Global Information Assurance Certification Paper

GIAC

Incident Handling and Hacker Exploits

Certification Practical

Version 1.5c

Tamara Bowman

April 2001

Exploit Details

Name: UDP Flood Denial of Service
Variants : fraggle, pepsi , UDP charge (spj), Rhythem (spelling not mine) Collision UDP flooder, arnUDP
Operating System: Windows and Unix
Protocols/services: chargen(19) and echo(7)
Description: The UDP flood consists of crafting a packet with a source port of echo for target host A and a destination port of chargen or echo for target host B. This creates an endless loop where host A will echo a packet to host B. Host B then responds to host A's echo port. This continues until system or network resources are exhausted.

Protocol Description

The chargen protocol is described in RFC 864. The chargen protocol listens on port 19. The protocol responds to connections by sending a packet containing a random number of characters between 0 and 512. Any data in the received packet is discarded. Chargen will continue responding as long as packets are received on port 19.

The echo protocol is described in RFC 862. The echo protocol listens on port 7. The protocol echoes back whatever characters are sent to it. It will continue to echo responses as long as packets are received on port 7.

These protocols were all submitted in May 1983. Both of the protocols were described in their respective RFCs as "A useful debugging and measurement tool". Unfortunately both protocols share the same weakness that also make them useful in a DoS attack. They respond to received packets regardless of the data in the packet. Two other protocols daytime and qotd share this characteristic and could also be abused in this fashion. These protocols are among the "small services". Services with low port numbers that are rarely used.

The TCP versions of these protocols will not work in these attacks. The forged packets would not negotiate a three-way handshake properly so the connection would ultimately be reset. Since, TCP only responds to directed queries it also could not be used to amplify attacks.

Description of Variants

The original UDP flood DoS as reported in the CERT advisory http://www.cert.org/advisories/CA-1996-01.html involved chargen and echo. There is also a CVE entry CVE-1999-0103 - Echo and chargen, or other combinations of UDP services, can be used in tandem to flood the server, a.k.a.UDP bomb or UDP packet storm.

There aren't significant variants to the original flood. The attack can be distributed, using networks that allow broadcasts as amplifiers. The attack can use different UDP ports to initiate the attack. The fraggle attack is a variation on this theme. Fraggle is a smurf variant that relies on the echo port rather than ICMP.

How the Exploit works

The UDP flood exploit depends on a number of vulnerabilities in the network, the UDP transport and the implementation of chargen and echo. We will first examine the simplest attack, a non-distributed flood.

In this attack the NPC (networked problem child) uses one of the available tools to craft a packet with a forged source and destination address. For this scenario the source and destination are on two networks so there are two victims. The forged packet has a source of echo and a destination of chargen. This packet is now released on the network. The packet must now make its way through the following hurdles.

The router and firewall of the destination victim must permit packets to chargen. The packet has found its first vulnerability, inadequate filtering at the border. While echo and chargen were used in the past for debugging and primitive performance monitoring, they are rarely used for that now. There are many more robust tools to handle performance monitoring that won't open your network to this type of DoS attack.

The packet continues on to the host. For the attack to continue the destination victim must have chargen available as a service, and it does. The packet gets to exercise the second vulnerability, failure to disable unused services. It is common for default installations to have all services in inetd.conf enabled. Many of the services supported by inetd are not used. All of the unused services should be disabled.

The packet now goes on to exercise vulnerabilities in the UDP transport and the chargen implementation. UDP is a connectionless transport mechanism. It has considerably less overhead than TCP. UDP depends on the protocols or applications using it to supply authentication and verify communication integrity. The only verification UDP offers is a header checksum. TCP handles the communication integrity in part with the three-way handshake. If this were a TCP based communication the packet would be dropped because no channel had been established between the source and destination. But this is a UDP packet so no connection verification is required.

The forged packet is passed by UDP to chargen. Chargen does not verify the data being received. In fact it discards whatever data is received. It does not authenticate the communication. There is nothing within the protocol that requires a login or authentication of the requesting host. The only thing required of chargen is to generate some random characters and send them back to the apparent source.

