MTR 01W0000103

MITRE TECHNICAL REPORT

Engineering Issues for an Adaptive Defense Network

June, 2001

Alan Piszcz
Nicholas Orlans
Zachary Eyler-Walker
David Moore

Contract No.: DAAB07-00-C-C201

Approved for public release; distribution unlimited.

MITRE
Washington C3 Center
McLean, Virginia

MITRE Department
Approval:

Eileen M. Boettcher
Department Manager, W033
Distributed Systems and
Technologies

Acknowledgments

This work was sponsored by the MITRE Technology Program.

Table of Contents

Executive Summary

Engineering Issues for an Adaptive Defense Network (ADN) examines the ability of network systems to change behavior dynamically to sustain service in response to attacks. In order to focus the research problem, Distributed Denial of Service (DDoS) attacks were used as the threat. The primary issue was the capability to detect and defend against DDoS. Experimentation was performed with a packet filtering firewall, a network Quality of Service manager, multiple DDoS tools, and traffic generation tools. Related efforts, recommendations and experiments are covered in this paper.

Adapting to network events in degraded environments is a challenge for applications, services and systems where conditions are known. As network conditions change due to cyber attacks carried out by email viruses, application viruses, and denial of service attacks, there is typically instantaneous network confusion. Network operator reaction and control of these events can take hours to days for determination and resolution. This effort examines a severe threat, Distributed Denial of Service (DDoS) and potential techniques for an adaptive, automatic defense which would take place in seconds and represent the first level of defense until network operations or system administrator personnel can respond. The asymmetric nature of the DDoS threat allows an *individual* with minimal resources to disrupt or deny network service to critical information infrastructures.

Adaptive defense of networks requires automated response to current and future threats. This effort utilized DDoS threats to motivate adaptive defense behavior and experimentation. The focus of the following recommendations will address denial-of-service (DoS) issues related to the network node.

In order to provide guidance with respect to DdoS, a number of recommendations have been developed by information security organizations. Note that the following recommendations protect the packet producers versus the victim, however, they are applicable to all sites and should be implemented.

- Egress filtering: do not allow packets with invalid source addresses to exit your network. Deny invalid source addresses that include private, reserved address ranges and any address not defined in your organization's network.
- Disable directed broadcast on all systems.
- Employ Unicast Reverse Path Forwarding (RPF). The following is quoted from "Unicast Reverse Path Forwarding" white paper [CISCORPF2001]. The Unicast RPF feature helps to mitigate problems that are caused by the introduction of malformed or forged (spoofed) IP source addresses into a network by discarding IP packets that lack a verifiable IP source address. For example, a number of common types of denial-of-service (DoS) attacks, including Smurf and Tribe Flood Network (TFN), can take advantage of forged or rapidly changing source IP addresses to allow attackers to thwart efforts to locate or filter the attacks. For Internet service providers (ISPs) that provide public access, Unicast RPF deflects such attacks by forwarding only packets that have source addresses that are valid and consistent with the IP routing table. This action protects the network of the ISP, its customer, and the rest of the Internet."
- Service Level Agreements to reduce payment in cases of DDoS traffic loss.
- Shared ISP alerting among firewalls.
- Layer 2 analysis, using a transparent bridge on the network, monitor layer 2 traffic parameters. Determine if this approach would simplify the solution and provide adequate detection.
- Develop attack trees for DDoS threat type and relate costs to adaptive defense techniques.
- Perform QoS reservations and attacks using CISCO router and observe network behavior.
- Evaluate DARPA funded and other emerging adaptive firewalls in testbed.
- Develop custom Alert Event rules for ntop.
- Produce a stable and complete ntop data extraction. Probably through an enhanced "dumpdata" or a direct GDBM database read.
- Develop cross system correlation logic using data extracted from multiple instances of ntop.

Engineering Issues for an Adaptive Defense Network
June 2001

Alan Piszcz, Nicholas Orlans, Zachary Eyler-Walker, David Moore
The MITRE Corporation
McLean, VA 22102

e-mail: { apiszcz | norlans| zach | davem }@mitre.org

ABSTRACT

Engineering Issues for an Adaptive Defense Network (ADN) examines the ability of network systems to change behavior dynamically to sustain service in response to attacks. In order to focus the research problem, Distributed Denial of Service (DDoS) attacks were used as the threat. The primary issue was the capability to detect and defend against DDoS. Experimentation was performed with a packet filtering firewall, a network Quality of Service manager, multiple DDoS tools, and traffic generation tools. Related efforts, recommendations and experiments are covered in this paper.

1 INTRODUCTION

Adapting to network events in degraded environments is a challenge for applications, services and systems where conditions are known. As network conditions change due to cyber attacks carried out by email viruses, application viruses, and denial of service attacks, there is typically instantaneous network confusion. Network operator reaction to and control of these events can take hours to days for determination and resolution. This effort examines a severe threat, Distributed Denial of Service (DDoS), and potential techniques for an adaptive, automatic defense which would take place in seconds and represent the first level of defense until network operations or system administrator personnel can respond. The asymmetric nature of the DDoS threat allows an *individual* with minimal resources to disrupt or deny network service to critical information infrastructures.

This effort does not expect to maintain full service capability, but instead a degraded subset of services until the problem can be resolved. Information assurance includes: policy, personnel, secure systems, remote access, and service providers. Achieving network defense in the future will involve: coordination between service providers, intrusion detection, auditing, automated organizational trust policies, firewall and quality of service tools. Fusion of these elements is the basis for adaptive behavior as dictated by the global or organization policy. This effort examined a range of capabilities to address defense for network targets against a Distributed Denial of Service attack.

1.1 ELEMENTS OF THE EXPERIMENTS

This problem encompasses a wide range of devices, software systems, and networking technology. The ADN project developed a testbed for experimentation and exploration of the hypothesis for adaptive defense. The ADN testbed consisted of two wide area network routers, several host organizations, a threat platform and measurement capabilities.

1.2 TESTBED ENVIRONMENT

ADN performed experiments with techniques that reacted automatically to network attacks. The network attacks under consideration were Distributed Denial of Service (DDoS). The ADN testbed consisted of a suite of SUN workstations and Intel machines running Linux representing organizations as multi-homed systems with stateful packet filtering firewalls. An Internet Service Provider (ISP) was simulated using a router and Solaris Bandwidth Manager 1.6 (SBM) to emulate router functions with respect to packet prioritization, channel capacity reservation, and channel bandwidth. Our preliminary experiment operated SBM on an Ultra10 300MHZ CPU with a quad 10/100 MBPS network interface card. This system acted as a five-port router (using the additional built-in interface) which routes for five organizations represented by SparcStation 5 or SparcStation 20 platforms. Organizations one through five represented the user community of the network. In the experiment, organization one was the victim, with control information being directed from organization five. Organizations two, three, and four were the DoS agent/zombie/client platforms which transmit packets to disable or degrade the victim or organization one. Figure

1.2-1 shows the configuration used for the initial experiments and tests. Appendix B provides a network mapping of services operating on each testbed platform.

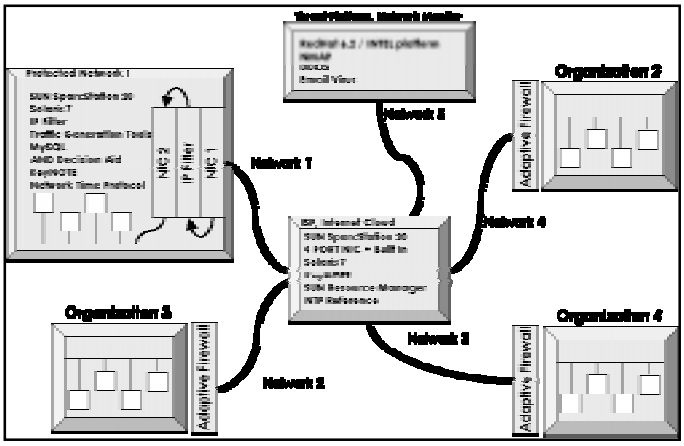

Figure 1.2-1 Testbed Environment

1.3 NOTIONAL COOPERATING FIREWALL

Figure 1.3-1 depicts the prerequisite ADN components for a cooperating firewall. The identified subsystems represent capabilities necessary to enable a wide area network defense.

Secure Communication Channel Proxy
This mechanism should provide a secure channel among cooperating firewalls to communicate policy information, firewall rules and control information. The channel requires network bandwidth reservation to guarantee availability during attacks.

Packet Filtering Firewall
A packet filtering firewall was used to demonstrate policy enforcement capability in this topology. Other techniques for network control include modification of kernel parameters, access control lists, router parameters, and platform operating system network services.

Decision Aid
Monitors effects of attacks and provides policy for courses of reaction. A few techniques for this area are explored later in this paper.

Network Time Reference
The Network Time Protocol (NTP) provides the capability to synchronize system clocks for time maintenance. In order to review system logs and events on different systems over a wide area network it is essential that time services be active. Greenwich Mean Time or other standard common reference time zone should be used.

Reactive Firewall Rule Database
Contains the required rule set groups in order to respond to known network threats, service restrictions and organization policies. Ideally these rules are self-contained and described by a unique identifier. Documentation and comments should also be in the rule set.

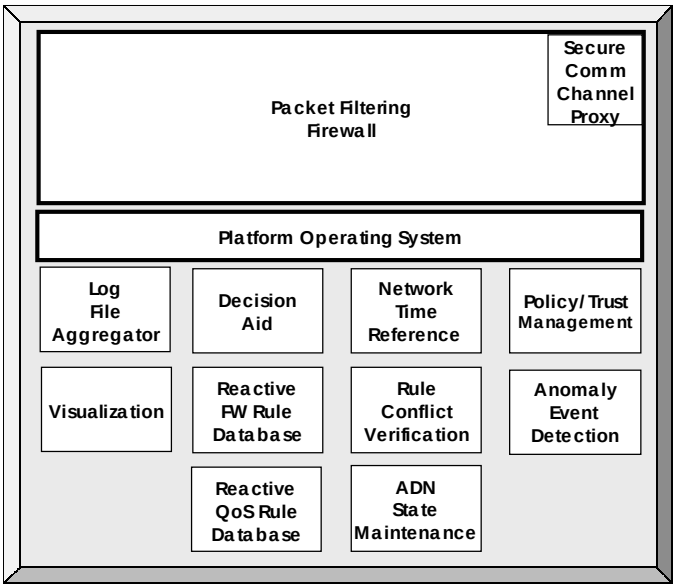

Figure 1.3-1 NOTIONAL COOPERATING FIREWALL

Rule Conflict Verification
This is an open research area being explored with techniques such as binary decision diagrams. Other techniques may emerge as this problem increases with the complexity of incompatible rule sets across vendor product lines.

Reactive Quality of Service (QoS) Rule Database
Service level agreements (SLA) and expected service availability information needs to be defined and stored by a cooperating firewall. In particular many attacks may require service providers to rate-limit traffic, disable services and otherwise override SLAs. The rule database should include approved responses for routers and QoS providers; additionally it would allow definition of SLA features for different threat combinations.

ADN State Maintenance
Historical and current network profiles of the cooperating nodes need to be captured. This information would be used by the decision aid to develop policy recommendations.

Policy/Trust Management
Hierarchical authorities are required to communicate and enforce network policies. Trust management is integral to

executing network policy changes. The Common Open Policy Server (COPS) addresses this area and is described later.

Anomaly Event Detection
Attacks such as DDoS should be detectable as anomalies. Understanding and classifying the type or severity of anomalies is required. For example, anomalies from a local hardware malfunction should not be construed as an attack.

Visualization
Currently management of a single firewall with hundreds or thousands of rules is a task which one can only grasp after spending endless hours with the domain, policy requirements and the service providers involved. Visualization applied to rule sets is an area in need of additional research and development.

2 DISTRIBUTED DENIAL OF SERVICE

DDoS attacks represent multi-network, multi-platform and multi-target network threats. The Computer Emergency Response Team Coordination Center (CERT/CC) held a workshop on DDoS tools during November 1999 to discuss DDoS and approaches for defense. The workshop output was a white paper titled, "Results of the Distributed-Systems Intruder Tools Workshop" [CERT/CC1999].

High profile attacks gained national recognition by media organizations during 2000. Figure 2-1 illustrates a few of the DDoS events that were publicized and the duration of each.

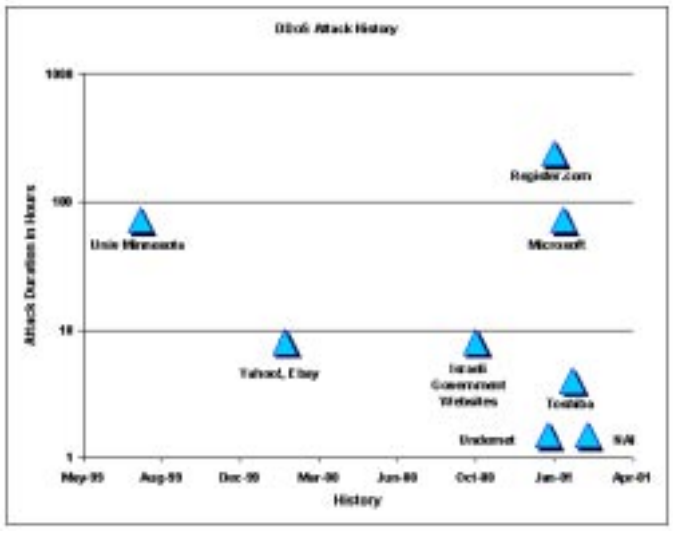

Figure 2-1 Publicized DDoS Attacks

The electronic business attack of February 2000 placed the DDoS threat as a top priority to the Alliance for Internet Security and other security aware partners. The October 2000 issue of Information Security Magazine [INFOSECMAG2000] contained a prominent announcement on the threat and urged organizations to develop counter measures. The YANKEE GROUP, a financial and strategic consultant, estimated the February 2000 DDoS attack caused a 1.2 billion dollar loss of revenues for the organizations involved.

