DC++ and DDoS Attacks

Ing. Adrian Furtunã
http://stormsecurity.wordpress.com
seastorm44@yahoo.com

ABSTRACT

The usage of peer-to-peer networks in massive distributed denial of service attacks is well known since the beginning of year 2007 when this kind of attack has often been observed [1][2] against many public servers. At the date of this article's writing (July 2008) there were not so frequent DC++ generated DDoS attacks reported.
But the big danger still remains because a great number of the DC++ hubs around the world are owned by people whose ethics is questionable and who could (any time) generate such an attack.
This article discusses in great depth the anatomy of a DC++ based DDoS attack and shows some measures that could be used to defend against it, including a tool to detect the attacker hubs.

The ideas presented in this article are based on practical experience during a confrontation with this type of attack.

1. INTRODUCTION

Distributed Denial of Service (DDoS) attacks are known for many years and they can be very effective even in nowadays. The main idea of a DDoS attack is to deplete the resources of the victim (bandwidth, CPU, memory, disk space, etc) and no matter how many resources the victim reallocates, the attack will still overwhelm it.

There are several methods to implement a DDoS attack nowadays. One of them is by commanding the bots (zombies) of a botnet to simultaneously send attack traffic against a victim. The attack intensity depends on the size of the botnet [3]. Several botnets are now disputing for supremacy around the world: Storm, Kraken, Srizibi, etc. [4]

A more stealthy method of doing a DDoS attack is to inject hidden code into well known sites that are vulnerable. The hidden code could contain instructions to initiate connections to the victim server. When the visitors visit those sites, they automatically execute the code and initiate legitimate connections to the victim server. [5]

Peer-to-peer networks can also be used in DDoS attacks. One of the most aggressive of these attacks exploits the DC++ network. This is different from a botnet attack because the attacker does not exploit any vulnerability in the clients that generate the attack traffic. He just instructs them to blindly connect to the victim through the DC++ hub. In the next parts of this article we will discuss in depth this DC++ based DDoS attack.

2. DC++ OVERVIEW

2.1 DC++ history and architecture

In 1999 a high-school boy named Jonathan Hess was creating a company named NeoModus, which had the objective to facilitate the file sharing over the Internet. He was using for this purpose a proprietary

protocol called Direct Connect. The first client application for this network was NMDC (NeoModus Direct Connect). The idea of sharing files over the Internet became very popular, so other Direct Connect (freeware) clients were written - DC++, StrongDC, oDC, Valknut, etc - by reverse engineering the protocol. The most popular of them is DC++, which gave the name of the network.

DC++ peer-to-peer network is composed of three entities: clients, hubs and hublist servers. The clients are the ones who want to share files between each other. The hubs are server applications (ex. Verlihub, YnHub, HexHub, Ptokax, etc) that facilitate the communication between the clients. For a client to know which hubs to connect to, it must know the hub's name or IP address and the hub's port. These information can be set manually or the client can download a list with hub information from specialized hublist servers. The architecture of a DC++ network is presented in Figure 1.