The reply to the forged packet is now on its way to the unsuspecting source. The forged source also has failed to secure their border and shut down unused services. The packet arrives at the echo port with its payload of random characters. Echo operates in much the same fashion as chargen. The data isn't checked and there is no attempt to authenticate the communication with the other host. Echo responds to the packet by echoing the random characters back to chargen.

The NPC has now made a successful attack. The two hosts will continue responding to each other as fast as their CPUs and the network will allow until they impact system resources and

someone attempts to figure out why the network or host is slow. The attacker can accelerate the DoS by releasing multiple forged packets on the network.

A variation on this attack would be to have the source and destination address be on the same network. This would allow the NPC to target a single victim with the same attack. In order for this attack to work the intended victim must permit packets with a source of their own network to pass through the border router. If the intended victim had taken measures against IP spoofing the attack would not work.

The distributed attack takes advantage of weaknesses in the victims network as well as in numerous networks used as amplifiers. In this attack our NPC crafts a packet with a source of the victim's host and a destination of a broadcast address to use as an amplifier. The source victim port is echo. There are databases of potential amplifiers or the NPC can discover amplifiers by using a port scanning tool. The site being used as an amplifier must allow UDP broadcast traffic through their border router. They don't necessarily need to have echo or chargen available. The ICMP unreachable responses flooding back to the victim source will have the same impact as getting a response from chargen or echo. This takes advantage of a second vulnerability in the UDP transport, broadcasts. Since UDP is connectionless it responds to undirected broadcasts. This allows for amplification of the UDP flood attacks. Allowing an attacker to more efficiently take advantage of the underlying weaknesses in UDP and the chargen and echo protocols.

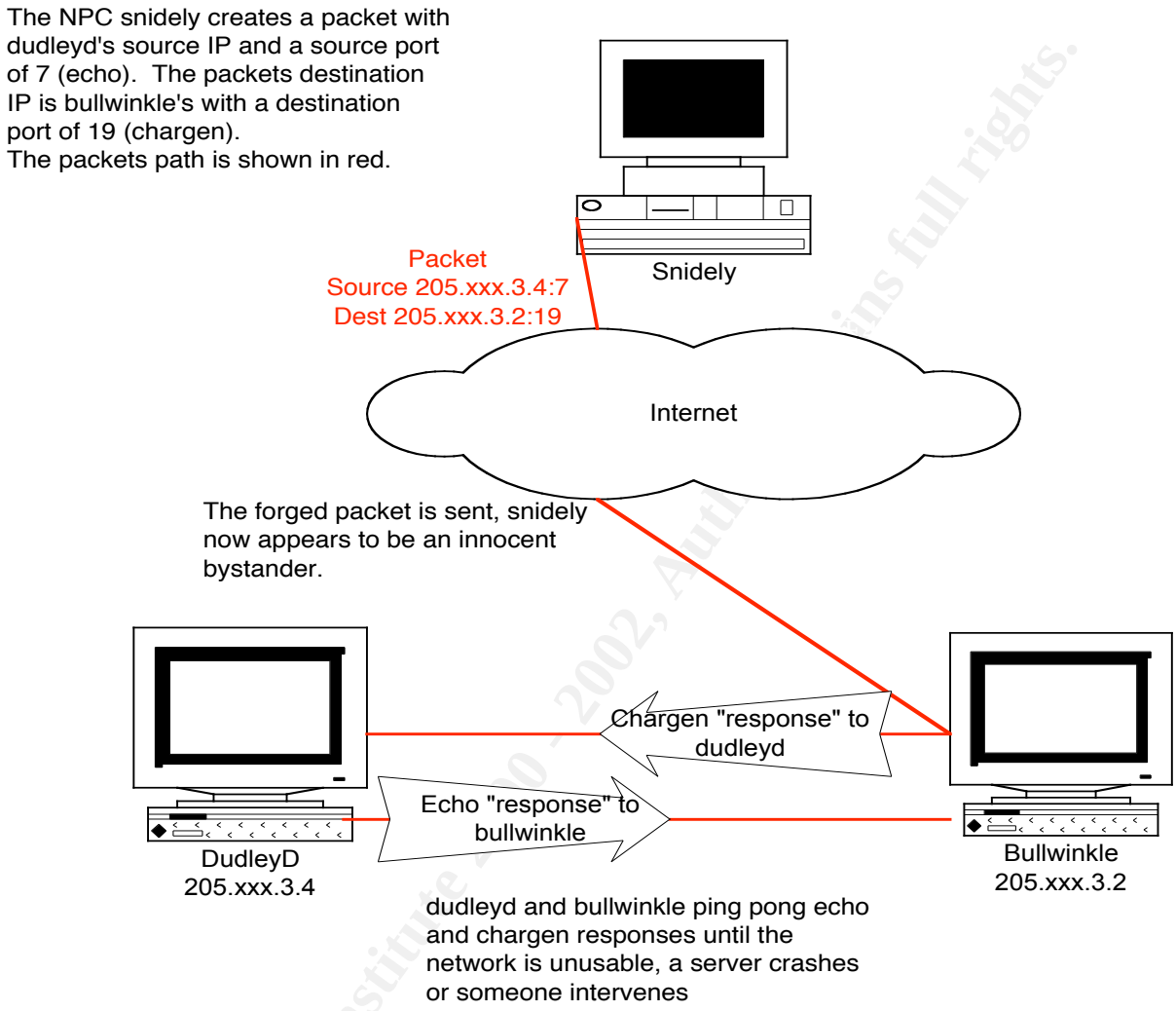
An example of what a Fraggle attack looks like when your site is being used as the amplifier is available at http://www.robertgraham.com/pubs/firewall-seen.html#5.4.1. The identifying features are packets received at network broadcast addresses with a destination port of echo.