During the month of October 2000 Israeli government websites were corrupted and disabled through denial of service techniques. News organizations labeled this the "Middle East cyber war."

Coincidentally, during that same week in October 2000 the Defense Advanced Research Projects Agency (DARPA) announced a 6 million dollar research effort for information security. One of the project goals is research of an autonomic distributed firewall, capable of reacting to network based attacks.

Figure 2-2 "Putting a Target Under Siege" [COMERFORD2001] illustrates a common use of the internet architecture and its relation to DDoS. In a distributed attack, an attacker first breaks into net-attached computers, quietly setting one up as a handler, or a controller as drones. Later, they bombard the target with traffic without the attacker's participation.

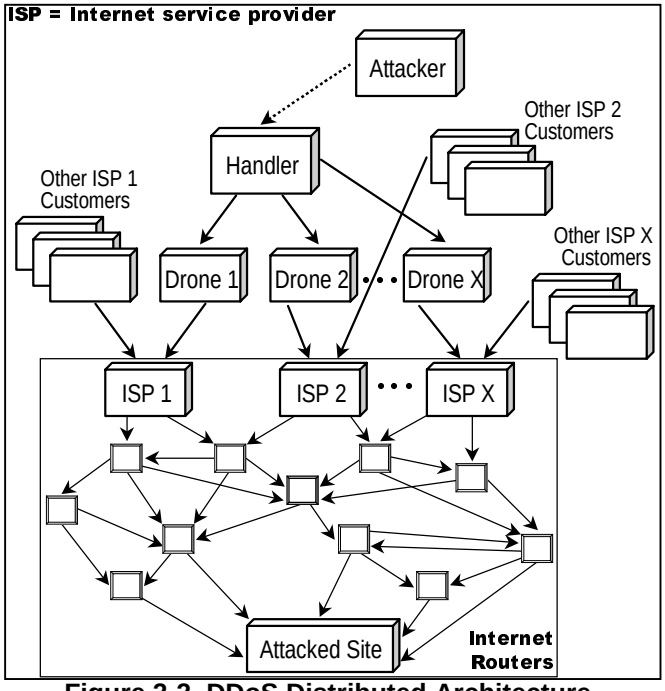

Figure 2-2 DDoS Distributed Architecture

There are numerous analyses of DDoS tools available on the internet, along with the tools themselves in some instances. Dave Dittrich [DITTRICH2000] is a security engineer at the University of Washington who has investigated a number of the DDoS tools. A number of attack tools exist, and more are in development. This effort was undertaken to understand the fundamental principles of DDoS and started with "trinoo," one of the first DDoS tools with a master and daemon or client/server capability. The following are some of the known tools which are available: fapi, blitznet, Tribe Flood Network (TFN), mstream, stacheldraht, Trinity, shaft, TFN2K, TRANK. In a few cases the authors of this paper have augmented the descriptions of the DDoS tools contained later in this paper.

3 RECOMMENDATIONS

Adaptive defense of networks requires automated response to current and future threats. This effort utilized DDoS threats to motivate adaptive defense behavior and experimentation. Figure 3-1 illustrates an organization for denial of service attacks. The focus of the following recommendations will address DoS issues related to the network node.

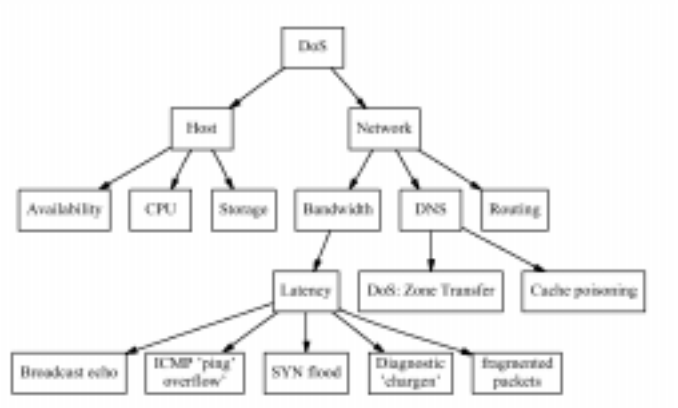

Figure 3-1 DoS Threat Types

3.1 MITIGATING ATTACK SOURCE CAPABILITIES

In order to provide guidance with respect to DdoS, a number of recommendations have been developed by information security organizations. Note that the following recommendations protect the packet producers versus the victim, however, they are applicable to all sites and should be implemented.

- Egress filtering: do not allow packets with invalid source addresses to exit your network. Deny invalid source addresses that include, private, reserved address ranges and any address not defined in your organization's network.

- Disable directed broadcast on all systems.

- Employ Unicast Reverse Path Forwarding (RPF). The following is quoted from "Unicast Reverse Path Forwarding" white paper [CISCORPF2001]. The Unicast RPF feature helps to mitigate problems that are caused by the introduction of malformed or forged (spoofed) IP source addresses into a network by discarding IP packets that lack a verifiable IP source address. For example, a number of common types of denial-of-service (DoS) attacks, including Smurf and Tribe Flood Network (TFN), can take advantage of forged or rapidly changing source IP addresses to allow attackers to thwart efforts to locate or filter the attacks. For Internet service providers (ISPs) that provide public access, Unicast RPF deflects such attacks by forwarding only packets that have source addresses that are valid and consistent with the IP routing table. This action protects the network of the ISP, its customer, and the rest of the Internet."

- Service Level Agreements to reduce payment in cases of DDoS traffic loss.

- Shared ISP alerting among firewalls.

3.2 NEXT STEPS IN ADN RESEARCH

- Layer 2 analysis, using a transparent bridge on the network, monitor layer 2 traffic parameters. Determine if this approach would simply the solution and provide adequate detection.

- Develop attack trees for DDoS threat type and relate costs to adaptive defense techniques.

- Perform QoS reservations and attacks using CISCO router and observe network behavior.

- Evaluate DARPA funded and other emerging adaptive firewalls in testbed.

- Develop custom Alert Event rules for ntop.

- Produce a stable and complete ntop data extraction. Probably through an enhanced "dumpdata" or a direct GDBM database read.

- Develop cross system correlation logic using data extracted from multiple instances of ntop.

4 TECHNIQUES AND TOOLS

The following sections contain a collection of tools, detection techniques and observations related to adaptive network defense. Organization of this material flows from infrastructure systems, tools, to techniques and experiments.

4.1 POLICY ARCHITECTURE AND MANAGEMENT

Common Open Policy Service (COPS) was analyzed for enabling policy coordination among distributed firewalls in an adaptive defense network. COPS is an emerging standard track defined by RFC 2748 and the related RFC 2753 (Framework for Policy-based Admission Control). The proposed messaging standard and associated terminology results from an industry consortium that includes Intel, Cisco, and AT&T. COPS defines message types and outlines an application interface and typical application sequences for Quality of service (QoS) programming and other policy based network applications.

4.1.1 COPS TERMINOLOGY

COPS terminology defines two primary functional entities, a Policy Decision Points (PDP) and a Policy Enforcement Point (PEP). PDPs are typically servers and PEPs are typically cooperating routers or firewalls.

There is no single reference enterprise level architecture, rather the COPS reference framework for policy-based admission control provides definitions of how PDPs and PEPs interact. Details of how system state is stored, how authentication occurs, and what additional mechanisms may be necessary for out-of-band communications is left to the application. Building, deploying, and configuring such a framework requires sound engineering judgment based on comprehensive and detailed information of the enterprise network topology, applications and traffic, and desired network policies.

COPS describes two primary modes of coordinated dynamic network responses, provisioning and outsourcing.

4.1.2 COPS PROVISIONING (PDP INITIATED)

COPS provisioning, as shown below in figure 4.1.2-1, is initiated by the PDP requesting a configuration change to one or more PEP. Typically this results from scheduled QoS provisioning, but could also be dynamically generated requests. The PEPs must have earlier issued a Request State message that establishes a passive wait condition for the PDP Decision message. The Decision message for each PEP includes client handle information context and configuration information for that particular client (device type). The PEP responds with a Report State message, indicating success or failure in carrying out each Decision message.

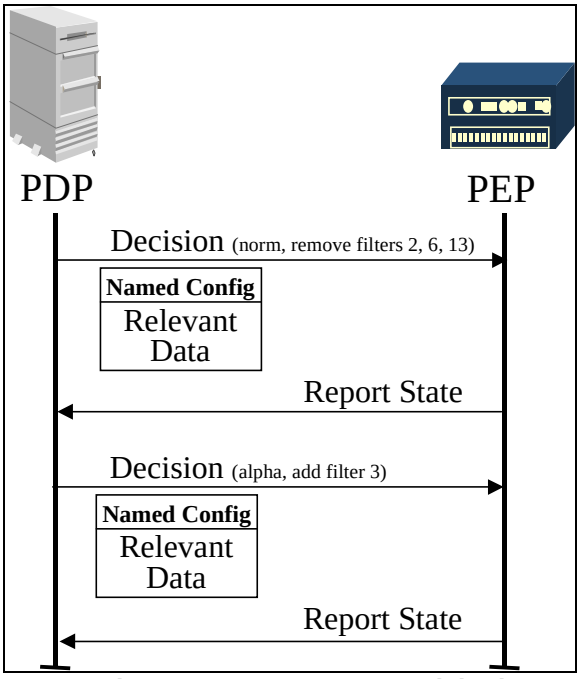

Figure 4.1.2-1. COPS Provisioning

4.1.3 COPS OUTSOURCING (PEP INITIATED)

An example of COPS outsourcing is shown below in figure 4.1.3-1. Outsourcing is initiated by the PEP and occurs whenever the PEP is unable to make a local decision. Outsourcing can also result from pre-programmed decision hierarchies. The PEP sends a Request message containing identification information and client specification information (e.g. router manufacture, model number, operating system, supported interfaces, and revision). The PDP responds with a Decision message, including the client handle and configuration context appropriate for that client. The exchange is concluded by the PEP returning a Report State message, reporting on the success or failure of the decision.

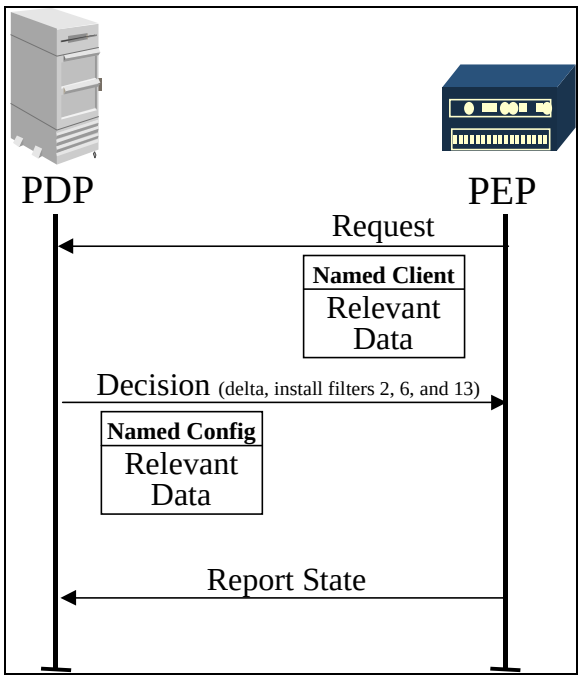

Figure 4.1.3-1. COPS Outsourcing

4.2 TRUST MANAGEMENT WITH KEYNOTE

Trust Management is an essential infrastructure component for adaptive firewall solutions. All network-critical message exchanges, such as those outlined within the COPS framework, depend on clearly defined trust and authorization relationships. Trust authorization and management tend to mimic human organizational hierarchies in terms of delegation and authority, but more importantly they must be make sense within a target network topology that includes boundary points and critical routes for egress and ingress.

Keynote is a trust management engine reviewed by the ADN team. Keynote provides a C-like assertion language designed to express and evaluate trust conditions. The Keynote system was produced by AT&T Research Laboratories in conjunction with the University of Pennsylvania [BLAZE2000]. It is available opensource and defined by RFC 2407 [KEYNNOTE]

The logical predicates used by Keynote for policy compliance testing are constructed from a flexible, yet rather abstract, collection of tag/value pairs. Principal names (hosts, PDPs, or PEPs) can be any nmemonic string and can also directly represent cryptographic keys. An important property of Keynote is that of 'assertion monotonocity', meaning compliance values are always derived from trusted sources and incomplete or missing assertions cannot result in spurious authorizations (higher compliance values). The Keynote abstraction environment requires a coherent nomenclature for "principals", "licensee", "conditions", and "actions". The level of granularity and semantics used to describe situational awareness (e.g. authority, trust conditions, current traffic states, candidate actions, and so on) must be defined by the application. Actions and transport mechanisms also must be provided by the application. A simple example root policy is shown in listin 4.2-1. The policy states that an ADN decision point is authorized to approve/accept *somthing* (where the something is an application action) from org1, org2, org3, and org4 if the state variable 'infocon' is set to "delta".

```
Keynote version: 2
Authorizer: "POLICY"
# This is the root policy
Licensees: "org1"||"org2"||"org3"||"org4"
Conditions: (appDomain == 'adn') &&
            (infocon == "delta")
            -> "Approve";
```

Listing 4.2-1 simple Keynote assertion

4.3 SOLARIS BANDWIDTH MANAGER

Solaris Bandwidth Manager (SBM) was experimented with as the Quality of Service router. In the ADN context it was used to evaluate channel reservation for the secure channel, and QoS for organizations that were routed through it. Experiment #2 later in this describes the experience with this capability. SUN Microsystems provides the following product overview.

