FEATURE 1

Tools of the DDoS Trade

Aleksander Czarnowski
Avet, Poland

Every once in a while we see a 'new technology' emerging which turns out to be nothing more than a bunch of old ideas in new packaging. This is the case with Distributed Denial of Service tools.

Like with viruses, the first versions of these tools were relatively simple – we had to wait a few months to see something more advanced, like TFN2000. Due to the widely available source codes of these tools, we can assume the existence of many 'private' versions which have never been used in the wild. This fact also made it easy to port tools like trinoo to the Win32 platform.

The Mechanism

Denial of Service (DoS) attacks are based on a very simple assumption that every system has limited resources. On the other hand, we tend to run more and more complicated network applications. More complicated applications usually mean more bugs and design/implementation flaws which can be exploited by an attacker.

It is worth mentioning that although it is possible to perform a DoS attack as a local user, in most cases such attacks are performed remotely. More often than not there is no point in crashing your machine locally while you are logged on to it.

DDoS Evolution

The first DoS tools were simple programs that exploited the misconfiguration of network services or bugs in the network application or TCP/IP stack. The next phase of their evolution was to create a shell from which an attacker could run few different DoS attacks. This method made it easier to perform an attack, but an attacker was still required to compile all the DoS attack tools.

Then came another generation of DoS tool – Targa. Targa was a compilation of a few different DoS attack tools integrated into one application. This made it easier to compile and run and to perform an attack. At this point it appeared that nothing more could be done to such tools except perhaps adding other types of attack and integrating them into one piece of code.

However, evolution must go on. The next logical step was the use of a distributed model to perform attacks. Again, this is nothing new. Distributed models have already been used in many security applications like intrusion detection systems or automatic virus analysis systems.

The Advantages of Distributed Models

Why would anyone want to use a distributed model to crash a machine remotely or to render the network connection unusable? The simple answer is that nowadays an attacker can bring down almost any network connection. Let me go back a little. To make a machine unusable remotely one has two choices; exploit some bugs in the configuration on the network application or flood the connection so that the machine or connection will exceed its limit.

The first choice is often used as it is very easy to find an exploit for a chosen platform. However, it can happen that there are no known bugs or that the machine is configured correctly. Sometimes there is quite simply no exploit to use against the machine. Even if there *is* an exploit, an attacker needs to identify the system platform that the machine is running. Not every exploit will work on every version of software. Some attacks can be blocked on an external router so it will never reach any machine in the Demilitarized Zone (DMZ) or LAN.

The second choice will always work: you only need to generate a stream of packets. If you can generate a number of packets greater than the target machine can handle, you have just performed a successful DoS attack. This can be seen sometimes with sites that have a high volume of traffic. They can be slow, or sometimes it is not possible to get a connection with the site. From an attacker's point of view the only challenge is the amount of network traffic that he must generate in order to bring the site down. Even in the case of middle-sized e-commerce sites it is not possible to flood the connection or external routers using a modem connection. More importantly, by using a distributed model, an attacker can remain unknown to the victims.

It is extremely hard to trace the origin of an attack but it is possible with the use of spoofed source IP addresses that are inserted into flooding packets. IPv4 allows such tricks. So, without the help of ISPs and telecommunications companies it is almost impossible to trace the real route of incoming packets. Since the attacker does not need to communicate with the victim machine, he also does not care about the packets received. An attacker can certainly spoof IP addresses, but from time to time he will need to use a real one.

Construction of DDoS Networks

To perform DDoS attacks an attacker needs to build a DDoS network. Such networks are built from three different types of component. The first component is an 'attacker'. Each network needs only one attacker. The attacker can then communicate with a 'handler' and an 'agent'. The attacker machine can *only* communicate with the agent through the handler – there is never any direct

communication between attacker and agent. This is significant because it means that an attacker does not leave any information about his identity on an agent's machine.