The vulnerabilities exercised in this attack fall into two categories, configuration weaknesses and protocol weaknesses. Configuration weaknesses are vulnerabilities that can be fixed or mitigated by making configuration changes. Vulnerabilities in this category are generally easily rectified. To protect against spoofed packets apply filters to the border router following best practices. Implement the appropriate deny rules in the firewall to drop connections to chargen and echo. If you don't have a firewall these protocols can be denied at the border router. When applying filters to the router be sure and apply egress filters as well to avoid having spoofed packets generated from your site. To avoid being used as an amplifier, turn off broadcasts at the border router. On servers eliminate services that aren't being used.

Protocol weaknesses are not as easily rectified. They generally require significant effort to rework the protocol. The effort to fix the security weaknesses in IPv4 has been ongoing for several years and IPv6 is still in the development phase. With this category your options are more limited. Use an application with features that counter the weaknesses in the protocol or use an application based on a stronger protocol. If you must use the existing application, control access by adding a layer of authentication using something like tcpwrappers or securelib or if the service needs to be accessible via the Internet use a proxy..

Vendors cannot fix the problems with chargen and echo but they could change their default install values to have those features disabled rather than activated by default.

Diagram of a UDP Flood



The diagram above shows the original UDP flood vulnerability. The NPC snidely crafts a packet and releases it on the Internet. The unsuspecting victims respond to the packet and ping-pong responses until something breaks or the problem is noticed.

While you might notice your host was sluggish there wouldn't be any error messages to indicate a problem. Executing netstat on one of the hosts involved might show you the chargen or echo port was active rather than idle. But the command would need to be executed right at the time a packet was being processed by echo or chargen. In the absence of an IDS the only clues would be on the network. A tcpdump of this attack, limiting the output to these two hosts, shows:

```
11:36:14.235711 192.168.4.96.7 > 192.168.4.67.19:  UDP 10 (DF)
11:36:14.236294 192.168.4.67.19 > 192.168.4.96.7:  UDP 74
11:36:14.236321 192.168.4.96.7 > 192.168.4.67.19:  UDP 74
11:36:14.236344 192.168.4.67.19 > 192.168.4.96.7:  UDP 74
11:36:14.236368 192.168.4.96.7 > 192.168.4.67.19:  UDP 74
```

```
11:36:14.236391 192.168.4.67.19 > 192.168.4.96.7:  UDP 74
11:36:14.236469 192.168.4.96.7 > 192.168.4.67.19:  UDP 74
11:36:14.236496 192.168.4.67.19 > 192.168.4.96.7:  UDP 74
11:36:14.236519 192.168.4.96.7 > 192.168.4.67.19:  UDP 74
```

The tcpdump output shows that in the space of a few seconds 9 packets are ping ponged between the target hosts. The tcpdump output has the following fields timestamp, source host and port, destination host and port, identifier that the communication is UDP and the number of bytes of user data. The very first packet also as the do not fragment flag set.

