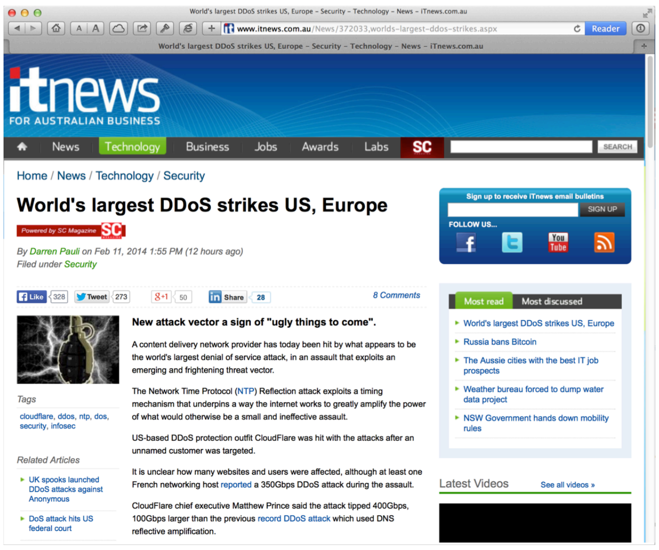
The ISP Column
A monthly column on things Internet

March 2014
Geoff Huston

NTP for Evil

There was a story that was distributed around the newswire services at the start of February this year, reporting that we had just encountered the "biggest DDOS attack ever" (http://rt.com/news/biggest-ddos-us-cloudflare-557/) This dubious distinction was as a result of the observation that this time around the attack volume got to 400Gbps of traffic, some 100Gbps more than the March 2013 DNS-based Spamhaus attack (http://www.spamhaus.org/news/article/695/answers-about-recent-ddos-attack-on-spamhaus).



Home / News / Technology / Security

World's largest DDoS strikes US, Europe

Powered by SC Magazine

By Darren Pauli on Feb 11, 2014 1:55 PM (12 hours ago)
Filed under Security

New attack vector a sign of "ugly things to come".

A content delivery network provider has today been hit by what appears to be the world's largest denial of service attack, in an assault that exploits an emerging and frightening threat vector.

The Network Time Protocol (NTP) Reflection attack exploits a timing mechanism that underpins a way the internet works to greatly amplify the power of what would otherwise be a small and ineffective assault.

US-based DDoS protection outfit CloudFlare was hit with the attacks after an unnamed customer was targeted.

It is unclear how many websites and users were affected, although at least one French networking host reported a 350Gbps DDoS attack during the assault.

CloudFlare chief executive Matthew Prince said the attack tipped 400Gbps, 100Gbps larger than the previous record DDoS attack which used DNS reflective amplification.

Tags
cloudflare, ddos, ntp, dos, security, infosec

Related Articles
- UK spooks launched DDoS attacks against Anonymous
- DoS attack hits US federal court

- World's largest DDoS strikes US, Europe
- Russia bans Bitcoin
- The Aussie cities with the best IT job prospects
- Weather bureau forced to dump water data project
- NSW Government hands down mobility rules

World's largest DDoS strikes US, Europe

What's going on? Why are these supposedly innocuous, and conventionally all but invisible services suddenly turning into venomous daemons? How has the DNS and NTP been turned against us in such a manner? And why have these attacks managed to overwhelm our conventional cyber defences?

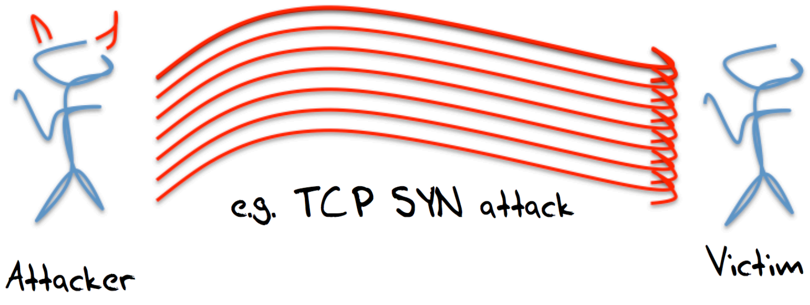
The Evolution of Evil

Attacks on the Internet have changed over time. The initial widely publicised attacks were quite sophisticated. While he was a student at Cornell University in 1988, Robert Morris released a piece of self-replicating code that exploited common coding errors in the versions of *sendmail*, *finger* and *rsh/rexec* that allowed data to be injected in such a way that the victim host's execution stack was corrupted and the victim host CPU started executing the injected data rather than the original code. Coupled with a relatively aggressive code replication algorithm, this worm quickly spread across the Internet at the time, infecting an estimated 6,000 Unix systems (http://en.wikipedia.org/wiki/Morris_worm).

