Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks

Felix C. Freiling, Thorsten Holz*, and Georg Wicherski

Laboratory for Dependable Distributed Systems, RWTH Aachen University
http://www-i4.informatik.rwth-aachen.de/lufg/

Abstract. Denial-of-Service (DoS) attacks pose a significant threat to the Internet today especially if they are distributed, i.e., launched simultaneously at a large number of systems. *Reactive* techniques that try to detect such an attack and throttle down malicious traffic prevail today but usually require an additional infrastructure to be really effective. In this paper we show that *preventive* mechanisms can be as effective with much less effort: We present an approach to (distributed) DoS attack prevention that is based on the observation that coordinated automated activity by many hosts needs a mechanism to remotely control them. To prevent such attacks, it is therefore possible to identify, infiltrate and analyze this remote control mechanism and to stop it in an automated fashion. We show that this method can be realized in the Internet by describing how we infiltrated and tracked IRC-based *botnets* which are the main DoS technology used by attackers today.

1 Introduction

An important witness of the increasing professionalism in Internet crime are so called *Denial-of-Service* (DoS) attacks. A DoS attack is an attack on a computer system or network that causes a loss of service to users, typically the loss of network connectivity and services by consuming the bandwidth of the victim network or overloading the computational resources of the victim system [13]. Using available tools [5], it is relatively easy to mount DoS attacks against remote networks. For the (connection-oriented) Internet protocol TCP, the most common technique is called *TCP SYN flooding* [19,4] and consists of creating a large number of “half open” TCP connections on the target machine, thereby exhausting kernel data structures and making it impossible for the machine to accept new connections. For the (connectionless) protocol UDP, the technique of *UDP flooding* consists of overrunning the target machine with a large number of UDP packets thereby exhausting its network bandwidth and other computational resources.

* Thorsten Holz was supported by Deutsche Forschungsgemeinschaft (DFG) as part of the Graduiertenkolleg “Software for mobile communication systems” at RWTH Aachen University.

Like spam, it is well-known that DoS attacks are extremely hard to prevent because of their "semantic" nature. In the terminology of Schneier [18], semantic attacks target the way we assign meaning to content. For example, it is very hard to distinguish a DoS attack from a peak in the popularity of a large website. Using authentication it is in principle possible to detect and identify the single origin of a DoS attack by looking at the distribution of packets over IP addresses. However, it is almost impossible to detect such an attack if multiple attack hosts act in a coordinated fashion against their victim. Such attacks are called *Distributed Denial-of-Service* (DDoS). DDoS attacks are one of the most dangerous threats in the Internet today since they are not limited to web servers: virtually any service available on the Internet can be the target of such an attack. Higher-level protocols can be used to increase the load even more effectively by using very specific attacks, such as running exhausting search queries on bulletin boards or mounting *web spidering attacks*, i.e., starting from a given website and then recursively requesting all links on that site.

In the past, there are several examples of severe DDoS attacks. In February 2000, an attacker targeted major e-commerce companies and news-sites [9]. The network traffic flooded the available Internet connection so that no users could access these websites for several hours. In recent years, the threat posed by DDoS attacks grew and began to turn into real cybercrime. An example of this professionalism are blackmail attempts against a betting company during the European soccer championship in 2004 [2]. The attacker threatened to take the website of this company offline unless the company payed money. Similar documented cybercrime cases happened during other major sport events. Furthermore, paid DDoS attacks to take competitor's websites down were reported in 2004 [1]. These type of attacks often involve so called *botnets* [11], i.e., networks of compromised machines that are remotely controlled by an attacker. Botnets often consist of several thousand machines and enable an attacker to cause serious damage. Botnets are regularly used for DDoS attacks since their combined bandwidth overwhelms the available bandwidth of most target systems. In addition, several thousand compromised machines can generate so many packets per second that the target is unable to respond to so many requests.

Defensive measures against DDoS can be classified as either preventive or reactive [14]. Currently, reactive techniques dominate the arena of DDoS defense methods (the work by Mirkovic *et al.* [13] gives an excellent survey over academic and commercial systems). The idea of reactive approaches is to detect the attack by using some form of (distributed) anomaly detection on the network traffic and then react to the attack by reducing the malicious network flows to manageable levels [15]. The drawback of these approaches is that they need an increasingly complex and powerful sensing and analysis infrastructure to be effective: the approach is best if large portions of network traffic can be observed for analysis, preferably in real-time.

Preventive methods either eliminate the possibility of a DDoS attack altogether or they help victims to survive an attack better by increasing the resources of the victim in relation to those of the attacker, e.g., by introducing some form of strong authentication before any network interaction can take place (see for example work by Meadows [12]). Although being effective in theory, these survival methods always boil down to an arms race between attacker and victim where the party with more resources wins. In practice, it seems as if the arms race is always won by the attacker, since it is usually easier for him to increase his resources (by compromising more machines) than for the victim, which needs to invest money in equipment and network bandwidth.