The following output is from tcpdump with hex and ascii of the packet. This is one of the responses from chargen. The ascii output shows the characters being sent back.

```
08:13:59.735363 192.168.4.58.19 > 192.168.4.174.7:  udp 74
0x0000   4500 0066 f25a 0000 1e11 1ff4 c0a8 043a      E..f.Z.........:
0x0010   c0a8 04ae 0013 0007 0052 61c3 2425 2627      .........Ra.$%&'
0x0020   2829 2a2b 2c2d 2e2f 3031 3233 3435 3637      ()*+,-./01234567
0x0030   3839 3a3b 3c3d                              89:;<=
```

Without an IDS to alert on packets destined to port 7 or port 19 this traffic wouldn't be noticed until it impacted network or server performance.

The same attack can be made using the broadcast address. This would amplify the impact on the network but still wouldn't be apparent early in the attack without an IDS. The tcpdump of the broadcast attack on a small network shows:

```
11:33:07.013217 192.168.4.96.7 > 192.168.4.255.19:  UDP 10 (DF) [ttl 1]
11:33:07.014305 192.168.4.108.19 > 192.168.4.96.7:  UDP 74
11:33:07.014543 192.168.4.67.19 > 192.168.4.96.7:  UDP 74
11:33:07.014651 192.168.4.122.19 > 192.168.4.96.7:  UDP 74
11:33:07.015573 192.168.4.90.19 > 192.168.4.96.7:  UDP 74
11:33:07.021666 192.168.4.63.19 > 192.168.4.96.7:  UDP 74
11:33:07.022595 192.168.4.52.19 > 192.168.4.96.7:  UDP 74
11:33:07.024449 192.168.4.58.19 > 192.168.4.96.7:  UDP 74
11:33:07.026729 192.168.4.77.19 > 192.168.4.96.7:  UDP 74
11:33:07.029037 192.168.4.59.19 > 192.168.4.96.7:  UDP 74
11:33:07.030734 192.168.4.45.19 > 192.168.4.96.7:  UDP 74
11:33:07.038487 192.168.4.193.19 > 192.168.4.96.7:  UDP 74
```

This tcpdump output shows the initial packet sent with a source address of 192.168.4.96 and source port of 7 to the broadcast address 192.168.4.255 and destination port of 19. Eleven hosts on the network respond back to port 7 on host 192.168.4.96.

This illustrates how easily the attack can be amplified using the broadcast address. Using this same method with the spoofed broadcast address of an Internet network that responds to directed broadcasts could result in hundreds of responses to a single forged packet. Sending multiple forged packets would just accelerate the impact.

How to use the exploit

There are numerous tools available to exploit this vulnerability. Since the vulnerability requires forging an ip packet header there isn't a manual exploit. In the absence of available code you would need to write your own code to create the forged packet.

In order for any of these tools to work the attacker must have root (or administrator if using Windows) access. A standard user cannot forge ip headers. A Windows user would also need additional network libraries to support creating IP headers.

The tools that are available for UDP flood attacks share one common characteristic. They are very user friendly. At least if you don't take some of the error messages personally. All of the tools give usage syntax if they are issued with no arguments or with incorrect arguments. Most have rudimentary checking to make sure the command line arguments given are valid. I am not a C programmer so my debugging skills with C programs are limited, however, I was able to compile two of these programs with minimal effort. Generally, I was not able to resolve library issues on the programs I was unable to compile. Overall, the lack of expertise required to build and the ease of use with these tools was frightening.

The pepsi tool crafts a packet with a source of echo for the first target host and a destination of chargen on the second host. They continue to ping-pong data until resources are exhausted. The source and destination ports can be overridden on the command line. The pepsi tool quite possible offers the greatest flexibility for crafting udp packets. In addition to specifying source and destination hosts and ports it also takes several other command line arguments. Flags are available to override defaults for packet size, time between packets and number of packets generated.

