GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

Trends in Bot Net Command and Control

GIAC General Security Essentials (GSEC)
Practical Assignment Version 1.4C
Option 1 – Research on Topics in Information Security

Will Longman
March 17, 2005

© SANS Institute 2000 - 2005, Author retains full rights.

Abstract

Since 1999, hackers have harnessed the power of compromised computer systems by arraying them into networks that are intended to launch attacks and gather intelligence. The ability of these illicit networks to function while remaining covert is primarily a function of their command and control architecture. That architecture has evolved in sophistication over the years in terms of detectability and overall robustness as attackers attempt to evade improved defensive security strategies and aggressive rival hacker takeovers. This paper will follow that evolution by analyzing representative command and control topologies and then providing a glimpse into some new "bleeding edge" approaches.

Introduction

Hackers assemble captive computer systems into networks in order to accomplish a variety of tasks, initially and most notably the launching of distributed denial of service (DDOS) attacks. Originally known as DDOS nets, these networks are now more commonly called bot nets in reference to the automated software agents or bots that are embedded in each of the captive hosts. The ability of these networks to carry out their functions and remain covert is primarily a function of their command and control or "c^2" architecture.

Bot net c^2 systems all share a common objective – they provide the network with a communications channel and a command set protocol that together enable the individual bots to accomplish the intended tasks. Each of the representative c^2 topologies detailed in this paper will be shown to possess the following three basic attributes in support of this objective:

- A control channel to provide a conduit for reports, commands, and acknowledgements.
- A method for the bot to join the control channel.
- A command shell that will enable the bot to receive commands and transmit acknowledgements as well as status reports.

© SANS Institute 2000 - 2005 Author retains full rights.

The manner in which a specific bot net implements these attributes can be used to place them in an evolutionary taxonomy. Using this taxonomic approach, c^2 topologies can be divided into three main classes: handler/agent, IRC-based, and peer-to-peer.

Handler / Agent Control Topology

As documented by the CERT Coordination Center [1], the first real DDOS net command and control architecture emerged in 1999 and was a three tiered structure referred to as a handler / agent topology. At the top level was the "master," a system operated directly by the hacker and which in turn controlled a small number of "handler" systems. These handlers formed the middle tier and each served to relay the master's commands to a subset of the remaining captive host systems. That latter group of hosts comprised the third tier and contained the software "agents" used to receive, acknowledge, and execute the master's instructions as relayed by the handlers. These commands were sent and received using custom TCP (Transmission Control Protocol), UDP (User Datagram Protocol), or ICMP (Internet Control Message Protocol) services.

Using these nonstandard ports as communication channels, the middle tier systems acted as communications hubs by listening for connections from the master on one channel while simultaneously operating a bi-channel structure with its agents. The master would contact the handlers in order to determine how many active agents there were, to cause them to execute specific tasks, to turn them off, and in some cases, even to download updated versions of agent code. An agent would contact the handler on one channel in order to identify itself to the network and then listen on a second for commands back from the handler.

Although effective for controlling the DDOS net, this command structure also left the illicit network quite vulnerable to detection and elimination. Network intrusion detection systems were able to detect the nonstandard ports as they were being used to relay the commands. Port scanners could also be used to identify systems that were listening for these same custom services. The discovery of just one agent system could then be used to unravel the entire hacker network. This is because each agent held the hard coded IP (Internet Protocol) address of at least one handler in order to communicate with the middle tier. Conversely, a handler system had to contain the IP addresses of every one of its agents. So, by detecting and then analyzing just one agent, one or more handlers could be revealed which in turn compromised most of the network.

© SANS Institute 2000 - 2005 Author retains full rights.

Handler / Agent Example – The Stacheldracht DDOS Tool

This section will detail a typical implementation of handler / agent bot net control architecture. The “stacheldracht” (German for “barbed wire”) DDOS tool was selected as an example of this particular c^2 topology as it was a mature variant that also possessed several unique and interesting features. The technical information that follows was extracted from an analysis performed by Dave Dittrich of the University of Washington [2].

In late 1999, the stacheldracht tool began to emerge as a method of delivering various forms of DDOS attacks. This tool established a three tiered bot net that was implemented through the following programs embedded in infected hosts:

- “telnetc/client.c” - the master.
- “mserv.c.” - a handler.
- “leaf/td.c” - the agent.