This form of buffer overflow, where an internet server is presented with a large input block that corrupts the in-memory image of the server's code has been exploited in many ways and forms since then. One of the more notable subsequent attacks was the SQL slammer worm of 2003, which was so virulent because of the combination of the worm's small size (376 bytes), and an aggressive replication algorithm. The basic nature of the attack was the much same: a relatively sophisticated attack that was based in a detailed analysis of the server's code (exposed, incidentally, by a Microsoft patch). It was a common reluctance of the installed base to track the latest code releases from the vendor that opened up the vulnerability to this worm (http://en.wikipedia.org/wiki/SQL_Slammer).

Such attacks are perhaps more of an anomaly than the general character of attacks in this period. Internet attacks certainly become more commonplace over the following years, and one general rule is that while they became more commonplace, at the same time they became less sophisticated. The Estonian attack of 2007 was largely an attack by generating overwhelming volumes of unsolicited traffic towards the set of victims, and this attack was based predominately on the use of ping (http://en.wikipedia.org/wiki/2007_cyberattacks_on_Estonia).

It appears to support the more general principle that attackers are no more sophisticated than they need to be. If ping works, then ping will be used. Other attack vectors will be used only when there is general rate limiting on ping, or when an even easier attack vector is promulgated, or there are more effective attack vectors. The problem with the ping attack of 2007 was the need to organise a large set of attackers, each to individually generate hostile traffic, but once the set of attackers was organised, then the attack was easy to execute.

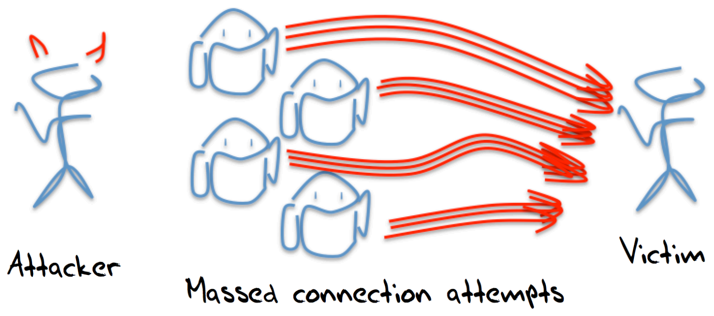


One-on-one attacks

In general terms the weakness of these one-on-one forms of attack is that expose the attacker as much as they inconvenience the victim, and the volume of the attack is limited by the volume that the attacker can harness.

It's a common computer science observation, perhaps even a computer science axiom, that anything can be solved by another level of indirection. In the case of attacks the application of this axiom led to the introduction of the bot army as the means of attack. Rather than using one's own systems to launch the attack, its possible to coopt an army of corrupted systems that are capable of executing a simple script.

These bot systems are fragile: if you overwork them their legitimate owners may notice the anomalous behaviour and clean the host. If you try to use privileged access in the bot's environment you may trigger alarms or simply not be able to obtain that access. So accessing a "raw" interface and hand-crafting attack packets is not so common. More common are bots that can turn on small packet bursts to a particular victim address. Individually this may be unremarkable, and its desirable that this individually unnoticeable, but with enough bots, the collective result can be toxic. This could be as simple as getting the bots to retrieve a given URL, or pinging a particular address. The advantages of this form of attack is that the attacker hides behind the bot control channel, and is not directly involved, and the scale of the attack is based on the potential size of the assembled bot army.

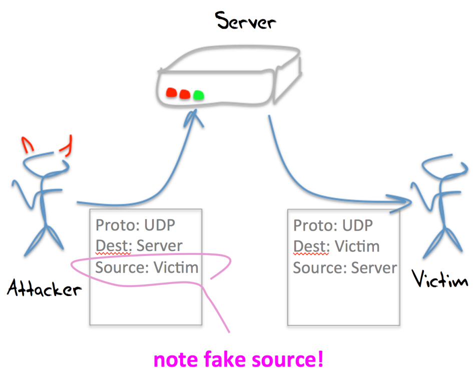


Many-to-one attacks

The downside is the need to enlist new bots into the bot army, and treat them carefully. The bots are created by entering code into the host through known exploits, and these attacks only work as long as the bot host is not well maintained with regular software updates and bot scanning. If the infected host is over-worked then its degraded operating state may indicate to the host's owner that the system has been compromised and requires a software refresh. Rather than relying on the ability to infect hosts and coopt them into a bot army, is it possible to use the normal operation of uncorrupted servers and pull them into the attack?

This is the next evolution of the DDOS attack model where we are using the approach of co-opting the innocent and uncorrupted to be a party to the attack, and using these third parties to act as attack amplifiers.