Solaris[tm] Bandwidth Manager is a software solution regulating bandwidth usage in LANs and WANs. It provides high-quality network service by controlling the bandwidth assigned to applications and users, prioritizing traffic, and building advanced bandwidth management policies.

Features include:
Controls incoming as well as outgoing traffic.
Manages any type of IP traffic: classification can be done on source or destination addresses, application type, URL address (with wildcards), and IP Type of Service (ToS).

Interoperates with routers (ToS field marking, NetFlow data exporting). Provides detailed flow and class based accounting information, through ASCII output, NetFlow protocol, and Java and C APIs for interfacing with billing applications. Java[tm]-based GUI for remote monitoring and configuration. Customizable: Java configuration APIs

and dynamically configurable Policy Agent using Java Dynamic Management Kit[tm].

5 THE DDOS THREAT

DDoS network attack applications are emerging from university and individual research efforts. These tools coordinate various network related abuse as described in Appendix A to deny service to the target(s). Attack applications are maturing in sophistication and the expertise required to launch an attack is decreasing.

The two tools selected for DDoS, trinoo and TFN2K, represent different levels of capability and threat potential for ADN attack testing. TFN2K was selected to represent an advanced DDoS tool. "trinoo" represents one of the most simple DDoS tools, except perhaps for broadcast pings.

5.1 TRINOO

In operating and reviewing trinoo, additional details were discovered that deserve description beyond what was available to the authors. The overall description of trinoo from Dittrich is good, but not complete enough to actually install and operate the software. Nevertheless, the CERT and Dittrich descriptions of trinoo are required reading for those interested in trinoo. The Dittrich Trinoo analysis is at "http://staff.washington.edu/dittrich/misc/trinoo.analysis". To learn more, a internet search will produce other analysis papers, most of which are similar in content.

For this project the plan was to operate a few DDoS capabilities on an isolated (network) testbed. The testbed consists of two router machines and a number of organizations with dual home machines operating a firewall. In the first series of experiments the plan is to have "n" organizations attack one. Two types of components are to required launch a trinoo attack. "ns" what Dittrich refers to as the daemon, which can also be thought of as a zombie or agent. It will be referred to as the *client* in this description. This module operates in conjunction with a "master" process to carry out DoS attacks on a target. In order to use this application it is necessary to edit the source code and update the static array initialization assignment with the IP addresses of the configuration's master. The version of ns.c used in this project is 184 lines (including comments) of C. "master" is the central control process, or as the authors prefer, the "server" of the trinoo configuration. The server

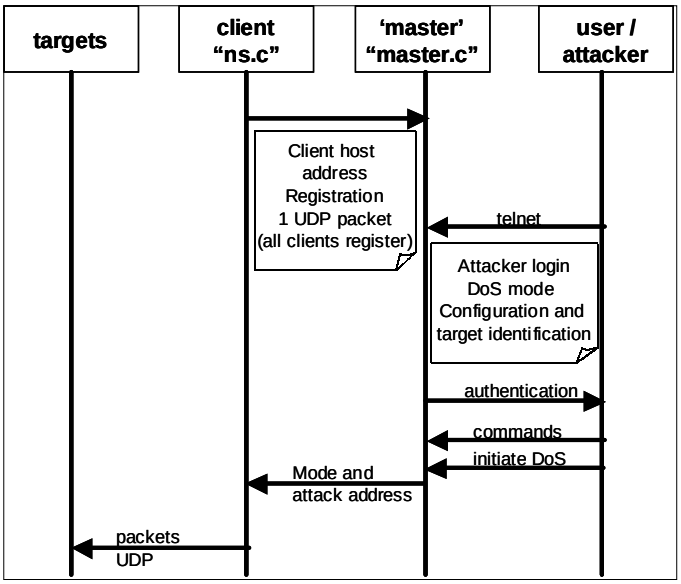

Figure 5.1-1. trinoo message sequence

accepts telnet connections to a reserved port and authenticates the user via a password. It then provides the user with a command prompt for communication with the clients. The file master.c contains 450 lines of C with comments. 'master' performs user authentication and command processing, maintains a list of available clients and communicates instructions to clients. Figure 5.1-1 illustrates the control message sequence. Commands are documented in the Dittrich document and are also available by entering "help." The trinoo master process provides timers, attack modes, client health check, client registry check, client kill command, and the ability to quit.

Note each entity; targets, user, master and clients, will typically be operating on different networks/hosts. In operational attack networks there are typically thousands of clients. The assumption is the client machines have been compromised by other exploits allowing the master and client to operate on unauthorized platforms.

Locating trinoo server on a host can be done by searching for its well-known, hard-coded network service ports. Ports can be modified to obscure detection. In cases where the default ports are not modified identification is possible by detection tools.

Listing 5.1-1 documents the transaction as a result of the trinoo mping command. "mping" uses the cached information the master has collected to verify the state of the clients. "tcpdump" produced the output that shows the ping request from the source (org5) host to the destination (org2) host. Org2 responds with a packet

indicating to the master that the client is active. Org2 is listening on port 27444 for UDP packets and receives an 11-byte payload. Org2 responds with a 4-byte payload back to the mping requestor. Listings 5.1-2 and 5.1-3 show the result of a UDP nmap scan using a port range. The first case shown is with the port range limited to 27400-27600. The second is a scan of all ports. “nmap” detects and reports the trinoo client as a ‘trinoo_Bcast’ service.

```
COMMAND: mping trace analysis
  VIEW: tcpdump output

09:03:32.081329 org5.org.1025 > org2.org.27444:
udp 11

09:03:32.098006 org2.org.34714 > org5.org.31335:
udp 4 (DF)
```

Listing 5.1-1 trinoo mping message exchange

```
[root@redhat1 bin]# ./nmap -v -v -p'27400-27600' -
sU 40.0.0.1

Starting nmap V. 2.54BETA7 ( www.insecure.org/nmap/
)
Host org4.org (40.0.0.1) appears to be up ... good.
Initiating UDP Scan against org4.org (40.0.0.1)
Too many drops ... increasing senddelay to 50000
Too many drops ... increasing senddelay to 100000
Too many drops ... increasing senddelay to 200000
The UDP Scan took 297 seconds to scan 201 ports.
Interesting ports on org4.org (40.0.0.1):
(The 200 ports scanned but not shown below are in
state: closed)
Port    State    Service
27444/udp open    Trinoo_Bcast

Nmap run completed -- 1 IP address (1 host up)
scanned in 297 seconds
```

Listing 5.1-2 trinoo restricted UDP nmap scan

```
[root@redhat1 bin]# ./nmap -sU 20.0.0.1

Starting nmap V. 2.54BETA7 ( www.insecure.org/nmap/
)
Interesting ports on org2.org (20.0.0.1):
(The 1422 ports scanned but not shown below are in
state: closed)
Port    State    Service
7/udp   open    echo
9/udp   open    discard
13/udp   open    daytime
19/udp   open    chargen
37/udp   open    time
42/udp   open    nameserver
111/udp  open    sunrpc
123/udp  open    ntp
161/udp  open    snmp
177/udp  open    xdmcp
512/udp  open    biff
514/udp  open    syslog
517/udp  open    talk
520/udp  open    route
928/udp  open    unknown
970/udp  open    unknown
4045/udp  open    lockd
27444/udp open    Trinoo_Bcast
32771/udp open    sometimes-rpc6
32772/udp open    sometimes-rpc8
32773/udp open    sometimes-rpc10
32774/udp open    sometimes-rpc12
32775/udp open    sometimes-rpc14
32776/udp open    sometimes-rpc16
32777/udp open    sometimes-rpc18
32778/udp open    sometimes-rpc20
32779/udp open    sometimes-rpc22
32787/udp open    sometimes-rpc28

Nmap run completed -- 1 IP address (1 host up)
scanned in 1607 seconds
```

Listing 5.1-3 trinoo full UDP nmap scan

5.2 TFN2K

The following information is from the TFN2K README.

Using distributed client/server functionality, stealth and encryption techniques and a variety of functions, TFN can be used to control any number of remote machines to generate on-demand, anonymous Denial Of Service attacks and remote shell access. The new and improved features in this version include:

Functionality additions:
* Remote one-way command execution for distributed execution control
* Mix attack aimed at weak routers
*Targa3 attack aimed at systems with IP stack vulnerabilities
* Compatibility to many UNIX systems and Windows NT

Anonymous stealth client/server communication using:
* spoofed source addresses
* strong advanced encryption
* one-way communication protocol
* messaging via random IP protocol
* decoy packets

Technology description

TFN consists of a client and an unlimited number of servers that are each installed on different hosts. Each one of these servers is utilized to commence floods with spoofed source IPs. Communication between client and server is realized using a randomly chosen protocol, TCP, UDP or ICMP, with internal values optimized so that no recognizable pattern can be found in client/server communication and that the packets easily pass through most filtering mechanisms. The actual Tribe Protocol (tm) is contained in the packet payload. It is CAST-256 encrypted and base64 encoded, and is decoded by the TFN servers in first place. The payload then consists of

the header, which is the command ID surrounded by two equal characters, and followed by the target or option string. The clients source IP address is generally spoofed, but a custom IP may be used for purposes like evasion of RFC2267 ingress/egress filtering, as well as a custom protocol. Additionally, any amount of decoy packets can optionally be sent out with every real packet, in order to obscure the real servers locations, thereby completely obscuring the client/server communication.

6 DETECTION TOOLS

DDoS detection does not comprehensively attempt to address the initial system intrusions and set-up stages preceding actual DDoS attacks. It is assumed that there is an ample supply of already compromised hosts (zombies) and that the supply pool of vulnerable hosts is in fact likely to increase. For the purposes of this investigation, DDoS detection considers the main traffic stream of the attack itself, and, to the extent possible, the control traffic that instigates a given attack.

As is the case with network monitoring and intrusion detection tools, DDoS detection involves the capture and classification of current network traffic based on “normal” profiles or expected operational boundaries.

Several experiments were undertaken to determine the usefulness of various techniques and tools. Remote Intrusion Detection (RID), a public domain DoS detection tool was also investigated. “netstat”, a Unix utility for displaying network data, was used to feed a covariance classifier. A technique used for profiling ports on single hosts on a per connections basis [GOTO1999] was modified to include network features aggregated at the interface level and extended to include UDP and ICMP protocol traffic features common to DDoS attacks.

6.1 REMOTE INTRUSION DETECTOR

The Remote Intrusion Detector (RID) [RID2000] tool is a configurable packet snooper and generator. It works by sending out packets defined in the rid.conf file, then listening for appropriate replies. The ADN effort reviewed this in effort to understand its maturity and ability in detecting DDoS framework components. Version 1.11 is a freely distributed tool that comes with a configuration file that is setup to detect trinoo, rootshell, stacheldraht and TFN. It was built for LINUX and Solaris 7 in the ADN environment and tested against systems operating the trinoo handler/master and agent/client. It was not successful in identifying the trinoo master and did locate the agent. The configuration file defines a send/reply sequence for a particular DDoS tool signature. The detection description syntax and example are provided in listing 6.1-1.

```
 begin
   send
   recv nmatch =
  end

  PROTOCOL=: TCP | UDP | ICMP
  OPTION =: ICMP_OPTIONS | UDP_OPTIONS |
TCP_OPTIONS
  ICMP_OPTIONS =: seq= | id= | type=
   | code= | data=""
  UDP_OPTIONS =: sport= | dport = | data=""
   | code= | data="string"
   *TCP_OPTIONS=: NOT IMPLEMENTED
```

Listing 6.1-1 RID detection syntax description

Using RID involves defining detection signature descriptions, starting the tool with configuration options and specifying the target host or host range. Listing 6.1.-2 provides a detection description for the trinoo Agent/Client and Handler/Master. In each case the send command outputs a payload to the daemon and specifies an expected reply. If any of the values were modified for the trinoo software this pattern would fail. Items that can be modified in the trinoo source code include port number, command names, passwords, and response tokens.

```
# agent signature
start AgentTrinoo
 send udp dport=27444 data="png l44adsl"
 recv udp data="PONG" nmatch=1
end AgentTrinoo

# master signature
start HandlerTrinoo
 send tcp dport=27665 data="betaalmostdone"
 recv tcp data="trinoo" nmatch=1
end HandlerTrinoo
```

Listing 6.1-2 RID detection description example

Sample run output is shown in Listing 6.1-3 and 6.1-4. The first case is scanning a host with the agent running and the detection is prefixed with ‘****’. In the second scan, RID is scanning a host with the Handler running. The trinoo Handler detects it was being accessed and is shown in Listing 6.1-5. Listing 6.1-5 shows NMAP detecting the Handler (last port) on the same host that RID missed. As noted earlier (listing 5.1-2) the trinoo agent can also be detected by NMAP. NMAP also has the advantage of scanning all ports where RID does not provide that capability. Listing 6.1-6 displays the trinoo alert during the RID scan.

```
./rid -v -c rid.conf 20.0.0.1
```

```
No mask given, assuming host scan (/32)
Kernel filter, protocol ALL, raw packet
socket
1 hosts responded during pingsweep.
Running icmp tests
  Sending HandlerStacheldraht probe ...
  Sending AgentTFN probe ...
  Sending AgentStacheldraht4 probe ...
  Sending AgentStacheldraht probe ...
Running udp tests
  Sending AgentShaft probe ...
  Sending WinTrinoo probe ...
  Sending AgentTrinoo probe ...
Running tcp tests
**** 20.0.0.1 infected with WinTrinoo
**** 20.0.0.1 infected with AgentTrinoo
  Sending HandlerStacheldraht4 probe ...
  Sending HandlerStacheldraht4 probe ...
  Sending HandlerTrinoo probe ...
  Sending rootshell probe ...
```