Pepsi was included in Trin00 as part of the rape utility. There is also a DOS and Windows variant. I was not able to compile this code in my environment. The program takes a minimum of destination IP address as input. It appears that if no source host is given one is chosen at random.

The Rhythem (sp?) Collision code is a variation on pepsi without the flexibility. It also uses the echo and chargen ports but only allows the user to specify source and destination host and the number of forged packets to send. I was able to compile Rhythem (sp?) Collision. The executable was compiled as rc8. Executed without arguments the command will output the syntax.

```
# ./rc8
Rythem Collision [v0.8] -- Coded, Nso
usage: rc <from host> <to host> <how many>
```

Once you have chosen your victim(s) execute the command with the source and destination host and the number of packets you wish to send. The command gives positive feedback that the packets are on their way.

```
#./rc8 192.168.4.174 192.168.4.96 10
Rythem Collision [v0.8] -- Coded, Nso
```

```
IP_HDRINCL: found!
..........
number of packets sent: 10
```

Fraggle is a rewrite of smurf. Smurf used the ICMP echo/echo-reply instead of the UDP echo. The fraggle variant crafts a packet with a source and destination port of echo. Like smurf the fraggle attack uses broadcast destination addresses to amplify the attack. There is a fraggle fragment available from http://www.attrition.org/security/denial/w/fraggle.dos.html. From the comments it would take a target and broadcast amplifier as input.

UDP charge (spj) has comments in the code that it is a variant of fraggle but it seems to be mutation of fraggle and pepsi. UDP charge uses the echo and chargen ports, like pepsi but uses broadcast addresses to distribute the attack like fraggle. Thus, depending on your outlook, getting the best or worst of both worlds. I could not compile this program in my environment. The program does take a target and broadcast amplifier as arguments.

The arnupd code allows the user to set a source and destination port as well as the IP address. The ports are set as UDP. It can effectively be used as a tool for any of the UDP floods already mentioned. This tool was also included in Trin00 as part of the rape utility. This program did compile in my environment. When executed without arguments arnupd gives the appropriate syntax.

```
# ./arnupd
usage: ./arnupd sourcename sourceport destinationname destinationport
```

After executing arnupd with source IP and port and destination IP and port the program creates the packet, sends it on and lets you know its on its way.

```
# ./arnupd 192.168.4.96 7 192.168.4.67 19
we have IP_HDRINCL :-)

datagram sent without error:
```

The arnupd program offers more flexibility than most of the other programs. Without modifying the source code arnupd can be used to launch attacks using echo, chargen or any other combination of UDP ports.

Why are these programs checking for IP_HDRINCL? IP_HDRINCL is the socket interface to the raw IP layer. This allows the program to create and prepend IP headers without the intervening transport layer. IP_HDRINCL allows the programs to create their own IP header.

This socket does have legitimate uses. The traceroute program uses the same feature to build its datagrams. The arnupd code contains comments that it was modeled on the code for traceroute. IP_HDRINCL also supplies a method for an application to create IP headers when a protocol is not yet supported in the kernel. Once the protocol matures and stabilizes it should be integrated with the kernel. In the Solaris implementation of IPv6 there is not an analog to IP_HDRINCL.

Signature of the Attack

The UDP flood attack is virtually invisible if you aren't running an intrusion detection system or sniffing the target network at the time of the attack.

The attacks generated no log messages on the Solaris hosts in my test group. The syslog configuration on the test hosts logged all kern facility messages at debug and above, all daemon facility messages at notice and above and all other facilities at err and above. No error messages were generated at the hosts. The only visible negative impact on the host being attacked was the load tripled from 1 to 3.2. This isn't useful for determining that the host is a victim of UDP flood since a higher than average load on a host can point to any number of potential problems. The output from netstat could not be relied on to give any information about the attack. Netstat would only show the echo or chargen port in use if you happened to enter the command at just the right time. It could be possible to see sustained activity if the attack continued, but at that point the response from the host would likely be hampered.