The attack is a "reflection and amplification" attack, and relies on leverage of otherwise innocent servers to take an incoming stream of packets, and reflect this stream as a larger stream of packets that are directed towards the victim. This is not a novel approach to launching a DDOS attack, and descriptions of this form of attack date back at least 8 years (for example, a 2006 presentation from Verisign (http://www.potaroo.net/iepg/july2006/1-frank-scalzo.pdf) described a 5Gbps attack).



Reflection attacks

UDP and Reflection Attacks

The User Datagram Protocol is a thin level abstraction of the underlying IP datagram model. As with IP itself, the service is a very simple datagram model. There is no session management, no reliability, and no flow control in UDP. UDP is used whenever there is a need for a fast and highly efficient short transaction protocol. A typical form of UDP application is for the client to send a single packet to a well known UDP port on a server, with the query contained in the packet's payload, and for the server to respond to the client with an answer. It's common to see the answer contained in a single packet, but this depends on the size of the answer, and larger responses may well invoke IP-level fragmentation or multi-packet responses.

In IPv4 it is a requirement for IP hosts to accept IP packets up to 576 bytes in size. Many hosts accept far larger packets these days, but the basic common requirement is to accept 576 bytes packets. This seems a somewhat arbitrary size, but in reading what's left of the origin material in design of IP, there was an intent to allow all hosts to accept basic datagrams with a 512 byte payload.

The IP header is 20 bytes in size, the IP options are up to 40 bytes in size, so this leaves 516 bytes of IP payload. The UDP header is 8 bytes long, so, strictly speaking, if a host is presented with a packet with 512 bytes of payload, 8 bytes of UDP header, 40 bytes of IP header options and 20 bytes of IP header, that makes 580 bytes, which overshoots the target by 4 bytes.

Maybe this 4 byte discrepancy would've been an important nit in the original specification for DNS, which called for 512 byte payloads in DNS over UDP, but it would only have been critical if we all used a full set of 40 bytes of IP options. These days IP options are rare. So rare that packets with IP options are regularly discarded by various firewall filters, so the missing 4 bytes is not really an issue.

To pack large answers into UDP packets the DNS went down a path of using IP-level fragmentation, while NTP went down the path of multi-packet responses. There are tradeoffs here involving the point in the protocol stack where fragmentation and reassembly takes place, and the issues of negotiating firewalls and filters where the multi-packet response invokes a different filter function than IP packet fragments.

The critical aspect of UDP, and the reason why it has been used for reflection attacks is that an interaction between a client and a server does not require a "handshake" as a precondition of service. The client sends the server a query over UDP, and the server assembles the response and sends it back to the client. How does the server know that this really is the client? At the UDP level, the server does not even try to validate the client. The server uses the source address in the IP header of the query packet, places this address into the destination field of the IP header of the response packet. And sends back its response

This allows for a simple reflection attack. If the attacker can craft the IP headers of an outbound packet, then all the attacker needs to do is send query packets to the server, but set the source address of these query packets to the IP address of the intended victim. When the server generates a response it will use this faked address as its destination address, and send the response to the intended victim.

In this case the server is not corrupted in any way. The attack actually counts on the server operating precisely the way it was intended to operate, and indeed can take advantage of the server being overprovisioned as a safeguard against the server itself being subject to some form of direct DOS attack.

What services work "best" for this form of UDP reflection attack? The kinds of requirements we are looking for to mount a very large scale reflection attack is to leverage a UDP service that meets the following criteria:

- The service is widely used
- The servers are widely deployed
- The servers are poorly maintained, or, even better, operated without active management

- The service's clients are not "qualified" by the server. That is, anyone can pose a question to the server. In other words these servers are "promiscuous"
- The service answer is far bigger than the question

The DNS is an ideal candidate service for DNS reflection attacks. When we compare DNS against these criteria for attack we can see:

- widely used
  The DNS is ubiquitous. Perhaps it is the most widely used set of servers on the Internet!

- servers are widely deployed
  Because it is widely used, the deployment of DNS resolvers and authoritative name servers is similarly widely deployed.

- servers are poorly maintained
  Many forms of consumer premises equipment include recursive name resolver code, and, inevitably, most folk are unaware that they are running an open recursive name resolver, let alone understand how to configure the device to shut down this exposure to user the resolver in a reflection attack.

- servers are "promiscuous"
  Authoritative name servers are conventionally required to be in a position to provide a response to anyone who presents them with a query. While there is no absolute requirement for recursive resolvers to be promiscuous, there are some 30 million such open resolvers on the Internet today!

- the answer is far bigger than the question
  In the case of the DNS it's just a matter of asking the right question, or using the right query options. DNSSEC tends to generate large responses, and queries for ANY can elicit large responses. Another approach is to craft a large TXT record in a domain specially crafted for a reflection attack, and pose queries for that domain to open recursive resolvers to generate the reflection attack.