Listing 6.1-3 RID trinoo agent scan example

```
./rid -v -c rid.conf 50.0.0.1
No mask given, assuming host scan (/32)
Kernel filter, protocol ALL, raw packet
socket
1 hosts responded during pingsweep.
Running icmp tests
  Sending HandlerStacheldraht probe ...
  Sending AgentTFN probe ...
  Sending AgentStacheldraht4 probe ...
  Sending AgentStacheldraht probe ...
Running udp tests
  Sending AgentShaft probe ...
  Sending WinTrinoo probe ...
  Sending AgentTrinoo probe ...
Running tcp tests
  Sending HandlerStacheldraht4 probe ...
  Sending HandlerStacheldraht4 probe ...
  Sending HandlerTrinoo probe ...
  Sending rootshell probe ...
```

Listing 6.1-4 RID trinoo Handler scan example

```
[root@redhat1 bin]# /disk1/net*/bin/nmap -sT
50.0.0.1

Starting nmap V. 2.54BETA7 (
www.insecure.org/nmap/ )
Interesting ports on org5.org (50.0.0.1):
(The 1518 ports scanned but not shown below
are in state: closed)
Port    State    Service
21/tcp   open    ftp
23/tcp   open    telnet
25/tcp   open    smtp
79/tcp   open    finger
98/tcp   open    linuxconf
111/tcp  open    sunrpc
113/tcp  open    auth
513/tcp  open    login
514/tcp  open    shell
515/tcp  open    printer
961/tcp  open    unknown
1024/tcp  open    kdm
1026/tcp  open    nterm
1032/tcp  open    iad3
6000/tcp  open    X11
27665/tcp open    Trinoo_Master

Nmap run completed -- 1 IP address (1 host up)
scanned in 3 seconds
```

Listing 6.1-5 trinoo Handler NMAP detection

```
telnet localhost 27665
Trying 127.0.0.1...
Connected to localhost.localdomain.
Escape character is '^]'.
betaalmostdone
trinoo v1.07d2+f3+c..[rpm8d/cb4Sx/]

trinoo> Warning: Connection from
44.251.255.191
```

Listing 6.1-6 trinoo alert message

6.2 COVARIANCE EXPERIMENT USING NETSTAT

"netstat" is a Unix utility that displays various network-related data structures, state information, routing tables, and per-protocol statistics. The objective was to periodically sample summary statistics (`netstat –s` output), collect the output samples into a state vector and then use the samples to define classes of network conditions. Full netstat statistics contain a rich set of statistic variables (3 udp, 52 tcp, 29 ip, 33 icmp, and 9 igmp). The listing below in 6.2-1 is the select set used for as features in the discriminator.

```
% netstat -s
UDP
udpInDatagrams
udpOutDatagrams

TCP
TcpHalfOpenDrop
tcpInAckBytes
tcpInAckUnsent
tcpInErrs
tcpOutDataBytes

IP
ipInDelivers
ipInReceives
ipInUnknownProtos
ipOutRequests
rawipInOverflows

ICMP
icmpInMsgs
icmpInErrors
icmpInOverflows
```

```
icmpOutMsgs
```

Listing 6.2-1 Netstat (select variables)

TCP and UDP traffic generators based on `netperf` and `mgen` were used to produce six classes of network conditions as shown in listing 6.2-2. “3 to 1” indicates a ratio of attacks sources to target.

Traffic	DDos
Light traffic	none
Medium traffic	none
Heavy traffic	none
Light traffic	trinoo (3 to 1)
Medium traffic	trinoo (3 to 1)
Heavy traffic	trinoo (3 to 1)

Listing 6.2-2 Detection Classes

6.2.1 Observations and Discussion of results

Two runs of the experiment were made, collecting data at both one-second intervals and five-second intervals. Successive differences of netstat values were captured (not the raw absolute values). Initial observations of the one-second interval data revealed duplicate samples and low variances. These data required additional processing to eliminate duplicates and linear dependencies, accidental or otherwise. The five-second interval data was slightly less problematic in terms of duplicate samples. However, for both sample rates certain features had little or no variance (i.e., were not well chosen features). In part this is a reflection of the traffic generators, but also a result of netstat’s internal data structures. In both cases the covariance matrix was prone to contain linear dependent rows, making the input features (netstat capture) unsuitable for this technique. The expectation that the detection problem does not have generalized solutions is evident. Further effort with alternate inputs and feature selection is required to validate this technique.

6.3 Chi-Square DDoS detection with argus

This implementation of the “Goto and Iguchi” [GOTO1999] method computes the chi-square of two vectors of equal length, one holding short term information and one long term. The former might be updated every second while the latter once an hour or once a day. At initialization the sampling rate and the total number of slots in each vector is set. As initialization continues, the maximum number of UDP packets arriving in a single sampling period is discovered; each slot in the vectors is then set to correspond to an equal-sized subset of the range from 0 to MaxUDP. For example, if the MaxUDP is 100 packets/sampling period, and there are 10 slots in each vector, the first would cover the range from 0 to 10, the second from 10 to 20, and so on, with the tenth covering from 90 to 100. Once initialized the implementation watches the UDP packet stream during each sampling period, and then increments the appropriate bin of the vector -- if the number of packets is 17, the second bin is incremented; if the number of packets is 97, the tenth bin is incremented. At each update, the vector is aged or decayed, so that past records do not overwhelm current records. As time goes on, each of the vectors provides a snap shot of the distribution of the number of packets arriving per sampling period. After each update, the chi-square value of the vectors is computed, essentially comparing corresponding slots between the two vectors. Because the data in each slot may be sparse, the chi-square value has no particular statistical validity, but by manually setting a threshold the chi-square can still be used as an alarm trigger.

Argus is a UNIX-based network auditing tool developed at Carnegie Mellon University. Argus captures detailed information about all IP datagrams on the subnet. Argus runs as a daemon, reading all packets promiscuously over a specified interface. A wide variety of information is kept about each datagram, including both source and destination addresses, its size in bytes, the IP options in use, and for UDP/TCP datagrams, the source and destination port numbers. Bundled with Argus are several tools for examining the logged data, as well as templates for building custom tools.

Argus keeps track of the state of TCP connections. This is useful for the purpose of detecting some kinds of DDoS attacks. For example, some attacks generate large numbers of falsified TCP packets, for which no connection exists. Argus could detect this kind of anomalous behavior, potentially providing useful evidence of a DDoS attack. Similarly, Argus attempts to correlate UDP packets with each other, inferring state about possible UDP connections by patterns in the packet stream.

Argus provides two means of accessing the network data it logs: binary audit log files; and clients, that directly connect to the Argus daemon, which acts as a server. Clients receive the data more or less in real time, which suits the needs of a DDoS detection system.

6.3.1 Experiment Description

This experiment was designed to explore two issues: a preliminary method to automatically detect the presence

of a trinoo attack and whether Argus is a suitable traffic sniffer for the desired rapid detection of the attack.

The first step in an automatic response to a DDoS attack will be the detection of the attack itself. “Goto and Iguchi” lay out a technique for detecting anomalous activities on a port by port basis, by comparing a short term profile to a long term profile and using these to generate a chi-square value. The technique can be used to examine the behavior of any number of features of a system, from packets per second to different ports in use per time step. “Goto and Iguchi” obtained good results using this method to watch for relatively low volume changes in output on each port, and when turned toward the problem of DDoS detection, there is a far less subtle pattern to decipher. Traffic will suddenly increase, and the chi-square values with it.

Using the client template provided with Argus, an implementation of the “Goto and Iguchi” method was integrated with the Argus sniffer. Watching specifically for changes in the number of bytes of UDP traffic being passed into the target machine, Org1, per time step. All ports were lumped together for this initial experiment, and the traffic generation scripts were turned on.

The client system was allowed to reach a baseline equilibrium before the trinoo attack began. The short-term profile was updated up to once per second, less often if no new UDP packets arrived. While in practice the long-term profile would be updated only once per day, here, for the sake of time, it was updated once every one-hundred seconds. When the client had stabilized with a chi-square value averaging 0.250. A 10 second trinoo attack was launched from the three slave machines.

With the Argus server set to not attempt to correlate UDP packets with each other, it was able to immediately pass details about incoming packets to the client. Within four seconds the chi-square value spiked to 240. From there it steadily rose to over 1000. The values actually obtained, in the hundreds and thousands, make it clear beyond that something strange is occurring. It is still possible that this sort of heavy UDP traffic could have a legitimate reason, but it would be highly anomalous.

The results of this experiment show that Goto and Iguchi’s method of detecting strange network behavior is applicable to DDoS detection. Figure 6.2-1 the chi-square value computed from the short and long term UDP traffic data, immediately before and during a brief trinoo attack. At time 8883, the chi-square increases, approaching 1000 in a matter of seconds.

Figure 6.2-1 is interesting for two reasons. First, it illustrates some of the intensity of the attack and the ease with

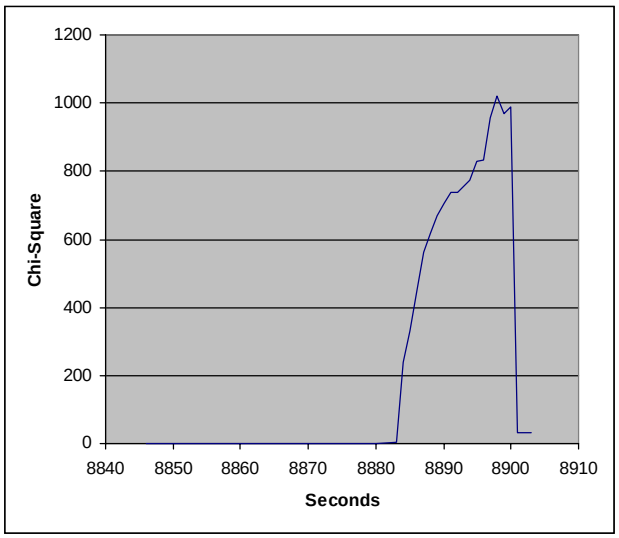

Figure 6.2-1 Chi-Square plot of UDP traffic

which it can be detected. Second, the precipitous fall-off at 8900 seconds displays some of the flexibility of the Goto and Iguchi method of anomaly detection: At 8900 seconds, the long-term data was updated to include the beginning of the trinoo attack. By placing an update in the middle of the attack, it can be seen how easily the algorithm is able to compensate for any level of baseline (long-term) traffic, even if it’s quite high. This approach should prove useful when the network is under heavy use, because it detects the deviation from the norm rather than just looking for network traffic overload.

6.4 NTOP

“ntop” is an open source network monitoring application under continuing development by Luca Deri. Luca began ntop initially while at the University of Pisa. “ntop” can be obtained from Luca’s web site http://www.ntop.org/. “ntop” is based on libpcap and has been written to be portable. It can be run on most UNIX and WIN32 platforms.

The ntop application captures packets at the Internet Protocol (IP) level for analysis. These packets are then categorized based upon type, source, and destination. Packet data content is not kept, only the resultant statistics. Traffic analysis is performed upon predefined cyclic timelines.

“ntop” provides the ability to view network traffic sorted by protocol, traffic, distribution among protocols, and source/destination usage. Two fundamental user

interfaces are provided as shown in figure 6.4-1; Web, and SQL via UDP. In addition, an event alerting capability is supported based upon packet filtering criteria.

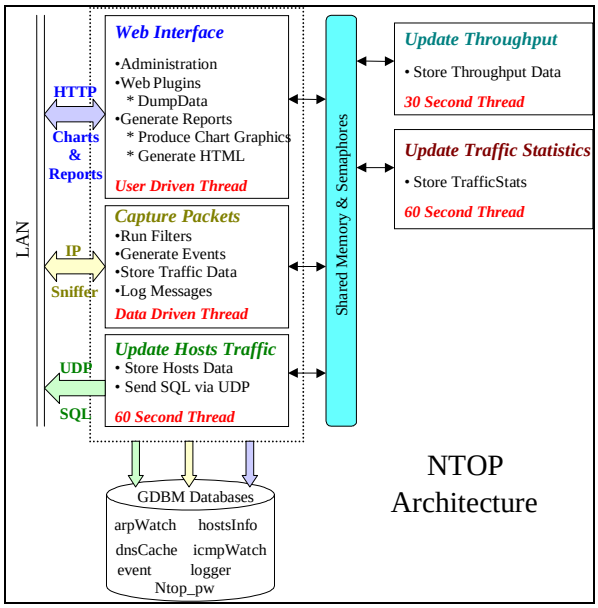

Figure 6.4-1 NTOP Architecture

6.4.1 Web Interface

The built-in web server provides formatted statistics and graphs on-the-fly to any browser. The default port used is port 3000, but this may be changed at start-up. Lower level statistics, such as individual machines, may be selected simply by clicking on selected fields in higher level overview statistics. Alert events, (if enabled), are displayed at the individual machine detail level.

The web interface also provides a "dump" request to allow external analysis of the collected data. This is provided as an HTML formatted string of a hierarchical Perl hash declaration. A Perl program was developed to use this interface and unwrap the hierarchical structure to produce a flat ASCII delineated file suitable for import into applications such as Excel. Multiple flat file dumps over time allow snapshot deltas to be computed from the NTOP continuous statistics.

6.4.2 UDP SQL Interface

One of the start-up options is to define a UDP target machine and port for SQL commands. When this interface is enabled, ntop will generate SQL to create, delete, and update data table content. Perl and Java listeners were developed to accept the UDP messages and interface to MS Access, and MySQL databases.

6.4.3 Observations

"ntop" is a work in progress. The current capabilities are very useful and informative within its predefined implementation but the customization features still need work. This is exacerbated by convoluted code and few useful comments. The ADN project team forwarded several code fixes to Luca dealing with erroneous SQL generation, hash key overlays, and other functional errors.

Stopping and restarting ntop results in a loss of most collected statistics. The one exception is alert events. There is a flag that is supposed to allow maintaining history over a restarts, but to date we have not been able to make this work.

The SQL generated does not represent complete data content. In addition, the SQL commands are based upon sample deltas to the internal data structure. This means that ntop restarts invalidate the collected SQL data. It also requires that the SQL listener be running before the ntop start-up. Extending this to the concept of a single SQL database for multiple ntop sources becomes infeasible due to this approach by ntop.

The Perl dump data interface is more complete than the SQL. Dump data coincides with the fundamental ntop approach of maintaining running statistics. At any given moment it allows a dump of the current state. In addition to the hash key fixes we provided for the dump data interface, ntop still needs additional work here as well. Dump does not include the alert event data at all. Additionally, there appears to be a buffering problem in the dump code. Repetitive dump requests randomly shows some data elements as zero or null. The rate at which this data error occurs correlates to the dump repetition cycle.

The ntop data is maintained as a cluster of GDBM databases. It may be possible to extract data directly from these databases but we have not yet tried this approach.

7 Anti-DDoS Tool

Tools are emerging to counter-act some DDoS attack systems and frameworks. This class of tool is aimed at disabling the DDoS infrastructure. It is analogous to virus eradicators. A review provides an insight to some techniques for proactive defense. These tools emulate components of a DDoS system, therefore their operation may be confused with actual DDoS events, triggering Intrusion Detection Systems (IDS).

7.1 ZOMBIE ZAPPER

Zombie Zapper (ZZ) [NOMAD2000] is a tool that operates on UNIX and Windows NT operating systems. This tool emulates the "master" protocol source providing a "quit" or "die" command to the clients. This tool is dependent upon default parameters specified in the source code of the DDoS. ZZ is less effective where the clients are sending packets from behind a firewall that blocks the incoming ports. This tool is not effective in cases where alternate ports via proxy servers or tunneling are used. The tool was built and tested on Solaris 7 and RedHat Linux 6.2 operating systems. The Solaris version of ZZ did not appear to stop the default trinoo clients when asserted. Operating from the Linux platform it did operate as expected. The UNIX version can select different physical network interfaces for its messages. Other options include ability to send to a class C broadcast address, set a timeout for sending, selecting the UDP send port (for trinoo only). Listing 7.1-1 is the UNIX command line interface summary of the tool.

```
./zz [-a 0-5] [-c class C] [-d dev] [-h] [-m host]
[-s src] [-u udp] [-v]
hosts

 -a antiddos type to kill:
    0 types 1-4 (default)
    1 trinoo
    2 tfn
    3 stacheldraht
    4 trinoo on Windows
    5 shaft (requires you use the -m option)
 -c class C in x.x.x.0 form
 -f time in seconds to send packets (default 1)
 -d grab local IP from dev (default eth0)
 -h this help screen
 -m my host being flooded (used with -a 5 above
   only one host)
 -s spoofed source address (just in case)
 -u UDP source port for trinoo (default 53)
 -v verbose mode (use twice for more verbosity)
 host(s) are target hosts (ignored if using –c)
```

Listing 7.1-1 ZZ UNIX command line parameters

The sample runs included below provide insight into the payload statistics, data, and the DDoS target type.

```
[root@redhat1 zombie-1.2]# ./zz -v 20.0.0.1
Zombie Zapper v1.2 - DDoS killer
Bugs/comments to thegnome@razor.bindview.com
More info and free tools at
http://razor.bindview.com
Copyright (c) 2000 BindView Development

Sending packets to stop these possible
daemons from flooding
    Trinoo, TFN, Stacheldraht, Troj_Trinoo

Building anti-Trinoo packets
 Payload is "d1e l44adsl d1e"
 Data length of 15
 Packet size is 43
 48 packets sent in 1 seconds
Building anti-TFN packets
 Payload is "12345"
 Data length of 5
 Packet size is 33
 50 packets sent in 1 seconds
Building anti-Stacheldraht packets
 Payload is NULL
 Data length of 0
 Packet size is 28
 49 packets sent in 1 seconds
Building anti-Troj_Trinoo packets
 Payload is "d1e [].Ks l44"
 Data length of 14
 Packet size is 42
 48 packets sent in 1 seconds
Complete
```

Listing 7.1-2 ZZ UNIX example output

The following output sequence displays trinoo mping output indicating that one of its broadcasts hosts has been disabled. In many cases the receiving trinoo client would need to be restarted, as an example this could be accomplished as a UNIX CRON job every minute. However it is unlikely one would be able to detect that the clients had been terminated in any way except reduction of UDP (in trinoo's case) packets against the victim system. In summary there is no formal acknowledge of client termination.

BEFORE zombie zapper run in listing 7.1-2

```
trinoo> mping
mping: Sending a PING to every Bcasts.
trinoo> PONG 1 Received from 40.0.0.1
PONG 2 Received from 30.0.0.1
PONG 3 Received from 20.0.0.1
```

AFTER zombie zapper run in listing 7.1-2

```
trinoo> mping
mping: Sending a PING to every Bcasts.
trinoo> PONG 1 Received from 40.0.0.1
PONG 2 Received from 30.0.0.1
```

Figure 7.1-1 is a screen dump of the tool's graphical user interface for Windows NT. This version does not allow selection of different physical network interfaces.

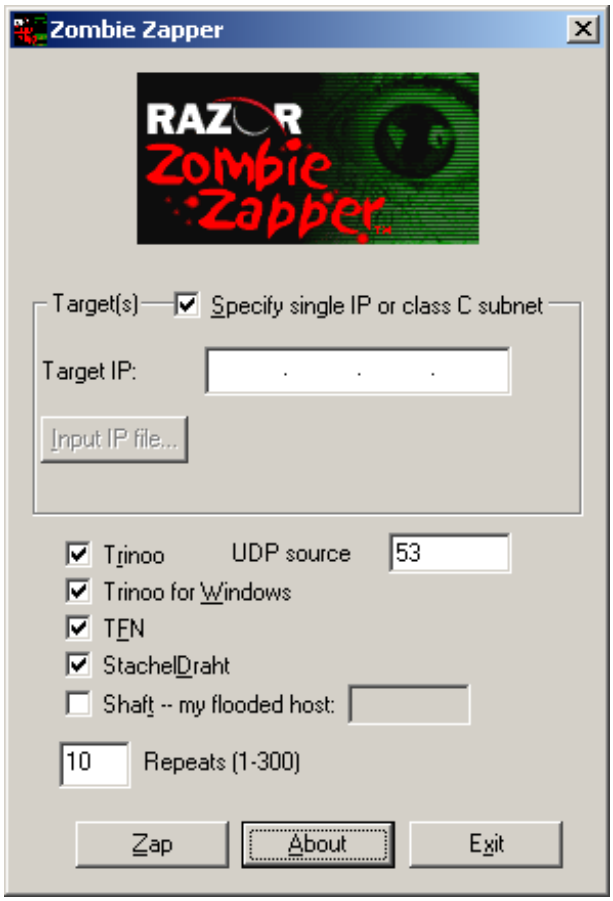

Figure 7.1-1 ZZ Graphical User Interface

Zombie zapper offers one early approach for responding to DDoS attacks. It is conceivable and practical to automate response with the UNIX version of the tool. One side effect was noted when operating the network interface monitor utility "snoop" on Solaris 7. When sending Zombie Zapper packets to the Solaris 7 target which was also being monitored with snoop, it caused snoop to core dump. The network utility tcpdump-3.5.2 operated without error.

8 ADN Concept Experiments

These experiments operate on the testbed to validate and explore techniques for ADN. They represent fundamental elements to an ADN solution.

8.1 Secure Channel Bandwidth Reservation

This experiment focused on performing a test utilizing Quality of Service classes for channel reservation of control traffic while under DDoS attack. Solaris Bandwidth Manager (SBM) allows network traffic types to be defined into seven priority classes. These classes are used as filters to provide prioritization and scheduling. The attributes used for class definition include, IP addresses (source, destination), IP Protocol type (UDP, TCP, other), ports for TCP/UDP (source, destination), Type of Service value (TOS) and URL or URL groups. SBM was configured to reserve 15% bandwidth allocation for telnet traffic and Secure Shell (SSH).

SBM was installed and operating on the pseudo internet service provider platform in the assessment environment as described in section 1.2. Parameters were also set to simulate 10 MBPS ISP link rates from Organizations 2, 3, 4 with Organization 1 having a 1.544 MBPS link rate. SBM includes a Java based management graphical user interface to configure and control the capabilities of the tool. During a DDoS attack on organization 1 the routing platform completely froze or stalled all processing. No keyboard, or mouse events were processed. Additionally no packets were able to pass through the router and SBM.

This behavior continued until the DDoS stopped, and even a few seconds after the DDoS termination. "trinoo" was used as the DDoS attack tool and was operated for 20-60 seconds and represents an early and immature capability compared to currently available tools. The leverage of bandwidth is relatively light compared to possible DDoS architectures, where the target system may see packets from hundreds or thousands of sources. Testing has also revealed that the SBM product while under attack will not allow organization 1 to emit and receive ping responses to organization 5. The same test was performed without SBM and pings operated properly between organization 1 and 5.

8.2 Adaptive Firewall Experiment #2

SBM was removed for the following experiment due to reasons described in the section 8.1. It was tested during attack and did not allow any outbound connections from organization 1 to organization 2. A primary focus of this project is to understand the behavior issues when dynamically updating rules in firewalls. The very first experiment was created to explore this using IP Filter as the stateful packet filtering firewall. This project developed a set of tools for dynamic modification of rulesets across the network. This experiment presumes the firewalls are across a virtual private network that allows dynamic rule modifications.

Figure 8.2-1 depicts the set of messages to support this experiment. A set of tests were performed and timed to provide a response time estimate while background TCP and UDP traffic generators were operating to load the network from organizations 2, 3, and 4. Traffic generators

were developed and operated to generate profile based network loads. In this experiment organizations 2, 3, and 4 were generating 3 to 5 megabits per second of background traffic. The attack traffic was also produced from organizations 2, 3, and 4 using the remaining bandwidth. During the test the nominal ping time, normally 1 millisecond from organization 1 to organization 5, climbed to 70-150 milliseconds during the DDoS attack. While this delayed the communication between organizations it did not stop it, allowing firewall updates to be made. Table 8.2-1 lists the reaction times for firewall modification. Figure 8.2-1 and 8.2-2 illustrate the message sequence as a result of the attack.

Table 8.2-1 Firewall Reaction Times

	Insert Block Rule (seconds)	Remove Block Rule (seconds)
Traffic load	10	11
Traffic load with DDoS	14	13

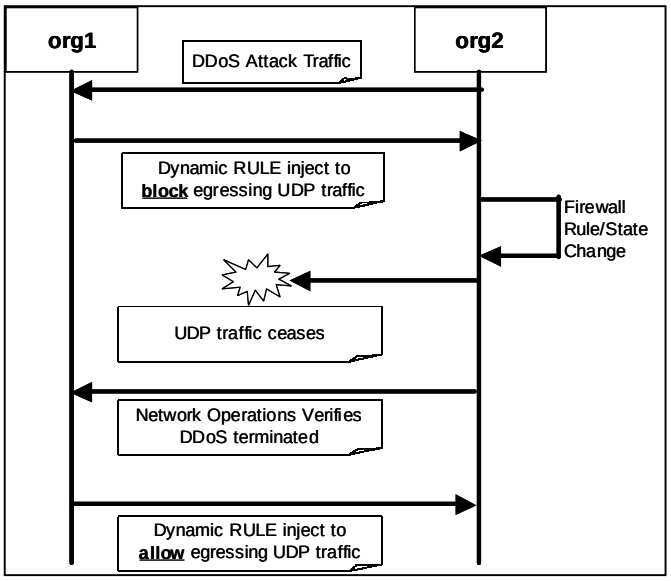

Figure 8.2-1 Dynamic Firewall Rule Message Sequence

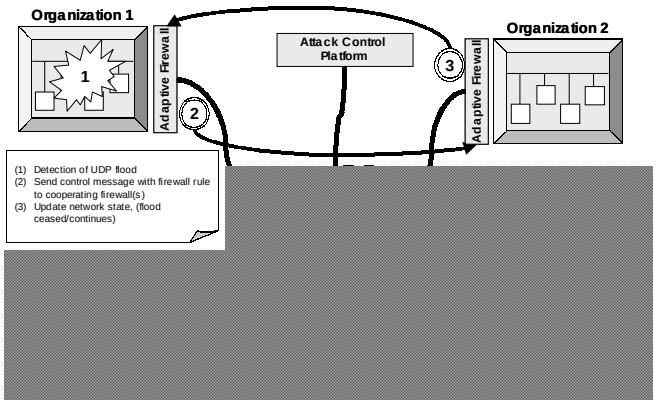

Figure 8.2-2 Overview of Experiment #2

REFERENCES

[BLAZE2000]
Blaze, Matt, et. al., “The Role of Trust Managment in Distributed Systems Security,” AT&T Research Labs and Distibute Sytems Lab, Univ. Penn.

[CERT/CC1999]
CERT/CC et. al., “Results of the Distributed-Systems Intruder Tools Workshop,” CERT/CC, November 1999.

[CISCORPF2000]
“Unicast Reverse Path Forwarding,” CISCO, May 2000.

[COMERFORD2001]
Comerford, Richard, “No Longer in Denial,” IEEE Spectrum, January 2001.

[DITTRICH2000]
Dave Dittrich Denial of Service information site, http://www.washington.edu/People/dad/

[GOTO1999]
Detecting Malicious Activiites Through Port Profiling, Goto, Shigeki, and Iguchi, Makoto, IEEE trans. Information and systems, vol E82-D, NO. 4 April 1999.

[IOANNIDIS2000]
Ioannidis S., Keromytis D., Bellovin S., Smith J., “Implementing a Distributed Firewall”, ACM Converence on Computer Communications Security, Athens, Greece, November 2000.