First, the attacker finds a handler machine. It must be sufficiently vulnerable that he can illegally achieve root privileges. Then the handler application can be copied and compiled. The next step is to find an agent machine – again this means breaking into it and installing the appropriate software. Once the attacker has gained root privileges he can install rootkits to hide his presence on the system. He can also install a sniffer to monitor network traffic and to move further into the network.

It might look very time- and energy-consuming, but in fact almost every part of this process can be automated. Finding handler and agent machines can be achieved by scanning a wide range of IP addresses. It is time-consuming but it does not need any human interaction. Then a list of vulnerable hosts (one can look for vulnerable versions of BIND or *SendMail* for example – identification of those is relatively simple and fast) must be generated.

The next step is to exploit a given vulnerability. With Unix, all this, plus installation and compilation on target machines can be performed using scripts. Almost the same functionality can be achieved under *Windows NT*.

Ready to Attack

When the network is set up, the attack can be launched. By issuing the appropriate command the attacker tells his handler to start the attack. The list of handlers is kept on the attacker's machine. Subsequently, the handler sends the same command to its agents. The list of agents is kept on the handler machine. Agents are always the ones who actually perform the attack. It is the role of an agent to generate a stream of packets with spoofed IP addresses.

Handlers do not communicate with each other, they only communicate with the attacker and their agents. Thanks to such a set-up, even if we were to find and isolate one particular agent we would still not know anything about the associated handlers.

Defending Your Network

A stream of packets will come from different places and all of the packets are sent to one IP address. Usually, the stream kills the external router long before it can reach a firewall. In such cases the firewall mechanism in place to protect from the attack is rendered useless.

If the stream can reach the firewall it is possible to defend the network by using techniques like dynamic host blocking, but this will only work in the case of several packets with the same source IP address. If every packet is sent with a random IP source address this technique will not work. Even if dynamic host blocking or any other technique is implemented on the firewall, it is still possible that all the firewall's resources will be consumed, again rendering it unusable. This problem is easy to solve: simply limit the number of connections being serviced.

Another method of defence is based on routing. The trick is to switch between two networks. One will be flooded by packets and the other one can be used normally. The drawback to this method is the need of ISP support.

If we can manage to disable the handlers, the agents will become useless. So, the first task is to isolate and remove the handlers. There are already tools like ZombieZapper that can send a signal to the DDoS network remotely to stop an attack. The current version of ZombieZapper works against trinoo (and its Win32 version), TFN, Stacheldraht and Shaft. Unfortunately, it will never work against TFN2K due to the way the TFN2K network communicates with each particular component.

Host Level Detection

While it is hard to detect such attacks at network level, it is easy to do it at host level. Furthermore, anti-virus software is ideal for the job due to the use of powerful and advanced scanning engines and wide infrastructure. Most scanners should not encounter problems even if DDoS tools employ the stealth techniques used currently by viruses or worms, or if/when polymorphic engines are built into them.

The recent development of worms and other malware for the Unix platform could pose a real threat. It is simple to hide a potential DDoS attack in a large chunk of virus code. If the virus code is analysed by some automatic analysis system its hidden 'weapon' will probably be missed. Even if a human took on the analysis, it is still possible that it would be missed.

What is even worse is the fact that viral worms can spread very quickly. Worms and viruses can be used as a very powerful method of building huge DDoS networks. The best case scenario would see a firewall filtering out all the control requests for DDoS network components like handlers or agents, rendering them unusable.

What is more important is that most machines are still running under *Windows 9x* which is very unstable and not as powerful as other Unix-based systems. While it is true to say that *Windows 9x* is insecure by default, its suitability for building DDoS network is somewhat limited.

Conclusion

We still do not have one proper method of dealing with DDoS attacks, and what we have seen up to now might not be the end of it. The danger from Unix malware is growing. More and more hacking tools are being ported to Win32. In too many environments a lack of security policy or incident response plan is the norm. Security in educational environments is an almost impossible task and computers will always have limited resources allocated to them.