On the network the attack was more visible. Snort was used for intrusion detection. Two snort rules were created to log potential fraggle or UDP flood attempts. In the first rule all UDP packets from any source with a source port of 7 and a destination within our network of port 7 are logged with the message Possible Fraggle.

In the second rule all UDP packets from any source with a source of 7 and a destination within our network of port 19 are logged with the message Possible UDP Flood. These can also be alerts, just change the log keyword to alert.

```
log UDP any 7 -> $HOME_NET 7 (msg: "Possible Fraggle";)
log UDP any 7 -> $HOME_NET 19 (msg: "Possible UDP Flood";)
```

Snort log output

```
[**] Possible Fraggle [**]
05/01-13:16:14.956240 192.168.4.67:7 -> 192.168.4.96:7
UDP TTL:255 TOS:0x0 ID:23015  DF
Len: 18
```

Snort logged several hundred of these errors, one is sufficient for an example. In the log message snort provides the message text provided. The message text is optional but if you have a monitoring system that can be configured to act upon keywords the message is very useful. Snort gives the date and time the packet was received, source and destination address, protocol type, ttl, packet ID, fragment info and length. Ordinarily, this information would be very useful in determining where the packet originated. If this were a real fraggle attack the source would be a legitimate address being used as an amplifier.

How to Protect Against it

For this attack you need to protect against becoming a victim or a potential amplifier. To avoid becoming a victim, first control your borders. Create incoming (ingress) filters for your router. Certain networks should never appear as the source IP in packets coming into your border router. Networks to filter would include your own network, any of the private networks defined in RFC 1918 and any other networks reserved by IANA.

Before applying the filter to your router be sure you understand the full impact on your network. On a Cisco router filters are evaluated in the order written, first match wins. All filters also have an implicit deny at the end of the filter. So if you apply a filter denying access from packets with a source of your network or the private networks the filter must contain an explicit allow statement at the end. This will allow all the denied networks to be dropped while still permitting legitimate traffic. Without the explicit allow you'll cut off all communication to the net. An example standard ingress access list:

```
access-list 99 deny   ip 127.0.0.0 0.255.255.255 log
access-list 99 deny   ip 224.0.0.0 31.255.255.255 log
access-list 99 deny   ip MY.NET.58.0 0.0.0.255 log
access-list 99 deny   ip MY.NET.149.0 0.0.0.255 log
access-list 99 deny   ip 169.254.0.0 0.0.255.255 log
access-list 99 deny   ip 192.168.0.0 0.0.255.255 log
access-list 99 deny   ip 172.16.0.0 0.15.255.255 log
access-list 99 deny   ip 10.0.0.0 0.255.255.255 log
access-list 99 permit ip any
```

The above access-list is a standard access list for a Cisco router. The fields in a standard access list are the command access-list, a number between 1 – 99 identifying access-list number, action to take (deny or permit), protocol impacted (ip, tcp , udp, icmp), network, wildcard mask and log keyword. The Cisco wildcard mask is not equivalent to a netmask. It is essentially a mirror image of the netmask that identifies the significant bits in an address. The log keyword in the access list statement is optional. In this example all denied traffic is logged. The explicit deny statement is added to capture log data. If the statement is omitted the traffic will still be denied but there will be no log record. This list could be improved by using an extended access list. The extended access list would allow a more precise permit statement that would only allow packets with a destination of MY.net.

If you do not have a firewall consider filtering out UDP services between 1 – 19. Be aware of the current load on your router when considering adding this filter. The ingress filter for networks can be handled with a standard filter. All the filter examines is the source IP address. To implement a filter to drop UDP traffic with destination ports between 1 – 19 requires an extended access list. An extended access list requires the router to look deeper into the header at destination IP and port. This increases the load on the router and depending on the current load may negatively impact network performance. The following is an example of an extended access list.

```
access-list 101 deny   ip 127.0.0.0 0.255.255.255 any  log
access-list 101 deny   ip 224.0.0.0 31.255.255.255 any log
access-list 101 deny   ip MY.NET.58.0 0.0.0.255 any log
access-list 101 deny   ip MY.NET.149.0 0.0.0.255 any log
access-list 101 deny   ip 169.254.0.0 0.0.255.255 any log
access-list 101 deny   ip 192.168.0.0 0.0.255.255 any log
access-list 101 deny   ip 172.16.0.0 0.15.255.255 any log
access-list 10 deny   ip 10.0.0.0 0.255.255.255 any log
access-list 101 deny   udp any any lt 20 log
access-list 101 permit ip any MY.NET.58.0 0.0.0.255
access-list 101 deny   ip any any log
```