The control channel between the master and handlers used a fixed nonstandard TCP port (default was 16660). One interesting improvement over earlier variants of this c^2 architecture was the encryption of this connection. By hardening the master/handler control channel in this manner, susceptibility to TCP reset attacks and session hi-jacking by rival hackers was eliminated. The “telnetc/client.c” program initiated the symmetric key encryption for these sessions. Authentication via an encrypted password was required for the master to execute commands at the handler. The password itself was then Blowfish encrypted using the passphrase "authentication" before being sent over the network to the handler.

The “mserv.c” program for each handler was coded to control a maximum of one thousand agents. A bi-channel structure was used for communication between a handler and its agents. Commands were sent to the agents over a TCP port different than that used for master/handler communications (default 65000) while responses from the agents came back embedded in ICMP echo_reply packets. Use of ICMP was another development that was unique to stacheldracht among handler / agent command and control systems. All these communications were also Blowfish encrypted using the same passphrase as above.

With the control structure thus established, the next functionality required of a bot net c^2 architecture is a means for an agent to join the network. Following the initial attack against a victim host that ended with the stacheldracht tools downloaded to it, the agent program “leaf/td.c” would begin to execute. One of the first activities initiated was for the agent to join the bot net. To join the network, an agent needed the IP address of a handler that could control it. These addresses were maintained in a master server configuration file also

© SANS Institute 2000 - 2005 Author retains full rights.

deployed to the victim. This file was Blowfish encrypted using a second passphrase, “randomsucks.” As back-up for a missing server file, there were several default handler addresses hardcoded in the agent program itself.

Having read the IP addresses from either source, the agent then attempted to contact the handlers one at a time by sending an ICMP echo_reply packet to them with the value 666 embedded in the ID (identifier) field and the string value “skillz” in the data field. Acknowledgement from a handler via echo_reply indicated that the packet had been received and that the agent was enrolled in the network. Interestingly, in comparison with the TCP control channels, no stacheldracht ICMP transmissions were either authenticated as to source or encrypted.

Once the agent joined the bot net, it then attempted to shift to a more secure communications mode by spoofing its own IP address. This was only possible if the host or victim network allowed spoofed outbound packets. So, the agent tested for this condition by transmitting an ICMP echo_reply packet with a source IP of 3.3.3.3 along with the values of 666 and the agent’s actual IP address loaded into the data field. If the packet was successfully transmitted out of the host network, the handler would reply directly back to the agent via another echo_reply packet that had the values of 1000 and the string “spoofworks” in the data field. Upon receiving this confirmation from the bot net handler, the agent then set an internal variable indicating that all four octets of its own source address could be fully spoofed. A time out on receipt of a reply from the master caused the agent to set that same variable to a different value that meant only the final octet could be spoofed.

With a communications structure established and the bot net populated with registered agents, the only remaining element required to have a fully functional c^2 architecture was a rich command set. Again, thanks to David Dittrich’s work, there is excellent documentation of the stacheldracht command shell. A summary of the key network management and attack commands that could be sent to agents by the master via a handler follows.

Key Stacheldracht Network Management Commands

Command	Purpose
.distro user server	Instructs the agent to install and run a new copy of itself using the Berkeley "rcp" (remote file copy) command, on the system "server", using the account "user." E.G. "rcp user@server:linux.bin ttymon"
.killall	Kills all active agents.

© SANS Institute 2000 - 2005 Author retains full rights.

.mping	Broadcasts a ping to all agents to see if they are alive.
.msadd	Adds a new handler server (handler) to the list of available servers.
.msort	Broadcasts a ping to all agents to sort out dead from alive agents. Shows counts/percentage of dead/alive agents.
.msrem	Removes a handler server from the list of available servers.
.showalive	Broadcasts a list of all alive agents.
.showdead	Broadcasts a list of all dead agents.