DNS reflection attacks are now very commonplace. They have been seen to operate at sustained gigabit speeds, and the efforts to mitigate them through throttling the response rate from servers (DNS Response Ralt Limiting, or RRL, described at http://ss.vix.su/~vixie/isc-tn-2012-1.txt) appears to be deployed in a disappointingly piecemeal fashion, and many open resolvers without any form of rate limiting still populate the Internet.

However, once this kind of UDP reflection attack was identified, the search was on for other UDP-based services that appear to offer a good match against this list of reflection attack criteria. The *chargen* service is another potential DDOS reflection attack candidate. In response to an incoming UDP packet of any size, a *chargen* server will respond with a UDP packet which has a payload of a randomly selected size between 0 and 512 bytes. However, open *chargen* servers are thankfully rare, and so far large scale DDOS attacks based on *chargen* are uncommon.

The *Simple Network Management Protocol* (SNMP) is also a UDP-based protocol that can be used in a manner where a short query generates a large response. SNMP is used in many contexts, including within routers and switches, servers and end hosts. In theory SNMP is protected by a relatively robust security framework, but earlier versions of the protocol used a common plaintext password, and there is a suspicion that there is still a significant population of SNMP servers running old versions of SNMP that will respond to an SNMP query using a community string of "public".

But there is an easier target out there, and it sits behind UDP port 123. It's the *Network Time Protocol.*

NTP and Reflection Attacks

NTP is the Network Time Protocol, and it is used by computers to synchronize their time of day clocks (http://www.potaroo.net/ispcol/2014-03/ntp.html). NTP is a UDP-based protocol, and it is widely deployed. So how does it fare against our criteria for being used for DDOS attacks?

- widely used

  Time is important for all types of network function, including basic security functions. NTP servers are widespread, from managed services attached to reference time sources through to edge consumer product that is intended to relay time for local clients.

- servers are widely deployed

  Yes, NTP servers are widely deployed.

- servers are poorly maintained

  While the reference clock sources are managed with care, the time relay NTP servers tend to be embedded in other units, such as routers, network gateways and customer CPE. in the latter categories many folk are unaware that they are running an open NTP server, let alone understand how to configure the device to shut down this exposure to user the server in a reflection attack.

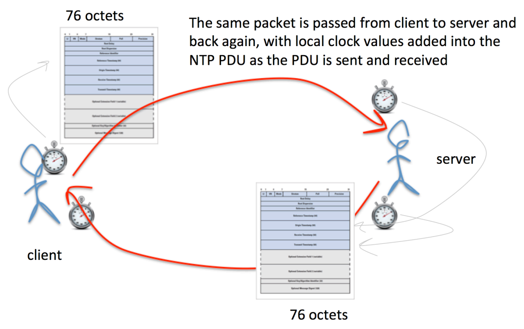
- Servers are "promiscuous"

  Unlike authoritative name servers in the DNS, there is no protocol or service requirement for NTP servers to be promiscuous. However, many servers are configured without filters, and are, in effect, promiscuous servers.

- the answer is far bigger than the question

  This is not normally the case.