These commands accomplish the same thing as the standard access list with some additional benefits. The deny statement in bold denies any udp packets with a port number less than 20. This encompasses ports 1 – 19. The lt is a keyword for less than. The permit statement following this statement can be more specific and allow only traffic with a destination of our network. The final deny statement logs packets that would fall through to the implicit deny statement.

To avoid being used as an amplifier, apply an outbound (egress) filter to prevent IP spoofing. The egress filter should specifically permit packets with a source address of your network. You have the option of leaving the implicit deny or adding an explicit deny with logging. In this case, its recommended to include the explicit deny statement.

Packets with a different source address than your network are a sign of a problem. Either you've been cracked or NATed addresses are leaking. The implicit deny does not log matches. With an explicit deny statement there's at least a trail to investigate. To get meaningful log information for these packets you need to use an extended access list. The extended access list allows the use of the keyword log-input. This logs not only the message but the MAC of the system generating the message. The MAC address will be your only method of tracking down what host is generating the bogus packets.

To stop broadcast traffic from propagating through your border router turn off ip directed broadcast on each interface. On version 12 of Cisco IOS this is the default. Check your router to confirm that broadcasts are disabled.

If you have a firewall, block inbound access to ports not being used. Also, as a second measure to prevent being an amplifier, block outbound access to chargen, echo and the similar services daytime and qotd. Blocking broadcasts at the router should take care of this, however, defense in depth is about not relying on any one countermeasure. On a PIX firewall it is necessary to put an explicit deny to block these services outbound. The PIXs default is to have all inbound connections denied and all outbound connections permitted. Other firewalls (Raptor and Checkpoint) default with all connections inbound or outbound denied. Use nmap or some other port scanning utility to verify that you're only allowing access to services you intend to allow access to. Regularly schedule an nmap scan to audit how the access rules are changing and to verify your access policy.

On the servers turn off services that aren't being used. In Windows systems go to the Network Control Panel and turn off "Simple TCP Services". This will turn off echo and chargen as well as other services like daytime, discard and qotd.

To disable the services on Unix systems edit the inetd.conf file. Comment out the echo and chargen services and any other unused services such as daytime, discard and qotd. After commenting out the services send a HUP to the inetd daemon so the inetd.conf file will be re-read. Use netstat to check to make sure the changes are active. The services that were commented out should not appear in the netstat output.

On a Solaris server you can also disable forwarding of ip broadcast traffic using the ndd command. On a Solaris system to make the changes survive a reboot edit the /etc/system file. Other Unix versions should have similar configuration options.

If for some reason your site needs to use echo or chargen add some authentication by using tcpwrappers to control access to the services. With tcpwrapper installed you can specify who is permitted to access those services and deny anyone else. There are some limitations with using tcpwrappers with UDP services. One of these limitations is that UDP services frequently wait around to service another request after finishing with an initial request. The tcpwrapper will only see the initial request.

Other options to control access would be securelib if you were running a Sun system. Securelib is a drop in replacement library that replaces the system calls accept, recvfrom, and recvmsg. These calls then reference a configuration file that determines if the source host is permitted to access the services. Another option would be running xinetd in place of inetd. Xinetd is essentially inetd implemented with logging and access control.