Key Stacheldracht Attack Commands

Command	Purpose
.madd ip1[:ip2[:ipN]]	Add IP addresses to list of attack victims.
.mdos	Begin DOS attack.
.micmp ip1[:ip2[:ipN]]	Begin ICMP flood attack against specified hosts.
.mlist	List IP addresses of hosts being DOS attacked.
.mstop ip1[:ip2[:ipN]] or .mstop all	Stop attacking specific IP addresses or all.
.msyn ip1[:ip2[:ipN]]	Begin SYN (Synchronize) flood attack against specified hosts.
.mtimer seconds	Set timer for attack duration.
.mudp ip1[:ip2[:ipN]]	Begin UDP flood attack against specified hosts.
.setisize	Sets size of ICMP packets for flooding. Max = 1024.
.setusize	Sets size of UDP packets for flooding Max = 1024.
.sprange lowport-highport	Sets the range of ports for SYN flooding (defaults to lowport:0, highport:140).

Note that the network command ".distro" could be used by the master to direct agents to delete their current image and then download updated agent code from misappropriated space on a server cache. This ability to provide improved code versions was yet another unique feature of the stacheldracht DDOS tools.

© SANS Institute 2000 - 2005 Author retains full rights.

The stacheldracht tool is an advanced example of the handler / agent c^2 model. First, it satisfies the basic requirements for a bot net command and control architecture by establishing a fundamental communication structure, providing a method for newly infected hosts to join the bot net, and having a rich command shell with which to direct the agents. Second, the network of captive hosts that was established used the three tiered master / handler / agent hierarchy characteristic of this first wave of DDOS nets. Finally, the stacheldracht network displayed certain advanced features of which the most notable were encrypted communications and the ability to upgrade agent programs.

IRC-Based Control Topology

A strikingly different approach to bot net command and control began to appear as early as August 2000 [1]. This new topology replaced the middle tier of handler systems with one or more IRC (Internet Relay Chat) servers such that the master could use the IRC protocol to communicate with the software agents embedded in victim hosts.

As defined by IETF RFC 2810 [3], IRC is a client-server application that provides text-based, real time, Internet conferencing services. Individual users run a client program which connects to a server in an IRC network. That server passes chat message traffic between users by communicating with other interconnected IRC servers and by acting as the central management hub for hundreds of clients. There are several IRC networks in existence that are available for public use. According to the mIRC FAQ on IRC [4], the key networks in descending order of size (number of users) are EFnet (Eris Free net), Undernet, and Dalnet.

A named communications “channel,” essentially nothing more than a logical grouping of interconnected clients, can be established over either a public or private IRC network. This channel is a key functionality that has been appropriated by hackers for control of their illicit networks. Communication from a single client (the master) can be sent out simultaneously to all the other clients on that channel. When the hacker establishes a channel for this purpose, he becomes the operator or “chan op” of that channel. The chan op can use the /mode command to increase the covertness of the bot net control channel in several ways. For example, the channel can be set up to hide the user list (“/mode+u”) which will shield the nicknames or individual client identifiers and their IP addresses from external scrutiny and as secret (“/mode+s”) so that it will not show up on channel lists. Also bots can be required to join using a key (“mode/+k”) and joining can be limited to only those clients having chan op privileges by setting the channel type as moderated (“mode/+m”). As additional security, the chan op may use special (non-text) characters in the channel name and require authentication from a client to join [5].

© SANS Institute 2000 - 2005 Author retains full rights.

At about the same time (2000), the word “bot” (short for robot) began to be used to describe the automated IRC client software loaded on the victims. IRC bots had initially been developed as a method for IRC server operators and clients to script automatic responses to specific activities that were occurring on IRC channels. Hacked versions of these bots were used to automate the establishment and maintenance of the IRC control channels that linked the bots as well as the accomplishment of various attack tasks. As the target systems became more hardened, the insertion of these malicious bots through normal IRC activity became increasingly difficult. In response, the hackers became more sophisticated and these same IRC bots began to show up as the payloads of virus, worms, and Trojans. The collection of captive hosts controlled through the use of IRC bots gradually became known as a “bot net” rather than by the older term of DDOS net.

When an IRC bot has been loaded into a victim host, it will seek to contact a predetermined IRC server through an outbound connection. IRC servers typically listen on TCP ports in the 6,000 to 7,000 range with a default port of 6667. Once connected to the server, the bot will then join the master’s private channel through use of an authenticating key and will have a unique nickname as an identifier. The master will use IRC text messaging to accomplish an authenticated log-in to the bot as a precursor to issuing commands. This authentication prevents other hackers from seizing control.