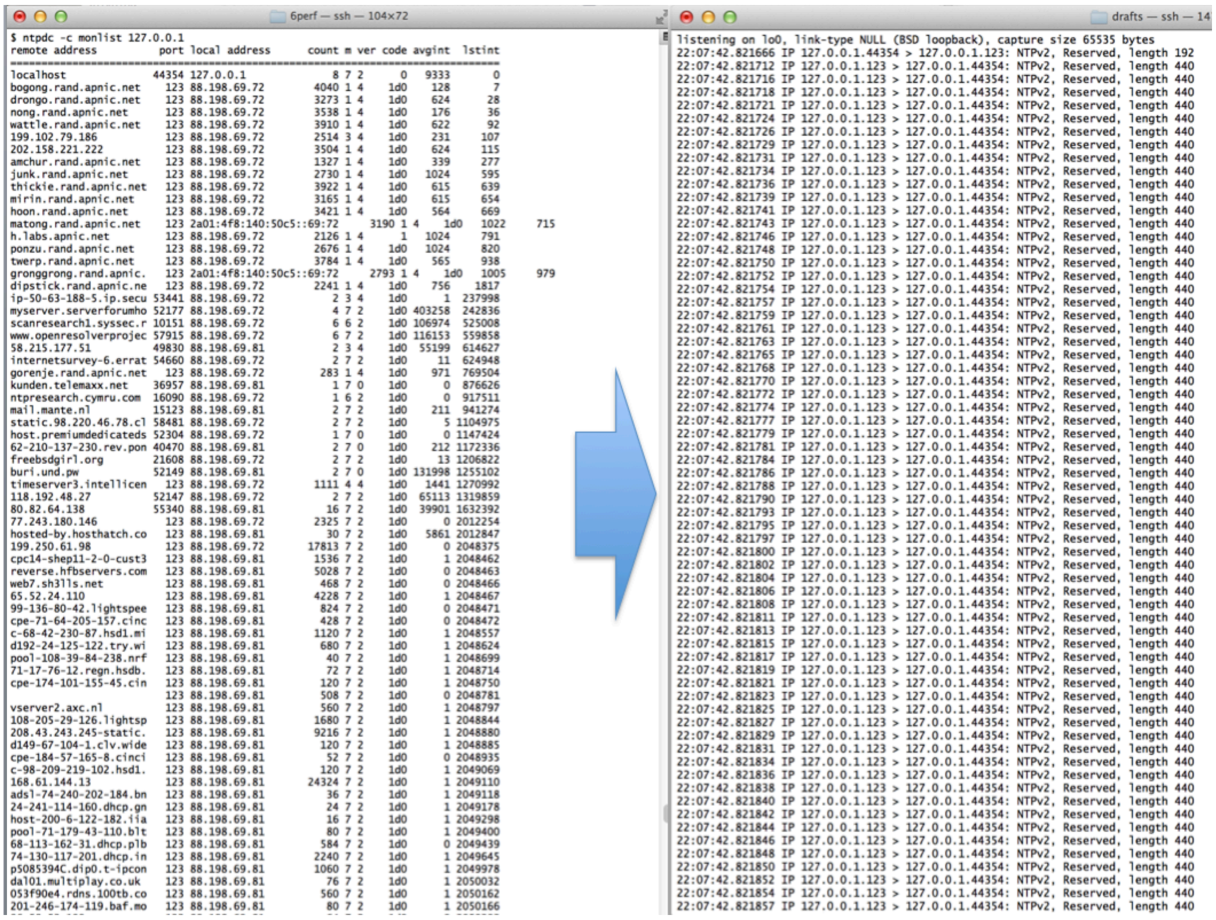
This last point has deterred the use of NTP for DDOS. NTP servers operate as a packet "reflector" in that when an NTP receives an incoming NTP packet it writes its time values upon packet reception and packet transmission, swapping the IP headers. NTP packets are relatively small, and without any optional extensions the incoming and outgoing packets are 76 octets in length. This implies that while NTP is a packet reflector, its not normally an amplifier as well, so attackers have preferred to use DNS instead.


76 octets
The same packet is passed from client to server and back again, with local clock values added into the NTP PDU as the PDU is sent and received
server
client
76 octets

NTP packet exchange for clock synchronization

But there is one weakness in NTP. NTP supports a query and control channel so that an NTP server can be managed remotely. The utility is called *ntpdc* and it takes a user command, converts into a NTP type 7 UDP packet, and send this packet to the NTP server using UDP and addressed to port 123. This is the same port as conventional clock synchronization traffic, so the conventional network-visible outer packet headers do not readily show the distinction between clock synchronization packets and these NTP type 7 packets. The NTP server will respond in UDP with its response. Unlike DNS, the response is conventionally broken into multiple packets, rather than relying on IP level fragmentation. NTP has been careful to ensure that configuration commands that are passed to a server using this control mode must contain a pre-configured key, so that hijacking an NTP server using this approach would normally require obtaining knowledge of this key. However, the NTP monitoring commands are read-only commands, and they are unprotected by this NTP authentication mechanism. Without explicit control anyone can send a read-only monitor command to an NTP server, and the NTP server will respond, as it is evidently promiscuous by default.

Most of the *ntpdc* commands elicit larger responses than the query, but the command which appears to have the largest response in the *monist* command. This command is a request for the server to report on the last 600 systems that this NTP server has communicated with. In this case a single UDP packet that is the encoding of the *monist* command, 220 bytes in length, will generate 100 UDP packets in response, each of which are 468 bytes in length. That is an amplification factor of 100 in terms of packets, and 212 in terms of bytes.

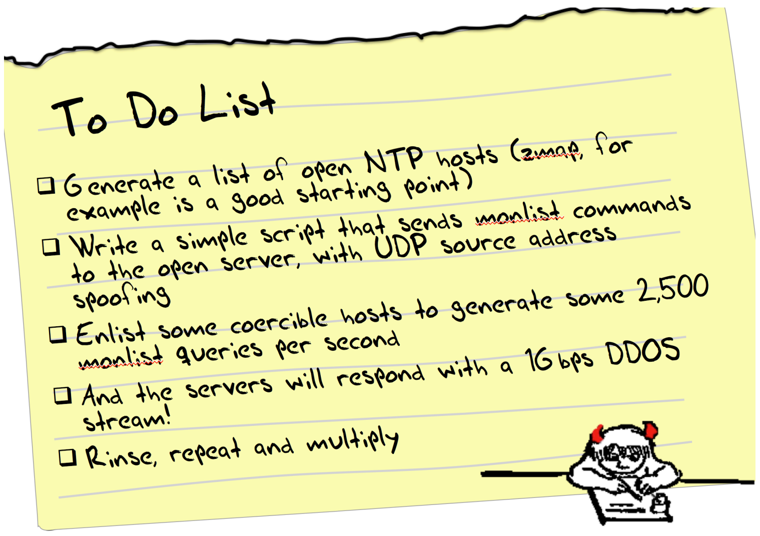


```
$ ntpdc -c monlist 127.0.0.1
remote address          port local address      count m ver code avgint  lstint
===============================================================================
localhost              44354 127.0.0.1              8 7 2      0   9333       0
bogong.rand.apnic.net    123 88.198.69.72        4040 1 4    1d0    128       7
drongo.rand.apnic.net    123 88.198.69.72        3273 1 4    1d0    624      28
nong.rand.apnic.net      123 88.198.69.72        3538 1 4    1d0    176      36
wattle.rand.apnic.net    123 88.198.69.72        3910 1 4    1d0    622      92
199.102.79.186           123 88.198.69.72        2514 3 4    1d0    231     107
202.158.221.222          123 88.198.69.72        3504 1 4    1d0    624     115
amchur.rand.apnic.net    123 88.198.69.72        1327 1 4    1d0    339     277
junk.rand.apnic.net      123 88.198.69.72        2730 1 4    1d0   1024     595
thickie.rand.apnic.net   123 88.198.69.72        3922 1 4    1d0    615     639
mirin.rand.apnic.net     123 88.198.69.72        3165 1 4    1d0    615     654
hoon.rand.apnic.net      123 88.198.69.72        3421 1 4    1d0    564     669
matong.rand.apnic.net    123 2a01:4f8:140:50c5::69:72      3190 1 4    1d0   1022       715
h.labs.apnic.net         123 88.198.69.72        2126 1 4      1   1024     791
ponzu.rand.apnic.net     123 88.198.69.72        2676 1 4    1d0   1024     820
twerp.rand.apnic.net     123 88.198.69.72        3784 1 4    1d0    565     938
gronggrong.rand.apnic.   123 2a01:4f8:140:50c5::69:72      2793 1 4    1d0   1005       979
dipstick.rand.apnic.ne   123 88.198.69.72        2241 1 4    1d0    756    1817
ip-50-63-188-5.ip.secu 53441 88.198.69.72           2 3 4    1d0      1  237998
myserver.serverforumho 52177 88.198.69.72           4 7 2    1d0 403258  242836
scanresearch1.syssec.r 10151 88.198.69.72           6 6 2    1d0 106974  525008
www.openresolverprojec 57915 88.198.69.72           6 7 2    1d0 116153  559858
58.215.177.51          49830 88.198.69.81           2 3 4    1d0  55199  614627
internetsurvey-6.errat 54660 88.198.69.72           2 7 2    1d0     11  624948
gorenje.rand.apnic.net   123 88.198.69.72         283 1 4    1d0    971  769504
kunden.telemaxx.net    36957 88.198.69.81           1 7 0    1d0      0  876626
ntpresearch.cymru.com  16090 88.198.69.72           1 6 2    1d0      0  917511
mail.mante.nl          15123 88.198.69.81           2 7 2    1d0    211  941274
static.98.220.46.78.cl 58481 88.198.69.72           2 7 2    1d0      5 1104975
host.premiumdedicateds 52304 88.198.69.72           1 7 0    1d0      0 1147424
62-210-137-230.rev.pon 40470 88.198.69.81           2 7 0    1d0    212 1172336
freebsdgirl.org        21608 88.198.69.72           2 7 2    1d0     13 1206822
buri.und.pw            52149 88.198.69.81           2 7 0    1d0 131998 1255102
timeserver3.intellicen   123 88.198.69.72        1111 4 4    1d0   1441 1270992
118.192.48.27          52147 88.198.69.72           2 7 2    1d0  65113 1319859
80.82.64.138           55340 88.198.69.81          16 7 2    1d0  39901 1632392
77.243.180.146           123 88.198.69.72        2325 7 2    1d0      0 2012254
hosted-by.hosthatch.co   123 88.198.69.81          30 7 2    1d0   5861 2012847
199.250.61.98            123 88.198.69.72       17813 7 2    1d0      0 2048375
cpc14-shep11-2-0-cust3   123 88.198.69.81        1536 7 2    1d0      1 2048462
reverse.hfbservers.com   123 88.198.69.81        5028 7 2    1d0      0 2048463
web7.sh3lls.net          123 88.198.69.81         468 7 2    1d0      0 2048466
65.52.24.110             123 88.198.69.81        4228 7 2    1d0      1 2048467
99-136-80-42.lightspee   123 88.198.69.81         824 7 2    1d0      0 2048471
cpe-71-64-205-157.cinc   123 88.198.69.81         428 7 2    1d0      0 2048472
c-68-42-230-87.hsd1.mi   123 88.198.69.81        1120 7 2    1d0      1 2048557
d192-24-125-122.try.wi   123 88.198.69.81         680 7 2    1d0      1 2048624
pool-108-39-84-238.nrf   123 88.198.69.81          40 7 2    1d0      1 2048699
71-17-76-12.regn.hsdb.   123 88.198.69.81          72 7 2    1d0      1 2048714
cpe-174-101-155-45.cin   123 88.198.69.81         120 7 2    1d0      1 2048750
                         123 88.198.69.81         508 7 2    1d0      0 2048781
vserver2.axc.nl          123 88.198.69.81         560 7 2    1d0      1 2048797
108-205-29-126.lightsp   123 88.198.69.81        1680 7 2    1d0      1 2048844
208.43.243.245-static.   123 88.198.69.81        9216 7 2    1d0      1 2048880
d149-67-104-1.clv.wide   123 88.198.69.81         120 7 2    1d0      1 2048885
cpe-184-57-165-8.cinci   123 88.198.69.81          52 7 2    1d0      0 2048935
c-98-209-219-102.hsd1.   123 88.198.69.81         120 7 2    1d0      1 2049069
168.61.144.13            123 88.198.69.81       24324 7 2    1d0      1 2049110
adsl-74-240-202-184.bn   123 88.198.69.81          36 7 2    1d0      1 2049118
24-241-114-160.dhcp.gn   123 88.198.69.81          24 7 2    1d0      1 2049178
host-200-6-122-182.iia   123 88.198.69.81          16 7 2    1d0      1 2049298
pool-71-179-43-110.blt   123 88.198.69.81          80 7 2    1d0      1 2049400
68-113-162-31.dhcp.plb   123 88.198.69.81         584 7 2    1d0      0 2049439
74-130-117-201.dhcp.in   123 88.198.69.81        2240 7 2    1d0      1 2049645
p5085394C.dip0.t-ipcon   123 88.198.69.81        1060 7 2    1d0      1 2049978
dal01.multiplay.co.uk    123 88.198.69.81          76 7 2    1d0      1 2050032
053f90e4.rdns.100tb.co   123 88.198.69.81         560 7 2    1d0      1 2050162
201-246-174-119.baf.mo   123 88.198.69.81          80 7 2    1d0      1 2050166
```

```
listening on lo0, link-type NULL (BSD loopback), capture size 65535 bytes
22:07:42.821666 IP 127.0.0.1.44354 > 127.0.0.1.123: NTPv2, Reserved, length 192
22:07:42.821712 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821716 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821718 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821721 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821724 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821726 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821729 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821731 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821734 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821736 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821739 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821741 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821743 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821746 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821748 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821750 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821752 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821754 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821757 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821759 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821761 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821763 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821765 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821768 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821770 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821772 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821774 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821777 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821779 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821781 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821784 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821786 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821788 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821790 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821793 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821795 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821797 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821800 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821802 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821804 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821806 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821808 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821811 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821813 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821815 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821817 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821819 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821821 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821823 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821825 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821827 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821829 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821831 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821834 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821836 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821838 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821840 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821842 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821844 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821846 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821848 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821850 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821852 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821854 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
22:07:42.821857 IP 127.0.0.1.123 > 127.0.0.1.44354: NTPv2, Reserved, length 440
```