Source code/ Pseudo code:

The source code, for the exploits discussed, is available at www.technotronic.com. I was not able to locate the complete source for fraggle. There is a fraggle fragment available at http://www.attrition.org/security/denial/w/fraggle.dos.html

Arnupd source is available at ftp://ftp.technotronic.com/unix/UDP-exploits/UDPdata.c

Arnupd Pseudo code

1. Check for correct syntax, if not correct echo correct syntax
2. Collect command line arguments
3. Attempt to resolve hostnames, err if hostnames cannot be resolved
4. Check for IP_ HDRINCL so we can create the forged packet, err if not present
5. Create UDP IPv 4 packet with source and destination hosts and ports given at command line
6. Echo datagram sent without error

Pepsi source is available at ftp://ftp.technotronic.com/unix/UDP-exploits/pepsi.c. There is also an executable for Windows available as well in the same directory.

Pepsi Pseudo code

1. Check for correct syntax and print usage if not correct
2. Collect command line arguments
3. Options that can be set with this command are:
   - -s <src> : source where packets are coming from
   - -n <num> : number of UDP packets to send
   - -p <size> : Packet Size [Default is 1024]
   - -d <port> : Destination Port [Default is 7]
   - -o <port> : Source Port [Default is 19]

- -w <time> : Wait time between packets [Default is 1]
- <dest> : destination

4. Attempt to resolve hostnames, err if hostnames cannot be resolved
5. Create UDP IPv4 packet using command line arguments or defaults
6. Send crafted packet out.

Rhythem Collision source code is available at ftp://ftp.technotronic.com/unix/UDP-exploits/rc8.c.

Rhythem Collision pseudo code:

1. Check for correct syntax and print usage if not correct
2. Collect command line arguments (source IP and port, destination IP and port and number of packets to generate) Rhythem Collision does not have defaults.
3. Attempt to resolve hostnames, err if hostnames cannot be resolved
4. Check for IP_ HDRINCL so we can create the forged packet, err if not present
5. Create UDP IPv4 packet using command line arguments
6. Echo number of packets sent

UDP Charge source code is available at ftp://ftp.technotronic.com/unix/chargen-exploits/spj-003.c.

UDP Charge pseudo code:

1. Check for correct syntax and print usage if not correct
2. Collect command line arguments. Specify target, ports default to dest 19 and src 7
3. Check to make sure haven't set my host as target, err if my host is target
4. Check for IP_ HDRINCL so we can create the forged packet, err if not present.
5. Create UDP IPv4 packet using command line arguments and defaults.

Additional Information

For information on how UDP works and all the basics
Stevens, W. Richard, 1995. "TCP/IP Illustrated, Volume 1 The Protocols" Addison-Wesley Profession Computing Series

For information on programming with UDP
Stevens, W. Richard; Wright, Gary R., 1995. "TCP/IP Illustrated, Volume 2 The Implementation" Addison-Wesley Profession Computing Series

Resource for exploit code, among other things. All code, with the exception of fraggle, was retrieved from this site
http://www.technotronic.com

Port Knowledge Data Base
http://advice.networkice.com/advice/Exploits/Ports/

Site with information on various exploits and an example of a site being used as a Fraggle amplifier

http://www.robertgraham.com/pubs/firewall-seen.html

Database of DoS contained fraggle code fragment
http://www.attrition.org/security/denial/

Whitepaper on Fraggle & Smurf
http://www.codetalker.com/whitepapers/dos-smurf.html

Analysis of trin00 DDoS tool by Dave Dittrich posted to Bugtraq
http://www.securityfocus.com/archive/1/37706

February 8, 1996 CERT Advisory UDP Denial of Service (chargen and echo)
http://www.cert.org/advisories/CA-1996-01.html

September 19, 1996 CERT Advisory TCP SYN Flooding and IP Spoofing Attacks
http://www.cert.org/advisories/CA-1996-21.html

Last Updated: July 25th, 2017

Upcoming Training

Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Mentor Session AW - SEC504	Milwaukee, WI	Aug 23, 2017 - Sep 29, 2017	Mentor
Mentor Session AW - SEC504	New York, NY	Aug 24, 2017 - Sep 08, 2017	Mentor
Mentor Session - SEC504	Denver, CO	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	SEC504 - 201709,	Sep 05, 2017 - Oct 12, 2017	vLive
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session AW - SEC504	Houston, TX	Oct 02, 2017 - Dec 11, 2017	Mentor
Mentor Session - SEC504	Columbia, SC	Oct 03, 2017 - Nov 14, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS