Denial of Service Mitigation Importance in Cloud Computing

Benjamin Rose
30 November, 2009
CS-573
Professor Amoroso

During the second half of 2010, Google will be releasing their revolutionary new operating system, Chrome OS. This operating system will run applications “in the cloud” with the user's computer, more than likely simply a netbook with minimalist hardware; playing as simply a pretty version of yesteryear's dumb-terminals. However, with this paradigm shift in computing about to take place, very special considerations must be placed on denial of service mitigation. An attack of great magnitude could render thousands of peoples' computers completely inoperable. Such an attack may also be extremely imminent, a single keystroke by one person's hand could leave even a computing giant like Google, and henceforth their cloud, in the dark. This could spell disaster for not only the Google, but also for every one of their constituents, possibly even many of the critical services in this country.

Many years ago, computers were primarily thin-client machines, running only the most basic software imaginable, with the heavy-lifting performed by a large mainframe located somewhere else. The mainframe was tended to by only the most brilliant minds available in the computing field. Computers soon became less expensive and more powerful, giving birth to the fat-client computer system. Today, nearly everyone and their grandparents owns their own computer running its own operating system and software, essentially making every owner their own mainframe administrator. Riddled with viruses and spyware, hundreds of thousands of these fat-clients have become zombies in an ever-waging botnet war. This war is truly gargantuan in scope, and to understand how lucrative it is to be a master of a botnet, one must first understand the fundamentals of a denial of service attack and how it works. Diagrams referenced throughout this document can be found in appendix A, examples are found in appendix B.

The vast majority of computers are connected to the Internet. Some directly, some through a router or LAN. In order to connect to most services on the Internet, such as a website

over HTTP or a chatroom via IRC, computers will use a protocol called the Transmission Control Protocol (TCP), which is carried by the Internet Protocol (IP). The combination thereof results in the acronym TCP/IP, meaning, quite literally, Transmission Control Protocol over Internet Protocol. Application data is encapsulated in a TCP datagram, and TCP traffic is in turn encapsulated within the data section of the IP packet. A more detailed diagram of the encapsulation process can be found in Figure 1, showing the roles of the application and the operating system. There are alternatives to TCP, such as the User Datagram Protocol (UDP). The main difference between these two protocols is that TCP provides delivery and ordering assurance, while UDP does not. This naturally makes TCP a more desirable choice for critical applications. However, it is also a much more vulnerable protocol in terms of fundamental design weaknesses.

When this suite of protocols was first developed, security wasn't a large concern, and in fact IPv6 does serve to correct some of these inherent weaknesses. However, IPv4, the version of which the vast majority of modern connections uses, and this document references, is vulnerable to a plethora of attacks. To understand these attacks, though, one must first examine how the protocol works in an ordinary situation. Figure 2 shows the three initiating packets in a typical TCP connection, known as the TCP handshake process. The client sends a synchronize request to the server (SYN), which responds with an acknowledgment (ACK) and a synchronize request of it's own. The client sends an acknowledgment to the server's provoked synchronize request, and the connection is now completed. At this point, data can now be passed across the connection as a series of small packets. Note that data cannot be passed on a SYN packet as it would be a violation of RFC's to send data into a possibly closed window, and to do so would be considered a TCP anomaly.

When a new SYN packet is received by a server, it must prepare to send data across the new connection. The daemon responsible for the new connection, hypothetically an HTTP connection, must fork a copy of itself to compile and serve the to-be requested data. This takes both processor time and a healthy amount of RAM to accomplish. Suppose now the situation highlighted in figure 3 were to occur, wherein the initial SYN packet were to have a forged source address. The server will now fork and attempt to complete the connection initiation. Since the Internet is a very lossy environment, it will send a SYN/ACK in an attempt to solicit the final ACK several times before finally timing out. If this were to occur constantly in a flood of forged SYN packets, the server would rapidly exhaust it's resources trying to cope with the oncoming flood. No legitimate users would be able to contact the daemon, so real people trying to contact the HTTP server to view a hypothetical website would be lost in a flood of forged requests and would probably rarely, if ever, receive a response.

This type of attack is a basic SYN-flood, and can in fact be blocked quite simply through the use of a firewall. A hashlimit in iptables as shown in example 1 can remedy this type of flood as far as the server's allocation of resources is concerned. It is noteworthy, though, that while the allocation of resources may not take place, a large amount of data is still being passed along the network link, and can be responsible for saturating the service in question. In fact, in theory, as long as no rate-limits are put in place, and the attacker has more bandwidth than the victim, a full denial of service can be performed using completely garbage datagrams with a valid IP header.

This type of attack worked exceedingly well back in the days of dial-up and T1 connections, but have lately been largely mitigated by larger bandwidth connections, firewall deployment, and overall awareness of the issue. It has been replaced, however, by a new type of attack, the distributed denial of service attack, as can be seen in the second graphic of figure 4.

Note that figure 4 refers to the attacking machines as zombies, and an equally accepted term for this type of machine is a bot. The attack control mechanism can be any number of mediums equally accessible by all the bots – as of recent, even Twitter has been used as a control channel medium.

The bots are not highly specialized machines, either. In fact, in many cases, it is just the average computer user running a copy of Windows riddled with viruses and spyware. These viruses are responsible for connecting to the control channel and awaiting further instructions. With just a few keystokes, the controller of the botnet can unleash an attack on a target of absolutely massive order, using the combined total bandwidth of all the individual bots in the botnet.

The effect can be seen in figure 5, where no matter what hardware resources the victim has, the network link it is attached with has absolutely no chance of competing. This can actually be done without the need for viruses and botnets, it actually happens quite frequently when a high-traffic site features a lower-traffic and less-connected site. It is often referred to by its unofficial nicknames, the slashdot effect or the digg effect. Note also that this document will not even discuss the Distributed and Reflected Denial of Service (DRDoS) attacks, as it is far beyond it's scope. Mitigating Distributed Denial of Service (DDoS) attacks has become commonplace on many servers, regardless of the number of constituents. Reference examples 2 and 3, which both serve to mitigate a DDoS attack. Example 2 is a small script written quickly in an effort to mitigate a DDoS with a particular pattern, and example 3 is a simple tc, the linux kernel-level packet scheduler, rule structure to prevent an unintentional denial of service on a high-traffic server. Such fixes for resource allocation are trivial, yet fixes for the issue of bandwidth are hard to find. Hardware solutions can easily cost hundreds of thousands of dollars for a large-scale

network, and engineers to work with this equipment can be even harder to find. Yet, the cost of this equipment and the personnel required to operate said equipment may be well worth it, especially as the world moves towards cloud computing; back to yesteryear's thin-client.

Google's Chrome OS is only an example of thin-client applications that are beginning to spring up throughout modern computing. Many modern services are becoming more and more reliant on network connections and remote storage to operate. Performing GUI operations locally but doing the heavy lifting or storage elsewhere is now more commonly known by it's buzzword, “cloud computing”. Chrome OS will be one of the most thin-client operating systems of the modern era, relying on a network connection to Google's server farm to perform many common tasks. For example, the operating system will provide nearly no driver support for many peripherals other than thumb drives and common items, it will sport only the most basic window manager, and will run minimal services. This allows an extraordinary boot time of 7 seconds, which Google has announced it will attempt to further reduce.

Chrome OS will not provide much software, either. For example, instead of providing an office suite such as OpenOffice or Microsoft Office, Chrome OS will simply rely on Google Docs for an office suite. The hardware will also be minimalist, Google plans to run Chrome OS on mainly netbooks with very basic specifications. This will make the entire experience very lightweight and snappy, but this relies very heavy on one main element – a stable and persistent network connection.

One can begin to see the problem that can be posed by considering what would happen if Google's network services were to be interrupted once Chrome OS becomes mainstream. If a full network connection were to be completely unobtainable, especially through the fault of Google and not the end user, every constituent of Chrome OS could essentially own a several hundred

dollar paperweight. Documents and files would become unavailable, work would be impossible to perform. There would be a very negative backlash towards the company that went down, even if it were due to a massive scale attack.

At this point, it must seem like an unlikely scenario for a botnet to be able to take down a computing powerhouse like Google. It is estimated that around 6% of all traffic on the Internet goes to or through Google via their tight connections with Internet Service Providers (ISPs) and massive quantities of “dark fiber” [1]. With this sort of bandwidth capability, it seems like no attack from any botnet could DoS Google, yet, nothing could be further from the truth. There are many botnets operating in present day that are more than capable of such a feat. One such botnet is known as the Storm Network.

The Storm botnet is absolutely gargantuan in scale. The most conservative widely-published estimate of it's size to date puts it in the range of 160 thousand zombie computers [3]. More liberal estimates have placed it in the multi-million territory, one estimate even claiming it may be running on more than 50 million machines [3]. There is clearly a huge differential between these two figures, as it is nearly impossible to estimate the size of a botnet like Storm. Storm communicates not via an Internet Relay Chat (IRC) Command and Control (C&C) medium, it is entirely peer-to-peer (P2P), relying on other compromised hosts to spread instructions via the eDonkey protocol.

This protocol is odd at best. This is not to mention, all communications between Storm nodes are fully encrypted, making traffic analysis nearly impossible. Storm is very crafty, it constantly changes its own instruction base, sometimes on a per-minute basis, to avoid detection by Heuristic virus scanners. It has the ability, via a rootkit, to catch fork calls and return 0 automatically, leading virus scanners like Norton and McAfee essentially brain-dead, thinking a

scan was performed and nothing was found. There exists no Intrusion Detection System (IDS) rules to catch all communication made by the storm worm either, the best set as of current day is the Emerging Threats rules for Snort, and they are not even close to a 100% catch-all rate.

Storm spreads primarily via email virus attachments, and once it has infected a system, uses that system to send spam emails to email addresses around the world. Some estimates say that the Storm botnet is responsible for generating nearly a quarter of the spam the world sees daily. [3] Storm is truly massive in scale. Taking the most modest figures available, assume Storm does consist of 160 thousand infected zombie machines. Now assume an average Internet connection of a conservative 3 Megabits per second. This results in Storm controlling over 450 Gigabits per second of theoretical bandwidth. This is no small amount, and is based on extremely conservative estimates. Using the liberal estimates of 50 million zombies, each with a 6 mbps connection to the internet, Storm controls a ridiculous near-300 terabits per second of theoretical bandwidth. Surely the liberal estimate could take even a powerhouse such as Google, probably even entire countries of the world, completely offline. The conservative estimates, probably not to as great an extent, yet it's certainly more than enough bandwidth to at least interrupt cloud services for users.

So far, this document has examined only examples, with botnets such as Storm and cloud services offered by only Google though their up-and-coming Chrome OS. However, the real struggle will come with the smaller cloud services that are starting up on a massive scale. Small companies looking to get their foot in the door are opening services such as network storage, content delivery networks, and render/cluster farms. All of these services can be seen as taking part “in the cloud”, and all of them are highly susceptible to denial of service attacks.

As the world moves back towards the thin-clients of yesteryear, and relies more and more

on a network connection to deliver their respective content, security – particularly denial of service mitigation – has never been more critical. Previous thin-clients had less of an emphasis on security, as it was not as widely-known of a topic and issue. It could not be more critical in today's world, though. The bottom line is, if a company, whether well-established and massive or a brand new start-up, wants to provide cloud services, very special attention needs to be paid to the importance of mitigation of denial of service attacks.

Appendix A: Diagrams

Figure 1: Data encapsulation. [4]

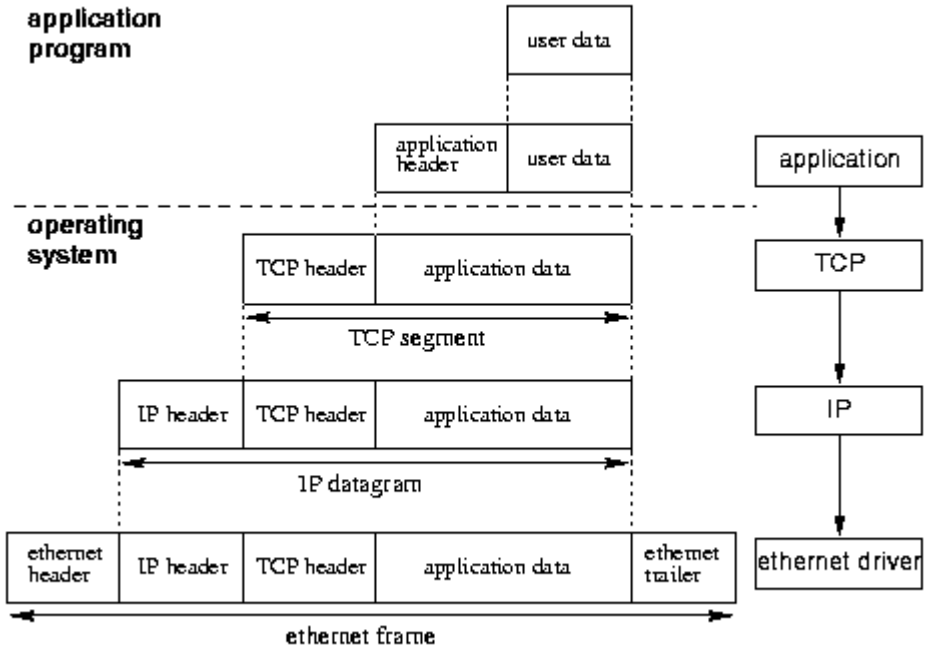


Figure 2: A typical TCP connection. [2]

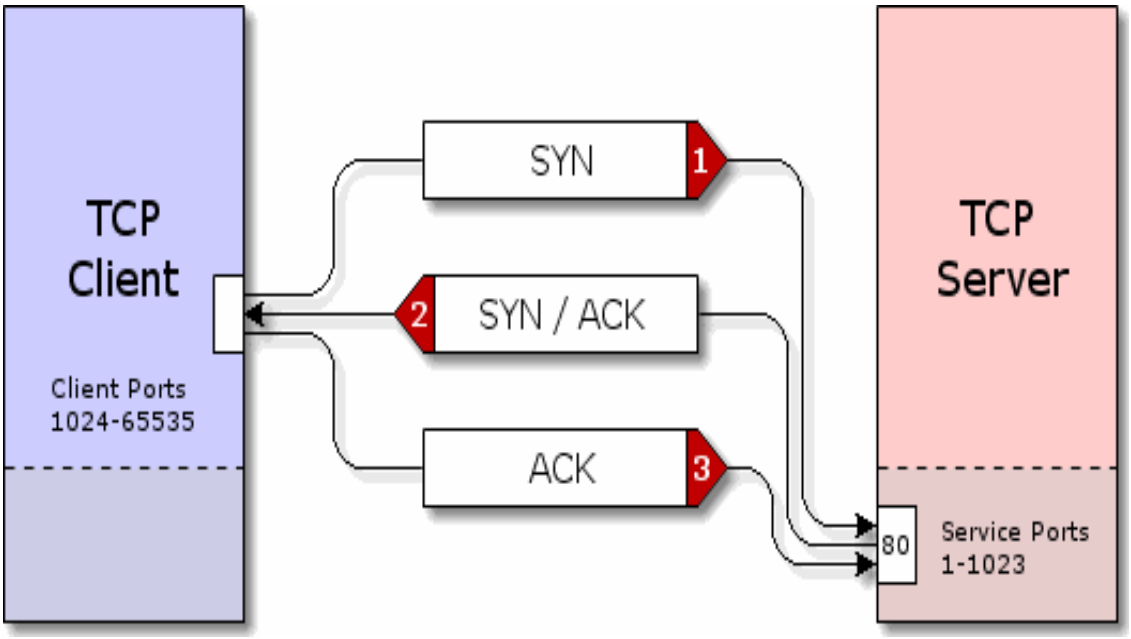


Figure 3: A forged TCP SYN packet. [2]

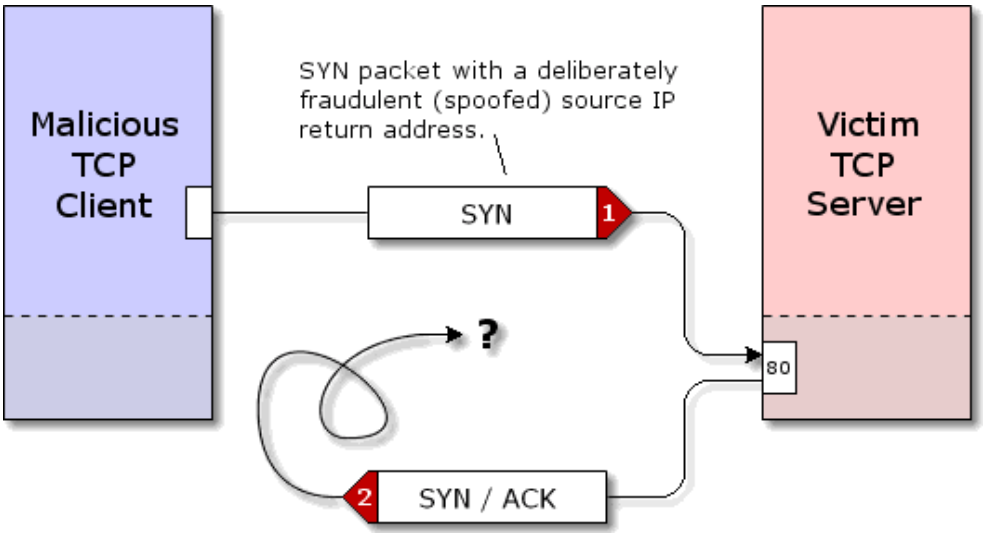


Figure 4: Two different approaches to attacker relationships, the DoS and the DDoS (distributed). [2]

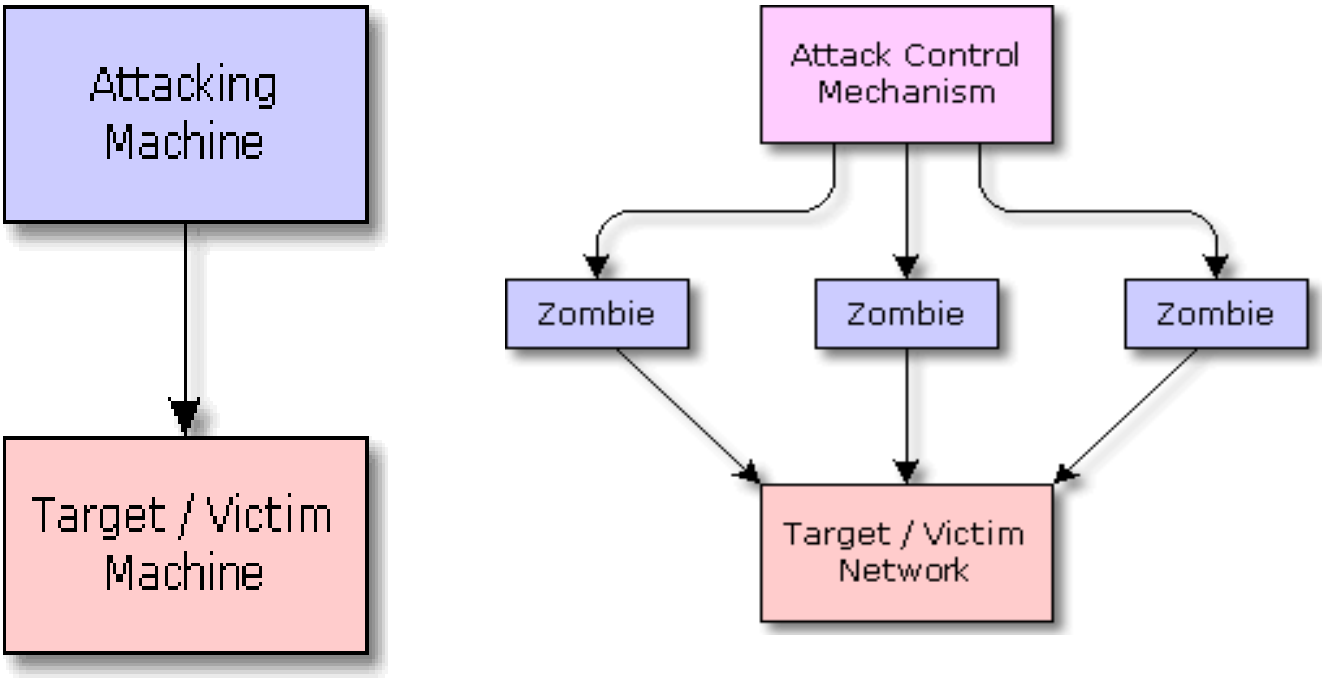
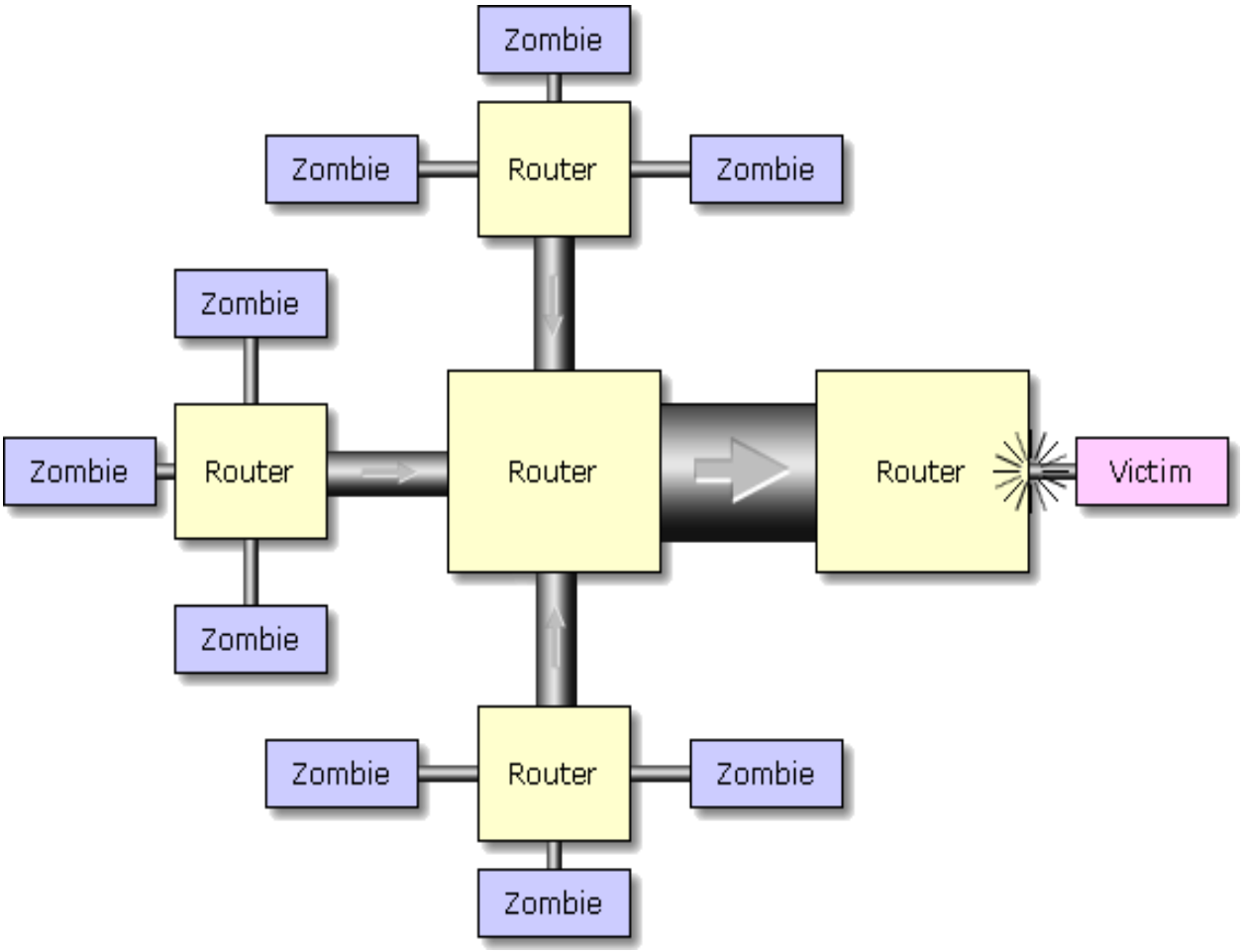


Figure 5: Distributed Denial of Service Attack. [2]



Appendix B: Code and textual examples

Example 1: iptables rule to prevent single-source SYN-flood.

-A INPUT -p tcp --source 0/0 --destination-port 80 -m hashlimit --hashlimit 1/second --hashlimit-burst 10 --hashlimit-mode srcip --hashlimit-name http -m state --state NEW -j ACCEPT

Example 2: PHP/bash script to ban IP's with several parallel connections.

```
<?
error_reporting(E_ERROR);
while(1)
{
unset($final);
$output = shell_exec("netstat -ntu | awk '{print \$5}'| cut -d\: -f1 | sort | uniq -c | sort -nr | grep -v Servers | grep -v Address | grep -v 192.168");
$output = explode("\n", $output);
for($i = 0; $i <= sizeof($output); $i++)
        {$final[] = explode(" ", $output[$i]);}

for($i = 0; $i <= sizeof($output); $i++)
{
        if($final[$i][sizeof($final[$i])-2] > 20 && $final[$i][sizeof($final[$i])-1] != "")
        {
                shell_exec("iptables -A Blacklist-INPUT -s ".$final[$i][sizeof($final[$i])-1]." -j DROP");
                echo("Banned ".$final[$i][sizeof($final[$i])-1].", had ".$final[$i][sizeof($final[$i])-2]." parallel connections, max is 20\n");
        }
}

sleep(30);
}
?>
```

Example 3: tc rules to prevent an unintentional denial of service (slashdot effect).

tc qdisc add dev eth0 root handle 1: cbq avpkt 1000 bandwidth 1000mbit
tc class add dev eth0 parent 1: classid 1:1 cbq rate 3mbps allot 1500 prio 5 bounded isolated
tc filter add dev eth0 parent 1: protocol ip prio 16 u32 match ip dst 155.246.0.0/16 flowid 1:1

Works Cited:

[1] http://www.nytimes.com/external/readwriteweb/2009/10/13/13readwriteweb-google-accounts-for-6-of-all-internet-traff-90323.html

[2] http://grc.com/

[3] http://en.wikipedia.org/wiki/Storm_botnet

[4] http://www.rogerclarke.com/