Preventive techniques that aim at DDoS attack avoidance (i.e., ensuring that DDoS attacks are stopped before they are even launched) have received close to no attention so far. One reason for this might be the popular folklore that the only effective prevention technique for DDoS means to fix all vulnerabilities in all Internet hosts that can be misused for an attack (see for example Section 5 of [14]). In this paper we show that this folklore is wrong by presenting an effective approach to DDoS prevention that neither implies a resource arms race nor needs any additional (authentication) infrastructure. The approach is based on the observation that coordinated automated activity by many hosts is at the core of DDoS attacks. Hence the attacker needs a mechanism to remotely control a large number of machines. To prevent DDoS attacks, our approach attempts to identify, infiltrate and analyze this remote control mechanism and to stop it in an automated and controlled fashion. Since we attack the problem of DDoS at the root of its emergence, we consider our approach to be a root-cause method to DDoS defense.

It may seem unlikely that it is possible to automatically analyze and infiltrate a malicious remote control method crafted by attackers for evil purposes. However, we provide evidence of the feasibility of our strategy by describing how we successfully tracked and investigated the automated attack activity of botnets in the Internet. The idea of our methods is to “catch” malware using *honeypots*, i.e., network resources (computers, routers, switches, etc.) deployed to be probed, attacked, and compromised. Honeypots run special software which permanently collects data about the system behavior and facilitates automated post-incident forensic analysis. From the automated analysis we derive the important information necessary to observe and combat malicious actions of the botnet maintainers. In a sense, our approach can be characterized as turning the methods of the attackers against themselves.

The paper is structured as follows: Section 2 gives a brief overview over botnets and their usage for DDoS attacks. In Section 3 we introduce a general methodology to prevent DDoS attacks and exemplify a technical realization in Section 4. We present our results in Section 5 and conclude this paper with Section 6.

2 Distributed Denial-of-Service Using Botnets

In this section we give a brief overview over botnets and how they can be used to mount DDoS attacks. More technical details can be found in [22]. A botnet is a network of compromised machines running programs (usually referred to as *bot*, *zombie*, or *drone*) under a common *Command and Control* (C&C) infrastructure. Usually, the controller of the botnet compromises a series of systems using various tools and then installs a *bot* to enable remote control of the victim computer via Internet Relay Chat (IRC).

Newer bots can even automatically scan whole network ranges and propagate themselves using vulnerabilities and weak passwords on other machines. After successful invasion, a bot uses Trivial File Transfer Protocol (TFTP), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), or CSend (an IRC extension to send files to other users) to transfer itself to the compromised host. The binary is started and tries to connect to the hard-coded master IRC server on a predefined *port*, often using a *server password* to protect the botnet infrastructure. This server acts as the C&C server to manage the botnet. Often a *dynamic DNS name* is provided rather than a hard coded IP address, so the bot can be easily relocated. Using a *specially crafted nickname*, the bot tries to join the *master's channel*, often using a *channel password*, too. In this channel, the bot can be remotely controlled by the attacker.

Commands can be sent to the bot in two different ways: via sending an ordinary command directly to the bot or via setting a special topic in the channel that all bots interpret. For example, the topic

```
advscan lsass 200 5 0 -b
```

tells the bots to spread further with the help of a known vulnerability (the Windows *lsass* vulnerability). The bots should start 200 concurrent threads that should scan with a delay of 5 seconds for an unlimited time (parameter `0`). The scans should target machines within the same Class B network (parameter `-b`). As another example, the topic

```
http.update http://<server>/rBot.exe c:\msy32awds.exe 1
```

instructs the bots to download a binary from the Internet via HTTP to the local filesystem and execute it (parameter `1`).

If the topic does not contain any instructions for the bot, then it does nothing but idling in the channel, awaiting commands. That is fundamental for most current bots: they do not spread if they are not told to spread in their master's channel. Figure 1 depicts the typical communication flow in a botnet.

In order to remotely control the bots, the controller of a botnet has to authenticate himself before issuing commands. This authentication is done with the help of a classical authentication scheme. At first, the controller has to login with his username. Afterwards, he has to authenticate with the correct password to approve his authenticity. The whole authentication process is only allowed from a predefined domain, so that only certain people can start this process. Once an attacker is authenticated, he has complete control over the bots and can execute arbitrary commands.