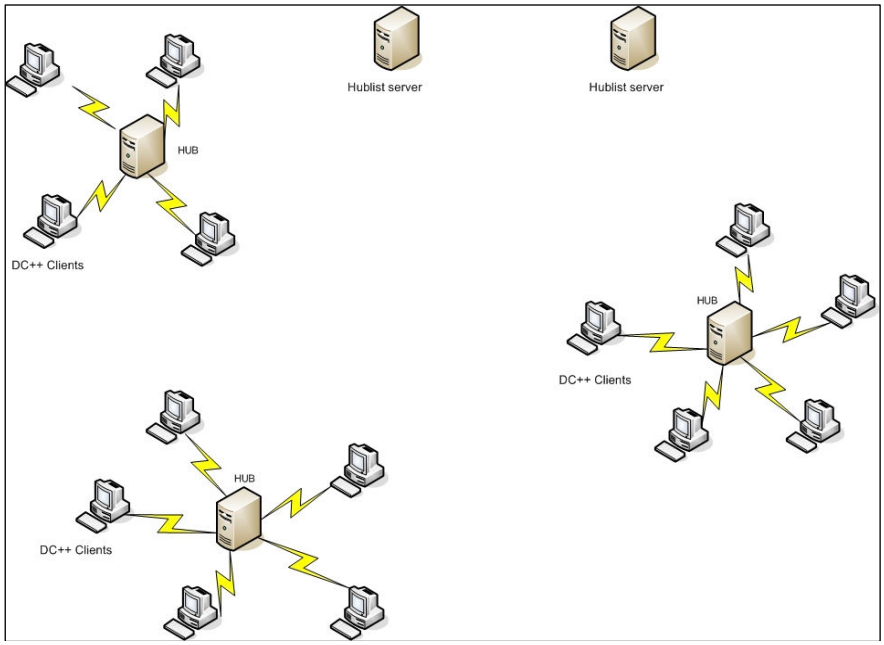

Figure 1. DC++ network architecture

DC++ clients identify themselves to the hub and to the other clients by a so called *nickname*. Some hubs impose restrictions for the nickname to have a specific format (ex. [RO][B][CZONE]xxx) but others allow random nicknames.
Clients can connect to the DC++ hubs in two ways: passive or active. A *passive* client is the one that connects to the hub from behind a firewall or from a LAN with private IP addresses. A passive client cannot receive direct connections from the Internet. An *active* client is the one that has a public IP address and is connected directly to the Internet. It can receive direct connections from other clients.
An active client can download files from any other client of any type but a passive client can download files only from active clients. Two passive clients cannot download files from each other using DC++ protocol unless they are on the same LAN.

There are millions of DC++ hubs in the Internet and each of them can have thousands of clients, depending on its resources.

2.2 DC++ usage in DDoS attacks

At the beginning of year 2007 there were many reports of DDoS attacks against web servers, generated by DC++ clients.
This kind of attack can have a very large scale, respectively several thousand computers that send traffic to a victim server resulting in about 25.000 connections/second for a moderately big attack.
While a typical server can handle a few hundred connections/sec before its performance begins to degrade, most web servers "die" almost instantly when they have to face six or seven hundred connections/sec. So, in case of a DDoS attack implemented with DC++ clients, most of the web servers can be completely disabled.
The classical defense mechanisms are not effective against them and the really effective ones are very expensive.

3. ATTACK DESCRIPTION

3.1 Direct Connect

In order to fully understand this kind of attack, the Direct Connect protocol must be understood first.
Direct Connect protocol has no standard version and it was initially documented by reverse engineering the first Direct Connect client application – NeoModus. Nowadays it is being maintained and developed by various groups from the Internet.

Direct Connect (DC) is an application level protocol that uses TCP for transport. It is a clear text protocol, unencrypted, that uses commands of the following form: **$<command>|**, where '|' is the command delimiter.
In DC protocol there are four communication types (usage scenarios):

1. Hub ←→ Client
2. Client ←→Client
3. Hub ←→ Hub (still in development)
4. Hub ←→ Hublist server

For the purpose of this article we will explain the second scenario, the Client ←→ Client communication, which is exploited to generate DDoS attacks.

DC++ clients communicate directly with each other when they want to download files. The communication between two clients is initiated through the hub to which are both connected, because this is their only common point. As we mentioned before, if the Downloader client and the Uploader client are both passive, the file transfer between them is not possible using Direct Connect protocol. If the Downloader is active and the Uploader is passive, then the Downloader cannot initiate a connection to the Uploader in order to transfer files. So, in order to do the file transfer, it will give the Uploader a command (through the hub) to initiate back a connection to the active Downloader and this way the file transfer can begin.

These are the steps of a file download in DC protocol:

D = downloader
U = uploader
H = hub

1. D>H: $ConnectToMe <U's username> <D's IP and port>|
2. H>U: $ConnectToMe <U's username> <D's IP and port>|
3. U>D: TCP Connection to D's IP and port
4. U>D: $MyNick <U's nick>|$Lock <new lock with pk>|
5. D>U: $MyNick <D's nick>|$Lock <new lock with pk>|$Direction Upload

<anumber>|$Key <key for U's lock>|
6. U>D: $Direction Download <anumber>|$Key <key for D's lock>|
7. D>U: $Get <filepath + filename>$<start at byte (1=beginning of file)>|
8. U>D: $FileLength <length of the requested file>|
9. D>U: $Send|
10. U>D: Data, in many chunks.
11. D>U: $Send| <- when 40906 bytes are sent, ask for more

We will explain the first four steps, because they are relevant to the DDoS attack. Steps 5-11 are also protocol specific and they deal with the transfer of the file bytes, after the direct connection between the two clients has been established. A full Direct Connect command reference can be found in [6].

