeypots are being installed to distract attackers from "real" network resources.

We believe these solutions leave much room for improvement. We want to be able to control the view of our network that can be obtained by outside users, whether malicious or not. We believe that by limiting, and even creating, the view from the outside, we can severely restrict an attacker's ability to conduct reconnaissance of our network and exploit weaknesses or vulnerabilities found.

In our research we combined the common network security ideas of network countermeasures, automated response, and honeypots to create Adaptive Network Countermeasures (ANC). Our goal was threefold: to provide a comprehensive and robust network countermeasure system, to dynamically (in real-time) detect and respond to malicious traffic directed to nonexistent and existent machines on our network, and to design and build an extensible architecture for conducting the aforementioned activities.

In this paper we discuss the background surrounding network scanning, network countermeasures, and automated response. In the following section we lay out our ANC architecture and our working implementation of that architecture. We discuss in depth the design of the countermeasure system, our system for making decisions about the type and timing of network countermeasures, and the glue that integrates the two systems into ANC. We then provide results describing the performance of ANC as well as our evaluation of its effectiveness in repelling malicious users. We finish by providing our thoughts regarding future improvements to the ANC architecture and our prototype implementation.

2 Background

The concept of network scanning has been around for a long time. Network scanning tools including `Nessus` and `Nmap` have made network reconnaissance easy. The concept of fingerprinting operating systems using specially crafted TCP/IP datagrams is newer as explained by Fyodor in [4]. After an attacker has performed a network scan, he/she will further profile machines based on which interesting ports were reported open. This further profiling can include OS fingerprinting, grabbing banners from running services, and other techniques. `Nmap` is the best known of these OS fingerprinting programs, using specially crafted TCP and UDP datagrams to gather information about a remote operating system. Ofir Arkin wrote `Xprobe 1` and `Xprobe2` [5] to fingerprint remote machines using mostly ICMP messages. Arkin first described his methods in [6]. These two fingerprinting tools each include a database of the mappings between a particular operating system and responses to probes. We used these databases essentially in reverse, crafting our own TCP/IP datagrams according to the database information provided for a chosen operating system.

Using automated response on a production network is risky because of potential interference with legitimate (non-attack) network traffic. In order to mitigate potential interference with the production network, [7, 8] allow some leeway in their responses. We take this approach as well, requiring a given threat to be of a sufficient weight before we issue a response on the network. Additionally, we make use of user-defined white lists of Internet Protocol (IP) addresses to avoid interfering with certain critical machines inside and outside of our network.

Many of the most popular intrusion detection systems (IDSs) provide methods to automatically block IP addresses and IP address ranges [9, 10, 11]. Some even provide deterministic network responses directed toward the offending IP [12]. In our architecture we introduce a layer of intelligence on top of our implementations of these functions. We will begin by describing our innovations including

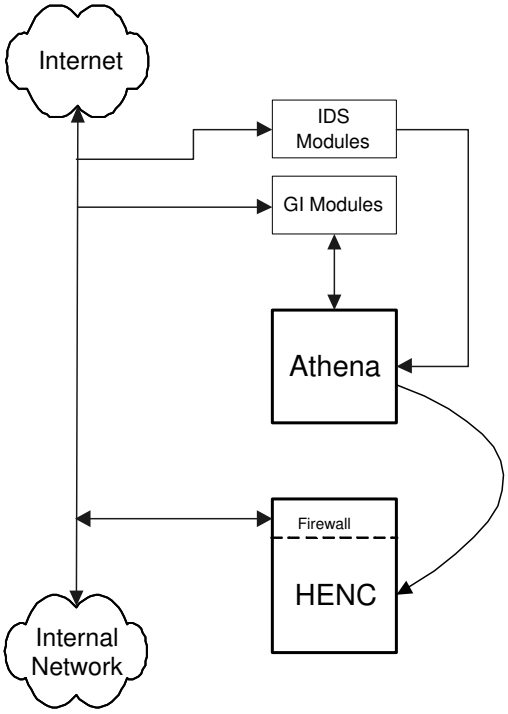

Figure 1: ANC Architecture

dynamically configurable network countermeasures, our pluggable architecture, our attempts to limit false positives to the bare minimum, and our information correlation intelligence.

3 Accomplishments, Design, and Implementation

Our design, and thus our project, was split into two main parts: the countermeasures component (HENC - Honeyd Enabled Network Countermeasures (pronounced "Hank")), and the decision-making component (Athena). As seen in Figure 1, the two components each run on their own machine, using a back channel for unidirectional communication.

3.1 Countermeasures

We designed HENC based on an existing production system at Sandia National Laboratories. Laboratory staff had written individual programs to provide protocol and service specific countermeasure protection for the networks. We wanted to combine these programs into a larger system capable of sending countermeasures using TCP, UDP, and ICMP as well as emulating various network services over

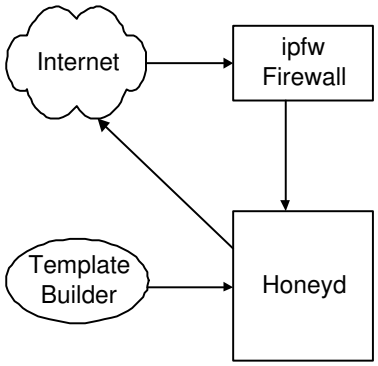

Figure 2: HENC Architecture

these protocols. As we developed our plans, we realized that we wanted to essentially recreate all the deterministic responses produced by TCP/IP network stacks when stimulated by all possible inbound datagrams. Our goal was to build our own IP stack without actually offering any real services.

We decided to build our system to emulate the IP stacks of as many operating systems as possible. This would be accomplished by reversing the use of the `Nmap` and `Xprobe` OS fingerprint databases. By responding to network traffic consistent with the way a real OS would respond, our program would provide countermeasures indistinguishable from the response traffic of a real operating system. Our program needed to provide countermeasures that would create the appearance of fully functioning Windows 2000, Mac OS X, Linux, or other machines residing on our network.

We chose `Honeyd` [13], a virtual honeypot software package, to act as the base for our countermeasure system. Initially `Honeyd` could read from the `Nmap` fingerprint file and send TCP and UDP datagrams based on the "recipes" in that file. In order to make our countermeasure system more robust, we added support to `Honeyd` for reading the `Xprobe2` fingerprint file and transmission of ICMP datagrams according to that file.

Our countermeasure system was designed to run on a single machine as a part of the larger adaptive network countermeasures framework. We architected our system to run on FreeBSD 4.7 and 4.8, but it would be easy to make changes to work with other UNIX-like operating systems. In order to allow the sending of countermeasures with an IP ID field that is predetermined by our system or with an ID of zero (as many Linux systems do), we wrote a small patch for the raw sockets portion of FreeBSD's IP stack (see Appendix B).

The HENC architecture has `Honeyd` at its base

as shown in Figure 2. `Honeyd` builds network datagrams and sends them out. The software knows how to build those datagrams based on input from both the `Nmap` and `Xprobe2` fingerprint files. We map individual operating systems and services to IP addresses on our network using `Honeyd`'s configuration file.

We wrote a utility named Template Builder to aid in creating host templates for the `Honeyd` configuration file. The utility actually creates a complete configuration, but its focus is on creating a specified distribution of operating system templates across all possible IP addresses being protected by ANC. For instance, a system administrator can create a configuration file for Template Builder similar to the one shown below:

```
# cat net.cfg
iprange 10.3.4-5.1-254

40%     Microsoft Microsoft NT 4.0 SP5-SP6
10%     NetBSD 1.6
10%     Sun Solaris 2.3 - 2.4
3%      3com Office Connect Router 810
2%      Compaq Tru64 UNIX V5.1A (Rev. 1885)
25%     Linux Kernel 2.4.0 - 2.5.20
5%      Apple Mac OS X 10.1.5
2%      Apple Mac OS 9 - 9.1
3%      SGI IRIX 6.5.14
```

For the given range(s) of IP addresses, the utility will create random operating system templates based on the supplied percentages. Template Builder also has a limited capability to modify configuration files that already exist.

We modified `Honeyd` to provide a mechanism whereby the binding of an operating system template to an IP address changes randomly, at intervals specified by the system administrator. The standard `Honeyd` syntax for setting a template personality and binding an IP address to it is as follows:

```
set my_template personality \
      "Microsoft NT 4.0 SP5-SP6"
bind 10.3.1.3 my_template
```

Using our new keyword, `mutate`, we bind the above template so that it will change to a random personality (operating system) every 100 seconds:

```
bind 10.3.1.3 my_template mutate 100
```

If no value is specified after the `mutate` keyword, a default of 300 seconds between personality changes is used.

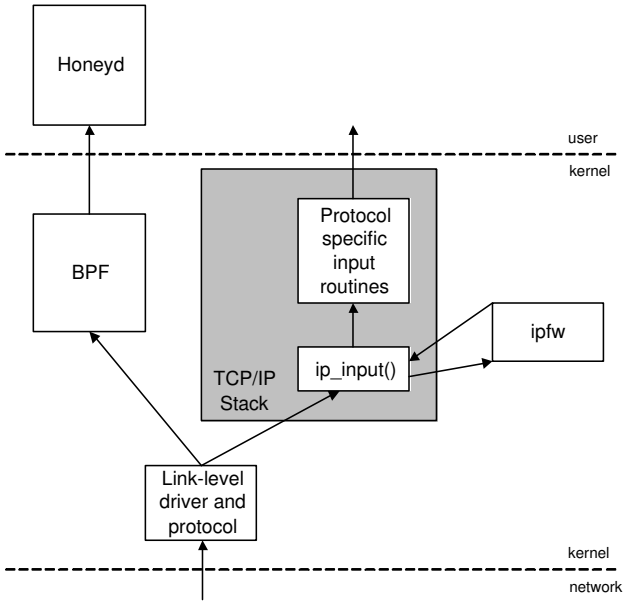

Figure 3: Typical datagram flow to `Honeyd`
BPF makes copies of the datagrams arriving on the machine [15]. `Honeyd` receives these datagrams via the libpcap library. Datagrams not destined for this machine are filtered out by the `ip_input()` function.

We configured `Honeyd` to respond for every address that we wanted to protect on our network. This included both unused and used address space. The `ipfw` firewall mechanism [14] provides an easy way for us to regulate the outside IP addresses that receive countermeasures. We used the standard `ipfw` program, but `ipfw2` should work as well. HENC's firewall rules are written so as to allow all outbound traffic while allowing inbound traffic only from previously determined malicious users. If a network datagram makes its way into our system, it is going to be responded to by our implementation of `Honeyd`.

When running `Honeyd` on FreeBSD, it relies on libpcap [16], which in turn relies on BPF (The BSD Packet Filter) [15], for receiving its IP datagrams. The FreeBSD IP stack is written such that BPF intercepts and copies IP datagrams before the `ipfw` packet filter is called (Figure 3). BPF works this way by design of course, but we needed the ability to block most inbound datagrams so that our countermeasures did not interfere with production network operations. So we added an interface to `Honeyd` giving it the capability of receiving datagrams via a FreeBSD divert socket in addition to the existing libpcap interface (Figure 4). Use of these interfaces is mutually exclusive. We enable this feature

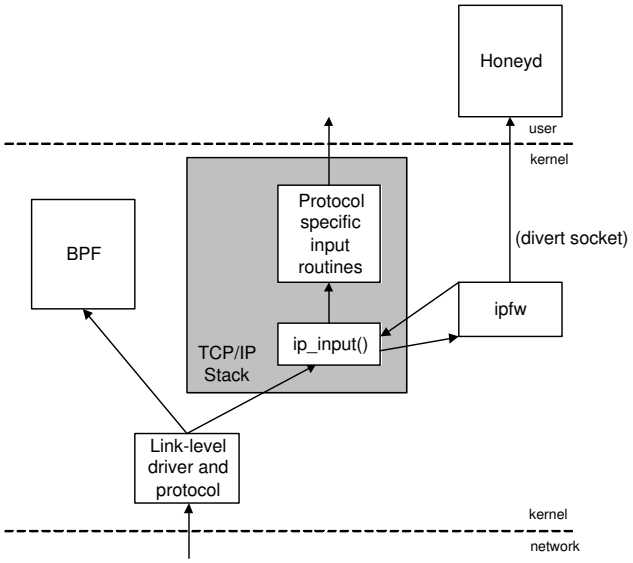

Figure 4: Modified datagram flow to `Honeyd`
BPF makes copies of the datagrams arriving on the machine [15]. No applications are listening to BPF descriptors. Datagrams are passed from the link-level driver to the `ip_input()` function in the TCP/IP stack. `ip_input()` sends the datagrams to `ipfw` before deciding whether to send them up the stack. `Honeyd` receives datagrams from `ipfw` via a divert socket.

by using `-d divert_port_number` in the standard `Honeyd` command-line. Our inbound `ipfw` rules are generated to include the `divert divert_port_number` syntax to direct accepted datagrams to `Honeyd`.

HENC generates its `ipfw` rules upon startup and when told to do so by Athena. Athena talks to HENC using ANCP (Adaptive Network Countermeasures Protocol). ANCP is built on top of UDP and consists of a C client library used by Athena and a standalone PERL server running on HENC. The server configures the `ipfw` firewall upon startup and maintains the firewall rules thereafter. ANCP can also be used to call the Template Builder utility in order to generate new configurations dynamically. Currently ANCP can only alter IP-to-template bindings by adding and removing the `mutate` keyword, but as the capabilities of Template Builder are expanded, the ANC protocol will be extended as well. It should be noted that the `ipfw` portion of ANCP affects outside (supposed malicious) IP addresses and the interface to Template Builder affects inside IP addresses (countermeasure sources).

Figure 5 is a flow chart of HENC's operations starting with the ANCP client in Athena and ending with a network-level response directed toward a

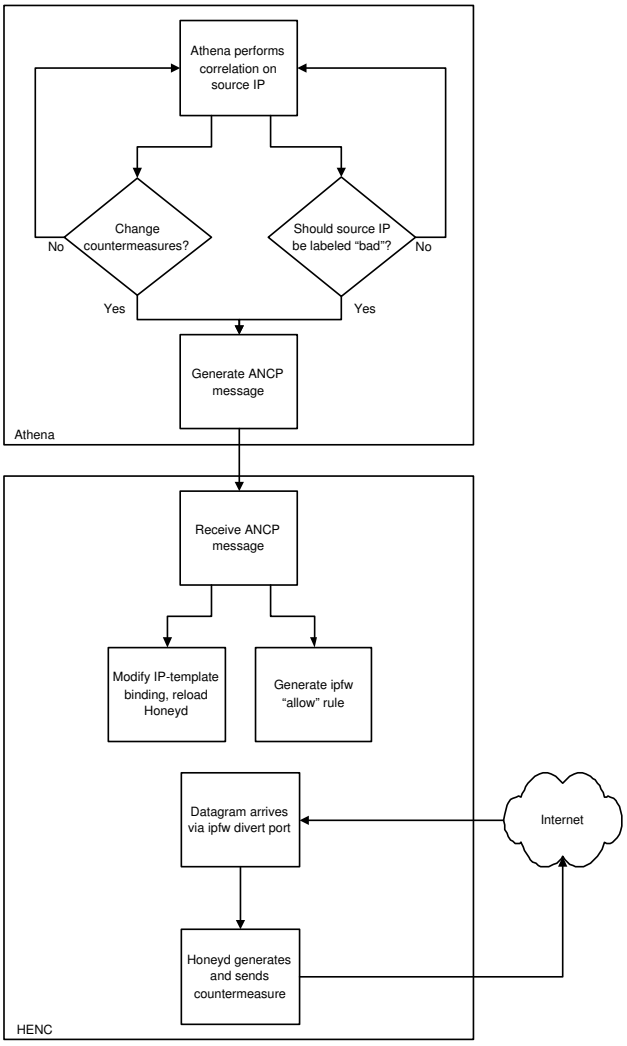

Figure 5: Flow of datagrams from Athena to HENC

malicious user. ANCP currently supports sending messages of two different types. In the first scenario, Athena makes a decision that an outside IP address is bad and generates an ANCP message indicating that the address should be blocked. The ANCP server generates a firewall rule allowing inbound traffic from the malicious IP address. Inbound datagrams are then diverted to the divert port being used by `Honeyd`. `Honeyd` will craft a response based on the destination IP-to-template binding in its configuration. HENC then sends the response, destined for the malicious user.

In the second scenario, Athena generates a message that changes the characteristics of countermeasures being sent from an inside IP address. The ANCP server calls Template Builder, passing a local IP address as a parameter, and changes the corresponding IP-to-template binding. A successful execution of Template Builder triggers a SIGHUP to `Honeyd` and subsequent reload of `Honeyd`'s configuration. Countermeasures continue to be crafted and sent according to the new configuration.

3.2 Athena

3.2.1 Design

Athena is the part of the ANC system that makes decisions about when to send countermeasures and what the countermeasure behavior should be. Athena makes ANC adaptive by changing how it responds based upon input from different modules. Athena accepts two different types of input: that from Intrusion Detection System (IDS) modules and input from General Information (GI) modules. When information from these two module sets are combined and correlated by Athena, Athena determines what countermeasures, if any, HENC will send.

All of the decision making of Athena revolves around a single piece of data, the source (outside) IP address of a network datagram. Processing is initialized when an IDS module sends an alert to Athena. Athena then creates a cumulative score for the source IP address, potentially using information from GI modules, and adds the information to a database. If an IP address already has an associated cumulative score, the score is updated in the database. If the cumulative score for an IP address ever exceeds a (user-defined) threshold, ANC initiates countermeasures against that IP address.

The behavior of an outside IP address that has already generated an IDS alert may cause an alteration of the countermeasures used against it. If a source IP address is continually exhibiting "bad behavior," it will continue to be responded to with countermeasures. If a source IP address is occasionally committing "small offenses," these offenses will build up over time and the source IP address may be responded to with countermeasures. These two results are possible because Athena adds to the IP address's cumulative score when an offense has been committed. Finally, over time and through "good behavior," an IP address can be removed from the list of addresses to which network countermeasures are directed. This is carried out by a separate "aging" thread within Athena which periodically decrements the cumulative score associated with each outside address.

Determining which countermeasures to send is a simple task. Whenever the cumulative score is within a certain range, the system will perform a specific countermeasure. This allows ANC to respond with more aggressive countermeasures whenever a source IP address has a very high cumulative score, and less aggressive countermeasures when a source IP address has a low cumulative score. The actual countermeasures and threshold ranges are customizable by the site implementing the ANC system.

3.2.2 Modular Code

ANC was architected with a very specific goal: adaptability. There are two ways in which Athena can be considered adaptive. First, as described above, Athena will adapt countermeasures based upon different behavior exhibited by a source IP address. The second way Athena is adaptive is by providing a level of code modularity. When new network security technology is created, particularly in the realms of intrusion detection and network statistics information, these technologies can be plugged into the ANC framework. Thus ANC retains the possibility of remaining a useful layer of network security long into the foreseeable future.

First, when new intrusion detection systems are developed or improved, they can be used to send alerts to Athena. Fortunately, this does not require modification of the IDS code itself. The new IDS system can be used as an additional IDS module by writing a wrapper for the IDS. The wrapper will translate alerts generated by the IDS into a common format that Athena understands. This is the Athena Alert Protocol (AAP). The new IDS and wrapper are usually run on a computer separate from Athena, and AAP messages are sent over a local, usually private, network.

The second form of modularity comes from the

ability to add new General Information (GI) modules. Athena queries these modules, and the response is used in the information correlation process. To add a new GI module, the implementer must create or modify a function that performs all of the correlation. The `correlation()` function retrieves information from the information modules, each with its own specific method of retrieval, and then correlates all of the retrieved information to form a number used to update the cumulative score.

The third and final form of modularity in the ANC framework is the addition and modification of what countermeasures to use and when to use them. As described above, different countermeasures can be used when the cumulative score falls within a specific range. When new countermeasures are developed, they can be integrated into the ANC framework to increase the adaptability.

3.2.3 Implementation

IDS Modules – In our prototype implementation of ANC, we use `Snort`, a popular open source intrusion detection system, as our solitary IDS. We wrote a wrapper to allow `Snort` to communicate with Athena via the Athena Alert Protocol. The threat level of an alert is specified in a configuration file. When the `Snort` wrapper sees a specific string, specifying the type of threat, the wrapper converts that to a static number, which becomes the threat level of that alert.

GI Modules – We use three different sources of General Information for correlation: Country Check, NetState, and AthenaDB. The first two are considered General Information modules. Country Check, when queried with a source IP address, will return a number that indicates the sensitivity of that country according to the Department of Energy's Sensitive Countries List. A zero value means the country is not sensitive, one represents a moderately sensitive country, two is sensitive, and a value of three represents a highly sensitive country. The module is queried via a direct network connection.

NetState is a different type of information module. NetState, internally developed Sandia software, can be used outside of ANC for the purpose of passively gathering information about the "demographics" of a network. NetState records what services are running on networked computers and what ports they are running on. NetState also stores information about the operating system running on each computer and summaries of previous network activity. Athena uses the NetState GI module to get specific information about the target of an attack. This information includes the most recent time of connection to the target port, the first recorded time of connection to the target port, and whether the port and recorded service correspond to the IANA standard port assignment [17]. Information is gathered over the network via a direct connection to the NetState MySQL database.

Athena – Athena is the most complex part of our implementation of the ANC framework. It was designed with efficiency, adjustability, and modularity in mind. Multiple threads, implemented with `pthreads` (POSIX threads), handle the many tasks of Athena. When first initialized, Athena's Scheduler thread initializes variables and then starts the remaining threads. When finished performing initialization, it waits for a signal from the operating system to close all threads and clean up when Athena terminates.

During initialization, Athena creates an AlertHandler thread. This thread listens on a local UDP socket for Athena Alert Protocol messages. When an AAP message is received, the following operations are performed: First, the alert message is added to a cache of alerts, which is then used to detect floods of attacks that could potentially overwhelm normal Athena processing. Next, the threat level of the alert is checked to see if it is already above the threat threshold. If so, Athena uses ANCP to request that HENC respond to the attacker. If not, the alert is queued.

Athena also starts MessageHandler threads, which dequeue alerts from the alert queue and perform the information correlation in order to determine if the threat is high enough to warrant countermeasures. Both AlertHandler and MessageHandler threads update the AthenaDB with the new cumulative score for future correlation as well as other statistics.

AthenaDB uses a custom PostgreSQL database that holds attack statistics. In particular, the AthenaDB is queried for the most recent time of threat from the attacker, the number of attacks from the same subnet, the total number of recent threats, and the number of recent threats of the same type. This information, along with information from the GI modules is all gathered in the `correlation()` function, which returns a number representing the threat level and in turn is used to modify the cumulative level. (See Appendix C: Information Correlation)

Finally, Athena starts up the FloodHandler thread, which performs two tasks. First, it monitors

the threat cache. If a large number of threats are received in a small amount of time, it sets up variables to indicate that the AlertHandler should look for multiple threats of the same type in the cache and handle such threats as an attack flood. FloodHandler's second task is to periodically decrement all of the cumulative scores stored in the AthenaDB according to the aging policy. This allows false positives to be removed from the malicious address list over time.

4 Evaluation and Performance

4.1 ifpw

FreeBSD's `ipfw` works well within HENC for controlling the breadth of datagrams responded to. Adding and removing `ipfw` rules via ANCP is fast. Work done by Mikula, Tracy, and Holling [18] on dynamic spam blocking indicates that `ipfw` can filter efficiently up through at least 1,000-1,500 firewall rules. Their work was performed on a machine consisting of a 1GHz Athlon processor, 640MB of RAM, and FreeBSD 2.2.8-STABLE. Our tests never exceeded this number of rules, so we were not able to push the limits of `ipfw`. In addition, we expect our system to scale even better because we are using a newer version of `ipfw` on more powerful hardware. We did not investigate `ipfw` in depth enough to determine if performance differences will surface because we relied on a default "deny" policy in conjunction with "allow" rules and the previous work used a default "allow" policy with "deny" rules.

4.2 Snort

Currently we use the default rule-set for `Snort`. We didn't want to spend time tuning the rule-set for our network since this is an art already established by system administrators. Our `Snort` system was overwhelmed on our network with traffic averaging 8Mb/s. It is obvious that the rule-set needs to be trimmed in order for `Snort` and the `Snort` alert wrapper to run effectively. The rule-set should be similar to that of the IDS system being used for production network monitoring.

4.3 Athena

We tested the rate at which IDS alerts could be received by Athena. Athena was run on a machine with a 1.7GHz Athlon XP2000+ CPU, 512MB of RAM, and a Fast Ethernet (100Mb/s) network interface card. We ran a test program from another machine that simply spit out IDS alerts as fast as it could. We ran several tests sending 10,000 alerts – a number we decided was unrealistically high for a real enterprise network at Gigabit Ethernet speeds – as fast as possible. Athena was able to handle every single alert without dropping any.

Next we tested sending 1,000 alerts – still a high number except when under a Distributed Denial of Service (DDoS) attack – as fast as our testing tool would generate them. We ran this test four different times without Athena dropping any alerts. We restarted Athena in between tests to clear out its cache. The total time to process the alerts was only recorded for two of the tests. It took about 10.5 seconds to process the alerts both times. This showed that Athena could process each alert in little more than 1/100 of a second. It appears then, that Athena will be able to process somewhere between 90 and 100 threats per second on any similar machines. We anticipate that IDS alerts (on a "tuned" IDS machine) will not occur nearly this often, even on an enterprise network with a Gigabit Ethernet infrastructure.

We noticed three obvious processing bottlenecks in Athena. The first bottleneck is a result of slow interactions with the databases. The database queries to both NetState and AthenaDB take the most time within Athena, even with AthenaDB running on the local Athena machine.

Because of the time spent querying the databases, the processing of alert messages causes a backlog since receiving alerts is much faster. This second bottleneck could potentially fill up Athena's alert cache, causing alerts to be dropped.

The third bottleneck appears when Athena is run with multiple MessageHandler threads. On our single processor system, multiple threads tend to slow down the overall speed of processing messages. We believe that multithreaded or multiprocessor systems will be able to handle multiple threads within Athena more efficiently.

We also noticed that processing the first alert after starting Athena routinely takes more time than subsequent alerts. The first alert often took between 1.5 and 3.5 seconds for Athena to process. Further alerts were processed in the 10,000 microsecond range.

4.4 ANC as a Whole

We ran two different tests to evaluate the overall performance of our ANC prototype on our production network. The first test determined whether or not we could send countermeasures to attackers

from a particular address while someone performed real work across the Internet on the computer with that same network address. Our test showed that we could effectively send countermeasures to protect a computer while that computer participated in production work via the Internet. HENC and Athena were each running on their own computer– a 2.8GHz Intel Xeon CPU, 1 GB of RAM, and Gigabit Ethernet NICs. `Snort` was run on a machine with dual 2.8GHz Intel Xeon CPUs, 1 GB of RAM, and Gigabit Ethernet NICs.

To do this, we wrote a rule for `Snort` that would interpret standard ICMP echo request datagrams as malicious packets. Each time `Snort` received one of these datagrams, it notified Athena, which in turn incremented the malicious IP address's cumulative level by a value of 5. Our threat level threshold was set at 25, so that six ICMP echo requests (within the time limit of the threat level aging routine) would cause the offending IP address to be added to HENC's list of hosts to respond to with countermeasures. We configured `Honeyd` to respond with network traffic typical to a specific operating system (randomly chosen by us) as if no TCP or UDP ports were open. Using an attacking host under our control on the Internet, we began sending ICMP echo requests to the computer we were trying to protect. After sending six echo requests and receiving six echo replies from the real computer, we started receiving duplicate echo replies. This was a result of both HENC and the real computer replying to the attack echo requests. We verified that HENC had received the appropriate information from Athena by listing its `ifpw` rules. We tried to connect to the protected computer via `ssh` and were not able to connect. Using a good host under our control on the Internet, we were able to connect to the protected computer immediately via `ssh` and perform our work.

We changed the test slightly so that both our attacking host and our good host were connected to the protected computer via `ssh` before `Snort` detected our malicious echo requests. We initiated the echo requests from the attacking host, and after six requests, HENC terminated the `ssh` connection to that host while we continued to conduct our work from the good host.

Our second test of ANC determined the average time between detecting an attack and sending the first countermeasure in response to the attack. We used the same `Snort` rule of six ICMP echo requests to identify our "attack." We determined this time by measuring the latency between the time of the sixth datagram arriving and the time of the first countermeasure datagram (echo reply) leaving. We used a `tcpdump` network tap at our border router to take these measurements.

We ran six tests:

Run	ANC Latency (in seconds)
1	.720362
2	.690251
3	.510538
4	.450718
5	.300322
6	.633722
Mean	.551

Notice that there is a rather large deviation from the mean in most of the data points. We were more interested in the order of magnitude of the latency rather than precise values.

There may be many causes for the large deviations, assuming static time for the cost of correlation and communication over the control network. For one, the reliability of packet delivery over the Internet can affect the results. For example, if during a given run datagrams six through ten were lost, then the start time of the run would actually be the reception time of the eleventh packet, not the sixth. This would actually cause our calculation to understate the latency.

Another (more likely) cause of the deviations is that `Snort` is not receiving all of the ICMP echo requests. Because our `Snort` process is utilizing the CPU close to 100%, it is highly likely that the IDS is not running fast enough to keep up with the stream of incoming datagrams. The result is that datagrams are being dropped from the receive buffer. Some of those dropped datagrams would likely be the malicious ICMP echo requests. Due to the dynamics of Internet traffic, it would be very difficult to predict which of the echo requests is being dropped. Our sixth recorded datagram may actually be the tenth datagram sent, for instance. This phenomenon would also cause our calculation to understate the actual latency.

We believe the latency of 551 milliseconds is acceptable in a real-time countermeasure system. Optimizations could be made to the `Snort` ruleset, the control network, and the ANC software itself, if less latency were required.

5 Future Work

While working on this project, we discovered several areas where we would like to spend more time. Be-

low we outline several areas that we believe deserve further investigation.

5.1 HENC

Currently, countermeasures are sent in response to inbound datagrams as determined by the binding of the destination IP address to a `Honeyd` template. Using the `mutate` keyword, we can randomly change that binding at a specified interval. We would like to allow Athena to choose a specific `Honeyd` template (and thus operating system) to rebind an IP address to. In addition, we would like to dynamically open and close virtual ports of machines on the protected network.

We envision countermeasures that are even more dynamic in nature than they currently are. We would like to investigate sending different countermeasures from the same IP address when the source of the malicious datagrams differs. Doing this would require a substantial overhaul of the underlying `Honeyd` code in order to allow for a binding between source and destination addresses. Athena's scoring system, with its varying threat levels, would allow easy use of these types of countermeasures.

As noted in the performance evaluation of ANC, attackers will receive two responses to each of their datagrams when attacking real machines. This is because both HENC and the real computer respond. A smart attacker may analyze his/her probe or attack results at the network level (as opposed to blindly accepting results given by the attack applications) and thereby see that he/she is receiving countermeasures. The attacker could then alter his/her attack methods to slip into the network undetected, without being subjected to countermeasures. HENC could be modified to communicate with a firewall on the protected network. The firewall would filter out network datagrams directed from the real computer to the attack host once countermeasures have been initiated to protect the real computer.

Also, we would like to investigate the performance of `ipfw` further to determine its scalability on the newest CPUs with the most up-to-date FreeBSD operating system. We were not able to find other recent research in this area.

5.2 IDS Modules

We currently only support receiving IDS alerts from `Snort`. We would like to write additional IDS module wrappers to support Sandia's `mon` system and Vern Paxson's `Bro`.

5.3 GI Modules

Currently, to incorporate additional GI modules, the user is required to modify the Athena source code directly. To increase the usability and modularity of the system, we want to create a standard interface to the modules using an Expect script. The wrapper would listen for a standardized request, query the GI modules, and then translate the response from the GI modules to a strictly formatted message useable within Athena.

5.4 Athena

Athena relies on the IDS modules for recognition of malicious patterns of traffic on the network. If the IDSs do not detect a threat, Athena will not have a stimulus to begin correlation. We would like to initiate the Athena correlation engine based on other stimuli to the system. Pattern recognition could take place within Athena as well, tipping the system off to threats before an IDS has picked them up. Using sources other than those typically used by an IDS could allow earlier detection of threats.

We would like Athena to gather more statistics about its performance. Athena already computes and uses the real-time load of the system, but it would also be useful to compute the CPU load for each type of alert and keep a history of measured loads. Athena could use these values to adjust thresholds at various times during the course of a day or month when the load is abnormally high or low. The system would essentially adapt to varying network conditions, changing the priority and/or types of threads to appropriately handle the load (or expected load) at any given time.

It is difficult to handle real-time threats because alerts can arrive at gigabit speed from the network, and analysis and processing of the alert may require many network based queries and numerous calculations. We could speed up Athena's analysis by replacing the PostgreSQL back-end database with a real-time database such as Prevayler [19], which is fully ACID compliant and much faster.

Another way to speed up real-time threat handling would be to have Athena adapt to suit the need of the given configuration. Since the GI modules are user configurable, we cannot predict how long it would take to query them all. Having Athena perform this calculation ahead of time would provide the system administrator a means for deciding which GI modules Athena should query based on the expected slowdown.

We could also speed our system up by load balancing the Athena duties. We could link multi-

ple systems running Athena, thereby adding redundancy and increasing capacity for analysis of IDS alerts.

6 Conclusion

The Adaptive Network Countermeasure system has proven to work well in our test environment and in limited production testing. We believe that a full-scale implementation of this system on a production network would produce significant results in repelling Internet attacks while not interfering with legitimate use of the network.

Through our work we have shown that we can provide a comprehensive and robust countermeasure system. The system responds quickly enough to be effective in repelling active attackers. Network datagrams generated by the system appear authentic and succeed in leading malicious users in the direction of our choosing.

The ANC system is able to dynamically (in real-time) detect and respond to malicious traffic directed to nonexistent and existent machines. This behavior allows us to send countermeasures to protect a machine that is conducting typical business via the network without interfering with that legitimate communication.

The Athena portion of ANC has proven to be an intelligent system for detecting malicious behavior and acting on it dynamically. Athena can take information from several different sources, correlate that information to provide a threat assessment of a potential malicious user, and issue commands to the countermeasure system to act on that threat. The code was written modularly so that additional information sources can be added easily to the existing set of sources. Athena's "brain" can be easily reprogrammed by modifying or rewriting the `correlation()` function.

7 Acknowledgments

The authors acknowledge the effort of Steve Hurd, Jim Hutchins, and Tim Toole in writing the funding proposal for this project. We also thank Jim and Tim for their insights into intrusion detection as performed by `mon` at Sandia National Laboratories - California. Jim's contributions to the realm of network countermeasures have been invaluable.

We also thank Sandia CCD (Center for Cyber Defenders) interns Derek Cotton, Chris Kolina, and Yuqing Mai for their assistance in adding ICMP response capabilities to `Honeyd`.

References

[1] Symantec Corporation. (2003, August) W32.blaster.worm. [Online]. Available: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.wor%m.html

[2] Renaud Deraison. (2003) Nessus. [Online]. Available: http://www.nessus.org/

[3] Fyodor. (2003) Nmap. Insecure.org. [Online]. Available: http://www.insecure.org/nmap/

[4] Fyodor. (1998, October) Remote os detection via tcp/ip stack fingerprinting. Insecure.org. [Online]. Available: http://www.insecure.org/nmap/nmap-fingerprinting-article.html

[5] Ofir Arkin. (2003) Xprobe/xprobe2. Sys-Security Group. [Online]. Available: http://www.sys-security.com/html/projects/X.html

[6] Ofir Arkin. (2000, July) Icmp usage in scanning. Sys-Security Group. [Online]. Available: http://www.sys-security.com/html/projects/icmp.html

[7] Anil Somayaji and Stephanie Forrest, "Automated response using system-call delays," in *Proceedings of the 9th USENIX Security Symposium*. The USENIX Association, August 2000.

[8] Jamie Twycross and Matthew M. Williamson, "Implementing and testing a virus throttle," in *Proceedings of the 12th USENIX Security Symposium*. The USENIX Association, August 2003.

[9] (2003) Snort. [Online]. Available: http://www.snort.org/

[10] (2003) Snortsam. [Online]. Available: http://www.snortsam.net/

[11] Vern Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, December 14, 1999.

[12] ForeScout Technologies Inc. (2003) Activescout site. [Online]. Available: http://www.forescout.com/activescout.html

[13] Niels Provos, "Honeyd: A virtual honeypot daemon," presented at 10th DFN-CERT Workshop, February 2003.

[14] Ugen J. S. Antsilevich, Poul-Henning Kamp, Alex Nash, Archie Cobbs, and Luigi Rizzo. (2003) Freebsd hypertext man pages: ipfw. Lawrence Berkeley National Laboratory Network Research Group. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=ipfw

[15] Steven McCanne and Van Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *Proceedings of the 1993 Winter USENIX Conference*. The USENIX Association, January 1993.

[16] Lawrence Berkeley National Laboratory Network Research Group. (2003) libpcap. [Online]. Available: http://ftp.ee.lbl.gov/nrg.html

[17] (2003) Port numbers. IANA. [Online]. Available: http://www.iana.org/assignments/port-numbers

[18] Deeann M. M. Mikula, Chris Tracy, and Mike Holling, "Spam blocking with a dynamically updated firewall ruleset," in *Proceedings of LISA 2002: 16th Systems Administration Conference*. The USENIX Association, November 1992.

[19] (2003) Prevayler. [Online]. Available: http://www.prevayler.org/

Appendix A

Setting up an ANC System

1. Overall design – The ANC architecture is composed of many modular components with specific functions. In our prototype system, each of these parts is implemented by software with communication channels between each part. Some of the software is open source, off-the-shelf software, and some is internally developed software. Our working prototype mirrors very closely the logical model that we architected (Figure 1).

   Corresponding with each of the four parts in the model – IDS modules, GI modules, Athena, and HENC – are four different physical computers that, in our prototype, provide each of the functions. Our communication channels between systems and the greater network are closely analogous as well.

2. Configuring the Basic system – Each of the four computers in the prototype has the same basic underlying hardware and operating system. They are fairly high-performance server-like systems with 2.8 GHz Intel Xeon processors, 1GB of RAM, and 36 GB SCSI hard-drives. Two of the computers, for `Snort` and NetState, require one Gigabit Ethernet card and one Fast Ethernet card each. The HENC computer requires two Gigabit Ethernet cards and one Fast Ethernet card. The Athena system requires only one Fast Ethernet card. The internal communication can be set up in many different ways. We chose to have all four computers attached to the same 10/100 Ethernet switch for internal communication. We do not anticipate that traffic loads will require more bandwidth on this internal network.

   Each of the four systems is running FreeBSD 4.8-RELEASE. The default kernels are acceptable, with two exceptions: the HENC computer needs to have the IPFW and DIVERT options turned on. IPFW is used to filter out traffic that HENC is not responding to, and Honeyd uses divert sockets to receive network datagrams.

3. Configuring the `Snort` computer – `Snort` can be installed from the FreeBSD ports tree. Run `Snort` according to the supplied documentation. We use the default rule set that comes with `Snort`. `Snort` gets its incoming traffic from the GigE network.

   `Snort` needs to be run with the '-A unsock' flag so that messages are sent to a local Unix socket. Our program, snort_wrapper, listens on the other end of that Unix socket and sends Athena Alert Protocol messages to Athena over the Fast Ethernet control network.

4. Configuring the NetState computer – NetState is Sandia-developed software and must be installed from source. The computer with NetState must have MySQL installed, which can be installed from the ports tree. Follow the NetState-supplied instructions for installing and running NetState.

   NetState does have one flaw when combined with network countermeasures. The countermeasures themselves, if seen by NetState, will affect how NetState views the network. The solution is to place NetState in a location in the network architecture such that it does not receive traffic that is the result of network countermeasures.

   Additionally, the NetState computer runs the Country Check program, which has its own set of installation instructions. Athena communicates with Country Check using a TCP/IP connection over the control network.

5. Configuring the Athena computer – Athena should be compiled from source and run on the Athena computer. All of Athena's communication happens over the Fast Ethernet control network. Both PostgreSQL and MySQL need to be installed on the Athena computer; they can be installed from the ports collection. The Athena configuration files offer many variables that can be changed to tune for performance and/or desired modes of operation.

6. Configuring the HENC computer – The modified version of Honeyd needs to be compiled from source and run on the HENC computer. This computer also runs the countermeasure logger (cmlog) and the ANCP server (server.pl). Some of the configuration variables can be changed in the configuration files. Template Builder (tb.pl) can be used to build the configuration files.

7. Putting it all together – All of the services and programs should be set up to be run when the computer is booted. When the network connections are all set up properly, the HENC, NetState, and `Snort` computers should all be

receiving network traffic. The HENC computer should also be able to send countermeasure datagrams. Finally, all computers should be able to communicate on the internal control network.

Appendix B

FreeBSD IP Stack Patch

(available from http://erie.ca.sandia.gov/anc/)

This patch has been successfully tested with both FreeBSD 4.7 and 4.8. Other versions of FreeBSD should work, but care should be taken to ensure the patch is applied successfully.

You must have the FreeBSD source installed to apply this patch and be comfortable with compiling your own kernel. Please look in /usr/src/Makefile for steps to compile the kernel. Basic directions to apply the patch and recompile are as follows:

```
# patch < /path/to/stack.patch
# cd /usr/src/
# make buildkernel KERNCONF=YOUR_KERNEL_HERE (default is GENERIC)
# make installkernel KERNCONF=YOUR_KERNEL_HERE (default is GENERIC)
# reboot
```

Patch:

```
--- /usr/src/sys/netinet/raw_ip.c       Fri Feb 15 13:25:24 2002
+++ /usr/src/sys/netinet/raw_ip.c       Fri Sep  6 09:53:19 2002
@@ -239,12 +239,14 @@
                        m_freem(m);
                        return EINVAL;
                }
+/*  FreeBSD patch for proper honeyd functionality
                if (ip->ip_id == 0)
 #ifdef RANDOM_IP_ID
                        ip->ip_id = ip_randomid();
 #else
                        ip->ip_id = htons(ip_id++);
 #endif
+*/
                /* XXX prevent ip_output from overwriting header fields */
                flags |= IP_RAWOUTPUT;
                ipstat.ips_rawout++;
```

Appendix C

Information Correlation

Athena receives information for characterizing threats from multiple entities in the ANC system. This information is correlated to determine the threat level of a particular outside IP address. The threat level describes how bad or malicious the remote host is. When the threat level has crossed a given threshold, countermeasures are sent to the offending host. The types and frequency of countermeasures may change based on the system administrator's configuration of ANC.

In addition to the alert information send by an IDS module, Athena retrieves information from several built in GI (General Information) modules: CountryCheck, a NetState module, and an internal database, AthenaDB. The AthenaDB database is used to maintain statistics about the behavior of remote hosts. The querying and correlation of all of this information occurs in the `correlation()` function.

The first step in information correlation is the retrieval of information. The `correlation()` function receives the IDS alert from the MessageHandler thread as a parameter. This message contains the source and destination IP addresses and ports, the time of receipt of the alert (in microseconds), the threat type, the threat level, and the protocol of the datagram(s) that triggered the alert. Some of this information is used to retrieve information from the NetState database. In particular, Athena retrieves the time of the most recent connection to a local IP/port pair, the time of the first connection to a local IP/port pair, and whether the local destination port matches the recorded service according to the IANA registry of well-known ports.

Next, the local AthenaDB is queried for specific recent (within the last 60 seconds) statistics. These statistics include the number of IDS alerts, the number of IDS alerts of the same threat type, and the number of IDS alerts whose source IP address is within the same 24-bit subnet as the IP address that generated the IDS alert in question.

Finally, Athena retrieves statistics related to the specific remote IP address. Athena retrieves the current cumulative level for that IP address, the last time a threat was received for that IP, and the last type of threat received from that IP. The Country Code level is also retrieved.

Six-Step Correlation

The computed threat level always starts off at the level retrieved from AthenaDB or zero if the IP address has never caused an alert before. This computed threat level is then modified by the results of different correlations, with each correlation multiplying their contribution by a user configurable weight value.

The first modifier is the threat level of the IDS alert message itself. This number is weighted and then added to the computed threat level.

Next, the country code modifier, a number between zero and three, is weighted and added to the computed threat level.

Next, Athena checks the port-to-service match modifier by querying NetState. If the port does not match the service running on that port according to the list of well-known ports, this is considered a bad situation. It may mean that the destination host is compromised and running a backdoor (Trojan Horse) program that allows remote access. If that port was opened recently (within the last 60 seconds), the situation is worsened because it could mean that the host is actively being compromised. Numerical values associated with these conditions are weighted and added to the computed threat level. This correlation is limited by the fact that NetState is only able to recognize a limited set of services.

The fourth modifier is related to detecting malicious intent. A remote host may accidentally trigger an alert once, but alerts become increasingly less likely to be accidental if the host continues to trigger the same alert. So, if the current threat type is the same as the last threat type from the specified IP address, the computed threat level is updated to reflect this possibly bad behavior.

The next modifier is similar to the backdoor detection modifier above, except that there is no correlation to whether the service matches the port. Though not necessarily a sign of malicious intent by itself, this information may signify bad behavior when combined with other alerts generated for that remote source IP address.

The final correlation determines whether the alert has been generated during normal business hours. Some attackers may opt to perform malicious activities at night because they believe nobody will be watching their actions. This modifier penalizes this traffic based on that line of thought.

When all of the information is correlated, the new cumulative threat level is set to the computed threat level for the remote IP address. This new

level may trigger changes in response. After any changes have been initiated via ANCP, the database is updated to reflect the new threat level.

Distribution:

1 Chris Tracy
1001 Milton Street #1
Pittsburgh, PA 15218

1 MS 0455 R. S. Tamashiro
1 MS 0630 D. H. Schroeder
1 MS 0801 A. L. Hale
1 MS 0801 W. F. Mason
1 MS 0801 M. R. Sjulin
1 MS 0806 J. A. Hudson
1 MS 0806 P. C. Jones
1 MS 0806 M. M. Miller
1 MS 0806 L. Stans
1 MS 0812 R. L. Adams
1 MS 0813 R. M. Cahoon
1 MS 0813 S. R. Carpenter
1 MS 0813 D. M. Kayatt Jr.
1 MS 0813 A. A. Quintana
1 MS 0813 R. A. Suppona
1 MS 9003 J. L. Handrock
1 MS 9003 C. M. Hartwig
1 MS 9003 K. E. Washington
1 MS 9011 N. A. Durgin
1 MS 9011 B. V. Hess
1 MS 9011 J. D. Howard
1 MS 9011 S. A. Hurd
1 MS 9011 J. A. Hutchins
1 MS 9011 E. D. Thomas
1 MS 9011 T. J. Toole
5 MS 9011 J. A. Van Randwyk
1 MS 9012 J. A. Friesen
1 MS 9012 S. C. Gray
1 MS 9012 P. E. Nielan
1 MS 9019 S. C. Carpenter
1 MS 9019 B. A. Maxwell
1 MS 9037 J. C. Berry
1 MS 9217 S. W. Thomas
1 MS 9914 C. D. Ulmer
1 MS 9915 N. M. Berry
1 MS 9915 H. Y. Chen
1 MS 9915 M. L. Koszykowski

3 MS 9018 Central Technical Files, 8945-1
1 MS 0899 Technical Library, 9616
1 MS 9021 Classification Office, 8511/Technical Library, MS 0899, 9616
DOE/OSTI via URL
1 MS 0323 D. Chavez, LDRD Office, 1011

This page intentionally left blank