ntpdc monlist command and packet trace

This is the essence of the NTP DDOS attack. Using a tool such as zmap (https://zmap.io) it's possible to assemble a list of open NTP servers, and these can be tested to see if they are responsive to the *monlist* NTP control packet. The attacker then needs to assemble a set of coopted attack systems that are capable of accessing a raw IP socket, allowing a script to generate UDP packets with a spoofed source address. If the attacker can assemble a set of attackers that collectively send some 2,500 queries per second to a set of NTP servers, then what would result is a 1Gbps UDP traffic stream being directed to the victim. Larger volume attacks are a case up scaling up the total query rate accordingly.



To Do List

- ❑ Generate a list of open NTP hosts (zmap, for example is a good starting point)
- ❑ Write a simple script that sends monlist commands to the open server, with UDP source address spoofing
- ❑ Enlist some coercible hosts to generate some 2,500 monlist queries per second
- ❑ And the servers will respond with a 16 bps DDOS stream!
- ❑ Rinse, repeat and multiply

Defence against NTP Attacks

As usual, the best defence is to run a current version of the NTP code. NTP version 4.2.7 (or later) has disabled the *monlist* command. If for some reason that's not an option, then the next best thing to do is to restrict the folk who can present these commands to the server by using the following configuration commands in `/etc/ntp.conf`

```
# By default, exchange time with everybody, but don't allow configuration.
#
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery
#
# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1
```

NTP also exists on many routers, and access lists should be used on the router to again filter who can have access to the NTP query channel. For a Cisco router running ios, the configuration recipe would look a lot like the following:

```
access-list 46 remark utility ACL to block everything
access-list 46 deny any
!
access-list 47 remark NTP peers/servers we sync to/with
access-list 47 permit 10.0.0.1
access-list 47 permit 10.0.0.2
access-list 47 deny any
!
! NTP access control
ntp access-group query-only 46   ! deny all NTP control queries
ntp access-group serve      46   ! deny all NTP time and control by default
ntp access-group peer       47   ! permit sync to configured peer(s)/server(s)
ntp access-group serve-only 46   ! deny NTP time sync requests
```

On a Juniper router the recipe is similar in intent:

```
term ntp {
               from {
                 source-address {
                     0.0.0.0/0;
                     /* NTP servers to get time from */
                     10.0.0.1 except;
                     10.0.0.2 except;
                     }
                 protocol udp;
                 port ntp;
                 }
                then {
                  discard;
                  }
               }
```

This config script assumes a default permit on the loopback filter. If the loopback is a default deny filter then the sense of the ntp config needs to be reversed to allow for a specific `accept`:

```
term ntp {
               from {
                 source-address {
                     10.0.0.1/23;
                     10.0.0.2/32;
                     }
                 protocol udp;
                 port ntp;
                 }
                then {
                  count ntp-requests;
                     accept;
                  }
               }
```

BCP38

After deploying the mitigation measures in the DNS, these measures of plugging up the leaks in NTP may start to feel like sticking one's fingers in the emerging holes in the dyke. It's a case of using incremental answers to a much larger and broader problem.

Given the level of embedded functionality in various consumer products that operate in a semi-sealed or fully sealed manner, than it also appears to be the case that these pinpoint filters can only be partially effective in any case. Open UDP-based servers appear to be a constant factor that the network has to deal with, and while open UDP-based servers are a problem, it's the combination of these servers and the ability to propagate packets with a spoofed source address that are the common elements of this form of reflection attack.

So if we are resigned to living with a continuing significant population of open UDP-based servers, then the corollary appears to be that we should not also be resigned to living in a network with a continuing significant population of networks that allow the propagation of packets with spoofed source addresses.

The reason why this section is titled "BCP38" is that the document that describes an approach to network ingress filtering to prevent source address spoofing is RFC2827 by Paul Ferguson and Daniel Senie, published in May 2000, and listed as a "Best Current Practice" document, hence the title "BCP38". That document is now 14 years old, and the set of networks that implement this practice still appears to be small. (The phrase "appears to be small" is somewhat unsatisfactory, and there have been some efforts to measure the extent to which BCP38 filtering has been deployed in today's Internet. One notable current effort can be found at http://spoofer.cmand.org)

Some argue that trying to get the entire set of networks to implement practical forms of source address spoofing filtering is a task akin to boiling the ocean, and the economics of cost vs benefit mean that many network operators are simply inadequately motivated to incur the incremental operational costs of deploying an additional filter mechanism. Others argue that the risks associated with UDP reflection attacks are so insidious and the pervasive use of sealed consumer equipment that appears to be intractably broken (such as the 30 million open DNS recursive resolvers indicated by the open resolver project, at http://openresolverproject.org) is just such an intractable problem that BCP38 is the only available mitigation left to us.

Irrespective of the various mitigation efforts its depressing to note that the evolution of evil on the net appears to be tracking the evolution of network capacity, and at the same time as we are approaching terabit transmission systems we are also approaching terabit attacks. Depressingly there is no real grounds to think that this will stop anytime soon. A bleak view sees that the scale of growth of the network's infrastructure capacity will continue to be matched by the ability to turn this infrastructure into an attack vector, and the dubious claims of the "world's largest attack" will continue to be short-lived exploits that will be consistently surpassed.

Further Reading

Description of NTP attack
    http://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks

Sealing up NTP – a template for ntp.conf
    http://www.team-cymru.org/ReadingRoom/Templates/secure-ntp-template.html

Cert Advisory
    https://www.us-cert.gov/ncas/alerts/TA14-013A

Open NTP servers
    http://openntpproject.org

Open Recursive DNS Resolvers
    http://openresolverproject.org

BCP 38
    http://bcp38.info

BCP 38 tracking
    http://spoofer.cmand.org

Disclaimer

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of Trustees of the Internet Society from 1992 until 2001.

www.potaroo.net