We can see in the first step that the Downloader sends the command $ConnectToMe to the hub. The command parameters are the Uploader's nickname and the Downloader's IP address and port. The hub must send this command unaltered to the Uploader (identified by its nick name) – step 2. When a client (Uploader) receives a $ConnectToMe command, it must initiate a TCP connection to the client that sent this command (identified by its IP address and port) – step 3. As we already said, this behavior is necessary when direct connection between two clients is not possible because of the network topology (one of the clients is behind of a NAT device or firewall and the other has public IP address).

After the TCP connection has been established, the Uploader sends to the Downloader the command $MyNick which is used to identify itself. The rest of the commands (steps 5-11) are used to effectively do the data transfer, between the two clients directly.

3.2 The Attack

The attack uses a vulnerability in the DC++ hubs (Verlihub-0.9.8c, Verlihub-0.9.8d-rc1, Ynhub < 1.0306, Ptokax < 0.3.5.2), respectively in the Client-to-Client communication described above.
The vulnerability is in step 2, when the hub forwards the $ConnectToMe request to the Uploader client without verifying it. So the Downloader can put any IP address and port it wants in the $ConnectToMe request and the receiving client (Uploader) will connect to that address, trying to continue the file download protocol.

It is very easy to make a tool that generates a DDoS attack using this vulnerability. All the tool needs to do is connect to several DC++ hubs (which are vulnerable) and repeatedly send forged $ConnectToMe requests to each of the hub's clients. The forged requests must have the Downloader's IP address and port set to victim server's IP address and port. That way all the hub clients that receive this message will initiate connections to the victim and try to continue the file download (steps 3 and 4).

D = downloader (attacker)
U = uploader (DC++ client)
H = hub
V = victim

1. D>H: $ConnectToMe<U's username, **Victim's IP and port**>
2. H>U: $ConnectToMe<U's username, **Victim's IP and port**>
3. U>V: TCP Connection
4. U>V: $MyNick <U's nick>|$Lock <new lock with pk>|

This behavior is presented in Figure 1.

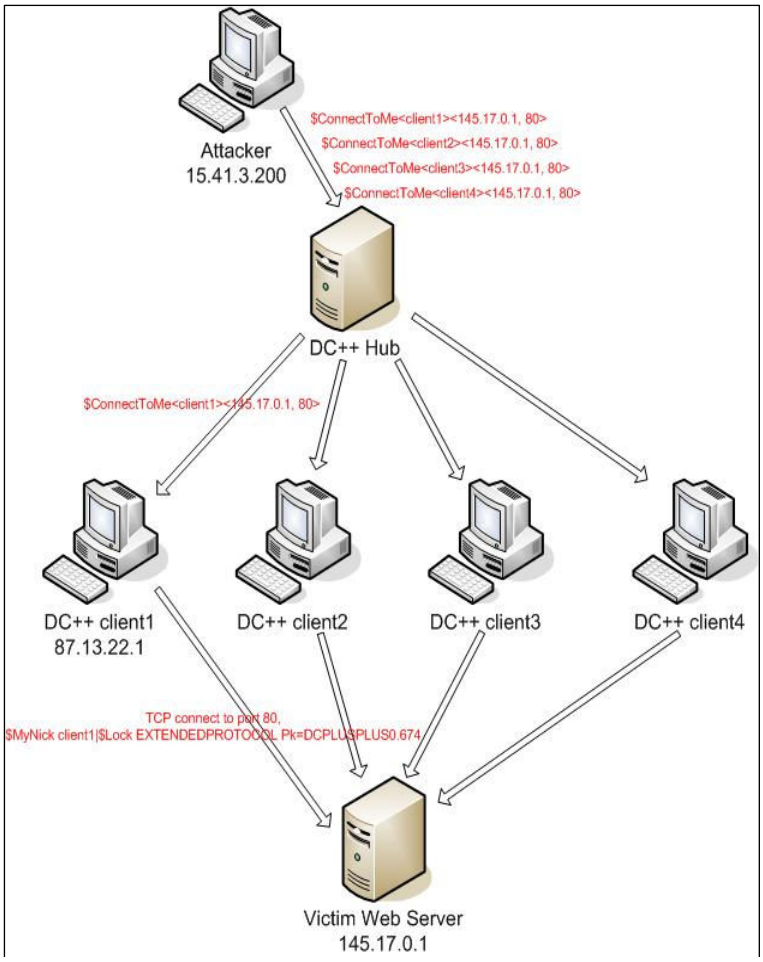

Figure 1. A DC++ DDoS attack in action

So, from the attacker's point of view it is a low bandwidth attack but the effects on the victim side are maxim.