[INTERNET2000]
D. Durham, J. Boyle, R. Cohen, S. Herzog, et al “The Common Open Policy Service Protocol (Request for Comments: 2748)", January 2000.

[KEYNOTE]
Blaze, Matt, et. al., “The Keynote Trust Management System,” work in progress. Internet Draft, April 1998, http://www.cis.upenn.edu/~angelos/draft-angelos-spki-keynote.txt.gz

[NOMAD2000]
Simple Nomad, Zombie Zapper, RAZOR, BindView http://razor.bindview.com/tools/ZombieZapper_form.shtml

[RID2000]
Remote.Intrusion.Detector, TheoryGroup, http://www.theorygroup.com/

APPENDIX A

Table A-1 describes systems and platforms that were used to generate the results in this appendix. Firewall rules were set to block all incoming and outgoing packets on the interface.

Table A-1. Platform Overview

Item Reference	Type	Description
Firewall	IP Filter 3.4.14	Open source firewall system software
Platform S1 192.168.168.8	Redhat 7.0, Pentium III, 533 MHZ/133FSB, 256 MB RAM, 100MBPS NIC	Attack machine for DoS exploits Source of attacks
Platform FW1 192.168.168.254	Solaris 8, Sparc Classic, 40MHZ SuperSparc, 96MB RAM, 10MBPS onboard NIC	Host firewall platform, target of all attacks
Platform FW2 192.168.168.8	Solaris 8, SparcStation 20, Dual 75MHZ SuperSparc, 256MB RAM, 100MBPS NIC	Host firewall platform, used for comparison in severe DoS situations.

The other row of information below each event is the firewall block log. The fields that are common in the log output from the 'ipmon' utility are:

Field Description

1 The date of packet receipt. This is suppressed when the message is sent to syslog.

2 The time of packet receipt. This is in the form HH:MM:SS.F, for hours, minutes seconds, and fractions of a second (which can be several digits long).

3 The name of the interface the packet was processed on, e.g., we1.

4 The group and rule number of the rule, e.g., @0:17. These can be viewed with ipfstat -n.

5 The action: p for passed or b for blocked.

6 The addresses. This is actually three fields: the source address and port (separated by a comma), the -> symbol, and the destination address and port. E.g.: 209.53.17.22,80 -> 198.73.220.17,1722.

7 PR followed by the protocol name or number, e.g., PR tcp.

8 len followed by the header length and total length of the packet, e.g., len 20 40.

Table A-2 is an overview of the firewall events examined in the remainder of the appendix A.

Table A-2. Attack/Probe Event Overview

Event	Attack/Probe
001	Ping
002	ping flood
003	telnet to a non standard telnet port
004	ftp
005	telnet (connection sequence)
006	ping maximum packet size
007	Flood ping with maximum packet size
008	**arnup** [CODE COMMENT] sends a single UDP datagram with the source and destination address and port set to whatever you want. datagram passed to kernel: 45002600 00000000 ff110000 c0a8a8fc c0a8a8fc 03e807d0 00120000 31323334 35363738 3930
009	**jolt2** [CODE COMMENT] This is the proof-of-concept code for the Windows denial-of-serice attack described by the Razor team (NTBugtraq, 19-May-00) (MS00-029). This code causes cpu utilization to go to 100%.
010	**kod** [CODE COMMENT] bluescreens windows users(98/98se) and kills tcp stack windows handles igmp badly and this is the result
011	**kox** [CODE COMMENT] this was a successful attempt to duplicate klepto/defile's kod win98 exploit and add spoofing support to it. affected systems: windows 98, windows 98 SE, windows 2000 build 2000 results: bluescreen, tcp/ip stack failure, lockup, or instant reboot
012	**nestea** [CODE COMMENT] This exploits the "off by one ip header" bug in the linux ip frag code. Crashes linux 2.0.* and 2.1.* and some windows boxes this code is a total rip of teardrop - it's messy
013	**newtear** [CODE COMMENT] this is a new version of teardrop. It affects NT 4 and Win95 machines with all current patches and hotfixes. Causes a bluescreen in both operating systems. Linux appears unaffected, other *NIXes untested. Differences are: Smaller padding data size (20 bytes instead of 28 in previous teardrop) Faked out UDP total length. (Increased reported UDP length to twice what it really is)

014	**sesquipedalian** Affects Linux kernels between 2.1.89 and 2.2.3. This sends a series of IP fragments such that a 0 length fragment is first in the fragment list. This causes a reference count on the cached routing information for that packet's originator to be incremented one extra time. This makes it impossible for the kernel to deallocate the destination entry and remove it from the cache. It is possible for a malicious attacker to send spoofed RPC datagrams to UDP destination port 135 so that it appears as if one RPC server sent bad data to another RPC server. The second server returns a REJECT packet and the first server (the spoofed serv er) replies with another REJECT packet creating a loop that is not broken until a packet is dropped, which could take a few minutes. If this spoofed UDP packet is sent to multiple computers, a loop could possibly be created, consuming processor resources and network bandwidth http://www.linuxgangster.com/dos.htm
015	**synk4** Any system providing TCP-based services to the Internet community are potentially vulnerable to this Denial of Service attack. The Synk4 DoS is considered a Syn Flooding type of attack. http://www.wwdsi.com/demo/saint_tutorials/synk4.html [CODE COMMENTS] Syn Flooder by Zakath * TCP Functions by trurl_ (thanks man). Random IP Spoofing Mode – ultima How To Use: Usage is simple. srcaddr is the IP the packets will be spoofed from. dstaddr is the target machine you are sending the packets to. low and high ports are the ports you want to send the packets to. Random IP Spoofing Mode: Instead of typing in a source address, just use '0'. This will engage the Random IP Spoofing mode, and the source address will be a random IP instead of a fixed ip.
016	**targa** [CODE COMMENTS] targa 1.2 by Mixter usage: ./targa1 <startIP> <endIP> [-t type] [-n repeats] interface to 8 multi-platform remote denial of service exploits usage: targa1 <startIP> <endIP> [-t type] [-n repeats] startIP - endIP: IP range to send packets to (destination) start and end must be on the same C class (1.1.1.X) repeats: repeat the whole cycle n times (default is 1) type: kind of remote DoS to send (default is 0) 1 = bonk ($) 2 = jolt (@) 3 = land (-) 4 = nestea (.) 5 = newtear (#) 6 = syndrop (&) 7 = teardrop (%) 8 = winnuke (*) 0 = use all remote DoS types at once
017	**targa2** [CODE COMMENTS] targa 2.1 by Mixter usage: ./targa2 <startIP> <endIP> [-t type] [-n repeats] startIP - endIP: IP range to send packets to (destination) start and end must be on the same C class (1.1.1.X) repeats: repeat the whole cycle n times (default is 1) type: kind of remote DoS to send (default is 0) 1 = bonk ($) 2 = jolt (@) 3 = land (-) 4 = nestea (.) 5 = newtear (#) 6 = syndrop (&) 7 = teardrop (%) 8 = winnuke (*) 9 = 1234 (!) 10 = saihyousen (+) 11 = oshare (|) 0 = use all remote DoS types at once
018	**targa3** [CODE COMMENTS] IP stack penetration tool / 'exploit generator' Sends combinations of uncommon IP packets to hosts to generate attacks using invalid fragmentation, protocol, packet size, header values, options, offsets, tcp segments, routing flags, and other unknown/unexpected packet values. Useful for testing IP stacks, routers, firewalls, NIDS, etc. for stability and reactions to unexpected packets. Some of these packets might not pass through routers with filtering enabled - tests with source and destination host on the same ethernet segment gives best effects.
019	**teardrop** Some implementations of the TCP/IP IP fragmentation re-assembly code do not properly handle overlapping IP fragments. Teardrop is a widely available attack tool that exploits this vulnerability. Exploits the overlapping IP fragment bug present in all Linux kernels and NT 4.0 / Windows 95 (others?)

020	**tentacle** NOTE: also exists as a virus for Windows 3.1 [CODE COMMENTS] proof-of-concept DoS against tcp (coded in 10 mins :p) open a huge number of sockets to a server,then terminate the process without closing the connection on the server side
021	**twinge** [CODE COMMENTS] this cycle through all the possible icmp types and subtypes and send to target host, 1 cycle == 1 run thru all of em twinge.c by sinkhole@dos.org - licensed for use by the public This is a PoC (Proof of Concept) program for educational uses. usage: ./twinge <dest> <cycles [0 == continuous]>
022	**winnuke** The WinNuke attack sends OOB (Out-of-Band) data to an IP address of a Windows machine connected to a network and/or Internet. Usually, the WinNuke program connects via port 139, but other ports are vulnerable if they are open. When a Windows machine receives the out-of-band data, it is unable to handle it and exhibits odd behavior, ranging from a lost Internet connection to a system crash (resulting in the infamous Blue Screen of Death). http://www.wwdsi.com/demo/tutorials/winnuke.html

Table A-3. IP Filter Log ping, telnet, ftp

Event	Source / Destination	Command Line	Command output / Notes
001	S1/FW1	`% ping FW1`	Standard ping command

```
03/11/2000 08:07:39.251985 le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:07:40.251727 le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:07:41.251475 le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:07:42.251218 le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:08:59.835366 le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
```

002	S1/FW1	`% ping –f FW1`	ping with flood option

```
03/11/2000 08:10:58.101776 99x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:10:59.092619   le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:10:59.101565   le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:10:59.112290   le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
03/11/2000 08:10:59.121495 99x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 84 icmp 8/0 IN
```

003	S1/FW1	`% telnet FW1 300`	telnet to port 300

```
03/11/2000 08:23:42.565436 le0 @0:2 b 192.168.168.8,33119 -> 192.168.168.252,300 PR tcp len 20 60 -S 3746563564 0 5840 IN
03/11/2000 08:23:45.558101 le0 @0:2 b 192.168.168.8,33119 -> 192.168.168.252,300 PR tcp len 20 60 -S 3746563564 0 5840 IN
03/11/2000 08:23:51.556599 le0 @0:2 b 192.168.168.8,33119 -> 192.168.168.252,300 PR tcp len 20 60 -S 3746563564 0 5840 IN
```

004	S1/FW1	`% ftp FW1`	Standard FTP session connection attempt

```
03/11/2000 08:40:51.182814 le0 @0:2 b 192.168.168.8,33196 -> 192.168.168.252,21 PR tcp len 20 60 -S 555229045 0 5840 IN
03/11/2000 08:40:54.179115 le0 @0:2 b 192.168.168.8,33196 -> 192.168.168.252,21 PR tcp len 20 60 -S 555229045 0 5840 IN
03/11/2000 08:41:00.177608 le0 @0:2 b 192.168.168.8,33196 -> 192.168.168.252,21 PR tcp len 20 60 -S 555229045 0 5840 IN
```

005	S1/FW1	`% telnet FW1`	Standard telnet session Note: the port was opened up and passed the packets so handshake could be analyzed

```
03/11/2000 08:49:04.611259 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 48 -S 1630947217 0 24820 IN
03/11/2000 08:49:04.613456 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 40 -A 1630947218 1606348785 24820 IN
03/11/2000 08:49:04.618522 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 64 -AP 1630947218 1606348785 24820 IN
03/11/2000 08:49:04.981992 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 40 -A 1630947242 1606348800 24820 IN
03/11/2000 08:49:04.983245 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 55 -AP 1630947242 1606348800 24820 IN
03/11/2000 08:49:05.078700 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 40 -A 1630947257 1606348815 24820 IN
03/11/2000 08:49:05.087354 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 57 -AP 1630947257 1606348833 24820 IN
03/11/2000 08:49:05.198663 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 40 -A 1630947274 1606348854 24820 IN
03/11/2000 08:49:05.277986 le0 @0:1 p 192.168.168.7,32882 -> 192.168.168.252,23 PR tcp len 20 46 -AP 1630947274 1606348860 24820 IN
```

Table A-4. IP Filter log, ping, arnup100, jolt2, kod

Event	Source / Destination	Command Line	Command output / Notes
006	S1/FW1	`% ping –s 65507 FW1`	ping with maximum size packets, note repeat blocks and fragmentation

`03/11/2000 09:19:41.143925 44x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (415) frag -395@65120 IN`
`03/11/2000 09:19:41.198094  le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 1500 icmp 8/0 IN`
`03/11/2000 09:19:42.144331 44x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (415) frag -395@65120 IN`
`03/11/2000 09:19:42.198499  le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 1500 icmp 8/0 IN`
`03/11/2000 09:19:43.144029 44x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (415) frag -395@65120 IN`
`03/11/2000 09:19:43.198206  le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 1500 icmp 8/0 IN`		

007	S1/FW1	`% ping –s 65507 FW1 -f`	ping and flood with maximum size packets, note repeat blocks and fragmentation

`03/11/2000 09:21:10.191229 31x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (1500) frag +-1480@11840 IN`
`03/11/2000 09:21:10.225882 11x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (1500) frag +-1480@59200 IN`
`03/11/2000 09:21:10.238569 10x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (1500) frag +-1480@35520 IN`
`03/11/2000 09:21:10.250136 12x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (1500) frag +-1480@50320 IN`
`03/11/2000 09:21:10.263012 10x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR icmp len 20 (1500) frag +-1480@63640 IN`		

008	S1/FW1	`% arnup100 FW1 1000 FW1 2000`	
`03/11/2000 16:54:32.456411 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN`			
009	S1/FW1	`% jolt2 –s FW1 –p 2000 FW1`	NOTE: 29209 packets. This tool generated approximately 5000 packets/second
`03/11/2000 17:02:05.929395 29203x le0 @0:2 b 127.0.0.1 -> 192.168.168.252 PR`			
010	S1/FW1	`% kod FW1 –p 0508 –t 10`	
`03/11/2000 17:09:12.253479 44x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR 2 len 20 (220) frag -200@14800 IN` `03/11/2000 17:09:12.484769 66x le0 @0:2 b 192.168.168.8 -> 192.168.168.252 PR 2 len 20 (220) frag -200@14800 IN`			

Table A-5. IP Filter log, kox, nestea

Event	Source / Destination	Command Line	Command output / Notes
011	S1/FW1	% kox FW1	

```
03/11/2000 17:14:28.609139 le0 @0:2 b 36.72.20.64 -> 192.168.168.252 PR 2 len 20 (1500) IN
03/11/2000 17:14:29.618277 le0 @0:2 b 36.72.20.64 -> 192.168.168.252 PR 2 len 20 (1500) frag +1480@1480 IN
03/11/2000 17:14:31.625837 le0 @0:2 b 36.72.20.64 -> 192.168.168.252 PR 2 len 20 (1500) frag +1480@2960 IN
03/11/2000 17:14:33.635330 le0 @0:2 b 36.72.20.64 -> 192.168.168.252 PR 2 len 20 (1500) frag +1480@4440 IN
```

012	S1/FW1	% nestea FW1 FW1 –s 1000 –t 2000 –n 100	NOTE: this is only a portion of the blocks

```
03/11/2000 17:17:06.189327 le0 @0:2 b 192.168.168.7,138 -> 192.168.168.255,138 PR udp len 20 261 IN
03/11/2000 17:17:26.738900 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:26.739346 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:26.739799 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:26.750839 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:26.751287 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:26.751739 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:28.611256 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:28.630433 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:28.630877 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:28.631331 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:28.650325 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:28.650776 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:28.651239 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:28.670427 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:28.670877 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:28.690296 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:28.690746 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:28.691204 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
03/11/2000 17:17:28.710325 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 38 IN
03/11/2000 17:17:28.710772 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 17:17:28.711226 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 60 284 IN
```

Table A-6. IP Filter log, newtear, sesquipedlian

Event	Source / Destination	Command Line	Command output / Notes
013	S1/FW1	% newtear FW1 FW1 –s 1000 –t 2000 –n 100	NOTE: this is only a portion of the blocks

```
03/11/2000 18:13:09.648279 10x le0 @0:2 b 20.0.0.0,20 -> 192.168.168.252,12 PR udp len 20 28 IN
03/11/2000 18:13:09.652291 10x le0 @0:2 b 30.0.0.0,30 -> 192.168.168.252,13 PR udp len 20 28 IN
03/11/2000 18:13:09.656091 10x le0 @0:2 b 40.0.0.0,40 -> 192.168.168.252,14 PR udp len 20 28 IN
03/11/2000 18:13:09.659896 8x  le0 @0:2 b 50.0.0.0,50 -> 192.168.168.252,15 PR udp len 20 28 IN
03/11/2000 18:13:09.663096   le0 @0:2 b 60.0.0.0,60 -> 192.168.168.252,16 PR udp len 20 28 IN
03/11/2000 18:13:09.663562 2x  le0 @0:2 b 70.0.0.0,70 -> 192.168.168.252,17 PR udp len 20 28 IN
03/11/2000 18:13:09.664398 2x  le0 @0:2 b 80.0.0.0,80 -> 192.168.168.252,18 PR udp len 20 28 IN
```

014	S1/FW1	% sesquipedalian -s 1080 -d 1080 -n 100 -u 0 S1 FW1	

```
03/11/2000 18:16:37.135490 le0 @0:2 b 192.168.168.102,1080 -> 192.168.168.252,1080 PR udp len 20 52 IN
03/11/2000 18:16:37.135928 le0 @0:2 b 192.168.168.102,18768 -> 192.168.168.252,19533 PR udp len 20 20 IN
03/11/2000 18:16:37.136359 le0 @0:2 b 192.168.168.102 -> 192.168.168.252 PR udp len 20 (52) frag 32@32 IN
03/11/2000 18:16:37.140463 le0 @0:2 b 192.168.168.103,1080 -> 192.168.168.252,1080 PR udp len 20 52 IN
03/11/2000 18:16:37.140897 le0 @0:2 b 192.168.168.103,18768 -> 192.168.168.252,19533 PR udp len 20 20 IN
03/11/2000 18:16:37.141319 le0 @0:2 b 192.168.168.103 -> 192.168.168.252 PR udp len 20 (52) frag 32@32 IN
03/11/2000 18:16:37.141768 le0 @0:2 b 192.168.168.104,1080 -> 192.168.168.252,1080 PR udp len 20 52 IN
03/11/2000 18:16:37.142204 le0 @0:2 b 192.168.168.104,18768 -> 192.168.168.252,19533 PR udp len 20 20 IN
03/11/2000 18:16:37.142625 le0 @0:2 b 192.168.168.104 -> 192.168.168.252 PR udp len 20 (52) frag 32@32 IN
```

Table A-7. IP Filter log, synk4, targa

Event	Source / Destination	Command Line	Command output / Notes
015	S1/FW1	`% synk4 S1 FW1 1000 1100`	NOTE: this is only a portion of the blocks

```
03/11/2000 18:19:47.190616 le0 @0:2 b 203.91.108.80,2071 -> 192.168.168.252,1000 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.201971 le0 @0:2 b 24.156.172.223,2071 -> 192.168.168.252,1001 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.223785 le0 @0:2 b 244.194.187.221,2071 -> 192.168.168.252,1002 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.241931 le0 @0:2 b 139.27.63.204,2071 -> 192.168.168.252,1003 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.261896 le0 @0:2 b 135.102.201.205,2071 -> 192.168.168.252,1004 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.281878 le0 @0:2 b 69.16.159.24,2071 -> 192.168.168.252,1005 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.301909 le0 @0:2 b 69.228.153.178,2071 -> 192.168.168.252,1006 PR tcp len 20 40 -S 674719801 0 65535 IN
03/11/2000 18:19:47.321972 le0 @0:2 b 217.202.208.134,2071 -> 192.168.168.252,1007 PR tcp len 20 40 -S 674719801 0 65535 IN
```

016	S1/FW1	`% targa FW1=1 FW1 –t 0 –n 2`	NOTE: this is only a portion of the blocks

```
03/11/2000 18:26:36.640244 le0 @0:2 b 94.145.87.47,53 -> 192.168.168.252,53 PR udp len 20 56 IN
03/11/2000 18:26:36.920775 172x le0 @0:2 b 216.134.171.91 -> 192.168.168.252 PR icmp len 20 (400) frag +380@376 IN
03/11/2000 18:26:36.981740 le0 @0:2 b 93.133.103.8 -> 192.168.168.252 PR icmp len 20 400 icmp 8/0 IN
03/11/2000 18:26:36.982189 116x le0 @0:2 b 93.133.103.8 -> 192.168.168.252 PR icmp len 20 (400) frag +380@376 IN
03/11/2000 18:26:37.023525 17x le0 @0:2 b 30.183.118.113 -> 192.168.168.252 PR icmp len 20 (400) frag +380@1520 IN
03/11/2000 18:26:37.106573 le0 @0:2 b 170.65.67.39,1079 -> 192.168.168.252,50602 PR udp len 20 38 IN
03/11/2000 18:26:37.107017 le0 @0:2 b 170.65.67.39 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 18:26:37.107474 le0 @0:2 b 170.65.67.39,1079 -> 192.168.168.252,50602 PR udp len 60 284 IN
03/11/2000 18:26:37.111230 le0 @0:2 b 170.65.67.39 -> 192.168.168.252 PR udp len 20 (136) frag 116@48 IN
03/11/2000 18:26:37.111681 le0 @0:2 b 170.65.67.39,1079 -> 192.168.168.252,50602 PR udp len 60 284 IN
```

Table A-8. IP Filter log, targa2, targa3

Event	Source / Destination	Command Line	Command output / Notes
017	S1/FW1	% targa2 FW1-1 FW1 -t 9 -n 1	NOTE: this is only a portion of the blocks

```
04/11/2000 08:01:53.187112 le0 @0:2 b 193.169.2.193,2 -> 192.168.168.252,2 PR udp len 20 36 IN
04/11/2000 08:01:53.187595 le0 @0:2 b 194.170.3.194,3 -> 192.168.168.252,3 PR udp len 20 36 IN
04/11/2000 08:01:53.188079 le0 @0:2 b 195.171.4.195,4 -> 192.168.168.252,4 PR udp len 20 36 IN
04/11/2000 08:01:53.188554 le0 @0:2 b 196.172.5.196,5 -> 192.168.168.252,5 PR udp len 20 36 IN
04/11/2000 08:01:53.189031 le0 @0:2 b 197.173.6.197,6 -> 192.168.168.252,6 PR udp len 20 36 IN
04/11/2000 08:01:53.189494 le0 @0:2 b 198.174.7.198,7 -> 192.168.168.252,7 PR udp len 20 36 IN
04/11/2000 08:01:53.190101 le0 @0:2 b 199.175.8.199,8 -> 192.168.168.252,8 PR udp len 20 36 IN
```

018	S1/FW1	% targa3 FW1 FW1 –c 1	NOTE: this is only a portion of the blocks

```
04/11/2000 08:07:10.531450 le0 @0:2 b 154.170.84.17 -> 192.168.168.252 PR 255 len 20 (460) frag 440@8 IN
04/11/2000 08:07:10.537001 le0 @0:2 b 125.129.179.43,47582 -> 192.168.168.252,25427 PR tcp len 20 460 -AF 3832265876 215595288
60236 IN
04/11/2000 08:07:10.539239 le0 @0:2 b 152.69.17.101 -> 192.168.168.252 PR udp len 20 (460) frag +440@65528 IN
04/11/2000 08:07:10.544699 le0 @0:2 b 245.33.69.53 -> 192.168.168.252 PR egp len 20 (460) frag 440@8 IN
04/11/2000 08:07:10.546834 le0 @0:2 b 160.65.209.49,38574 -> 192.168.168.252,21645 PR tcp len 20 460 -SF 4113714051 1509308798
50591 IN
04/11/2000 08:07:10.555565 le0 @0:2 b 204.182.213.71 -> 192.168.168.252 PR 2 len 20 (460) frag 440@8 IN
04/11/2000 08:07:10.557705 le0 @0:2 b 65.176.74.29,59947 -> 192.168.168.252,37697 PR udp len 20 460 IN
04/11/2000 08:07:10.566038 le0 @0:2 b 198.35.123.50 -> 192.168.168.252 PR 4 len 20 (460) IN
```

Table A-9. IP Filter log, teardrop, tentacle, twinge, winnuke

Event	Source / Destination	Command Line	Command output / Notes
019	S1/FW1	% teardrop FW2 FW1 –s 1000 –t 2000 –n 1	
04/11/2000 08:09:12.220497 le0 @0:2 b 192.168.168.252,1000 -> 192.168.168.252,2000 PR udp len 20 56 IN 04/11/2000 08:09:12.220939 le0 @0:2 b 192.168.168.252 -> 192.168.168.252 PR udp len 20 (24) frag 4@24 IN			
020	S1/FW1	% tentacle FW1 7777 10	
04/11/2000 08:10:57.333015 le0 @0:2 b 192.168.168.8,34580 -> 192.168.168.252,7777 PR tcp len 20 60 -S 3980288427 0 5840 IN 04/11/2000 08:11:00.329956 le0 @0:2 b 192.168.168.8,34580 -> 192.168.168.252,7777 PR tcp len 20 60 -S 3980288427 0 5840 IN			
021	S1/FW1	% twinge	NOTE: this is only a portion of the blocks
04/11/2000 08:13:46.867779 le0 @0:2 b 65.218.100.63 -> 192.168.168.252 PR icmp len 20 29 icmp 0/0 IN 04/11/2000 08:13:46.868216 le0 @0:2 b 251.68.242.51 -> 192.168.168.252 PR icmp len 20 29 icmp 1/0 IN 04/11/2000 08:13:46.869067 le0 @0:2 b 44.95.203.104 -> 192.168.168.252 PR icmp len 20 29 icmp 2/0 IN 04/11/2000 08:13:46.869527 le0 @0:2 b 153.255.169.29 -> 192.168.168.252 PR icmp len 20 29 icmp 3/0 for 138.0.13.71 - 5.0.0.0 PR ipv6-icmp len 0 (20556) IN 04/11/2000 08:13:46.870149 le0 @0:2 b 153.255.169.29 -> 192.168.168.252 PR icmp len 20 29 icmp 3/1 for 138.0.13.72 - 232.0.0.0 PR ipv6-icmp len 0 (20556) IN 04/11/2000 08:13:46.870632 le0 @0:2 b 153.255.169.29 -> 192.168.168.252 PR icmp len 20 29 icmp 3/2 for 116.32.99.111 - 0.0.0.32 PR hopopt len 0 (29584) frag 29584@17440 IN 04/11/2000 08:13:46.871109 le0 @0:2 b 153.255.169.29 -> 192.168.168.252 PR icmp len 20 29 icmp 3/3 for 138.0.13.76 - 147.0.0.0 PR ipv6-icmp len 0 (20556) IN			
021	S1/FW1	% winnuke	
04/11/2000 08:18:12.843355 le0 @0:2 b 192.168.168.8,34586 -> 192.168.168.252,139 PR tcp len 20 60 -S 145844525 0 5840 IN 04/11/2000 08:18:15.840448 le0 @0:2 b 192.168.168.8,34586 -> 192.168.168.252,139 PR tcp len 20 60 -S 145844525 0 5840 IN			

APPENDIX B

INITIAL ADN TESTBED NMAP SCAN

The following tables contain output from NMAP, a network scanning tool. It was used to characterize our initial testbed environment configuration.