Today, botnets are most often used to mount DDoS attacks in the Internet. All common bots include several different possibilities to participate in these attacks. Most commonly implemented, and also very often used, are TCP SYN [19,4] and UDP flooding attacks. For example, the command

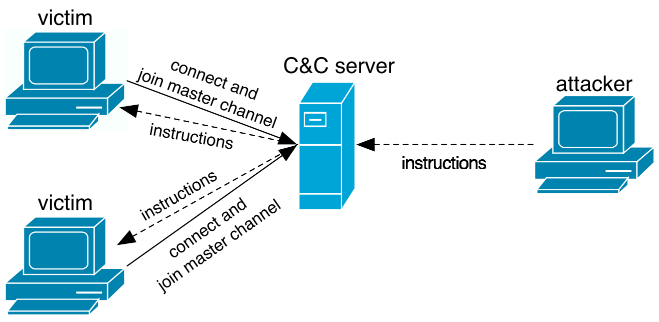

Fig. 1. Communication flow in a botnet

```
ddos.syn XXX.XXX.XXX.XXX 80 600
```

instructs the bots within the botnet to start a TCP SYN flooding attack against the specified IP address against TCP port 80 for 600 seconds. Another example is the following command:

```
udp XXX.XXX.XXX.XXX 18000 50000 100
```

It instructs the bots to mount a UDP flooding attack against the specified target with 18,000 packets of a size of 50,000 bytes using a delay of 100 milliseconds between each packet. Note that the C&C IRC server that is used to connect all bots is in most cases also a compromised machine.

3 Preventing Distributed Denial-of-Service Attacks

In this section we introduce a general methodology to prevent DDoS attacks. It is based on the following line of reasoning:

1. To mount a successful DDoS attack, a large number of compromised machines are necessary.
2. To coordinate a large number of machines, the attacker needs a remote control mechanism.
3. If the remote control mechanism is disabled, the DoS attack is prevented.

We will substantiate this line of reasoning in the following paragraphs.

3.1 A Large Number of Machines Is Necessary

Why does an attacker need a large number of machines to mount a successful DDoS attack? If an attacker controls only few machines, a DDoS attack is successful only if the total resources of the attacker (e.g., available bandwidth or possibility to generate many packets per second) are greater than the resources of the victim. Otherwise the victim is able to cope with the attack. Hence, if this requirement is met, the attacker can efficiently overwhelm the services offered by the victim or cause the loss of network connectivity.

Moreover, if only a small number of attacking machines are involved in an attack, these machines can be identified and counteractive measures can be applied, e.g., shutting down the attacking machines or blocking their traffic. To obfuscate the real address of the attacking machines, *IP spoofing*, i.e., sending IP packets with a counterfeited sender address, is often used. Furthermore, this technique is used to disguise the actual number of attacking machines by seemingly increasing it. However, IP spoofing does not help an attacker to conduct a DDoS attack from an efficiency point of view. It does not increase the available resources, but it even reduces them due to computing efforts for counterfeiting the IP addresses. In addition, several ways to detect and counteract spoofed sender address exist, e.g., ingress filtering [7], packet marking [20], or ICMP traceback [3,17]. The IP distribution of a large number of machines in different networks makes ingress filter construction, maintenance, and deployment much more difficult. Additionally, incident response is hampered by a high number of separate organizations involved.

So control over a large number of machines is necessary for a successful DDoS attack.

3.2 A Remote Control Mechanism Is Necessary

The success of a DDoS attack depends on the volume of the malicious traffic as well as the time this traffic is directed against the victim. Therefore, it is vital that the actions of the many hosts which participate in the attack are well-coordinated regarding the type of traffic, the victim's identity, as well as the time of attack.

A cautious attacker may encode all this information directly into the malware which is used to compromise the zombies that form the DDoS network. While this makes him harder to track down, the attacker loses a lot of flexibility since he needs to plan his deeds well in advance. Additionally, this approach makes the DDoS attack also less effective since it is possible to analyze the malware and then reliably predict when and where an attack will take place. Therefore it is desirable to have a channel through which this information can be transferred to the zombies on demand, i.e., a remote control mechanism.

A remote control mechanism has many more advantages:

1. The most effective attacks come by surprise regarding the time, the type and the target of attack. A remote control mechanism allows an attacker to react swiftly to a given situation, e.g., to mount a counterattack or to substantiate blackmail threats.
2. Like any software, malware is usually far from perfect. A remote control mechanism can be used as an automated update facility, e.g., to upgrade malware with new functionality.

In short, a DDoS attack mechanism is only effective if an attacker has some type of remote control over a large number of machines. Then he can issue commands to exhaust the victim's resources at many systems, thus successfully attacking the victim.

3.3 Preventing Attacks

Our methodology to mitigate DDoS attacks aims at manipulating the root-cause of the attacks, i.e., influencing the remote control network. Our approach is based on three steps:

1. Infiltrating the remote control network.
2. Analyzing the network in detail.
3. Shutting down the remote control network.

In the first step, we have to find a way to smuggle an *agent* into the control network. In this context, the term agent describes a general procedure to mask as a valid member of the control network. This agent must thus be customized to the type of network we want to plant it in. The level of adaptation to a real member of the network depends on the target we want to infiltrate. For instance, to infiltrate a botnet we would try to simulate a valid bot, maybe even emulating some bot commands.

Once we are able to sneak an agent into the remote control network, it enables us to perform the second step, i.e., to observe the network in detail. So we can start to monitor all activity and analyze all information we have collected.

In the last step, we use the collected information to shut down the remote control network. Once this is done, we have deprived the attacker's control over the other machines and thus efficiently stopped the threat of a DDoS attack with this network. Again, the particular way in which the network is shut down depends on the type of network.

3.4 Discussion

The methodology described above can be applied to different kinds of remote control networks and is thus very general. The practical challenge of the methodology is to automate the infiltration and analysis process as much as possible. In all these cases, the zombies need to establish a communication channel between themselves and the attacker. If it is possible to "catch" this malware in a controlled way, it is possible to extract a lot of information out of it in an automated fashion. For example, if contact to the attacker is set up by establishing a regular network connection, the network address of the attacker's computer can be automatically collected.

To many readers, the methodology may sound like coming directly from a James Bond novel and it is legitimate to ask for evidence of its feasibility. In the following section we give exactly this evidence. We show that this method can be realized in the Internet by describing how we infiltrated and tracked IRC-based botnets which are the main DDoS technology used by attackers today.

4 An Example: Tracking Botnets

In this section we exemplify a technical realization of the methodology we introduced above. We present an approach to track and observe botnets that is able to prevent DDoS attacks.

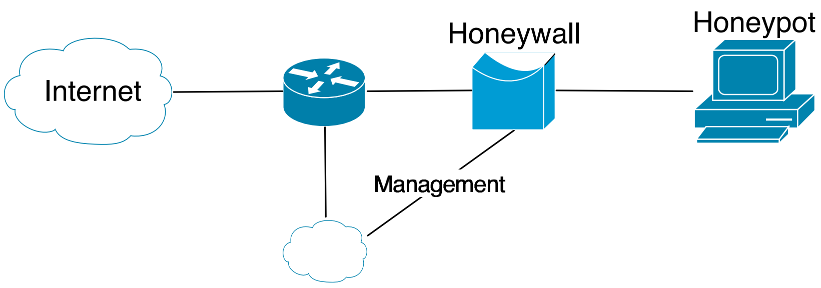

Fig. 2. Setup for tracking botnets

As already stated in the last section, tracking botnets is clearly a multi-step operation: First one needs to gather some data about an existing botnet. This can for instance be obtained with the help of botnets or via an analysis of captured malware. With the help of this information it is possible to smuggle a client into the network.

We first introduce two techniques to retrieve the necessary information from a botnet which enables us to infiltrate in it. The necessary information includes:

– DNS/IP-address of IRC server and port number.
– Password to connect to IRC-server (optional).
– Nickname of a bot and `ident` [10] structure.
– Name of IRC channel to join and (optional) channel password.

The first method to retrieve this information is based on *honeypot* technology and is presented in Section 4.1. The second method is more lightweight and presented in Section 4.2. Then we describe the observation and analysis process in which we collected further information (Section 4.3). Finally, in Section 4.4 we give a small overview of possible ways to shut down a botnet.

4.1 Collecting Malware with Honeypots

A *honeypot* is a network resource (computers, routers, switches, etc.) deployed to be probed, attacked, and compromised. A *honeynet* is a network of honeypots. Honeypots run special software which permanently collects data about the system behavior and facilitates automatic post-incident forensic analysis. The collected data enables us to determine the necessary information about an existing botnet. A detailed introduction to honeypots can for example be found in [6].

Using a so called *GenII Honeynet* [21] containing some Windows honeypots, we are able to collect all necessary information. We deployed a typical GenII Honeynet with some small modifications as depicted in Figure 4.1.

The Windows honeypot runs an unpatched version of Windows 2000 or Windows XP. This system is thus very vulnerable to attacks. It is located within the internal network of RWTH Aachen University. On average, the expected lifespan of the honeypot is less than ten minutes. After this small amount of time, the honeypot is often successfully exploited by automated malware. The shortest compromise time was only a few seconds: Once we plugged the network cable in, a bot compromised the machine and installed itself on the machine.

As explained in the previous section, a bot tries to connect to the C&C server to obtain further commands once it successfully attacked the honeypot. This is where the *Honeywall* comes into play. The Honeywall is a transparent bridge that enables the two tasks *Data Control* and *Data Capture*. Due to the Data Control facilities, it is possible to control the outgoing traffic. Using available tools for Data Control we can replace all suspicious in- and outgoing messages. A message is suspicious if it contains typical IRC messages for command and control, for example " `TOPIC` ", " `PRIVMSG` ", or " `NOTICE` ". Thus we are able to inhibit the bot from accepting valid commands from the master channel. It can therefore cause no harm to others and therefore we have caught a bot inside our Honeynet. As a side effect, we can also derive all necessary sensitive information for a botnet from the data we have obtained up to that point in time: The Data Capture capability of the Honeywall allows us to determine the DNS/IP-address the bot wants to connect to and also the corresponding port number. In addition, we can derive from the Data Capture logs the nickname, the ident information, the server's password, channel name, and the channel password as well. So we have collected all necessary information and the honeypot can catch further malware. Since we do not care about the captured malware for now, we rebuild the honeypot every 24 hours to have a "clean" system every day. This has proven to be a good time span since after this amount of time the honeypot tends to become unstable due to installed malware.

4.2 Collecting Malware with Mwcollect

The approach described in the previous section works, but has several drawbacks:

– A honeypot will crash regularly if the bot fails to exploit the offered service, e.g. due to a wrong offset within the exploit.
– The honeypot itself has to be closely monitored in order to detect changes on the system. Furthermore, these changes have to be analyzed carefully to detect malware.
– The approach does not scale well; observing a large number of IP addresses is difficult.

To overcome these limitations, we developed a program called *mwcollect* to capture malware in non-native environments. This tool simulates several vulnerable services and waits for them to be exploited. It is comparable to a *low-interaction honeypot* like *honeyd* [16]. In contrast to *honeyd* it is tailored to collecting of malware and offers possibilities that *honeyd* cannot offer, e.g. better packet handling and more flexibility.

mwcollect is based upon a very flexible and modularized design. The core module – the actual daemon – handles the network interface and coordinates the actions of the other modules. Furthermore, the core module implements a *sniffer mode* which records all traffic to a special log file. This can for example be useful if an unknown exploit is detected that needs to be further analyzed.

Several modules, which register themselves in the core, fulfill the actual tasks. There are basically four types of modules:

- *Vulnerability modules* open some common vulnerable ports (e.g. TCP Port 135 or 2745) and simulate the vulnerabilities according to these ports.
- *Shellcode parsing modules* analyze the shellcode, an assembly language program which executes a shell, received by one of the vulnerability modules. These modules try to extract generic URLs from the shellcode.
- *Fetch modules* simply download the files specified by an URL. These URLs do not necessarily have to be HTTP or FTP URLs, but can also be TFTP or other protocols.
- *Submission modules* handle successfully downloaded files, for example by writing it to disk or submitting it to a database.

Vulnerability modules seem to be the most important part of *mwcollect*, but in fact they are not more important than every other module, they all require each other. Moreover, the vulnerable service emulation is not very sophisticated, but functional: Often malware does not require an indistinguishable emulation of a real service but an approximation of it. In most cases it is thus sufficient to provide some minimal information at certain offsets in the network flow. This information is used by the malware to calculate the offsets it can use to exploit the service. Upon successful exploitation, the payload of the malware is passed to another kind of modules.

Currently there is only one *shellcode parsing module* that is capable of analyzing all shellcodes we have found up to now. The module first recursively detects *XOR decoders* in a generic way. An XOR decoder is a common way to encrypt the actual shellcode in order to evade intrusion detection systems. Afterwards the module decodes the code itself according to the computed key and then applies some pattern detection, for example `CreateProcess` and `URLDownloadToFileA` detection patterns. The results are further analyzed and if an URL is detected, it is passed to the fetch modules. A module that parses shellcodes in an even more generic way by emulating a Windows Operating System environment is currently under development.

Fetch modules have the simple task of downloading files from the Internet. There are currently three different fetch modules: one for TFTP, one for generic HTTP and FTP URLs and finally one for CSend and similar transfer methods used by different species of bots.

Finally, *submission modules* handle successfully downloaded files. Currently there are three different types of submission modules:

- A module that stores the file in a configurable location on the filesystem and is also capable of changing the ownership.
- A module that submits the file to a central database to enable distributed sensors with central logging interface
- A module that checks the file with the help of different anti-virus scanners for known malware. Optionally this module sends an alert to enable an early warning system. Therefore, *mwcollect* can also be seen as a kind of intrusion detection system.

Two further features of *mwcollect* are important to efficiently collect malware: *virtualized filesystem* and *shell emulation*.

A common technique to infect a host via a shell is to write commands for downloading and executing malware into a temporary file and then execute this file. Therefore a *virtual filesystem* was implemented to enable this type of attacks. Every shell session has its own virtual filesystem so concurrent infection sessions using similar exploits do not conflict. The temporary file is analyzed and the malware is downloaded from the Internet in an automated way.

Some Malware does not spread by download shellcodes but by providing a shell to the attacker. Therefore it is sometimes required to spawn and emulate a Windows shell. *Shell emulation* is centralized in the core module since only one type of shell is emulated. However, modules can register additional commands that extend the possibilities for the malware. *mwcollect* currently simulates a rudimentary shell and implements several commands: `echo`, `ftp.exe` and `tftp.exe`, as well as batch file execution.

The big advantage of using *mwcollect* to collect malware is clearly both stability and scalability: A bot trying to exploit a honeypot running Windows 2000 with payload that targets Windows XP will presumably crash the service. In most cases, the honeypot will be forced to reboot. In contrast to this, *mwcollect* can be successfully exploited by all of those tools and hence catch a lot more binaries this way. Furthermore, *mwcollect* can listen on many IP addresses in parallel. We tested the program with 256 IP addresses and it scaled well.

To derive the sensitive information of the botnet from the collected malware, a further analysis is necessary. A possible way to extract the information from the captured malware is *reverse engineering*, the process of carefully analyzing a program without having its source code. This process is time consuming, but we have developed some techniques that enables us to extract the information within a few minutes. A better approach is an automated analysis with the help of a honeynet. The setup depicted in Figure 4.1 can be used for this purpose. Upon startup, the Windows honeypot downloads a piece of malware from a database located somewhere in the Internet. It executes the file and reboots itself after a few minutes. During this time span, the bot installs itself on the honeypots and connects to the C&C server. With the help of the Honeywall, we are again able to extract all necessary information. In addition, the honeypot resets the hard disk during each reboot so that a clean image is booted each time.

In a third approach, we are currently implementing a virtual machine that implements an environments in which the bot can be executed. This virtualization emulates a Windows environment and enables us to efficiently analyze the malware.

4.3 Observing Botnets

Once we have collected all sensitive information of the botnet, we start to infiltrate the botnet as we have all the necessary data. In a first approach, it is possible to setup a normal IRC client and try to connect to the network. If the operators of the botnets do not detect this client, logging of all commands can be enabled. This way, all bot commands and all actions can be observed. If the network is relatively small (i.e., less then 50 clients), there is a chance that the bogus client will be identified since it does not answer to valid commands. In this case, the operators of the botnets tend to either ban and/or DDoS the suspicious client.

But there are many problems with this approach: Some botnets use very strongly stripped down C&C server which is not RFC compliant so that a normal IRC client can not connect to this network. A possible way to circumvent this situation is to find out what the operator has stripped out, and modify the source code of the IRC client to override it. Furthermore, this approach does not scale very well. Tracking more than just a few botnets is not possible since a normal IRC client will be overwhelmed with the amount of logging data and it does not offer a concise overview of what is happening.

Therefore we use an IRC client optimized for botnet tracking called *drone*. This software was developed by two members of the German Honeynet Project and offers several decent techniques for observing botnets:

– Multi-server support to track a large number of botnets in parallel
– Excessive debug-logging interface so that it is possible to get information about RFC non-compliance issues very fast and fix them in the client
– Automated downloading of malware identified within the botnet
– Modular interface to un/load modules at runtime

Furthermore, *drone* is capable of using SOCKS v4 proxies so we do not run into problems if it's presence is noticed by an attacker in a botnet. The SOCKS v4 proxies are on dial-in accounts in different networks so that we can easily change the IP addresses of our infiltrated bot.

When observing more than a couple of networks, we began to check if some of them are linked, and group them if possible. Link-checking is simply realizable: our client just joins a specific channel on all networks and detects if more than one client is there, thus concluding the the networks controlled by several C&C servers are linked. Surprisingly, many networks are linked.

4.4 Preventing DDoS Attacks Caused by Botnets

Several ways to prevent DDoS attacks caused by botnets exist that we want to sketch in this section. Since we observe the communication flow within the botnet, we are also able to observe the IP addresses of the bots unless this information is obfuscated, e.g., by modifying the C&C server. Thus one possible way to stop DDoS attacks with this methodology is to contact the owner of the compromised system. This is however a tedious and cumbersome job, since many organizations are involved and these organizations are spread all over the world. In addition, the large number of bots make this approach nearly infeasible, only an automated notification system could help.

Another approach to prevent DDoS attacks caused by botnets aims at stopping the actual infrastructure, in particular the C&C server, since this component is vital for the remote control network. One possible way to stop the C&C

server is described in [8]: Most botnets use a dynamic DNS name instead of a hard-coded IP address for the C&C server. So if the DNS name is changed so that it resolves to an IP address in a private subnet as defined in RFC 1918, the bots are not able to connect to the central server. Thus the remote control network is efficiently shut down. For this approach, the assistance of the DNS provider is needed, though.

In addition, the collected information about botnets enable another way to stop the botnet. We know the IP address of the C&C server and are thus able to locate it. If the operator of the network cooperates, it is possible to shut down this server and thus shutting down the remote control network.

5 Results

In this section we present some of the findings we obtained through our observation of botnets. Data is sanitized so that it does not allow one to draw any conclusions about specific attacks against a particular system, and protects the identity and privacy of those involved. The information about specific attacks and compromised systems was forwarded to DFN-CERT (Computer Emergency Response Team) based in Hamburg, Germany.

The results are based on the observations collected with just two sensors. One sensors uses the approach depicted in Section 4.1 and is located within the network of RWTH Aachen University. The other sensor is based on the technique and software introduced in Section 4.2 and is located within a dial-in network of a German ISP.

We start with some statistics about the botnets we have observed in the last five months:

– *Number of botnets*: We were able to track about 180 botnets during the last five months. Some of them went offline (e.g. C&C server went offline or inexperienced attackers) and at the time of writing (March 2005) we are tracking about 60 active botnets.
– *Number of hosts*: During these few months, we saw more than 300,000 unique IP addresses joining at least one of the channels we monitored. Seeing an IP means here that the C&C server was not modified to not send a `JOIN` message for each joining client. If an IRC server is modified not to show joining clients in a channel, we do not see IPs here. Furthermore some IRC server obfuscate the joining clients IP address and obfuscated IP addresses do not count as seen, too. This shows that the threat posed by botnets is probably worse than originally believed. Even if we are very optimistic and estimate that we track a significant percentage of all botnets and all of our tracked botnet C&C servers are not modified to hide `JOIN`s or obfuscate the joining clients IPs, this would mean that more than one million hosts are compromised and can be controlled by malicious attackers.
– *Typical size of Botnets*: Some botnets consist of only a few hundred bots. In contrast to this, we have also monitored several large botnets with up to 50,000 hosts. The actual size of such a large botnet is hard to estimate.

Often the attackers use heavily modified IRC servers and the bots are spread across several C&C servers which are linked together to form a common remote control network. We use link-checking between IRC servers to detect connections between different botnets that form one large botnet. Thus we are able to approximate the actual size.
As a side note: We know about a home computer which got infected by 16 different bots, so its hard to make an estimation about world bot population here.
– *Dimension of DDoS-attacks*: We are able to make an educated guess about the current dimension of DDoS-attacks caused by botnets. We can observe the commands issued by the controllers and thus see whenever the botnet is used for such attacks. From the beginning of November 2004 until the end of March 2005, we were able to observe 406 DDoS-attacks against 179 unique targets. Often these attacks targeted dial-up lines, but there are also attacks against bigger websites or other IRC server.
– *Spreading of botnets*: Commands issued for further spreading of the bots are the most frequent observed messages. Commonly, Windows systems are exploited and thus we see most traffic on typical Windows ports used for file sharing.
– *"Updates" within botnets*: We also observed updates of botnets quite frequently. Updating in this context means that the bots are instructed to download a piece of software from the Internet and then execute it.

We conclude that our general methodology described in Section 3 is feasible and the automated approach described in Section 4.2 is effective. We collected more than 5500 binaries (about 800 unique ones) with mwcollect in just one week on a single sensor. This sensor has only one IP address and is connected to the Internet via a German DSL dial-in provider with 4 MBit downstream and 2 MBit upstream. About five percent of the unique files were broken due to failures during TFTP transfer. We are currently in the process of analyzing these files. Once we have implemented a virtualization mechanism to efficiently and automatically analyze the collected files, we hope to be able to significantly increase the number of botnets we observe. In addition, this information can be used to prevent DDoS attacks by shutting down the C&C server.

6 Conclusion and Further Work

DDoS attacks have become increasingly dangerous in recent years and we are observing a growing professionalism in the type of Internet crime surrounding DDoS. In this paper we have introduced a technique for DDoS attack prevention that neither implies a resource arms race nor needs any additional infrastructure. In contrast to previous work in this area our approach is preventive instead of reactive. Our technique attacks a root-cause of DDoS attacks: in order to be effective, an attacker has to control a large number of machines and thus needs a remote control network. Our methodology aims at shutting down this control network by infiltrating it and analyzing it in detail.

We have exemplified a technical realization of this methodology considering as example the tracking of IRC-based botnets. Such a botnet is a network of compromised machines that can be remotely controlled by an attacker through Internet relay chat technology. Due to their immense size (tens of thousands of systems can be linked together), these botnets pose a severe threat to the Internet community, e.g., since their aggregated resources can be used to overwhelm most targets with a DDoS attack. We have shown that an automation of this approach is possible to a high degree. With the help of honeypots, i.e., network resources deployed to be compromised, we are able to automate the process of collecting sensitive information of the remote control network by automatically "collecting" malware. Via an automated analysis of the captured binaries we are furthermore able to extract the sensitive information that allow to shut down the control network.

With the help of just two sensors we were able to track a significant number of botnets within a few months. In the future we want to analyze how good our approach scales. Therefore we want to deploy more sensors within different networks. In addition, we aim at speeding up the automated analysis process so that it becomes even more effective. This can for example be achieved with the help of a generic shellcode parser or a virtual machine that analyzes and extracts the sensitive information from the captured binaries.

Moreover, the data we captured while observing the botnets show that these control networks are used for more than just DDoS attacks. Possible usages of botnets can be categorized as listed below. And since a botnet is nothing more then a tool, there are most likely other potential uses that we have not listed:

- *Spamming*: Some bots offer the possibility to open a SOCKS v4/v5 proxy – a generic proxy protocol for TCP/IP-based networking applications – on a compromised machine. After having enabled the SOCKS proxy, this machine can then be used for nefarious tasks such as sending bulk email (*spam*) or phishing mails. With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of spam. Some bots also implement a special function to harvest email-addresses from the victims.
- *Attacking IRC Chat Networks*: Botnets are also used for DDoS attacks against Internet Relay Chat (IRC) networks. Popular among attackers is especially the so called *clone attack*: In this kind of attack, the controller orders each bot to connect a large number of clones to the victim IRC network. The victim is overwhelmed by service request from thousands of (cloned) bots.
- *Manipulating online polls/games*: Online polls/games are getting more and more attention and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way.
- *Sniffing Traffic*: Bots can also use a packet sniffer to watch for interesting clear-text data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords.

– *Keylogging*: If the compromised machine uses encrypted communication channels (e.g. HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless since the appropriate key to decrypt the packets is missing. But most bots also implement functions to log keystrokes. With the help of a keylogger it is very easy for an attacker to retrieve sensitive information.
– *Harvesting of information*: Sometimes we can also observe the harvesting of information from all compromised machines. With the help of special commands the operator of the botnet is able to request a list of sensitive information from all bots.

With our method we can shut down the root-cause of all of these types of nuisances, and hence our method is not restricted to combat DDoS.

In the future, we hope to develop more advanced honeypots that help us to gather more information about threats such as botnets. Examples include *client-side honeypots* that actively participate in networks (e.g., by crawling the web, idling in IRC channels, or using P2P-networks) or modify honeypots so that they capture malware and send it to anti-virus vendors for further analysis. It is also to be expected that future botnets will use communication facilities other than IRC (like potentially decentralized P2P-communication or covert channels). Our methodology seems valid also for these scenarios, although more research in this area is still needed.

Acknowledgments

We thank Lucia Draque Penso for reading a previous version of this paper and giving valuable feedback that substantially improved it's presentation. In addition, we would like to thank the authors of *drone*, an IRC client optimized for botnet tracking, for their hard word in writing the program.

References

1. FBI report on Operation Cyberslam. Internet: `http://www.reverse.net/operationcyberslam.pdf`, Accessed March 2005, February 2004.
2. Hacker threats to bookies probed. Internet: `http://news.bbc.co.uk/1/hi/technology/3513849.stm`, Accessed March 2005, February 2004.
3. S. M. Bellovin. ICMP traceback messages, March 2001. Internet Draft.
4. Computer Emergency Response Team. CERT advisory CA-1996-21 TCP SYN Flooding Attacks. Internet: `http://www.cert.org/advisories/CA-1996-21.html`, 1996.
5. D. Dittrich. Distributed Denial of Service (DDoS) attacks/tools resource page. Internet: `http://staff.washington.edu/dittrich/misc/ddos/`, 2000.
6. M. Dornseif, F. C. Gärtner, and T. Holz. Vulnerability assessment using honepots. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 4(27):195–201, 2004.
7. P. Ferguson. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, May 2000. Request for Comments: RFC 2827.

8. T. Fischer. Botnetze. In *Proceedings of 12th DFN-CERT Workshop*, March 2005.
9. L. Garber. Denial-of-service attacks rip the Internet. *Computer*, 33(4):12–17, April 2000.
10. M. S. Johns. Identification protocol, February 1993. Request for Comments: RFC 1413.
11. B. McCarty. Botnets: Big and bigger. *IEEE Security & Privacy*, 1(4):87–90, 2003.
12. C. Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, pages 4–13. IEEE Computer Society Press, 1998.
13. J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall PTR, 2004.
14. J. Mirkovic and P. Reiher. A taxonomy of DDoS attacks and defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2):39–54, Apr. 2004.
15. J. Mirkovic, M. Robinson, P. Reiher, and G. Kuenning. Alliance formation for DDoS defense. In *Proceedings of the New Security Paradigms Workshop 2003.* ACM SIGSAC, Aug. 2003.
16. N. Provos. A virtual honeypot framework. In *Proceedings of 13th USENIX Security Symposium*, 2004.
17. S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, August 2000.
18. B. Schneier. Inside risks: semantic network attacks. *Communications of the ACM*, 43(12):168–168, Dec. 2000.
19. C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society, IEEE Computer Society Press, May 1997.
20. D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of IEEE Infocom 2001*, April 2001.
21. The Honeynet Project. Know Your Enemy: GenII Honeynets, November 2003. `http://www.honeynet.org/papers/gen2/`.
22. The Honeynet Project. Know your Enemy: Tracking Botnets, March 2005. `http://www.honeynet.org/papers/bots`.