This kind of attacks is usually done against web servers. During the attack, the web server first does legitimate TCP handshakes with the DC++ clients and then receives non-HTTP packets containing Direct Connect commands like: $MyNick clientxxx|$Lock EXTENDEDPROTOCOL Pk=DCPLUSPLUS0.674 (step 4).
The HTTP protocol makes the server wait for a configured period of time until it receives a valid HTTP request. So its resources for that connection will be unavailable until the timeout expires. The very big number of connections simultaneously established with the web server finishes its resources and makes it unavailable for any legitimate requests.

The number of connections/sec generated during an attack can be calculated after the following formula:

connections/sec = hub_no * hub_clients * $CTMs/sec

where:

hub_no = number of hubs participating in the attack
hub_clients = average number of clients on each hub
$CTMs/sec = number of $ConnectToMe commands received by each client per second

For a moderately big attack, the variables could be:

hub_no ~= 5
hub_clients ~= 5000
$CTMs/sec ~= 1
=> 25.000 connections/sec

Because of the big number of requests, the attack could easily be confounded with a SYN flood attack.

4. ATTACK MITIGATION

In this section we will present a number of methods that can be used on the victim side to mitigate the attack. Such an attack can last days or weeks and the administrator of the victim server has time to try different mitigation methods and to find the culprit. The ideas presented below come from practical experience during such an attack.

The first thing that comes in mind when dealing with a DC++ based DDoS attack is to block the IP addresses of the DC++ clients at network level using a firewall. But this method usually doesn't have any positive effect because the clients of a DC++ hub are very dynamic and the IP addresses change in matter of seconds or minutes. New clients from different IP blocks join the attacker hubs and the blacklisting method is not efficient. A big number of firewall rules would considerably slow down the firewall device resulting also in service unavailability.

After a little bit of thinking, another possible solution comes in mind. We can see that this is an IP based attack. So, in case of a web server, we could change periodically the IP address of the web server and modify the DNS resolution accordingly. Some attack tools are smarter but others are just for script kiddies. This solution could slow down a script kiddy that doesn't know how to modify the tool in order to target the current IP address of the victim.

In case of a web server being attacked, we can see that when it receives a non-HTTP packet, it waits for a timeout to expire before resetting the TCP connection. This timeout is specified in the configuration file of the web server. In case of Apache, this parameter can be found in httpd.conf file and is called *Timeout*. Its default value is 300 seconds (5 minutes!) and it should be lowered to 30 seconds. This way, the server will not keep its resources busy for that long time.

This kind of DC++ based DDoS attack is an application level attack. Older DDoS attacks were at network or transport level and they were easier to mitigate by using firewalls or other IP filtering mechanisms. But application level attacks are legitimate from the network or transport's layer point of view. So the only way to effectively mitigate this kind of attacks is by using an application level firewall that has **deep packet inspection** capabilities.

The easier and cheapest solution, considering a Linux/Unix machine, is to use the *string* module of *iptables*. This module searches the packet for a given string and can reset the connection if the string is found. The command is like:

IPTABLES -A INPUT -d $TARGET_IP -p tcp --dport $TARGET_PORT --tcp-flags ALL PSH,ACK -m string --algo bm --string MyNick --to 100 -j REJECT --reject-with tcp-reset

This method is effective for small scale DDoS attacks. But when the attack is bigger, the performance of the server lowers because it still has to make full TCP handshakes with the DC++ clients and reset the connections only when the first data packets come.

The most effective way to mitigate a DC++ based DDoS attack is to use an application level gateway, or proxy server, or a firewall that can do deep packet inspection. This kind of device knows how to interpret application level data (layer 7 protocols) and can reject malformed HTTP requests. But the disadvantage of this solution is that is very expensive.

Another (expensive) solution to this problem is to redirect the traffic of the attacked server to a Clean Pipe service provider. This type of service filtrates the malicious packets and allows only the legitimate ones to reach the customer.

5. FINDING THE ATTACKER

5.1 Method details - HubMonitor

In this type of attack is very difficult to detect the real attacker (the one running the attack tool) because the victim doesn't have any information about him. The packets that reach the victim are generated by the DC++ clients and contain no information about the real attacker.

But there is a nondeterministic method that a victim can use in order to gather evidence about the attacker during the attack and/or to stop the attack from its root point. By using this method, the victim could find the attacker hubs. When one of these hubs is found, the legal way to shut it down is to contact its service provider and give it the evidence of the attack and to contact hublist server owners to erase the hub from its list because of DDoS. Considering that the number of hubs participating in an attack is very small (comparing to the number of DC++ participating clients), shutting down one or two of them can significantly reduce the amount of attack traffic.

As we have seen until now, the real attacker is a modified DC++ client that sends $ConnectToMe requests to all of the other clients of the hubs to which it is connected. So all of the clients of the attacker hubs receive the $ConnectToMe messages that contain the victim's IP address and port. The method of detecting these attacker hubs uses also a modified DC++ client that connects simultaneously to multiple DC++ hubs and listens for $ConnectToMe ($CTM) messages that are targeted to the victim. In Figure 2 there is a visual description of this behavior.

I have implemented this method into a freeware tool called **HubMonitor** that can be found at [7]. HubMonitor is a command line tool that allows its operator to inspect automatically or manually the traffic of multiple hubs and detect the attacker ones. The algorithm used by HubMonitor is a parallelized version of this one (pseudo-code):

1. Read the list of hubs to connect to
2. For each hub H
   a. Connect to H
   b. Wait few minutes for $CTM packets
   c. IF receive $CTM and (IP == victim IP) AND (port == victim port)
      i. H is an attacker hub
      ELSE
      ii. H is not an attacker hub
   d. Disconnect from H
3. Exit

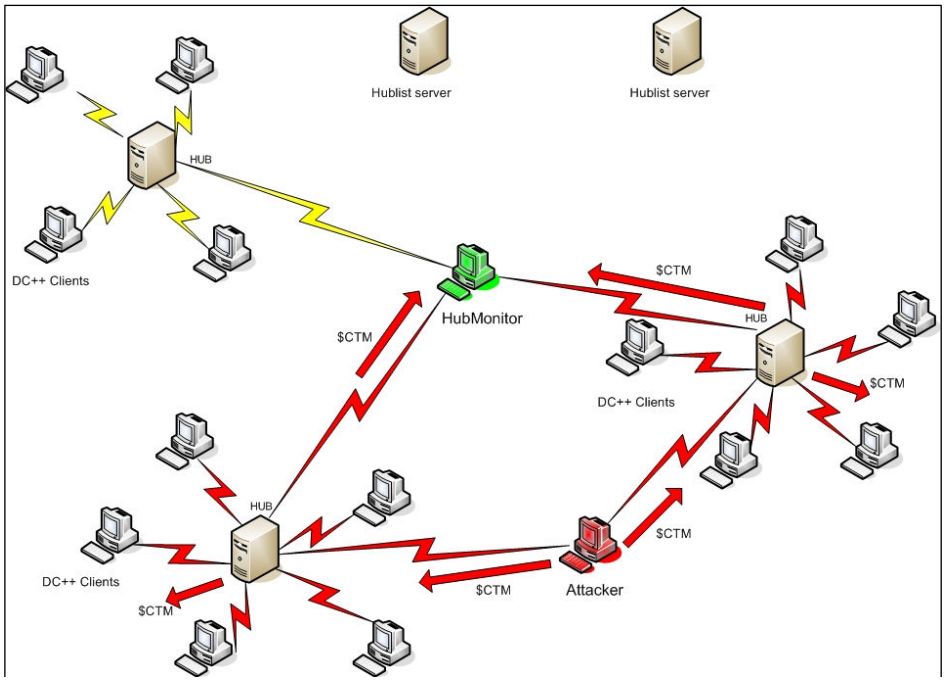

Figure 2. Detecting the attacker hubs

This very simplistic algorithm has one unknown element. This is the hub list to connect to. To get the hub list, the operator has at least two methods:

- get the hub list from a Hublist server
- compose manually the hub list based on observations

For the moment, HubMonitor only accepts a list of hubs given as an input file.