CERT (Computer Emergency Readiness Team) [1] has concluded that the use of IRC services affords greater security to bot net c[2] systems than previous approaches for the following reasons. In the earlier generation handler / agent-based architecture, the utilization of unusual TCP or UDP services often led to the detection of an agent by intrusion detection systems or through network scanners. With IRC-based control, bot net communications could occur on the same ports as legitimate chat traffic. This makes detection harder and countermeasures more difficult to enact. If all IRC traffic in the victim network were blocked as a defensive measure then legitimate users would also be denied service.

With the previous handler / agent-based architecture, forensic analysis of a compromised agent could uncover the IP address of the handler which itself might then reveal the addresses of all its agents. Should one of the captive hosts in an IRC-based bot net happen to be detected, analysis would reveal little more than an IRC server and channel name. This may or may not lead to any further action as the owners of public IRC servers are often reluctant to take a server off line merely to disable a bot net because of the overall impact to legitimate users. There remains a risk to the attacker though that these same public IRC servers are also subject to external security analysis. This risk can be mitigated through the utilization of private IRC servers and/or the use of a service that enables the attacker to elude security forces by rapidly shifting the control channel to a different IRC server.

© SANS Institute 2000 - 2005 Author retains full rights.

Regarding private IRC servers, Swatit.org [6] states that the IRCD (the IRC daemon or server application) "of choice" is Unreal IRCD [http://www.unrealircd.com] due to both its ease of installation and unique security features that can be used by hackers to increase the stealthiness of their control channels.

If an IRC control channel has been discovered by security administrators or if the channel has been banned from an IRC server, the technique of dynamic addressing enables a bot master to quickly and efficiently redirect the entire bot net to a different server. Dyndns.org is a common provider of free accounts that allow this functionality.

In summary, IRC-based c^2 is an improvement over the previous handler / agent topology in that it affords increased security to the bot net. By eliminating handler systems, the risk of compromise for an entire bot net is reduced. Also, the master of the bot net gained the ability to shift IRC servers and channels at will in order to maintain the control channel. Of course, IRC traffic is still subject to detection and blocking by network security administrators. The gains have outweighed the risks enough such that IRC-based bot net c^2 is still used by today's hackers although now it is often a secondary or alternate control channel.

IRC-Based Control Example – GT Bot

The GT Bot (global threat bot) tool is presented as the example of an IRC-based bot net c^2 system because of the widespread and continued use of its many variants. Except as otherwise noted, the following detailed information on the inner workings of GT Bot was provided by swatit.org [6] and David Dittrich at the University of Washington [7].

GT Bot is essentially a hacked version of mIRC, a legitimate shareware IRC client program for Windows hosts. mIRC enables GT Bot to connect to multiple IRC servers simultaneously. This allows for complexity in the bot net command and control system as communications can occur over different IRC networks and channels at the same time. Characteristic of the automated nature of bots, miRC also has events and commands handlers that provide automatic responses to channel activity and other IRC user actions. Another key feature is mIRC's built-in DCC (direct client-to-client) file server which enables bots to FTP files to each other [8].

When used legitimately, mIRC is operated via a display on the user's console. For hacker use, this same display must be hidden from the operator of a victim host in order to conceal the fact that the host is actively using IRC. GT Bot uses the HideWindow program to accomplish this.

© SANS Institute 2000 - 2005 Author retains full rights.

Interestingly enough, GT Bot's IRC functionality can also be used as one of its attack vectors. By deliberately resembling the legitimate mIRC program, innocent IRC users can be tricked into downloading GT Bot onto their systems.

Once GT Bot has been downloaded into a victim host, mIRC creates an IRC bot that then connects to an IRC server and channel. The GT Bot script usually lists several different servers for the bot to contact over both TCP ports 6667 and 7000. The IRC server address in the bot script is also dynamic so that the bot net can easily and quickly be pointed to another IRC server. The bot then uses the !join <channel name><key> command to connect to a specific IRC channel. Attempts at channel security by the bot net master become apparent at the IRC server level. Channel names and keys may be comprised of special characters rather than the standard ASCII text characters. One example from swatit.org [6] has a channel name of "#Ãßÿ¥€¢_¿øùô" and a key of "¥_æÅ_ ‡" which would then require the command string "!join #Ãßÿ¥€¢_¿øùô ¥_æÅ_ ‡" to connect to the channel.

After the IRC bot has joined the bot net's c^2 channel, the master will then log into the bot itself. From that point on, the bot will continue to run in the background and respond to the specific command strings that the script is monitoring the channel for. Commensurate with the increased capabilities of this hacker tool, GT Bot can be seen to have a richer command set than stacheldracht. A summary of the most significant commands identified by swatit.org [6] follows.

Key GT Bot Network Management Commands

Command	Purpose
!-	Executes any command on the remote computer/mIRC client for the master
!quit	Quits mIRC
!login	"!login grrrr yeah baby!" sets the user as master
!up	Attempts to op the $nick in the current channel.
!mode <#channel|nick> <+|-|smkiplnb> <address>	Sets a mode on a channel or nick
!jump-server <server> <port>	Tells the client to jump IRC servers
!add.server <host|ip> [port] [password]	Tells the client to add an IRC server to its server list
!update <url>	Attempts to get an update from a web page

© SANS Institute 2000 - 2005 Author retains full rights.

Key GT Bot General Attack Commands

Command	Purpose
!pfast stop !pfast <number of packets> <dest host> <dest port>	Start / stop UDP flood attack
!scan <ip.*> <port>	Execute a port scan
!packet <ip> <number>	Starts a denial of service (ping.exe) attack on the specified ip
!icppagebomb <uin> <amount> <email/name> <sub> <message>	Floods a certain user (uin) on ICQ (the original IRC instant messaging application) via www.icq.com with a page message

Key GT Bot Clone Commands

Clones are essentially multiple IRC clients that are spawned by a parent bot. They can be used to flood an IRC channel or server thus creating a denial of service condition. Clones can also harvest information from a victim host and then transmit that data back to the hacker via the bot net IRC channel.

Command	Purpose
!clone.load <hostname|ip> <port> <number of clones>	Attempts to load a set amount of clones on a selected server.
!clone.flood.ctcp.version <#channel|nick>	Attempts to flood a user or channel with CTCP (client to client protocol) ping requests.
!clone.flood.ctcp.time <#channel|nick>	Attempts to flood a user or channel with CTCP time requests.
!clone.service.killer	Attempts to flood ChanServ and NickServ by registering random channels and nicknames

© SANS Institute 2000 - 2005 Author retains full rights.

!clone.dcc.chat <nick> !clone.dcc.send <nick>	Attempts to flood a user with DCC sends/chats
!clone.c.flood	Constant flood, sets a timer to continually flood a channel or nick
!flood.stop	Stops the above flood.

Key GT Bot BNC Commands

BNC (Bounce for IRC) is an application used to load clones onto IRC networks and can circumvent a ban on the attacker's host or domain that has been implemented by an IRC server's operator.

Command	Purpose
!bnc stats	Shows statistics for the BNC (Bounce for IRC) application. See below for a more detailed explanation of BNC.
!bnc log	Starts logging to bnc.log.
!bnc start <port> <password>	Starts a bnc on <port> with <password>.
!bnc stop <port>	Kills the listening bnc on <port.>
!bnc kill users	Kill all listening and active bncs.
!bnc shutdown	Shuts down the bnc server.
!bnc list bnc	Lists all the listening bnc ports.
!bnc list users	Lists all the users currently using the bnc(s).
!bnc list servers	Lists all connects to remote servers.

Peer-to-Peer Control Topology

In mid-2004, a new form of bot net command and control emerged – the use of P2P (peer-to-peer) control channels.

According to the on-line encyclopedia Wikipedia [9], a P2P network consists of a group of peer nodes that are "…able to initiate or complete any supported transaction with any other node," essentially the antithesis of a client-server model. This architecture provides two key advantages for the bot net master – one, there are multiple paths through which commands can be issued and two, the loss of any one node has little or no effect on the rest of the network. To

© SANS Institute 2000 - 2005 Author retains full rights.

effectively cripple a P2P c^2 system, literally every single node has to be detected and taken off line.

The bot in a newly compromised victim host must discover the other nodes on the P2P network in order to join. This can be done by scripting the bot to contact a legitimate and otherwise uninvolved caching server in order to obtain a list of those other nodes. Once the bot is on the channel, the master merely logs into it in order to enable it to receive further commands. Those commands are transmitted as text messaging traffic.

Peer-to-Peer Control Example -- PhatBot

In March 2004, alarming reports like the following from washingtonpost.com [10] announced the arrival of a Trojan that used a P2P bot net topology:

> Computer security experts in the private sector and U.S. government are monitoring the emergence of a new, highly sophisticated hacker tool that uses the same peer-to-peer (P2P) networking abilities that power controversial file-sharing networks like Kazaa and BearShare.
>
> …The tool, a program that security researchers have dubbed "Phatbot," allows its authors to gain control over computers and link them into P2P networks that can be used to send large amounts of spam e-mail messages or to flood Web sites with data in an attempt to knock them offline.

According to the LURHQ Threat Intelligence Group [11], PhatBot was not the first hacker tool to use P2P for c^2 though. A predecessor named AgoBot had used P2P as an alternate control methodology while still utilizing IRC as its primary channel. PhatBot however used P2P as its primary control channel. The P2P application used by PhatBot is a hacked variant of WASTE which is a communications and file sharing application originally developed for AOL (America OnLine). AOL never used this application though and the code was made briefly available to the open source community by one of the programmers.

The following information on WASTE was obtained from the application's open source development website [12] and from an article posted on instantmessaging planet.com [13]. WASTE establishes a decentralized network of peer FTP (File Transfer Protocol)-like server nodes that provide a messaging and file sharing functionality. Unlike other P2P protocols, WASTE requires no central server to act as a repository for lists of connected hosts and files available for share. Instead, each WASTE bot provides the complete functionality creating a true distributed network. Traffic between nodes is routed along the path of least latency which results in a de facto form of simple load balancing.

© SANS Institute 2000 - 2005 Author retains full rights.

Although the links that are established between nodes can be encrypted using 128-bit RSA, the hacked variant of WASTE used in PhatBot has not been reported to use this feature. This is believed to be due to the difficulty involved in sharing public keys among the bots [11].

WASTE uses three general classes of messages for network management. A node can send out a broadcast message type when it wishes to notify or request information from every other node. A routed reply type message will be sent back to the host that initiated the broadcast. Finally, local management messages are exchanged between two nodes to set up the link between them. 1377 is the default port used by WASTE.

Although the WASTE P2P application was initially intended to create relatively small peer networks of up to fifty nodes, PhatBot nets of up to 400,000 linked bots have been reported by the IEEE [14].

The analysis by LURHQ [11] explains that when the bot in an infected host is ready to join the hacker's WASTE P2P network, it is scripted to contact a Gnutella (another popular and widely used P2P application) cache server. Using a CGI script provided by the server, PhatBot then registers itself as a fake Gnutella client, using TCP port 4387 instead of the standard Gnutella port, in order to access a list of peers placed there by other PhatBots. To join the P2P net, the new PhatBot node then merely connects to a peer from the list. With the bot now on the net's control channel, the master can take control by authenticating via a username and password that is embedded in the bot's binary as an MD5sum. Command and control has been established at that point.

PhatBot has an extremely rich command set illustrative of the wide range of destructive abilities for which the bot is also well known for. A list of representative commands showing each of these functional areas follows.

Key PhatBot P2P WASTE Commands

Command	Purpose
waste.server	Changes the server the bot connects to.
waste.reconnect	Reconnects to the server.
waste.quit	Quits the bot.
waste.privmsg	Sends a private message.
waste.netinfo	prints netinfo
waste.mode	Has the bot execute a mode change.
waste.join	Join a new channel.

© SANS Institute 2000 - 2005 Author retains full rights.

waste.gethost	Prints netinfo when host matches
waste.getedu	Prints netinfo when the bot is in an.edu host
waste.action	Has the bot execute an action.
waste.disconnect	Disconnects the bot from WASTE.

Key PhatBot Bot-level Management Commands

Command	Purpose
bot.unsecure	Enable shares / enable dcom (distributed component object model).
bot.secure	Delete shares / disable dcom.
bot.flushdns	Flushes the bot's DNS cache.
bot.quit	Quits the bot.
bot.longuptime	If uptime > 7 days then bot will respond.
bot.sysinfo	Displays the system info.
bot.status	Gives status.
bot.rndnick	Makes the bot generate a new random nickname.
bot.removeallbut	Removes the bot if id does not match.
bot.remove	Removes the bot.
bot.open	Opens a file (name).
bot.nick	Changes the nickname of the bot.
bot.id	Displays the id of the current node.
bot.execute	Makes the bot execute an .exe type file.
bot.dns	Resolves IP/hostname by DNS.
bot.die	Terminates the bot.
bot.about	Displays whatever info the author wants you to see.

Key PhatBot DDOS Attack Commands

Command	Purpose
ddos.phatwonk	Starts phatwonk flood.
ddos.phaticmp	Starts phaticmp flood.
ddos.phatsyn	Starts phatsyn flood.

© SANS Institute 2000 - 2005 Author retains full rights.

ddos.stop	Stops all floods.
ddos.httpflood	Starts a HTTP flood.
ddos.synflood	Starts an SYN flood.
ddos.udpflood	Starts a UDP flood.

Key PhatBot Data Harvesting Commands

Command	Purpose
harvest.aol	Harvest AOL (America Online) log-in data.
harvest.cdkeys	Harvest a list of cdkeys.
harvest.emailshttp	Harvest a list of emails via http.
harvest.emails	Harvest a list of emails.

Key PhatBot Redirect Commands

Command	Purpose
redirect.stop	Stops all redirects running
redirect.https	Starts a HTTPS proxy
redirect.http	Starts a HTTP proxy
redirect.gre	Starts a GRE (Generic Routing Encapsulation) redirect
redirect.tcp	Starts a TCP port redirect

Key PhatBot FTP Commands

Command	Purpose
ftp.update	Updates PhatBot from an FTP url.
ftp.execute	Executes a file from an FTP url.
ftp.download	Downloads a file via FTP.

© SANS Institute 2000 - 2005 Author retains full rights.

Emerging Control Trends

When one inquires about the trends in bot net c[2] that are currently being observed by the top U.S. forensics and security labs, a shroud of secrecy quickly descends as far as specifics are concerned. Much information is prevented from becoming open source in order to allow a full analysis and the development of countermeasures before the techniques under study are general knowledge in the hacker community. Yet by piecing together some general observations and the few details that are available, a picture does come into focus – it is that bot net masters are achieving new and challenging levels of stealthiness with their control channels.

In a presentation by CERT in November 2004 [[15]], two emerging and extremely covert bot net control channels were identified -- background noise and illegal fragmented traffic.

Background noise refers to the use of pulses (sudden increases and decreases) of DNS, HTTP, and other network traffic as a means of communication between the master and the bots. The bots are scripted to recognize specific thresholds in the rate that these packets are received and respond by performing certain actions. Detection of this use of these otherwise legitimate services is quite difficult as it requires an excellent sense of baseline traffic levels which can often vary widely from hour-to-hour and day-to-day during normal operations. In fact, even completely normal changes in traffic patterns could be used to trigger attack events.

The use of illegal and fragmented traffic as a control vector is another emerging trend. This involves the embedding of commands or data in hacked TCP packets. One specific example of this was recently described by Lancope, Inc., a security services vendor [[16]]. Lancope reported that a specific window size (55808 bytes) was being used in the headers of hacked TCP SYN packets to flag bots that those packets contained control information for them. This control information was encrypted and found in other header fields such as sequence number and port number. As to detection, Dr. John Copeland, founder and Chief Scientist at Lancope, observed, “This new generation of Trojan horses makes it far more difficult to detect either the Controller IP address or the Trojan-infected hosts…”

© SANS Institute 2000 - 2005 Author retains full rights.

Summary

In an evolutionary pattern similar to that evoked by Darwinian forces in the natural world, the command and control architecture of bot nets has increased in robustness over the last six years to enhance their survivability. Two key factors can be seen to influence the overall robustness of a control vector – detectability and redundancy.

A move away from the use of custom TCP and UDP services and toward IRC as a control vector made it much more difficult to separate illicit bot net traffic from legitimate activity thus hindering detection. The concurrent shift from a handler / agent topology to IRC and P2P-based channels also eliminated the requirement for mid-level command hosts that if compromised by network security administrators could ultimately reveal the IP addresses and host names of the entire bot net. The emergent use of noise- and fragmented traffic-based control channels will further hinder efforts to detect bot net control traffic and thus enhance their survivability.

The second key factor in preserving the bot net command structure has been the trend toward redundant control paths. With the handler / agent architecture, elimination of a single handler by security administrators or rival bot net masters would wipe out control of every agent that relied upon it. Subsequent IRC-based implementations enabled a bot net master to use and rapidly shift between multiple IRC servers and encrypted channels in order to preserve control. Optimum realization of redundant control paths has been achieved through the current use of peer-to-peer topologies. With this latest approach, no one server or link outage can take down a bot net's c^2 channel.

© SANS Institute 2000 - 2005 Author retains full rights.

References

1. Houle, Kevin, et al. "Trends in Denial of Service Technology." October 2001. CERT Coordination Center, Carnegie Mellon University. 29 January 2005. <http://www.cert.org/archive/pdf/DoS_trends.pdf >

2. Dittrich, David. "The "stacheldracht" Distributed Denial of Service Attack Tool." 31 December 1999. University of Washington. 29 January 2005. <http://www.sans.org/y2k/stacheldraht.htm>

3. Roeckx, Kurt, et al. "Internet Relay Chat: Architecture Request for Comments: 2810." April 2000. 29 January 2005. <http://www.irchelp.org/irchelp/rfc/rfc2810.txt>

4. Vonck, Tjerk. "IRC FAQ. Introduction to IRC for people using Windows." 2004. mIRC Co. Ltd. 29 January 2005. <http://www.mirc.com/irc.html>

5. Navratilova, Viki. "Today's Modern Network Killing Robot." 29 March 2003. University of Chicago. 31 January 2005. <www.uniforum.chi.il.us/slides/ killer_robots/network-killing-robot.ppt>

6. "Bots, Drones, Zombies, Worms and other things that go bump in the night. GT Bot (Global threat)." 2003. Swatit.org. 29 January 2005. <http://swatit.org/bots/gtbot.html>

7. Dittrich, David. "Subject: World-wide distributed DoS and 'warez' bot networks." May 2002. 29 January 2005. < www.derkeiler.com/Mailing-Lists/ securityfocus/incidents/2002-09/0044.html>

8. "Introduction to mIRC." mIRC Co. Ltd. 2004. 29 January 2005. <http://www.mirc.com/mirc.html>

9. "Peer to Peer." Wikipedia Free On-line Encyclopedia. 14 Mar 2005. 15 Mar 2005. <http://en.wikipedia.org/wiki/Peer_to_peer>

10. Krebs, Brian. "Hackers Embrace P2P Concept." washingtonpost.com. 17 March 2004. 1 March 2005. < http://www.washingtonpost.com/wp-dyn/articles/A444-2004Mar17.html>

11. "PhatBot." 15 March 2004. LURHQ Threat Intelligence Group. 3 February 2005. <http://www.lurhq.com/phatbot.html>

© SANS Institute 2000 - 2005 Author retains full rights.

12. "WASTE Information." 2004. WASTE Development Group. 10 March 2005. <http://waste.com/information.html>

13. Saunders, Christopher. "Not a WASTE of Time." 15 January 2004. Instantmessagingplanet.com.10 March 2005. <http://www.instantmessagingplanet.com/enterprise/article.php/3300391>

14. McLaughlin, Laurianne. "Bot Software Spreads, Causes New Worries." IEEE DISTRIBUTED SYSTEMS ONLINE Vol. 5, No. 6. June 2004. 13 December 2004. <csdl.computer.org/comp/mags/ds/2004/06/o6001.pdf>

15. "Cyber Security: A Global Perspective." CERT Coordination Center, Carnegie Mellon University. 2004. CERT Presentation to Evergreen State InfraGard Members Alliance. November 2004.

16. "VIRUS ALERT: Lancope Confirms Discovery of Third-Generation Internet Trojan Horse." Lancope Inc. 9 June 2003. 22 February 2005. <http://www.lancope.com/news/Virus_Alert_Trojan.htm>

© SANS Institute 2000 - 2005, Author retains full rights.