NMAP BASE SCAN OF ADN TESTBED		12-Dec-2000	COMMAND: **nmap –v –O –sS '10.0.0-1.*' '20.0.-1.*' '30.0.-1.*' '40.0.0-1.*' '50.0.0-1.*'**		
10.x.x.x	**10.0.0.254**	**10.0.1.254**	**20.0.0.1**	**20.0.0.254**	**20.0.1.254**
Starting nmap V. 2.54BETA7 (www.insecure.org/nmap/) Host (10.0.0.0) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (10.0.0.1) appears to be up ... good. Initiating SYN Stealth Scan against (10.0.0.1)	Host (10.0.0.254) appears to be up ... good. Initiating SYN Stealth Scan against (10.0.0.254)	Host (10.0.0.255) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (10.0.1.0) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (10.0.1.254) appears to be up ... good. Initiating SYN Stealth Scan against (10.0.1.254)	Host (10.0.1.255) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (20.0.0.0) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (20.0.0.1) appears to be up ... good. Initiating SYN Stealth Scan against (20.0.0.1)	Host (20.0.0.254) appears to be up ... good. Initiating SYN Stealth Scan against (20.0.0.254)	Host (20.0.0.255) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (20.0.1.0) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (20.0.1.254) appears to be up ... good. Initiating SYN Stealth Scan against (20.0.1.254)
7/tcp open echo	7/tcp open echo	7/tcp open echo			
9/tcp open discard	9/tcp open discard	9/tcp open discard	7/tcp open echo	7/tcp open echo	7/tcp open echo
13/tcp open daytime	13/tcp open daytime	13/tcp open daytime	9/tcp open discard	9/tcp open discard	9/tcp open discard
19/tcp open chargen	19/tcp open chargen	19/tcp open chargen	13/tcp open daytime	13/tcp open daytime	13/tcp open daytime
21/tcp open ftp	21/tcp open ftp	21/tcp open ftp	19/tcp open chargen	19/tcp open chargen	19/tcp open chargen
22/tcp open ssh		22/tcp open ssh	21/tcp open ftp	21/tcp open ftp	21/tcp open ftp
23/tcp open telnet	23/tcp open telnet	23/tcp open telnet	23/tcp open telnet	23/tcp open telnet	23/tcp open telnet
25/tcp open smtp	25/tcp open smtp	25/tcp open smtp	25/tcp open smtp	25/tcp open smtp	25/tcp open smtp
37/tcp open time	37/tcp open time	37/tcp open time	37/tcp open time	37/tcp open time	37/tcp open time
79/tcp open finger	79/tcp open finger	79/tcp open finger	79/tcp open finger	79/tcp open finger	79/tcp open finger
111/tcp open sunrpc	111/tcp open sunrpc	111/tcp open sunrpc	111/tcp open sunrpc	111/tcp open sunrpc	111/tcp open sunrpc
512/tcp open exec	512/tcp open exec	512/tcp open exec	512/tcp open exec	512/tcp open exec	512/tcp open exec
513/tcp open login	513/tcp open login	513/tcp open login	513/tcp open login	513/tcp open login	513/tcp open login
514/tcp open shell	514/tcp open shell	514/tcp open shell	514/tcp open shell	514/tcp open shell	514/tcp open shell
515/tcp open printer	515/tcp open printer	515/tcp open printer	515/tcp open printer	515/tcp open printer	515/tcp open printer
540/tcp open uucp	540/tcp open uucp	540/tcp open uucp	540/tcp open uucp	540/tcp open uucp	540/tcp open uucp
2049/tcp open nfs		2049/tcp open nfs	4045/tcp open lockd	4045/tcp open lockd	4045/tcp open lockd
4045/tcp open lockd	4045/tcp open lockd	4045/tcp open lockd	6000/tcp open X11	6000/tcp open X11	6000/tcp open X11
6000/tcp open X11	6000/tcp open X11	6000/tcp open X11	6112/tcp open dtspc	6112/tcp open dtspc	6112/tcp open dtspc
6112/tcp open dtspc	6112/tcp open dtspc	6112/tcp open dtspc	7100/tcp open font-service	7100/tcp open font-service	7100/tcp open font-service
7100/tcp open font-service	7100/tcp open font-service	7100/tcp open font-service	32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5

32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7
32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9
32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11
32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13
32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15
32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	32777/tcp open sometimes-rpc17	32777/tcp open sometimes-rpc17	32777/tcp open sometimes-rpc17
32777/tcp open sometimes-rpc17	32777/tcp open sometimes-rpc17	32777/tcp open sometimes-rpc17	32778/tcp open sometimes-rpc19	32778/tcp open sometimes-rpc19	32778/tcp open sometimes-rpc19
32778/tcp open sometimes-rpc19	32778/tcp open sometimes-rpc19	32778/tcp open sometimes-rpc19			
32779/tcp open sometimes-rpc21		32779/tcp open sometimes-rpc21			
	32786/tcp open sometimes-rpc25			32786/tcp open sometimes-rpc25	
The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled. Interesting ports on (10.0.0.1): (The 1504 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=42629 (Worthy challenge) Sequence numbers: C17469DC C1748DCA C1765B47 C1779B3A C1798010 C17B3A41 Remote operating system guess: Solaris 2.6 - 2.7	For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (10.0.0.254): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=23857 (Worthy challenge) Sequence numbers: B63F4698 B63FA5E8 B640A977 B6421AFD B6435AA1 B64440FC Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (10.0.1.254): (The 1504 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=25506 (Worthy challenge) Sequence numbers: C1D82675 C1D93585 C1D95336 C1D9EE1F C1DA16CB C1DB0946 Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (20.0.0.1): (The 1507 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=23256 (Worthy challenge) Sequence numbers: EA52A704 EA53E4C9 EA54BB16 EA563B23 EA57C371 EA5868B2 Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 5 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (20.0.0.254): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=37665 (Worthy challenge) Sequence numbers: B6A41B4A B6A46E19 B6A63307 B6A66DCC B6A6BDCB B6A75A5D Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (20.0.1.254): (The 1507 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=45395 (Worthy challenge) Sequence numbers: EA9FBE52 EAA04445 EAA06BBE EAA177E2 EAA342DE EAA53108 Remote operating system guess: Solaris 2.6 - 2.7

30.0.0.1	30.0.0.254	30.0.1.254	40.0.0.1	40.0.0.254	40.0.1.254
Host (20.0.1.255) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (30.0.0.0) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (30.0.0.1) appears to be up ... good. Initiating SYN Stealth Scan against (30.0.0.1)	Host (30.0.0.254) appears to be up ... good. Initiating SYN Stealth Scan against (30.0.0.254)	Host (30.0.0.255) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (30.0.1.0) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (30.0.1.254) appears to be up ... good. Initiating SYN Stealth Scan against (30.0.1.254)	Host (30.0.1.255) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (40.0.0.0) seems to be a subnet broadcast address (returned 2 extra pings). Skipping host. Host (40.0.0.1) appears to be up ... good. Initiating SYN Stealth Scan against (40.0.0.1)	Host (40.0.0.254) appears to be up ... good. Initiating SYN Stealth Scan against (40.0.0.254)	NOT CONFIGURED FOR THIS TEST
7/tcp open echo	7/tcp open echo	7/tcp open echo			
9/tcp open discard	9/tcp open discard	9/tcp open discard			
13/tcp open daytime	13/tcp open daytime	13/tcp open daytime	7/tcp open echo	7/tcp open echo	
19/tcp open chargen	19/tcp open chargen	19/tcp open chargen	9/tcp open discard	9/tcp open discard	
21/tcp open ftp	21/tcp open ftp	21/tcp open ftp	13/tcp open daytime	13/tcp open daytime	
23/tcp open telnet	23/tcp open telnet	23/tcp open telnet	19/tcp open chargen	19/tcp open chargen	
25/tcp open smtp	25/tcp open smtp	25/tcp open smtp	21/tcp open ftp	21/tcp open ftp	
37/tcp open time	37/tcp open time	37/tcp open time	22/tcp open ssh		
79/tcp open finger	79/tcp open finger	79/tcp open finger	23/tcp open telnet	23/tcp open telnet	
111/tcp open sunrpc	111/tcp open sunrpc	111/tcp open sunrpc	25/tcp open smtp	25/tcp open smtp	
512/tcp open exec	512/tcp open exec	512/tcp open exec	37/tcp open time	37/tcp open time	
513/tcp open login	513/tcp open login	513/tcp open login	79/tcp open finger	79/tcp open finger	
514/tcp open shell	514/tcp open shell	514/tcp open shell	111/tcp open sunrpc	111/tcp open sunrpc	
515/tcp open printer	515/tcp open printer	515/tcp open printer	512/tcp open exec	512/tcp open exec	
540/tcp open uucp	540/tcp open uucp	540/tcp open uucp	513/tcp open login	513/tcp open login	
4045/tcp open lockd	4045/tcp open lockd	4045/tcp open lockd	514/tcp open shell	514/tcp open shell	
6000/tcp open X11	6000/tcp open X11	6000/tcp open X11	515/tcp open printer	515/tcp open printer	
6112/tcp open dtspc	6112/tcp open dtspc	6112/tcp open dtspc	540/tcp open uucp	540/tcp open uucp	
7100/tcp open font-service	7100/tcp open font-service	7100/tcp open font-service	4045/tcp open lockd	4045/tcp open lockd	
32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	6000/tcp open X11	6000/tcp open X11	
32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	6112/tcp open dtspc	6112/tcp open dtspc	
32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	7100/tcp open font-service	7100/tcp open font-service	
32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	32771/tcp open sometimes-rpc5	32771/tcp open sometimes-rpc5	
32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	32772/tcp open sometimes-rpc7	32772/tcp open sometimes-rpc7	
32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	32773/tcp open sometimes-rpc9	32773/tcp open sometimes-rpc9	
	32777/tcp open sometimes-rpc17		32774/tcp open sometimes-rpc11	32774/tcp open sometimes-rpc11	
	32778/tcp open sometimes-rpc19		32775/tcp open sometimes-rpc13	32775/tcp open sometimes-rpc13	
	32786/tcp open sometimes-rpc25		32776/tcp open sometimes-rpc15	32776/tcp open sometimes-rpc15	

			32777/tcp open sometimes-rpc17	32777/tcp open sometimes-rpc17	
			32778/tcp open sometimes-rpc19	32778/tcp open sometimes-rpc19	
				32786/tcp open sometimes-rpc25	
The SYN Stealth Scan took 5 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (30.0.0.1): (The 1509 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=39607 (Worthy challenge) Sequence numbers: A46B8481 A46D8CB5 A46E73D0 A46EA1A5 A46FDCC7 A470B056 Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (30.0.0.254): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=32152 (Worthy challenge) Sequence numbers: B70631F2 B7066313 B7069129 B70769B9 B708EBE9 B7099075 Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (30.0.1.254): (The 1509 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=33112 (Worthy challenge) Sequence numbers: A4B015A7 A4B1841A A4B1994E A4B1E3D1 A4B2125D A4B23BB8 Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 3 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (40.0.0.1): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=41552 (Worthy challenge) Sequence numbers: A28079DF A280FF0C A2814DA8 A281652C A2834AD0 A283D5FB Remote operating system guess: Solaris 2.6 - 2.7	The SYN Stealth Scan took 4 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (40.0.0.254): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=26964 (Worthy challenge) Sequence numbers: B75B7A7F B75C96F3 B75D1C4F B75DE425 B75E0D71 B75F5E3C Remote operating system guess: Solaris 2.6 - 2.7	

50.0.0.1	50.0.0.254
Host (40.0.0.255) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (50.0.0.0) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host. Host (50.0.0.1) appears to be up ... good. Initiating SYN Stealth Scan against (50.0.0.1)	Host (50.0.0.254) appears to be up ... good. Initiating SYN Stealth Scan against (50.0.0.254)
21/tcp open ftp	7/tcp open echo
23/tcp open telnet	9/tcp open discard
25/tcp open smtp	13/tcp open daytime
79/tcp open finger	19/tcp open chargen
98/tcp open linuxconf	21/tcp open ftp
111/tcp open sunrpc	23/tcp open telnet
113/tcp open auth	25/tcp open smtp
513/tcp open login	37/tcp open time
514/tcp open shell	79/tcp open finger
515/tcp open printer	111/tcp open sunrpc
959/tcp open unknown	512/tcp open exec
1024/tcp open kdm	513/tcp open login
1025/tcp open listen	514/tcp open shell
1031/tcp open iad2	515/tcp open printer
6000/tcp open X11	540/tcp open uucp
	4045/tcp open lockd
	6000/tcp open X11
	6112/tcp open dtspc
	7100/tcp open font-service
	32771/tcp open sometimes-rpc5
	32772/tcp open sometimes-rpc7

	32773/tcp open sometimes-rpc9
	32774/tcp open sometimes-rpc11
	32775/tcp open sometimes-rpc13
	32776/tcp open sometimes-rpc15
	32777/tcp open sometimes-rpc17
	32778/tcp open sometimes-rpc19
	32786/tcp open sometimes-rpc25
The SYN Stealth Scan took 0 seconds to scan 1534 ports. For OSScan assuming that port 21 is open and port 1 is closed and neither are firewalled Interesting ports on (50.0.0.1): (The 1519 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=5852726 (Good luck!) Sequence numbers: 72044F1D 72044F1D 72E3D46E 72E3D46E 72751B3E 72751B3E Remote operating system guess: Linux 2.1.122 - 2.2.16	The SYN Stealth Scan took 3 seconds to scan 1534 ports. For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled Interesting ports on (50.0.0.254): (The 1506 ports scanned but not shown below are in state: closed) TCP Sequence Prediction: Class=random positive increments Difficulty=35177 (Worthy challenge) Sequence numbers: B792D87F B794C11A B796A5EA B7976108 B7983AC9 B799134E Remote operating system guess: Solaris 2.6 - 2.7
FINAL OUTPUT	
Host (50.0.0.255) seems to be a subnet broadcast address (returned 1 extra pings). Skipping host.	
Nmap run completed -- 2560 IP addresses (13 hosts up) scanned in 143 seconds	