But getting the hub list from a hublist server can be done automatically too. Hub lists can be found on XML format or on a special hublist.config format. The advantage is that the tool will crawl through a very big number of hubs and the chances of finding the attacker hubs are greater. But the method is slow because connecting to each hub is time consuming. Hublist servers can be easily found with a simple web search. Example: dchublist.com, adchublist.com, dchublist.ro, dchubs.ro, hublist.top25.ro. For better results, the hublist servers from the country where the victim resides should be searched first. The algorithm below is an improved version of the one HubMonitor currently uses:

1. Read the list of Hublist servers
2. For each Hublist server HB
   a. Connect to HB
   b. Download hub_list
   c. Parse hub_list and extract INFO: hub name, port, minimum share size
   d. Add INFO to hub_info_list
3. For each hub H in hub_info_list
   a. Connect to H
   b. Wait few minutes for $CTM packets
   c. IF receive $CTM and (IP == victim IP) AND (port == victim port)
      i. H is an attacker hub

      ELSE
      ii. H is not an attacker hub

d. Disconnect from H
4. Exit

When you want to compose a list of hubs to verify (with HubMonitor), there are some observations that can be useful:

- Look at the nicknames from the $ConnectToMe packet. Many nicknames contain additional information such as country, town, ISP (ex. [RO][B][RDS] xxx, [IT]kkk, [FI] ppp, etc). Based on the predominant country you can focus on hublist servers from that country. The country can also be obtained from the IP address.
- If you are an ISP and your server is attacked, there is a big probability that some of the attacking DC++ clients to be in your network (verify by source IP address). Then you can call on the phone the person who has that IP address and ask him to tell you all the hub names to which he is connected in that moment. Those suspect hubs can be used as input to the tool described above and the chances to find the attacker hubs are greatly improved.
- Think about your enemies and analyze any messages received during the attack that can point to the hackers performing it. There are specialized hacker groups in the Internet that own DC++ hubs and can generate such an attack against your server. You can find their hubs and try the tool on them. More about this subject on the next paragraph.

5.2 DC++ teams

When this attack was often used by various malicious people from the Internet, there were a lot of vulnerable hubs out there. Now their number is considerably lower because the administrators have upgraded the hub software and they cannot be used in the attack anymore. Furthermore, the vulnerable hubs have been blacklisted on some hublist servers.

But many of the hubs from the Internet are owned by individuals who are part of various hacker teams. So they can downgrade anytime their hub software to a vulnerable version or use a custom made hub software in order to use them in DDoS attacks.

One of these teams is called TeamElite (‘][’€AM € LiT €) and is composed of various young persons around the world (Romania, Latvia, United Kingdom, etc). The most skilled of them are good programmers, capable of building their own hacking tools (including viruses, worms, etc) and have deep knowledge of computer systems. One of their leaders is Cristian Albu (a.k.a Lord_Zero, Vektor) and he is the writer of HexHub software and other hacking utilities.
TeamElite has often been involved in many illegal activities, including web site defacements and DDoS attacks [8]. They are the owners of a couple of DC++ hubs and they also work on Direct Connect protocol development [9][10].

In case of an attack, system administrators should consider checking the hubs of the known hacker teams – using the method presented in 5.1 - who might use them in the attack.

6. CONCLUSIONS

Peer-to-peer networks have a big attack potential from the malicious people’s point of view. When hackers find flaws in these networks, the generated attacks can be very powerful because of the big number of clients participating at them.
The DDoS attack described in this article is very dangerous because it is easy to implement by the hackers and is very difficult to defend against it on the victim side. Above that, the victim has no way

of knowing directly who the attacker is because it has no evidence in the received packets.
Regarding defense methods, the more efficient they are, the more expensive they are.

This article provided an in depth description of the DC++ based DDoS attack, including measures to defend against it and a method to find the attackers behind the attack.

7. REFERENCES:

[1] http://www.prolexic.com/content/moduleId/tPjJLKRF/article/aRQNVcBH.html
[2] http://www.securityfocus.com/news/11466
[3] http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date
[4] http://www.computerworld.com.au/index.php/id;1183357273
[5] http://jeremiahgrossman.blogspot.com/2008/04/csrf-ddos-skeleton-in-closet.html
[6] http://www.teamfair.info/DC-Protocol.htm
[7] http://stormsecurity.wordpress.com/2008/08/11/dc-and-ddos-attacks/
[8] www.cert.org/archive/pdf/CERT-FIactivities_CERT-FI-Pub.pdf
[9] http://nemesis.te-home.net
[10] http://portal.te-home.net